USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

OPTIMIZACIÓN DE PROBLEMAS CON ECUACIÓNS DIFERENCIAIS ALXÉBRICAS

Iago Comesaña Soliño

Curso 2023/2024

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

OPTIMIZACIÓN DE PROBLEMAS CON ECUACIÓNS DIFERENCIAIS ALXÉBRICAS

Iago Comesaña Soliño

Xullo, 2024

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Traballo proposto

Área de Coñecemento: Estatística e Investigación Operativa
Título: Optimización de problemas con ecuacións diferenciais alxébricas
Breve descrición do contido
As ecuacións diferencias alxébricas (DAEs, polas súas siglas en inglés) son a forma máis natural de modelizar matematicamente moitos sistemas físicos complexos que xorden no campo da ciencia e enxeñaría. Ademais, permiten crear modelos separados para cada subcompoñente dun sistema que logo se poden conectar mediante unha rede. Unha vez tense modelizado un certo sistema complexo empregando DAEs, pódese formular un problema de optimización matemática que inclúa como parte das súas restricións tanto ecuacións diferenciais como alxébricas, que permitan optimizar o funcionamento do sistema físico subxacente. A clase máis xeral deste tipo de problemas son os coñecidos como problemas de control óptimo. Para poder resolver estes problemas é necesario empregar algoritmos específicos que combinen técnicas de optimización xerais con métodos de resolución de ecuacións diferenciais. Neste traballo farase unha revisión dos problemas de optimización con DAEs. A estrutura do mesmo centrarase nos seguintes aspectos: • Introdución aos problemas de optimización con DAEs e principais resultados teóricos. • Técnicas de resolución de problemas de control óptimo. • Aplicación en problemas baseados en control óptimo de procesos industriais.

Índice

Resumo

A optimización de problemas con restricións de tipo diferencial descríbese a través dos coñecidos como problemas de control óptimo, obxecto de estudo do traballo. Así pois, alén dunha presentación dos sistemas de ecuacións diferenciais e alxébricas, introdúcese esta clase de problemas. Apórtanse comentarios e interpretacións sobre a súa definición e lévase a cabo unha breve discusión teórica, na que se enuncia o principio do máximo de Pontryagin, resultado de grande importancia nesta área que significa unha primeira forma de atopar unha solución. A continuación, expóñense os métodos directos de resolución máis coñecidos, que se basean nunha discretización do problema orixinal, dando lugar a un de programación non linear atacable con ferramentas habituais de optimización como a programación cuadrática secuencial. Ademais, estes métodos compleméntanse con dous algoritmos iterativos de refinamento da malla para mellorar o erro da aproximación discreta obtida. Para finalizar, resólvense unha serie de exemplos mediante un código orixinal implementado en MATLAB.

Abstract

The optimization of problems with differential constraints is described through the so-called optimal control problems, focus of this work. Therefore, after a presentation of the systems of differential algebraic equations, this kind of problems is introduced. Remarks and interpretations on its definition are provided, including also a brief theorical discussion, in which the maximum principle of Pontryagin is formulated, a highly important result on this field that means a first way of finding a solution. Next, the most frequently used direct methods for resolution are explained, which are based on a discretization of the original problem, leading to a nonlinear programming one which can be solved by using usual approaches such as sequential quadratic programming. Moreover, these methods are complemented with iterative algorithms of mesh refinement in order to improve the error of the discrete approximation obtained. Finally, some illustrative examples are solved employing an original code implemented in MATLAB.

Introdución

Os problemas de programación matemática supoñen unha ferramenta realmente útil para construír modelos e resolver unha gran cantidade de desafíos en áreas académicas e industriais. Dende conceptos básicos que se ensinan no primeiro nivel universitario como a optimización linear ou convexa, até máis avanzados como a programación enteira ou non linear, constitúen un campo de investigación moi activo hoxe en día.

Neste sentido, a programación matemática considera problemas cun número finito de variables, cuxas restricións adoitan ser sistemas alxébricos de maior ou menor dificultade. Porén, en ocasións interesa estender esta metodoloxía para poder contemplar sistemas físicos cuxa modelización involucra ecuacións diferenciais. Así xorden os coñecidos como problemas de control óptimo, interese principal deste traballo e que se poden formular como problemas de programación matemática de dimensión infinita. As novas variables son funcións escalares ou vectoriais dependentes dunha variable real cuxo comportamento se quere optimizar respectando un conxunto de restricións diferenciais e alxébricas. Cabe mencionar que é habitual que a variable independente sexa o tempo, malia poder representar outras magnitudes físicas.

Deste xeito, os problemas de control óptimo supoñen a unión de dúas áreas que de por si soas requiren cantidade de estudo. Cuns inicios relativamente tardíos da man do cálculo de variacións, pasando por traballos coma os de Leonhard Euler, Joseph-Louis Lagrange, Carl Gustav Jacob Jacobi ou Richard Ernest Bellman, a súa explosión acadouse coa publicación do principio do máximo por Lev Pontryagin a mediados do século XX e o desenvolvemento da computación. Neste sentido, dende entón non escasean as publicacións na área, con grande interese tamén a nivel industrial.

Así pois, este traballo presume dunha presentación dos problemas de control óptimo no Capítulo 1 con apoio de [9], comezando cunha introdución aos sistemas de ecuacións diferenciais e alxébricas e os seus tipos de variables. Tras explicar os conceptos de traxectoria e restricións de puntos interiores, enúncianse tres exemplos que servirán de axuda para ilustrar aspectos teóricos e prácticos: un problema de creación propia, unha reacción química introducida en [3] e un movemento presentado en [7]. Entón, defínese o problema básico de control óptimo,

acompañado de comentarios sobre a formulación da función obxectivo, o concepto de solución óptima e a extensión a restricións por intervalos. Por último, tamén se ofrece a visión xeométrica de Pontryagin ([8]), que pode resultar de utilidade para probar resultados teóricos.

Nesta liña, no Capítulo 2 enúnciase un dos resultados máis importantes na teoría de control óptimo: o principio do máximo de Pontryagin. Este especifica unha serie de condicións necesarias que ten que verificar a solución óptima dun problema de control óptimo básico, converténdoo nun sistema de ecuacións diferenciais. Porén, malia que as extensións deste teorema cobren unha gran variedade de casuísticas, esta estratexia de resolución fica en moitas ocasións substituída polos métodos directos, baseados nunha transcrición do problema a dimensión finita. Noutros termos, o problema orixinal discretízase para obter un de programación non linear, que se pode abordar con ferramentas máis estudadas.

Daquela, no Capítulo 3 explícanse os métodos de *direct single shooting*, *direct multiple shooting* e *collocation*, xunto cunha sección adicada a dar unha idea de como se resolvería o problema discreto resultante. En particular, faise especial fincapé na *collocation*, ofrecendo varias posibilidades con esquemas Runge-Kutta ou interpolación de Lagrange, xa que nos Capítulos 4 e 5 será a estratexia empregada.

Así, no Capítulo 4 compleméntase a estratexia dos métodos directos con algoritmos iterativos propostos en [1] e [3], que pretenden refinar a discretización feita para obter solucións discretas máis próximas á solución real. En concreto, amósase un primeiro algoritmo que implica ir refinando progresivamente a partición do intervalo de definición das variables, para despois presentar un segundo algoritmo no que as lonxitudes dos elementos da partición se inclúen como variables. Neste sentido, mentres a primeira estratexia é máis doada de trasladar á práctica, a segunda ten tamén as súas vantaxes á hora de detectar descontinuidades e reducir os pasos de busca.

Por último, elaborouse o código de MATLAB ([6]) do Anexo A para ilustrar no Capítulo 5 os resultados de aplicar a primeira estratexia de resolución nos exemplos do traballo, xunto con outro problema de [3] para verificar a eficacia dos programas. Vese que a estratexia de discretización é válida e coméntanse as posibles melloras e ampliacións para os programas.

Co cal, o traballo abrangue dende a presentación do problema de optimización até a explicación completa de varias posibilidades de resolución, comentando e exemplificando as estratexias. Como traballo futuro existen varios puntos interesantes que non puideron ser tratados: a elaboración de código propio para o segundo algoritmo iterativo, a posta en práctica doutros métodos de resolución ou incluso o afondamento en resultados teóricos de caracterización da solución.

Capítulo 1

Problemas de control óptimo

Neste primeiro capítulo preséntase nas seccións 1.1 e 1.2 a forma xeral dos problemas obxecto de estudo e introdúcense tres exemplos prácticos que se analizan en paralelo. Ademais, na sección 1.3, unha visión xeométrica do problema axuda a comprender dende outra perspectiva o que se está a facer.

1.1. Ecuacións diferenciais alxébricas

Dende o cálculo de traxectorias na industria aeroespacial até os cambios na composición de aliaxes na enxeñaría química, unha gran cantidade de procesos complexos poden ser modelizados mediante sistemas de ecuacións diferenciais alxébricas (*differential algebraic equation*, DAE).

Dunha banda, neste tipo de sistemas, a variable independente $t \in [t_0, t_f] \subset \mathbb{R}$ adoita representar o tempo variando dende un instante inicial t_0 até o instante t_f. Doutra banda, as variables dependentes son as coñecidas como *variables de estado* $\boldsymbol{y} : [t_0, t_f] \to \mathbb{R}^{n_y}$, que describen o estado do sistema correspondente e son variables incógnita, e as *variables de control* $\boldsymbol{u} : [t_0, t_f] \to \mathbb{R}^{n_u}$, que na práctica poden ser introducidas manualmente e permiten controlar o proceso subxacente. Adicionalmente, poden existir variables de tipo parámetro, $\boldsymbol{p} \in \mathbb{R}^{n_p}$, que non dependen de t.

Neste contexto, un sistema de ecuacións diferenciais alxébricas en forma implícita pódese representar por:

$$\boldsymbol{0} = \tilde{\boldsymbol{f}}(t, \dot{\boldsymbol{y}}(t), \boldsymbol{y}(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{1.1}$$

onde $\dot{\boldsymbol{y}} = \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} : [t_0, t_f] \to \mathbb{R}^{n_y}$ e $\tilde{\boldsymbol{f}} : [t_0, t_f] \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_y}$.

Nalgunhas referencias como [9] é común atopar unha distinción entre variables de estado diferenciais e alxébricas (é dicir, se intervén a súa derivada temporal ou non). Desta situación é de

onde xorde o concepto de DAE e escríbese $\boldsymbol{y} = (\boldsymbol{x}, \boldsymbol{z})$, con $\boldsymbol{x}$ denotando as variables diferenciais e $\boldsymbol{z}$ as alxébricas. Porén, noutra literatura como [1], as variables diferenciais identifícanse coas de estado, e as alxébricas coas de control. Neste traballo en principio adoptarase a primeira convención, se ben non é de gran relevancia para o desenvolvemento de resultados teórico-prácticos, pois a diferenza atópase realmente na interpretación.

Ademais, para a formulación teórica é habitual escribir o sistema en xeito semi-explícito:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}), \\ \boldsymbol{0} &= \boldsymbol{g}(t, \boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}), \end{aligned} \tag{1.2}$$

con $\boldsymbol{f} : [t_0, t_f] \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$ e $\boldsymbol{g} : [t_0, t_f] \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_z}$. Non obstante, malia ser a forma que se empregará neste traballo, cómpre ter presente que na práctica non sempre se pode asumir expresar as derivadas temporais no lado esquerdo da ecuación.

En canto ás hipóteses sobre as variables, para facilitar a dedución de resultados teóricos asúmese que as funcións de control son medibles, $\boldsymbol{u} \in \mathfrak{U}_m = \{\boldsymbol{u} : [t_0, t_f] \to \mathbb{R}^{n_u}, \boldsymbol{u}(\cdot) \text{ medible}\}$, e que $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} \in \mathbb{R}^{n_z \times n_z}$ é regular, de xeito que a diferenciación con respecto a t transformaría o sistema (1.2) nun conxunto de ecuacións diferenciais ordinarias. O que é máis, para garantir a existencia e unicidade de solución $\boldsymbol{y}$, a función $\boldsymbol{f}$ suponse continua e Lipschitz a cachos.

Amais, en calquera sistema da forma (1.1) ou (1.2) pódese suprimir a dependencia explícita de t sen máis que engadir unha variable adicional coa ecuación:

$$\dot{x}_{n_x+1} = 1,$$

tal que $x_{n_x+1}(t_0) = t_0$, transformándose nun sistema *autónomo*. Daquela, de aquí en diante, t non será un argumento de $\boldsymbol{f}$ (nin de $\boldsymbol{g}$).

Definición 1.1 (Traxectoria)**.** Unha traxectoria do sistema (1.2) vén dada por:

$$\mathcal{T} = (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}) = \{(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}) |\ t \in [t_0, t_f]\},$$

onde as funcións $\boldsymbol{x} : [t_0, t_f] \to \mathbb{R}^{n_x}$, $\boldsymbol{z} : [t_0, t_f] \to \mathbb{R}^{n_z}$, $\boldsymbol{u} : [t_0, t_f] \to \mathbb{R}^{n_u}$ e os parámetros $\boldsymbol{p} \in \mathbb{R}^{n_p}$ satisfán (1.2). As compoñentes $\boldsymbol{x}(t)$ e $\boldsymbol{z}(t)$ denomínanse estados da traxectoria.

Á hora de atopar traxectorias do sistema (1.2) cómpre resolver un sistema de DAEs. Para este propósito, poden ser proporcionados valores iniciais de xeito implícito ou explícito. De forma xeral, dados $\{\bar{t}_1, \ldots, \bar{t}_{n-1}\}$ puntos interiores do intervalo $[t_0, t_f]$, imponse unha serie de restricións que relacionan os valores dos estados nos puntos mencionados:

$$\boldsymbol{0} = \boldsymbol{r}(\boldsymbol{x}(t_0), \boldsymbol{z}(t_0), \boldsymbol{x}(\bar{t}_1), \boldsymbol{z}(\bar{t}_1), \ldots, \boldsymbol{x}(\bar{t}_{n-1}), \boldsymbol{z}(\bar{t}_{n-1}), \boldsymbol{x}(t_f), \boldsymbol{z}(t_f), \boldsymbol{p}), \tag{1.3}$$

onde $\boldsymbol{r} : (\mathbb{R}^{n_x} \times \mathbb{R}^{n_z}) \times \overset{(n+1)}{\cdots} \times (\mathbb{R}^{n_x} \times \mathbb{R}^{n_z}) \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_r}$.

Agora ben, en moitos problemas os valores iniciais das variables diferenciais apórtanse explicitamente como:

$$\boldsymbol{x}(t_0) = \boldsymbol{x_0},$$

e os valores iniciais para as variables alxébricas, $\boldsymbol{z}(t_0)$, son determinados como a solución de:

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{x}(t_0), \boldsymbol{z}(t_0), \boldsymbol{u}(t_0), \boldsymbol{p}),$$

onde $\boldsymbol{u}(\cdot)$ e $\boldsymbol{p}$ son dados.

Daquela, preséntanse a continuación os Exemplos 1.2, 1.3 e 1.4 para ilustrar a utilidade dos sistemas de DAEs e traballar sobre eles ao longo do traballo.

Exemplo 1.2 (Freada dun bloque)**.** Para dar unha visión sinxela de como funcionan as ecuacións diferenciais alxébricas, optouse por introducir este exemplo de creación propia. Malia que o modelo subxacente é sinxelo e non de grande utilidade real, servirá para ilustrar como son os problemas de control óptimo suxeitos a sistemas de DAEs. Ademais, suporase que se traballa en [km] e [min] como unidades de referencia de espazo e tempo, respectivamente.

Suponse entón que un operario dunha canteira pretende axudar a aminorar a velocidade dun bloque dun quilogramo de pedra que se move a velocidade constante. Así, aplícalle unha aceleración contraria ao movemento, $u(t)$, xunto coa aceleración automática, $z(t)$, que lle aplica o mecanismo de freada habitual non controlado polo operario. En particular, a suma de cadrados das aceleracións ten que ser igual a 4 unidades, pois estudos da fábrica avalan que é o ideal para non romper a peza e aproveitar que as máquinas están acesas. Deste xeito, se $x(t)$ denota a velocidade da peza no instante t, esta situación pódese describir de xeito simplificado como o seguinte sistema de DAEs:

$$\begin{aligned} \dot{x}(t) &= -z(t) - u(t), \\ 4 &= (z(t))^2 + (u(t))^2, \end{aligned}$$

para todo t en $[0, t_f]$, con $t_f > 0$. Ademais, pódese complementar cunha condición inicial para a variable de estado $x(t)$, supoñendo que a velocidade inicial verifica $x(0) = 3$. Ilústrase a situación na Figura 1.1.

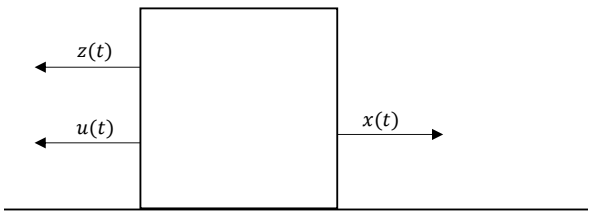

Figura 1.1: Representación das variables do problema de freada dun bloque do Exemplo 1.2.

Exemplo 1.3 (Reactor descontinuo)**.** Considérese agora un reactor químico de tipo intermitente ou descontinuo, onde teñen lugar dúas reaccións químicas paralelas de primeira orde: A $\rightarrow$ B e A $\rightarrow$ C. Este tipo de equipos caracterízanse porque non existe fluxo de entrada de reactivos nin de saída de produtos, é dicir, ao comezo do proceso introdúcense todos os elementos e materiais necesarios e non se extrae nada do interior até a finalización. Supóñase ademais que non é isotérmico (polo que a temperatura varía co tempo) e que o produto de interese é B.

Este sistema pódese modelizar, por exemplo, mediante un sistema de DAEs do tipo (1.2), en concreto mediante un sistema de ecuacións diferenciais ordinarias (véxase [3]). En efecto, sexan no instante t: $x_1(t)$ a cantidade do reactivo A, $x_2(t)$ a cantidade de produto B e $u(t)$ a temperatura do interior do reactor, que pode ser modificada polo usuario. Daquela, nun intervalo temporal $[0, t_f]$, as cantidades de reactivo A e produto B varían segundo o seguinte sistema:

$$\begin{aligned} \dot{x}_1(t) &= -x_1(t)\left(u(t) + \frac{(u(t))^2}{2}\right), \\ \dot{x}_2(t) &= x_1(t)u(t), \end{aligned}$$

e no caso de que se introduza unha unidade de reactivo A, as condicións inicias serían:

$$x_1(0) = 1, \quad x_2(0) = 0.$$

Nótese que a cantidade de produto C se obtén facilmente a partir de A e B baixo a suposición de que a masa do conxunto se conserva, como se observa na Figura 1.2.

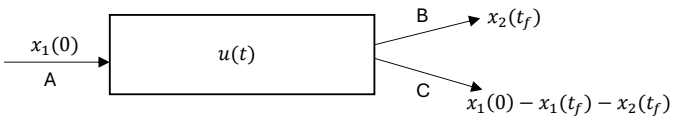

Figura 1.2: Representación das variables do problema do reactor descontinuo do Exemplo 1.3.

Exemplo 1.4 (Carro con péndulo)**.** Na mesma liña, no artigo [7] modelízase un sistema físico mediante ecuacións diferenciais. Este sistema consiste nun carro que se despraza ao longo dunha pista plana cun péndulo colgando libremente. O carro móvese grazas á forza dun motor e é posible partir dunha posición inicial de repouso, co péndulo colgando verticalmente cara abaixo, para finalizar o movemento co péndulo na posición oposta: vertical cara arriba.

Sexan, no instante de tempo t, $x_1(t)$ a posición do carro, $x_2(t)$ o ángulo que forma o péndulo respecto da posición inicial e $u(t)$ a forza exercida a través do motor. Sexan ademais m_1 e m_2 as masas do carro e do péndulo, l a lonxitude do péndulo e g a aceleración da gravidade. Deste xeito, con t en $[0, t_f]$, o sistema queda determinado polas seguintes ecuacións diferenciais de segunda

orde:

$$\ddot{x}_1(t) = \frac{lm_2 \operatorname{sen}(x_2(t))(\dot{x}_2(t))^2 + u(t) + m_2 g \cos(x_2(t)) \operatorname{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))},$$
$$\ddot{x}_2(t) = -\frac{lm_2 \cos(x_2(t)) \operatorname{sen}(x_2(t))(\dot{x}_2(t))^2 + u(t)\cos(x_2(t)) + (m_1 + m_2) g \operatorname{sen}(x_2(t))}{lm_1 + lm_2(1 - \cos^2(x_2(t)))},$$

que se poden reescribir como un conxunto de ecuacións diferenciais de primeira orde engadindo dúas variables de estado, $x_3(t)$ e $x_4(t)$:

$$\dot{x}_1(t) = x_3(t),$$
$$\dot{x}_2(t) = x_4(t),$$
$$\dot{x}_3(t) = \frac{lm_2 \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t) + m_2 g \cos(x_2(t)) \operatorname{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))},$$
$$\dot{x}_4(t) = -\frac{lm_2 \cos(x_2(t)) \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t)\cos(x_2(t)) + (m_1 + m_2) g \operatorname{sen}(x_2(t))}{lm_1 + lm_2(1 - \cos^2(x_2(t)))}.$$

Adicionalmente, engádense as condicións iniciais e finais para que se cumpra o descrito:

$$x_1(0) = 0, \quad x_2(0) = 0, \quad x_3(0) = 0, \quad x_4(0) = 0,$$
$$x_1(t_f) = d, \quad x_2(t_f) = \pi, \quad x_3(t_f) = 0, \quad x_4(t_f) = 0,$$

sendo d a posición final do carro que se pretende acadar. De novo, na Figura 1.3, amósase unha imaxe para entender ben o que está a suceder.

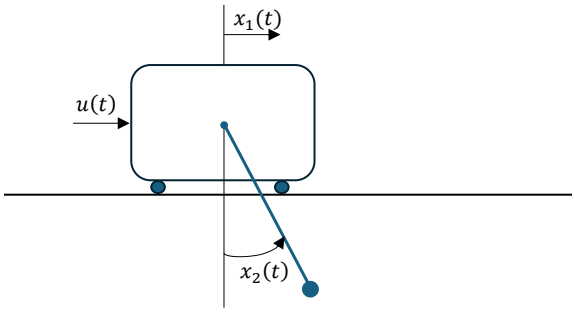

Figura 1.3: Representación das variables do problema do carro con péndulo do Exemplo 1.4.

1.2. Formulación do problema

Nun problema de control óptimo, o obxectivo é determinar as funcións de control $\boldsymbol{u}(\cdot)$, parámetros $\boldsymbol{p}$ e variables de estado $\boldsymbol{x}(\cdot)$ e $\boldsymbol{z}(\cdot)$ que minimicen unha certa función obxectivo e con traxectorias que verifiquen as ecuacións (1.2), (1.3) e restricións adicionais.

Definición 1.5 (Problema de control óptimo, OCP)**.** Un problema de control óptimo (*optimal control problem*, OCP) é un problema de optimización con restricións da forma seguinte:

$$\min_{\boldsymbol{x},\boldsymbol{z},\boldsymbol{u},\boldsymbol{p}} \Phi[\boldsymbol{x},\boldsymbol{z},\boldsymbol{u},\boldsymbol{p}] \tag{1.4a}$$

suxeito ao sistema de DAEs:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p}), \quad t\in[t_0,t_f], \\ \boldsymbol{0} &= \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p}), \quad t\in[t_0,t_f], \end{aligned} \tag{1.4b}$$

restricións de control e traxectoria:

$$\boldsymbol{0} \le \boldsymbol{c}(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p}), \quad t\in[t_0,t_f], \tag{1.4c}$$

e igualdades e desigualdades de puntos interiores:

$$\begin{aligned} \boldsymbol{0} &= \boldsymbol{r}(\boldsymbol{x}(t_0),\boldsymbol{z}(t_0),\boldsymbol{x}(\bar{t_1}),\boldsymbol{z}(\bar{t_1}),\dots,\boldsymbol{x}(t_f),\boldsymbol{z}(t_f),\boldsymbol{p}), \\ \boldsymbol{0} &\le \tilde{\boldsymbol{r}}(\boldsymbol{x}(t_0),\boldsymbol{z}(t_0),\boldsymbol{x}(\bar{t_1}),\boldsymbol{z}(\bar{t_1}),\dots,\boldsymbol{x}(t_f),\boldsymbol{z}(t_f),\boldsymbol{p}). \end{aligned} \tag{1.4d}$$

As variables t, $\boldsymbol{x}(t)$, $\boldsymbol{z}(t)$, $\boldsymbol{u}(t)$ e $\boldsymbol{p}$, e as funcións $\boldsymbol{f}$, $\boldsymbol{g}$ e $\boldsymbol{r}$ son as xa introducidas na sección 1.1, mentres que o resto das restricións están definidas como $\boldsymbol{c}:\mathbb{R}^{n_x}\times\mathbb{R}^{n_z}\times\mathbb{R}^{n_u}\times\mathbb{R}^{n_p}\to\mathbb{R}^{n_c}$ e $\tilde{\boldsymbol{r}}:(\mathbb{R}^{n_x}\times\mathbb{R}^{n_z})\times\overset{(n+1)}{\cdots}\times(\mathbb{R}^{n_x}\times\mathbb{R}^{n_z})\times\mathbb{R}^{n_p}\to\mathbb{R}^{n_{\tilde{r}}}$. Ademais, a función obxectivo:

$$\Phi[\boldsymbol{x},\boldsymbol{z},\boldsymbol{u},\boldsymbol{p}] := \int_{t_0}^{t_f} L(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p})\mathrm{d}t + E(\boldsymbol{x}(t_f),\boldsymbol{z}(t_f),\boldsymbol{p})$$

dise de *tipo Bolza*, cun *termo de Lagrange* $\int_{t_0}^{t_f} L(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p})\mathrm{d}t$ e un *termo de Mayer* $E(\boldsymbol{x}(t_f),\boldsymbol{z}(t_f),\boldsymbol{p})$. Tanto $L:\mathbb{R}^{n_x}\times\mathbb{R}^{n_z}\times\mathbb{R}^{n_u}\times\mathbb{R}^{n_p}\to\mathbb{R}$ coma $E:\mathbb{R}^{n_x}\times\mathbb{R}^{n_z}\times\mathbb{R}^{n_p}\to\mathbb{R}$ asúmense dúas veces diferenciables, así como as restricións $\boldsymbol{c}$, $\boldsymbol{r}$ e $\tilde{\boldsymbol{r}}$.

Exemplo 1.6 (Freada dun bloque)**.** No contexto do Exemplo 1.2, o obxectivo pode ser minimizar a velocidade despois dunha unidade de tempo, $t_f = 1$. Ademais, hai que ter en conta que a velocidade $x(t)$ vai diminuír até un máximo de repouso e que o operario e mecanismo da freada non son capaces de aplicar cada un máis de 2 de aceleración. Daquela, o problema queda enunciado da seguinte maneira:

$$\begin{aligned} \min_{x,z,u} \quad & x(1) \\ \text{s. a} \quad & \dot{x}(t) = -z(t) - u(t), \quad t\in[0,1], \\ & 4 = (z(t))^2 + (u(t))^2, \quad t\in[0,1], \\ & 0 \le x(t) \le 3, \quad 0 \le z(t), u(t) \le 2, \quad t\in[0,1], \\ & x(0) = 3. \end{aligned}$$

Exemplo 1.7 (Reactor descontinuo)**.** Continuando co Exemplo 1.3, un pode querer escoller un perfil de temperaturas que maximice a cantidade de produto B despois dunha hora ($t_f = 1$). Ademais, este perfil non pode ser inferior a 0 Kelvin nin superior a 5 Kelvin. Entón, o problema de control óptimo pódese expresar como:

$$\begin{aligned}
\min_{\boldsymbol{x},u} \quad & -x_2(1) \\
\text{s. a} \quad & \dot{x}_1(t) = -x_1(t)\left(u(t) + \frac{(u(t))^2}{2}\right), \quad t \in [0,1], \\
& \dot{x}_2(t) = x_1(t)u(t), \quad t \in [0,1], \\
& 0 \leq u(t) \leq 5, \quad t \in [0,1], \\
& x_1(0) = 1, \quad x_2(0) = 0.
\end{aligned}$$

Exemplo 1.8 (Carro con péndulo - Lagrange)**.** Sexa agora o sistema descrito no Exemplo 1.4. Un problema de control óptimo asociado pode consistir en escoller a forza do motor que minimice o esforzo total realizado. Neste sentido, tómase como función obxectivo do problema:

$$\Phi[\boldsymbol{x}, u] \equiv \Phi[u] = \int_0^{t_f} (u(t))^2 \mathrm{d}t,$$

que é de *tipo Bolza* sen *termo de Mayer*. Amais, interesa que o carro non saia dos límites da pista e que non se aplique unha forza que o motor non soporte, o que se pode incluír mediante as constantes positivas d_{max} e u_{max} e as restricións:

$$-d_{max} \leq x_1(t) \leq d_{max}, \quad -u_{max} \leq u(t) \leq u_{max}, \quad t \in [0, t_f].$$

Daquela, obtense o problema de control óptimo seguinte:

$$\begin{aligned}
\min_{\boldsymbol{x},u} \quad & \int_0^{t_f} (u(t))^2 \mathrm{d}t \\
\text{s. a} \quad & \dot{x}_1(t) = x_3(t), \quad t \in [0, t_f], \\
& \dot{x}_2(t) = x_4(t), \quad t \in [0, t_f], \\
& \dot{x}_3(t) = \frac{lm_2 \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t) + m_2 g \cos(x_2(t)) \operatorname{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))}, \quad t \in [0, t_f], \\
& \dot{x}_4(t) = -\frac{lm_2 \cos(x_2(t)) \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t)\cos(x_2(t)) + (m_1 + m_2) g \operatorname{sen}(x_2(t))}{lm_1 + lm_2(1 - \cos^2(x_2(t)))}, \quad t \in [0, t_f], \\
& -d_{max} \leq x_1(t) \leq d_{max}, \quad -u_{max} \leq u(t) \leq u_{max}, \quad t \in [0, t_f], \\
& x_1(0) = 0, \quad x_2(0) = 0, \quad x_3(0) = 0, \quad x_4(0) = 0, \\
& x_1(t_f) = d, \quad x_2(t_f) = \pi, \quad x_3(t_f) = 0, \quad x_4(t_f) = 0.
\end{aligned}$$

Observación 1.9. Unha vez presentada a Definición 1.5, nótese que se pode supor que, nun problema de control óptimo, a función obxectivo a minimizar só conta cun *termo de Lagrange* no canto dun *termo de Mayer*, e viceversa. Se se ten un *termo de Lagrange*:

$$\Phi[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}] = \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}) \mathrm{d}t,$$

basta introducir unha variable de estado adicional x_{n+1} xunto con:

$$\dot{x}_{n+1}(t) = L(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}), \quad x_{n+1}(t_0) = 0,$$

e a nova función obxectivo sería:

$$\tilde{\Phi}[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}] = x_{n+1}(t_f).$$

Reciprocamente, dada unha función obxectivo con *termo de Mayer*:

$$\Phi[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}] = E(\boldsymbol{x}(t_f), \boldsymbol{z}(t_f), \boldsymbol{p}),$$

pódese tomar:

$$\tilde{\Phi}[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}] = \int_{t_0}^{t_f} \frac{E(\boldsymbol{x}(t_f), \boldsymbol{z}(t_f), \boldsymbol{p})}{t_f - t_0} \mathrm{d}t.$$

Agora ben, tal e como se destaca en [1], aínda que este tipo de transformacións dan lugar a problemas equivalentes dende o punto de vista teórico, dende unha perspectiva práctica non teñen as mesmas características: distinto número de variables e tipo de aproximacións.

Exemplo 1.10 (Carro con péndulo - Mayer)**.** No caso do Exemplo 1.8, modificando o tipo de función obxectivo con axuda do anterior, unha reformulación do problema sería:

$$\begin{aligned}
\min_{\boldsymbol{x},u} \quad & x_5(t_f) \\
\text{s. a} \quad & \dot{x}_1(t) = x_3(t), \quad t \in [0, t_f], \\
& \dot{x}_2(t) = x_4(t), \quad t \in [0, t_f], \\
& \dot{x}_3(t) = \frac{l m_2 \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t) + m_2 g \cos(x_2(t)) \operatorname{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))}, \quad t \in [0, t_f], \\
& \dot{x}_4(t) = -\frac{l m_2 \cos(x_2(t)) \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t) \cos(x_2(t)) + (m_1 + m_2) g \operatorname{sen}(x_2(t))}{l m_1 + l m_2(1 - \cos^2(x_2(t)))}, \quad t \in [0, t_f], \\
& \dot{x}_5(t) = (u(t))^2, \quad t \in [0, t_f], \\
& -d_{max} \leq x_1(t) \leq d_{max}, \quad -u_{max} \leq u(t) \leq u_{max}, \quad t \in [0, t_f], \\
& x_1(0) = 0, \quad x_2(0) = 0, \quad x_3(0) = 0, \quad x_4(0) = 0, \quad x_5(0) = 0, \\
& x_1(t_f) = d, \quad x_2(t_f) = \pi, \quad x_3(t_f) = 0, \quad x_4(t_f) = 0.
\end{aligned}$$

Así a todo, os problemas da Definición 1.5 tamén se poden estender mediante a incorporación de variables de control e parámetros que só tomen valores enteiros, constituíndo os coñecidos como problemas de control óptimo mixtos. Porén, neste traballo enunciaranse só resultados para os problemas de tipo OCP, para outras extensións consúltese [9]. Nesta liña, nas seguintes definicións preséntanse nocións básicas para tratar estes problemas.

Definición 1.11 (Traxectoria admisible)**.** Unha traxectoria dise admisible se $\boldsymbol{x}(t)$ é absolutamente continua, $\boldsymbol{u}(t)$ é medible e esencialmente limitada e as funcións $(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p})$ verifican as restricións do problema (1.4). Dise que a función de control $\hat{\boldsymbol{u}}(\cdot)$ é admisible se existe cando menos unha traxectoria admisible $(\boldsymbol{x}(t), \boldsymbol{z}(t), \hat{\boldsymbol{u}}(t), \boldsymbol{p})$.

Definición 1.12 (Óptimo global)**.** A traxectoria $(\boldsymbol{x}^*, \boldsymbol{z}^*, \boldsymbol{u}^*, \boldsymbol{p}^*)$ dise globalmente óptima se é admisible e verifica:

$$\Phi[\boldsymbol{x}^*, \boldsymbol{z}^*, \boldsymbol{u}^*, \boldsymbol{p}^*] \leq \Phi[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}] \tag{1.5}$$

para todas as traxectorias admisibles $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p})$.

Definición 1.13 (Óptimo local)**.** A traxectoria $(\boldsymbol{x}^*, \boldsymbol{z}^*, \boldsymbol{u}^*, \boldsymbol{p}^*)$ dise localmente óptima se é admisible e existe un $\delta > 0$ tal que se $\|\cdot\|$ denota calquera norma do espazo euclidiano, entón (1.5) verifícase para toda traxectoria admisible $(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p})$ con:

$$\|\boldsymbol{u}^*(t) - \boldsymbol{u}(t)\| \leq \delta \quad \forall t \in [t_0, t_f], \quad \|\boldsymbol{p}^* - \boldsymbol{p}\| \leq \delta.$$

Agora ben, antes de adentrarse en enunciados teóricos e métodos de resolución, cómpre salientar que en moitas ocasións o comportamento do sistema físico ou matemático muda co tempo. Noutras palabras, o sistema de DAEs pode non ser o mesmo en todo o intervalo temporal. Logo, para aviarse ante esta situación, introdúcese un número finito m de nodos temporais intermedios $\{\tilde{t}_0, \dots, \tilde{t}_m\}$ no conxunto de puntos $\{t_0, \bar{t}_1, \dots, \bar{t}_{n-1}, t_f\}$ que xa se empregaba para (1.4d). Obtense así un conxunto de $N+1$ puntos ordenados:

$$t_0 \leq t_1 \leq \cdots \leq t_N,$$

tal que $\{\tilde{t}_0, \dots, \tilde{t}_m\} \cup \{t_0, \bar{t}_1, \dots, \bar{t}_{n-1}, t_f\} = \{t_0, t_1, \dots, t_N\}$, $\tilde{t}_0 = t_0$ e $\tilde{t}_m = t_N = t_f$. Deste xeito, alí onde exista un cambio de dinámica, márcase mediante un punto $\tilde{t}_j$, que pode estar sometido a restricións ou non, e resulta un problema como o da seguinte definición.

Definición 1.14 (Problema de control óptimo multi-estado)**.** Sexan $\tilde{I}_k = [\tilde{t}_k, \tilde{t}_{k+1}]$ aqueles intervalos onde non hai cambio de dinámica e $\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}$ as funcións de estado diferenciais definidas en $\tilde{I}_k$ (de xeito análogo denótanse $\boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}$, $\boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}$, $\boldsymbol{f}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}$, ...). Un problema de control óptimo multi-estado é un problema de optimización con restricións da seguinte forma:

$$\min_{\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{p}} \sum_{k=0}^{m-1} \Phi^{\tilde{I}_k}[\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}, \boldsymbol{p}] \tag{1.6a}$$

suxeito aos sistemas de DAEs:

$$\begin{aligned} \dot{\boldsymbol{x}}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t) &= \boldsymbol{f}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{p}), \quad t \in \tilde{I}_k, \quad k = 0, \dots, m-1, \\ \boldsymbol{0} &= \boldsymbol{g}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{p}), \quad t \in \tilde{I}_k, \quad k = 0, \dots, m-1, \end{aligned} \tag{1.6b}$$

restricións de control e traxectoria:

$$\boldsymbol{0} \leq \boldsymbol{c}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{u}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(t), \boldsymbol{p}), \quad t \in \tilde{I}_k, \quad k = 0, \dots, m-1, \tag{1.6c}$$

igualdades e desigualdades de puntos interiores, con k_i denotando o índice do intervalo temporal que contén ao punto t_i:

$$\begin{aligned} \boldsymbol{0} &= \boldsymbol{r}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_0}}}(t_0), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_0}}}(t_0), \dots, \boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_N}}}(t_f), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_N}}}(t_f), \boldsymbol{p}), \\ \boldsymbol{0} &\leq \tilde{\boldsymbol{r}}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_0}}}(t_0), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_0}}}(t_0), \dots, \boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_N}}}(t_f), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k_N}}}(t_f), \boldsymbol{p}), \end{aligned} \tag{1.6d}$$

e condicións de transición dos estados:

$$\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k+1}}}(\tilde{t}_{k+1}) = \boldsymbol{tr_k}(\boldsymbol{x}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(\tilde{t}_{k+1}), \boldsymbol{z}^{\tilde{\boldsymbol{I}}_{\boldsymbol{k}}}(\tilde{t}_{k+1}), \boldsymbol{p}), \quad k = 0, \dots, m-2. \tag{1.6e}$$

As funcións involucradas teñen unha definición análoga á empregada na Definición 1.5 e as condicións (1.6e) de transición dos estados, con $\boldsymbol{tr_k} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$, adoitan forzar a continuidade no cambio de intervalo. En relación coas dimensións, estas poden variar con k.

Observación 1.15 (Intervalo temporal nos problemas de control óptimo). A elección dos instantes de tempo t_0 e, en especial, do instante t_f non é necesariamente fixa. É dicir, o valor de t_f pode estar suxeito a modificacións segundo o problema, co propósito de mellorar a optimización da función obxectivo. Por exemplo, pode ser de interese minimizar o tempo de voo dun proxectil suxeito a unha serie de leis do movemento.

1.3. Equivalencia xeométrica

Un dos principais resultados teóricos sobre a teoría de control óptimo que se tratan neste traballo, o principio do máximo (Teorema 2.2), foi enunciado e demostrado por vez primeira en [8]. A dedución de dito principio inclúe un número elevado de lemas e desenvolvementos previos, mais un dos aspectos máis interesantes é a reformulación xeométrica do problema en cuestión, que se adapta neste traballo ás notacións empregadas.

Deste xeito, pártese dun problema de control óptimo simplificado, que só conta con variables de estado diferenciais $\boldsymbol{x}(t) = (x_1(t), x_2(t), \dots, x_{n_x}(t))$ e variables de control $\boldsymbol{u}(t)$. Así, dados $\boldsymbol{x^0}$ e $\boldsymbol{x^f}$ puntos de $\mathbb{R}^{n_x}$, de entre todas as funcións de control admisibles $\boldsymbol{u} = \boldsymbol{u}(t)$ tales que:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [t_0, t_f], \\ \boldsymbol{x}(t_0) &= \boldsymbol{x^0}, \\ \boldsymbol{x}(t_f) &= \boldsymbol{x^f}, \end{aligned} \tag{1.7}$$

preténdese achar aquela que minimice:

$$\Phi[\boldsymbol{x}, \boldsymbol{u}] := \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t)) \mathrm{d}t. \tag{1.8}$$

Entón, se $\boldsymbol{f} = (f_1, f_2, \dots, f_{n_x})$, introdúcese unha nova coordenada x_0 ás coordenadas dos estados tal que:

$$\frac{\mathrm{d}x_0}{\mathrm{d}t}(t) = L(x_1(t), x_2(t), \dots, x_{n_x}(t), \boldsymbol{u}(t)),$$

obténdose o seguinte sistema:

$$\frac{\mathrm{d}x_i}{\mathrm{d}t}(t) = f_i(x_1(t), x_2(t), \dots, x_{n_x}(t), \boldsymbol{u}(t)), \quad i = 0, 1, \dots, n_x, \tag{1.9}$$

onde se denota $f_0 := L$.

Así, $\hat{\boldsymbol{x}} = (x_0, \boldsymbol{x})$ e $\hat{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{u}) = (f_0(\boldsymbol{x}, \boldsymbol{u}), \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}))$ toman valores en $\mathbb{R}^{n_x+1}$ e pódese reescribir o sistema (1.9) como:

$$\frac{\mathrm{d}\hat{\boldsymbol{x}}}{\mathrm{d}t}(t) = \hat{\boldsymbol{f}}(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{1.10}$$

Daquela, fixada $\boldsymbol{u}(t)$, tense a correspondente solución $\boldsymbol{x}(t)$ de (1.7). Entón, para:

$$x_0(t) = \int_{t_0}^{t} L(\boldsymbol{x}(t), \boldsymbol{u}(t))\mathrm{d}t,$$

tense que $\hat{\boldsymbol{x}}(t) = (x_0(t), \boldsymbol{x}(t))$ é solución de (1.10) con condición inicial $\hat{\boldsymbol{x}}(t_0) = (0, \boldsymbol{x^0})$ e está definida en todo $[t_0, t_f]$. Deste xeito, $\hat{\boldsymbol{x}}(t_f) = (\Phi[\boldsymbol{x}, \boldsymbol{u}], \boldsymbol{x^f})$.

Ademais, considerando a recta $(n_x + 1)$-dimensional $l = \left\{(a, \boldsymbol{x^f}) : a \in \mathbb{R}\right\}$, que claramente é paralela ao eixo engadido de coordenadas x_0, obsérvase que $\hat{\boldsymbol{x}}(t)$ corta a l no instante t_f e no punto $(\Phi[\boldsymbol{x}, \boldsymbol{u}], \boldsymbol{x^f})$.

Reciprocamente, sexan $\boldsymbol{u}(t)$ e a correspondente solución $\hat{\boldsymbol{x}}(t) = (x_0(t), \boldsymbol{x}(t))$ de (1.10) con $\hat{\boldsymbol{x}}(t_0) = (0, \boldsymbol{x^0})$ e $\hat{\boldsymbol{x}}(t_f) = (x_0(t_f), \boldsymbol{x^f}) \in l$. Entón $\boldsymbol{x}(t)$ é solución de (1.7) e $\Phi[\boldsymbol{x}, \boldsymbol{u}] = x_0(t_f)$.

Polo tanto, coas notacións presentadas, o problema (1.7) - (1.8) enúnciase de xeito xeométrico equivalente a continuación. Unha ilustración gráfica do feito represéntase na Figura 1.4.

Definición 1.16 (Problema xeométrico equivalente)**.** Sexan $(0, \boldsymbol{x^0})$ un punto de $\mathbb{R}^{n_x+1}$ e l a recta paralela ao eixo de coordenadas x_0 que pasa polo punto $(0, \boldsymbol{x^f})$. O problema xeométrico de control óptimo consiste en atopar aquela función de control admisible $\boldsymbol{u}(t)$ tal que a respectiva solución $\hat{\boldsymbol{x}}(t)$ de (1.10), con condición inicial $\hat{\boldsymbol{x}}(t_0) = (0, \boldsymbol{x^0})$, corte a recta l en t_f e, de entre aquelas que o fagan, o punto de intersección teña primeira coordenada x_0 mínima.

Ademais, nas condicións do problema simplificado desta sección ((1.7)-(1.8), ou a Definición 1.16) é sinxelo probar a Proposición 1.17.

Proposición 1.17. *Toda porción dunha traxectoria óptima é tamén óptima.*

Demostración. Sexa a partición de $N+1$ puntos $\{t_0, t_1, \ldots, t_{N-1}, t_N = t_f\}$ en $[t_0, t_f]$ e denótense $I_k = [t_k, t_{k+1}]$, para $k = 0, 1, \ldots, N-1$. Considérese o conxunto de funcións $\left\{\boldsymbol{x^{I_k}}, \boldsymbol{u^{I_k}}\right\}_k$ e de puntos $\left\{\boldsymbol{x^k}\right\}_k$ verificando:

$$\begin{aligned}
\Phi[\boldsymbol{x^{I_k}}, \boldsymbol{u^{I_k}}] &= \phi^{I_k}, \\
\dot{\boldsymbol{x}}^{\boldsymbol{I_k}}(t) &= \boldsymbol{f}(\boldsymbol{x^{I_k}}(t), \boldsymbol{u^{I_k}}(t)), \quad t \in [t_k, t_{k+1}], \\
\boldsymbol{x^{I_k}}(t_k) &= \boldsymbol{x^k}, \\
\boldsymbol{x^{I_k}}(t_{k+1}) &= \boldsymbol{x^{k+1}},
\end{aligned}$$

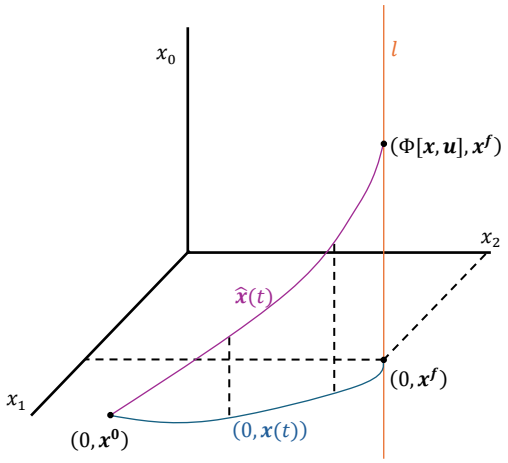

Figura 1.4: Representación gráfica do problema xeométrico equivalente da Definición 1.16 para o caso de dúas variables de estado.

para $k = 0, 1, \dots, N-1$. Entón, existen un control $\boldsymbol{u}(t)$ e un estado $\boldsymbol{x}(t)$ cumprindo:

$$\begin{aligned} \Phi[\boldsymbol{x},\boldsymbol{u}] &= \phi^{I_0} + \phi^{I_1} + \cdots + \phi^{I_{N-1}}, \\ \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{u}(t)), \quad t \in [t_0, t_f], \\ \boldsymbol{x}(t_0) &= \boldsymbol{x^0}, \\ \boldsymbol{x}(t_f) &= \boldsymbol{x^f}. \end{aligned} \tag{1.11}$$

En efecto, a partir dos controis $\boldsymbol{u^{I_k}}$, definidos en intervalos consecutivos, basta tomar unha definición de $\boldsymbol{u}$ a cachos. Non se extravían as propiedades de función medible e esencialmente limitada, e de igual xeito o estado resultante obedece o sistema (1.11).

Polo tanto, xa se teñen as condicións para probar o enunciado da proposición. Sexan respectivamente $\boldsymbol{u}(t)$ e $\boldsymbol{x}(t)$ un control e estado óptimos verificando (1.7) e supóñase que existe un intervalo $[t_1, t_2] \subseteq [t_0, t_f]$ tal que o control restrinxido non é óptimo.

O intervalo do problema queda dividido en $I_0 = [t_0, t_1]$, $I_1 = [t_1, t_2]$ e $I_2 = [t_2, t_f]$, e $\Phi[\boldsymbol{x},\boldsymbol{u}]$ pódese expresar como a suma dos obxectivos en cada anaco: $\Phi = \Phi^{I_0} + \Phi^{I_1} + \Phi^{I_2}$.

Agora ben, por hipótese de non ter unha restrición óptima en $[t_1, t_2]$, existen un control $\boldsymbol{v}(t)$ e un estado asociado $\boldsymbol{x^v}(t)$ verificando:

$$\begin{aligned} \dot{\boldsymbol{x}}^{\boldsymbol{v}}(t) &= \boldsymbol{f}(\boldsymbol{x^v}(t),\boldsymbol{v}(t)), \quad t \in [t_1, t_2], \\ \boldsymbol{x^v}(t_1) &= \boldsymbol{x^{I_0}}(t_1), \\ \boldsymbol{x^v}(t_2) &= \boldsymbol{x^{I_2}}(t_2), \end{aligned}$$

e tal que $\Phi^{\boldsymbol{v}} := \Phi[\boldsymbol{x}^{\boldsymbol{v}}, \boldsymbol{v}] < \Phi^{I_1}$. Daquela, definindo:

$$\tilde{\boldsymbol{u}}(t) = \begin{cases} \boldsymbol{u}(t), & t \in [t_0, t_1], \\ \boldsymbol{v}(t), & t \in [t_1, t_2], \\ \boldsymbol{u}(t), & t \in [t_2, t_f], \end{cases} \qquad \tilde{\boldsymbol{x}}(t) = \begin{cases} \boldsymbol{x}(t), & t \in [t_0, t_1], \\ \boldsymbol{x}^{\boldsymbol{v}}(t), & t \in [t_1, t_2], \\ \boldsymbol{x}(t), & t \in [t_2, t_f], \end{cases}$$

teríase que verifican (1.7) e ademais:

$$\Phi[\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}] = \Phi^{I_0} + \Phi^{\boldsymbol{v}} + \Phi^{I_2} < \Phi^{I_0} + \Phi^{I_1} + \Phi^{I_2} = \Phi[\boldsymbol{x}, \boldsymbol{u}].$$

Co que se chega a unha contradición coa hipótese do enunciado de solución óptima que vén de supor que existe unha porción de intervalo total onde a restrición non é óptima. □

Capítulo 2

O principio do máximo

O obxectivo deste capítulo é seguir o traballo de [9] e formular unha serie de condicións necesarias de solución óptima, cuxa análise e resolución serían unha forma de atacar o problema. Após os enunciados teóricos engadiranse complexidades ao problema, mais para comezar confróntase a seguinte formulación, que só conta con variables de estado diferenciais e controis:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & E(\boldsymbol{x}(t_f)) \\ \text{s. a} \quad & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [t_0, t_f], \\ & \mathbf{0} \leq \boldsymbol{c}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [t_0, t_f], \\ & \boldsymbol{x}(t_0) = \boldsymbol{x}^{\mathbf{0}}, \\ & \mathbf{0} = \boldsymbol{r}(\boldsymbol{x}(t_f)), \end{aligned} \tag{2.1}$$

para un intervalo temporal $[t_0, t_f]$ fixado. Os Exemplos 1.7 e 1.10 están nas condicións anteriores.

2.1. O principio do máximo

O principio do máximo sérvese da seguinte noción de Hamiltoniano e multiplicadores de Lagrange.

Definición 2.1. O Hamiltoniano do problema de control óptimo (2.1) defínese como:

$$\mathcal{H}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\mu}^T \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{u}),$$

e a función de punto final de Lagrange como:

$$\psi(\boldsymbol{x}(t_f), \boldsymbol{\nu}) := E(\boldsymbol{x}(t_f)) + \boldsymbol{\nu}^T \boldsymbol{r}(\boldsymbol{x}(t_f)),$$

onde as funcións $\boldsymbol{\lambda} : [t_0, t_f] \to \mathbb{R}^{n_x}$, $\boldsymbol{\mu} : [t_0, t_f] \to \mathbb{R}^{n_c}$ e $\boldsymbol{\nu} \in \mathbb{R}^{n_r}$ se denominan multiplicadores de Lagrange.

Deste xeito, estase en condicións de enunciar o seguinte teorema da teoría de control óptimo.

Teorema 2.2 (Principio do máximo)**.** *Sexan $\boldsymbol{u}^*(\cdot)$ unha función de control admisible e óptima do problema (2.1) e $\boldsymbol{x}^*(\cdot)$ o correspondente estado óptimo. Entón existen multiplicadores de Lagrange $\boldsymbol{\lambda}^*(\cdot)$, $\boldsymbol{\mu}^*(\cdot)$ e $\boldsymbol{\nu}^*$ tal que para case todo t en $[t_0, t_f]$ se verifican:*

$$\dot{\boldsymbol{x}}^*(t) = \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), \boldsymbol{\mu}^*(t)) = \boldsymbol{f}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t)), \tag{2.2a}$$

$$\dot{\boldsymbol{\lambda}}^{*T}(t) = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), \boldsymbol{\mu}^*(t)), \tag{2.2b}$$

$$\boldsymbol{x}^*(t_0) = \boldsymbol{x}^{\mathbf{0}}, \tag{2.2c}$$

$$\boldsymbol{\lambda}^{*T}(t_f) = -\frac{\partial \psi}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t_f), \boldsymbol{\nu}^*), \tag{2.2d}$$

$$\mathbf{0} \leq \boldsymbol{c}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t)), \tag{2.2e}$$

$$\mathbf{0} = \boldsymbol{r}(\boldsymbol{x}^*(t_f)), \tag{2.2f}$$

$$\boldsymbol{u}^*(t) = \arg\min_{\boldsymbol{u}} \mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}(t), \boldsymbol{\lambda}^*(t), \boldsymbol{\mu}^*(t)), \tag{2.2g}$$

$$\mathbf{0} = \boldsymbol{\mu}^{*T}(t)\boldsymbol{c}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t)), \tag{2.2h}$$

$$\mathbf{0} \leq \boldsymbol{\mu}^*(t). \tag{2.2i}$$

Cómpre salientar que as condicións (2.2h) se deben entender por compoñentes. É dicir, denotando as funcións $\boldsymbol{\mu}^* = (\mu_1, \dots, \mu_{n_c})$ e $\boldsymbol{c} = (c_1, \dots, c_{n_c})$, para cada índice i terase $\mu_i^* = 0$ ou $c_i = 0$. A proba deste resultado pódese consultar en [8].

Daquela, a partir do principio do máximo pódense deducir condicións necesarias de primeira e segunda orde para solucións óptimas do problema (2.1). En particular, grazas a (2.2g) tense en case todo punto o seguinte sistema:

$$\mathbf{0}^T = \frac{\partial \mathcal{H}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \boldsymbol{\lambda}^{*T}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*, \boldsymbol{u}^*) + \boldsymbol{\mu}^{*T}\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*, \boldsymbol{u}^*). \tag{2.3}$$

Ilústrase entón como se procedería cos Exemplos 1.7 e 1.10.

Exemplo 2.3 (Reactor descontinuo)**.** No problema de control óptimo asociado a un reactor descontinuo proposto no Exemplo 1.7, o Hamiltoniano vén dado por:

$$\mathcal{H}(\boldsymbol{x}, u, \boldsymbol{\lambda}, \boldsymbol{\mu}) = -\lambda_1(t)x_1(t)\left(u(t) + \frac{(u(t))^2}{2}\right) + \lambda_2(t)x_1(t)u(t) + \mu_1(t)u(t) + \mu_2(t)(5 - u(t)),$$

e a función de punto final de Lagrange por:

$$\psi(\boldsymbol{x}(1), \boldsymbol{\nu}) \equiv \psi(\boldsymbol{x}(1)) = -x_2(1).$$

Así, $\boldsymbol{\lambda} : [0,1] \to \mathbb{R}^2$, $\boldsymbol{\mu} : [0,1] \to \mathbb{R}^2$ e non existen multiplicadores $\boldsymbol{\nu}$.

Co cal, se $\boldsymbol{x^*}(t) = (x_1^*(t), x_2^*(t))$ e $u^*(t)$ son óptimas, segundo o Teorema 2.2 existen multiplicadores de Lagrange $\boldsymbol{\lambda^*}(t)$ e $\boldsymbol{\mu^*}(t)$ tales que, para case todo $t \in [0,1]$, verifican:

$$\dot{x}_1^*(t) = -x_1^*(t)\left(u^*(t) + \frac{(u^*(t))^2}{2}\right), \quad \dot{x}_2^*(t) = x_1^*(t)u^*(t),$$

$$\dot{\lambda}_1^*(t) = \lambda_1^*(t)\left(u^*(t) + \frac{(u^*(t))^2}{2}\right) - \lambda_2^*(t)u(t), \quad \dot{\lambda}_2^*(t) = 0,$$

$$x_1^*(0) = 1, \quad x_2^*(0) = 0,$$

$$\lambda_1^*(1) = 0, \quad \lambda_2^*(1) = 1,$$

$$0 \leq u^*(t), \quad 0 \leq 5 - u^*(t),$$

$$u^*(t) = \arg\min_u \left\{-\lambda_1^*(t)x_1^*(t)\left(u(t) + \frac{(u(t))^2}{2}\right) + \lambda_2^*(t)x_1^*(t)u(t) + \mu_1^*(t)u(t) + \mu_2^*(t)(5 - u(t))\right\},$$

$$0 = \mu_1^*(t) \quad \text{ou} \quad 0 = u^*(t),$$
$$0 = \mu_2^*(t) \quad \text{ou} \quad 0 = 5 - u^*(t),$$

$$0 \leq \mu_1^*(t), \quad 0 \leq \mu_2^*(t).$$

Ademais, a condición necesaria (2.3) neste caso é:

$$0 = -\lambda_1^*(t)x_1^*(t)(1 + u^*(t)) + \lambda_2^*(t)x_1^*(t) + \mu_1^*(t) - \mu_2^*(t).$$

Exemplo 2.4 (Carro con péndulo - Mayer)**.** Na formulación con función obxectivo de *tipo Mayer* do problema do carro con péndulo, Exemplo 1.10, o Hamiltoniano resulta en:

$$\begin{aligned}\mathcal{H}(\boldsymbol{x}, u, \boldsymbol{\lambda}, \boldsymbol{\mu}) = {} & \lambda_1(t)x_3(t) + \lambda_2(t)x_4(t) \\ & + \lambda_3(t)\frac{lm_2\,\mathrm{sen}(x_2(t))(x_4(t))^2 + u(t) + m_2 g\cos(x_2(t))\,\mathrm{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))} \\ & - \lambda_4(t)\frac{lm_2\cos(x_2(t))\,\mathrm{sen}(x_2(t))(x_4(t))^2 + u(t)\cos(x_2(t)) + (m_1 + m_2)g\,\mathrm{sen}(x_2(t))}{lm_1 + lm_2(1 - \cos^2(x_2(t)))} \\ & + \lambda_5(t)(u(t))^2 + \mu_1(t)(x_1(t) + d_{max}) + \mu_2(t)(d_{max} - x_1(t)) \\ & + \mu_3(t)(u(t) + u_{max}) + \mu_4(t)(u_{max} - u(t)),\end{aligned}$$

e a función de punto final de Lagrange en:

$$\psi(\boldsymbol{x}(t_f), \boldsymbol{\nu}) = x_5(t_f) + \nu_1(x_1(t_f) - d) + \nu_2(x_2(t_f) - \pi) + \nu_3 x_3(t_f) + \nu_4 x_4(t_f).$$

Nótese que xa son expresións máis difíciles de tratar de xeito teórico, igual que as condicións do Teorema 2.2, que se deducen a partir das expresións anteriores.

2.2. Extensións do principio do máximo

Está claro que o problema (2.1) é unha simplificación dos problemas de control óptimo, en xeral teranse casuísticas con máis variables e restricións como en (1.4) ou (1.6). Para cada un dos seguintes casos, hai distintas medidas a tomar para obter conclusións semellantes ás das dúas seccións anteriores. Unha explicación detallada de cada modificación no problema atópase en [5].

Termo de Lagrange. Se a función obxectivo a minimizar é de *tipo Bolza* completo, o principio do máximo mantense sempre que se engada o *termo de Lagrange* á definición de Hamiltoniano:

$$\mathcal{H}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := \lambda_L L(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\mu}^T \boldsymbol{c}(\boldsymbol{x}, \boldsymbol{u}).$$

Variables de tipo parámetro. Os parámetros $\boldsymbol{p}$ poden ser incluídos coma variables de estado con derivada temporal nula.

Variable de tempo final. Se t_f é susceptible a modificacións no proceso de optimización, outra condición debe engadirse ao Teorema 2.2:

$$0 = \left(\mathcal{H} + \frac{\partial E}{\partial t} + \boldsymbol{\mu}^{*T} \frac{\partial \boldsymbol{r}}{\partial t} \right)_{t=t_f}.$$

Problemas multi-estado. No Capítulo 1 probouse para un tipo de problema sinxelo que toda porción dunha traxectoria óptima era óptima na restrición. Pois ben, este principio pódese trasladar e basta concatenar as condicións do Teorema 2.2 para cada subintervalo. En particular, restricións nos extremos engádense para forzar a continuidade dos estados.

Variables de estado alxébricas. Na Definición 1.5 de problema de control óptimo estableceuse como hipótese a regularidade de $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} \in \mathbb{R}^{n_z \times n_z}$. Co cal, as variables alxébricas están determinadas pola parte alxébrica de (1.4b) e non inflúen nas condicións necesarias.

Restricións de fronteira. Se estas son enunciadas de xeito xeral, a función de punto final de Lagrange debe ser modificada adecuadamente. Por exemplo, se as restricións de fronteira son da forma:

$$\boldsymbol{0} = \boldsymbol{r}(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f)),$$

entón:

$$\psi(\boldsymbol{x}(t_f), \boldsymbol{\nu}) := E(\boldsymbol{x}(t_f)) + \boldsymbol{\nu}^T \boldsymbol{r}(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f)).$$

Restricións de punto interior. Se o problema presume de restricións do tipo:

$$\boldsymbol{0} = \boldsymbol{r}(\boldsymbol{x}(t_0), \boldsymbol{z}(t_0), \boldsymbol{x}(\bar{t_1}), \boldsymbol{z}(\bar{t_1}), \dots, \boldsymbol{x}(t_f), \boldsymbol{z}(t_f), \boldsymbol{p}),$$
$$\boldsymbol{0} \leq \tilde{\boldsymbol{r}}(\boldsymbol{x}(t_0), \boldsymbol{z}(t_0), \boldsymbol{x}(\bar{t_1}), \boldsymbol{z}(\bar{t_1}), \dots, \boldsymbol{x}(t_f), \boldsymbol{z}(t_f), \boldsymbol{p}),$$

outras ecuacións han de ser engadidas ao principio do máximo. Véxase [5].

Exemplo 2.5 (Freada dun bloque)**.** Daquela, malia que o Exemplo 1.6 non entra na estrutura de (2.1), a variable alxébrica $z(t)$ queda determinada a partir da ecuación:

$$0 = (z(t))^2 + (u(t))^2 - 4.$$

Así, tendo en conta que $u(t) \in [0, 2]$, obtense:

$$z(t) = \sqrt{4 - (u(t))^2},$$

e tomaríase como Hamiltoniano:

$$\begin{aligned}\mathcal{H}(x, u, \lambda, \boldsymbol{\mu}) =& \lambda(t)\left(-\sqrt{4-(u(t))^2} - u(t)\right) + \mu_1(t)x(t) + \mu_2(t)(3 - x(t)) \\ &+ \mu_3(t)\sqrt{4-(u(t))^2} + \mu_4(t)\left(2 - \sqrt{4-(u(t))^2}\right) + \mu_5(t)u(t) + \mu_6(t)(2 - u(t)),\end{aligned}$$

e como función de punto final de Lagrange:

$$\psi(x(1), \boldsymbol{\nu}) \equiv \psi(x(1)) = x(1).$$

Exemplo 2.6 (Carro con péndulo - Lagrange)**.** De igual xeito, aínda que o *termo de Lagrange* no Exemplo 1.8 non entra na estrutura de (2.1), as ecuacións do Teorema 2.2 dedúcense a partir da seguinte definición de Hamiltoniano:

$$\begin{aligned}\mathcal{H}(\boldsymbol{x}, u, \boldsymbol{\lambda}, \boldsymbol{\mu}) =& \lambda_L(t)(u(t))^2 + \lambda_1(t)x_3(t) + \lambda_2(t)x_4(t) \\ &+ \lambda_3(t)\frac{lm_2 \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t) + m_2 g \cos(x_2(t)) \operatorname{sen}(x_2(t))}{m_1 + m_2(1 - \cos^2(x_2(t)))} \\ &- \lambda_4(t)\frac{lm_2 \cos(x_2(t)) \operatorname{sen}(x_2(t))(x_4(t))^2 + u(t)\cos(x_2(t)) + (m_1 + m_2)g \operatorname{sen}(x_2(t))}{lm_1 + lm_2(1 - \cos^2(x_2(t)))} \\ &+ \mu_1(t)(x_1(t) + d_{max}) + \mu_2(t)(d_{max} - x_1(t)) \\ &+ \mu_3(t)(u(t) + u_{max}) + \mu_4(t)(u_{max} - u(t)),\end{aligned}$$

e da función de punto final de Lagrange:

$$\psi(\boldsymbol{x}(t_f), \boldsymbol{\nu}) = \nu_1(x_1(t_f) - d) + \nu_2(x_2(t_f) - \pi) + \nu_3 x_3(t_f) + \nu_4 x_4(t_f).$$

De tódolos xeitos, dada a súa similitude coa formulación con *termo de Mayer* do Exemplo 1.10 e polas facilidades que supón traballar con este último no código de MATLAB do traballo, tratarase dende este momento só a formulación do *termo de Mayer*.

Capítulo 3

Métodos directos de resolución

O achegamento tradicional á hora de resolver os problemas de control óptimo (1.4) e (1.6) baséase no principio do máximo, Teorema 2.2, que axuda a trocar o problema orixinal por un sistema de ecuacións diferenciais e alxébricas con restricións de fronteira. Porén, malia ser un procedemento que non implica aproximacións funcionais, un aumento do número de ecuacións ou restricións non é doado de tratar e require un coñecemento elevado do problema para poder obter as condicións necesarias.

Daquela, a familia de métodos empregada con maior frecuencia é a dos métodos directos, que levan o problema a un espazo de dimensión finita antes de proceder coa optimización mediante técnicas de programación non linear. En xeral, as funcións de control $\boldsymbol{u}(\cdot)$ son discretizadas e os distintos procedementos distínguense polo xeito de traballar coas funcións de estado.

A continuación, introdúcense os principais conceptos dos métodos directos de *direct single shooting*, *direct multiple shooting* e *collocation*. Estes dan lugar á formulación dun problema de programación non linear (*nonlinear programming*, NLP), da que se fará unha breve revisión na sección 3.4. En concreto, centrarase na programación cuadrática secuencial, un dos métodos empregados para resolver este tipo de problemas.

Neste sentido, trabállase dende este punto coa notación do problema (1.4), pois a extensión ao caso multi-estado é inmediata traballando con cada división do intervalo total.

3.1. Direct single shooting

Sexa o problema (1.4) e considérese unha partición temporal $t_0 \leq t_1 \leq \cdots \leq t_{n_{ss}} = t_f$. No canto de traballar con funcións de control arbitrarias, estas aproxímanse mediante unha serie de

parámetros $\boldsymbol{q^k}$ e funcións $\boldsymbol{\varphi^k}$ tal que:

$$\begin{aligned} &\boldsymbol{u}(t) \approx \boldsymbol{\varphi^0}(t, \boldsymbol{q^0}), \quad t \in [t_0, t_1], \\ &\boldsymbol{u}(t) \approx \boldsymbol{\varphi^k}(t, \boldsymbol{q^k}), \quad t \in (t_k, t_{k+1}], \quad k = 1, \dots, n_{ss} - 1. \end{aligned}$$

Normalmente as $\boldsymbol{\varphi^k}$ son funcións constantes ou lineais. Por exemplo, $\boldsymbol{\varphi^k}(t, \boldsymbol{q^k}) = \boldsymbol{q^k}$.

Entón, as funcións de estado $\boldsymbol{y}(\cdot)$ calcúlanse mediante integración substituíndo a aproximación dos controis no sistema de DAEs. Co cal, alén das discretizacións pertinentes, resulta un problema de dimensión finita nas variables $\boldsymbol{q} = (\boldsymbol{q^0}, \dots, \boldsymbol{q^{n_{ss}-1}})$, nos parámetros $\boldsymbol{p}$ e nos valores iniciais $\boldsymbol{y^0} = (\boldsymbol{x^0}, \boldsymbol{z^0})$.

En efecto, se $\boldsymbol{\xi} = (\boldsymbol{x^0}, \boldsymbol{z^0}, \boldsymbol{q^0}, \boldsymbol{q^1}, \dots, \boldsymbol{q^{n_{ss}-1}}, \boldsymbol{p})^T$, resulta un problema de programación non linear:

$$\begin{aligned} \min_{\boldsymbol{\xi}} \quad & F(\boldsymbol{\xi}) \\ \text{s. a} \quad & \boldsymbol{G}(\boldsymbol{\xi}) = \boldsymbol{0}, \\ & \boldsymbol{H}(\boldsymbol{\xi}) \leq \boldsymbol{0}, \end{aligned} \tag{3.1}$$

onde $F(\boldsymbol{\xi}) \equiv \Phi[\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p}]$ é a función obxectivo, con $\boldsymbol{x}$ e $\boldsymbol{z}$ determinadas por integración a partir de $\boldsymbol{\xi}$, e $\boldsymbol{G}(\cdot)$ e $\boldsymbol{H}(\cdot)$ inclúen as distintas restricións do problema orixinal.

O proceso de obtención de $\boldsymbol{x}$ e $\boldsymbol{z}$ pódese representar mediante unha aplicación $\boldsymbol{\phi}(\boldsymbol{\xi})$, que denota calquera proceso de resolución de sistemas diferenciais. Así, proporciona os valores das variables de estado necesarios a partir de $\boldsymbol{\xi}$ e do sistema de ecuacións diferenciais das restricións. Emprégase a notación:

$$\phi_x^k(\boldsymbol{\xi}) \approx x(t_k), \quad \phi_z^k(\boldsymbol{\xi}) \approx z(t_k).$$

Obsérvase entón a razón do nome do método, pois as variables de estado obtéñense nun só paso, xa que só se emprega a información dos controis e dos valores iniciais. Porén, no suposto de coñecer o valor dos estados en puntos interiores, non se pode aportar esta axuda, o que supón unha desvantaxe fronte a métodos posteriores.

Exemplo 3.1 (Freada dun bloque)**.** Para o problema presentado no Exemplo 1.6, se n_{ss} é o número de nodos temporais e discretízanse as variables de control como constantes a cachos:

$$\begin{aligned} &u(t) \approx q^0, \quad t \in [0, t_1], \\ &u(t) \approx q^k, \quad t \in (t_k, t_{k+1}], \quad k = 1, \dots, n_{ss} - 1, \end{aligned}$$

obtense $\boldsymbol{\xi} = (x^0, z^0, q^0, q^1 \dots, q^{n_{ss}-1})^T$. Daquela, mediante o método de *direct single shooting*, $\boldsymbol{\phi}(\boldsymbol{\xi}) = (\boldsymbol{\phi}_x, \boldsymbol{\phi}_z)(\boldsymbol{\xi}) = (\phi_x^0, \phi_x^1, \dots, \phi_x^{n_{ss}}, \phi_z^0, \phi_z^1, \dots, \phi_z^{n_{ss}})(\boldsymbol{\xi})$ e resulta o problema de programación

non linear:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & \phi_x^{n_{ss}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & 4 = \left(\phi_z^0(\boldsymbol{\xi})\right)^2 + \left(q^0\right)^2, \\
& 4 = \left(\phi_z^{k+1}(\boldsymbol{\xi})\right)^2 + \left(q^k\right)^2, \quad k = 0, \dots, n_{ss} - 1, \\
& 0 \leq \phi_x^k(\boldsymbol{\xi}) \leq 3, \quad 0 \leq \phi_z^k(\boldsymbol{\xi}) \leq 2, \quad k = 0, \dots, n_{ss}, \\
& 0 \leq q^k \leq 2, \quad k = 0, \dots, n_{ss} - 1, \\
& \phi_x^0(\boldsymbol{\xi}) = 3.
\end{aligned}$$

Exemplo 3.2 (Reactor descontinuo)**.** Para o problema presentado no Exemplo 1.7, con notacións análogas ás do Exemplo 3.1, obtense $\boldsymbol{\xi} = (x_1^0, x_2^0, q^0, q^1, \dots, q^{n_{ss}-1})^T$. Daquela, mediante *direct single shooting* precísase $\phi(\boldsymbol{\xi}) = (\phi_{x_1}^0(\boldsymbol{\xi}), \phi_{x_2}^0(\boldsymbol{\xi}), \phi_{x_2}^{n_{ss}}(\boldsymbol{\xi}))$ e resulta o problema:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & -\phi_{x_2}^{n_{ss}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & 0 \leq q^k \leq 5, \quad k = 0, \dots, n_{ss} - 1, \\
& \phi_{x_1}^0(\boldsymbol{\xi}) = 1, \quad \phi_{x_2}^0(\boldsymbol{\xi}) = 0.
\end{aligned}$$

Exemplo 3.3 (Carro con péndulo - Mayer)**.** Para o Exemplo 1.10, as variables discretas son $\boldsymbol{\xi} = (x_1^0, x_2^0, x_3^0, x_4^0, x_5^0, q^0, q^1, \dots, q^{n_{ss}-1})^T$ e queda a seguinte formulación:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & \phi_{x_5}^{n_{ss}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & -d_{max} \leq \phi_{x_1}^k(\boldsymbol{\xi}) \leq d_{max}, \quad k = 0, \dots, n_{ss}, \\
& -u_{max} \leq q^k \leq u_{max}, \quad k = 0, \dots, n_{ss} - 1, \\
& \phi_{x_1}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_2}^0(\boldsymbol{\xi}) = 0 \quad \phi_{x_3}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_4}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_5}^0(\boldsymbol{\xi}) = 0, \\
& \phi_{x_1}^{n_{ss}}(\boldsymbol{\xi}) = d, \quad \phi_{x_2}^{n_{ss}}(\boldsymbol{\xi}) = \pi \quad \phi_{x_3}^{n_{ss}}(\boldsymbol{\xi}) = 0, \quad \phi_{x_4}^{n_{ss}}(\boldsymbol{\xi}) = 0,
\end{aligned}$$

onde neste caso $\boldsymbol{\phi} = (\phi_{x_1}^0, \phi_{x_1}^1, \dots, \phi_{x_1}^{n_{ss}}, \phi_{x_2}^0, \phi_{x_2}^{n_{ss}}, \phi_{x_3}^0, \phi_{x_3}^{n_{ss}}, \phi_{x_4}^0, \phi_{x_4}^{n_{ss}}, \phi_{x_5}^0, \phi_{x_5}^{n_{ss}})$.

3.2. Direct multiple shooting

De igual xeito que no *direct single shooting*, a súa versión *multiple* baséase na discretización dos controis, mais agora a integración de estados faise en cada subintervalo da malla considerada. Sexan entón os nodos temporais $t_0 \leq t_1 \leq \cdots \leq t_{ms} = t_f$, e os controis aproximados:

$$\begin{aligned}
& \boldsymbol{u}(t) \approx \boldsymbol{\varphi^0}(t, \boldsymbol{q^0}), \quad t \in [t_0, t_1], \\
& \boldsymbol{u}(t) \approx \boldsymbol{\varphi^k}(t, \boldsymbol{q^k}), \quad t \in (t_k, t_{k+1}], \quad k = 1, \dots, n_{ms} - 1.
\end{aligned}$$

Nos nodos da malla aproxímanse os valores dos estados:

$$\boldsymbol{x^k} \approx \boldsymbol{x}(t_k), \quad \boldsymbol{z^k} \approx \boldsymbol{z}(t_k), \quad k = 0, 1, \dots, n_{ms},$$

que se empregan como valores iniciais para obter expresións mediante integración dos estados en cada $[t_k, t_{k+1}]$. Noutras palabras, no subintervalo $k-$ésimo resólvese o sistema de ecuacións correspondente con valores iniciais $\boldsymbol{x^k}$ e $\boldsymbol{z^k}$, e controis aproximados $\boldsymbol{\varphi^k}(t, \boldsymbol{q^k})$.

Deste xeito, empregando un lixeiro abuso de notación, vólvense empregar as mesmas notacións para $\boldsymbol{\phi}(\boldsymbol{\xi})$, se ben agora hai que ter en conta que $\boldsymbol{\phi_x^k}(\boldsymbol{\xi}) = (\phi_{x_1}^k, \dots, \phi_{x_{n_x}}^k)$ obtense a partir de $\boldsymbol{q^{k-1}}$, $\boldsymbol{x^{k-1}}$, $\boldsymbol{z^{k-1}}$ e do sistema diferencial correspondente. Ademais, co fin de garantir a continuidade das variables de estado diferenciais, engádense ao problema de optimización restricións nos extremos dos intervalos:

$$\boldsymbol{\phi_x^k}(\boldsymbol{\xi}) - \boldsymbol{x^k} = 0, \tag{3.2}$$

que habitualmente non se satisfán no proceso de resolución até que se chega ao óptimo desexado.

Co cal, denotando $\boldsymbol{\xi} = (\boldsymbol{x^0}, \boldsymbol{z^0}, \boldsymbol{q^0}, \boldsymbol{x^1}, \boldsymbol{z^1}, \boldsymbol{q^1}, \dots, \boldsymbol{x^{n_{ms}-1}}, \boldsymbol{z^{n_{ms}-1}} \boldsymbol{q^{n_{ms}-1}}, \boldsymbol{x^{n_{ms}}}, \boldsymbol{z^{n_{ms}}}, \boldsymbol{p})^T$, obtense un problema de programación non linear da forma (3.1).

Así, na Figura 3.1 ilústrase este proceso e a necesidade das restricións (3.2).

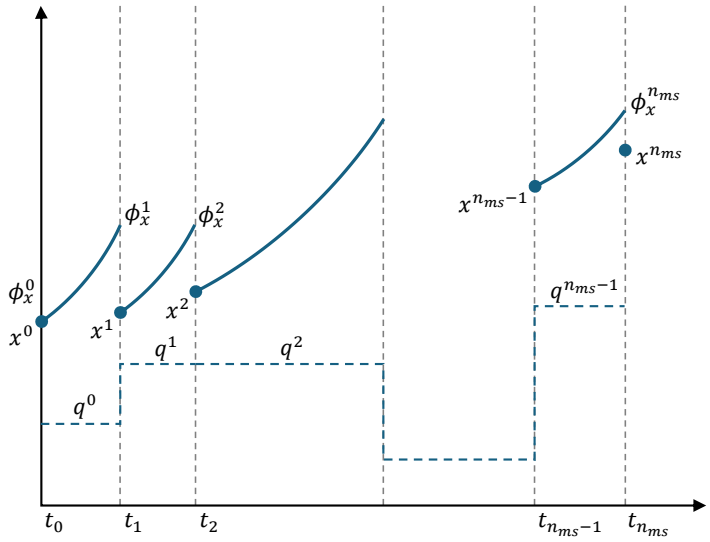

Figura 3.1: Ilustración do *direct multiple shooting*.

Exemplo 3.4 (Freada dun bloque)**.** Para o problema presentado no Exemplo 1.6, se n_{ms} é o número de nodos temporais, discretízanse as variables de control como constantes a cachos:

$$\begin{aligned} u(t) &\approx q^0, \quad t \in [0, t_1], \\ u(t) &\approx q^k, \quad t \in (t_k, t_{k+1}], \quad k = 1, \dots, n_{ms} - 1, \end{aligned}$$

e aproxímase o valor dos estados nos nodos temporais:

$$x^k \approx x(t_k), \quad z^k \approx z(t_k), \quad k = 0, \dots, n_{ms},$$

obtendo $\boldsymbol{\xi} = (x^0, z^0, q^0, \dots, x^{n_{ms}-1}, z^{n_{ms}-1}, q^{n_{ms}-1}, x^{n_{ms}}, z^{n_{ms}})^T$. Daquela, mediante *direct multiple shooting* e coa notación habitual para $\boldsymbol{\phi}(\boldsymbol{\xi})$ resulta o problema:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & \phi_x^{n_{ms}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & 4 = \left(\phi_z^0(\boldsymbol{\xi})\right)^2 + \left(q^0\right)^2, \\
& 4 = \left(\phi_z^{k+1}(\boldsymbol{\xi})\right)^2 + \left(q^k\right)^2, \quad k = 0, \dots, n_{ms}-1, \\
& \phi_x^k(\boldsymbol{\xi}) - x^k = 0, \quad k = 1, \dots, n_{ms}, \\
& 0 \le \phi_x^k(\boldsymbol{\xi}) \le 3, \quad 0 \le \phi_z^k(\boldsymbol{\xi}) \le 2, \quad k = 0, \dots, n_{ms}, \\
& 0 \le q^k \le 2, \quad k = 0, \dots, n_{ms}-1, \\
& \phi_x^0(\boldsymbol{\xi}) = 3.
\end{aligned}$$

Exemplo 3.5 (Reactor descontinuo)**.** No caso do Exemplo 1.7, mediante *direct multiple shooting* obtense $\boldsymbol{\xi} = (x_1^0, x_2^0, q^0, \dots, x_1^{n_{ms}-1}, x_2^{n_{ms}-1}, q^{n_{ms}-1}, x_1^{n_{ms}}, x_2^{n_{ms}})^T$, e resulta o problema:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & -\phi_{x_2}^{n_{ms}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & \phi_{x_1}^k(\boldsymbol{\xi}) - x_1^k = 0, \quad k = 1, \dots, n_{ms}, \\
& \phi_{x_2}^k(\boldsymbol{\xi}) - x_2^k = 0, \quad k = 1, \dots, n_{ms}, \\
& 0 \le q^k \le 5, \quad k = 0, \dots, n_{ms}-1, \\
& \phi_{x_1}^0(\boldsymbol{\xi}) = 1, \quad \phi_{x_2}^0(\boldsymbol{\xi}) = 0.
\end{aligned}$$

Exemplo 3.6 (Carro con péndulo - Mayer)**.** Por último, cando se traballa co Exemplo 1.10, queda $\boldsymbol{\xi} = (x_1^0, x_2^0, \dots, x_5^0, q^0, \dots, x_1^{n_{ms}-1}, x_2^{n_{ms}-1}, \dots, x_5^{n_{ms}-1}, q^{n_{ms}-1}, x_1^{n_{ms}}, x_2^{n_{ms}}, \dots, x_5^{n_{ms}})^T$ e:

$$\begin{aligned}
\min_{\boldsymbol{\xi}} \quad & \phi_{x_5}^{n_{ms}}(\boldsymbol{\xi}) \\
\text{s. a} \quad & \phi_{x_1}^k(\boldsymbol{\xi}) - x_1^k = 0, \quad k = 1, \dots, n_{ms}, \\
& \phi_{x_2}^k(\boldsymbol{\xi}) - x_2^k = 0, \quad k = 1, \dots, n_{ms}, \\
& \phi_{x_3}^k(\boldsymbol{\xi}) - x_3^k = 0, \quad k = 1, \dots, n_{ms}, \\
& \phi_{x_4}^k(\boldsymbol{\xi}) - x_4^k = 0, \quad k = 1, \dots, n_{ms}, \\
& \phi_{x_5}^k(\boldsymbol{\xi}) - x_5^k = 0, \quad k = 1, \dots, n_{ms}, \\
& -d_{max} \le \phi_{x_1}^k(\boldsymbol{\xi}) \le d_{max}, \quad k = 0, \dots, n_{ms}, \\
& -u_{max} \le q^k \le u_{max}, \quad k = 0, \dots, n_{ms}-1, \\
& \phi_{x_1}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_2}^0(\boldsymbol{\xi}) = 0 \quad \phi_{x_3}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_4}^0(\boldsymbol{\xi}) = 0, \quad \phi_{x_5}^0(\boldsymbol{\xi}) = 0, \\
& \phi_{x_1}^{n_{ms}}(\boldsymbol{\xi}) = d, \quad \phi_{x_2}^{n_{ms}}(\boldsymbol{\xi}) = \pi \quad \phi_{x_3}^{n_{ms}}(\boldsymbol{\xi}) = 0, \quad \phi_{x_4}^{n_{ms}}(\boldsymbol{\xi}) = 0.
\end{aligned}$$

3.3. Collocation

En termos do comportamento dos estados, na *collocation* xa non se tratan como variables dependentes. Daquela, considérase unha partición temporal $t_0 \leq t_1 \leq \cdots \leq t_{n_{col}} = t_f$, aproxímanse tanto os controis coma os estados nos nodos temporais e múdase o sistema de ecuacións:

$$\mathbf{0} = \dot{\boldsymbol{x}}(t) - \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p}), \quad t \in [t_0, t_f], \tag{3.3}$$

por sistemas discretos mediante esquemas de discretización como Runge-Kutta. De igual xeito, o resto de restricións e a función obxectivo discretízanse empregando o novo conxunto finito de variables. Así, o problema orixinal convértese nun problema de programación non linear de forma semellante a (3.1), mais desta vez *puro*, é dicir, no que non hai variables dependentes e polo tanto non é preciso proceder en pasos intermedios con integración.

En primeiro lugar, a función obxectivo a minimizar pódese tratar de diversas formas. No caso de que sexa de *tipo Mayer*, xa está en forma discreta. Porén, se existe un termo de *tipo Lagrange*, este pódese converter a *tipo Mayer* como xa se veu no Capítulo 1, ou empregar algunha fórmula de cuadratura como pode ser o método dos trapecios (véxase [1]).

En segundo lugar, cómpre atender á discretización das restricións de tipo funcional, é dicir, aquelas que non só involucran valores puntuais de funcións, pois estas xa están en forma discreta. As restricións de tipo alxébrico imponse nos puntos da malla e o máis complexo xorde cando aparecen ecuacións diferenciais.

3.3.1. Collocation con esquemas Runge-Kutta

Dunha banda, unha primeira opción, empregada en [1], é apostar por unha discretización baseada en métodos do tipo Runge-Kutta ou multipaso. Tómanse como variables:

$$\boldsymbol{u^k} \approx \boldsymbol{u}(t_k), \quad \boldsymbol{x^k} \approx \boldsymbol{x}(t_k), \quad \boldsymbol{z^k} \approx \boldsymbol{z}(t_k), \quad k = 0, \ldots, n_{col},$$

e denótase:

$$h_k = t_{k+1} - t_k, \quad \boldsymbol{f^k} = \boldsymbol{f}(\boldsymbol{x^k}, \boldsymbol{z^k}, \boldsymbol{u^k}, \boldsymbol{p}) \approx \boldsymbol{f}(\boldsymbol{x}(t_k), \boldsymbol{z}(t_k), \boldsymbol{u}(t_k), \boldsymbol{p}).$$

Así, (3.3) convértese, con k percorrendo os nodos da malla temporal, en:

- *Método de Euler:*

$$\mathbf{0} = \boldsymbol{x^{k+1}} - \boldsymbol{x^k} - h_k \boldsymbol{f^k}.$$

- *Método dos trapecios:*

$$\mathbf{0} = \boldsymbol{x^{k+1}} - \boldsymbol{x^k} - \frac{h_k}{2}\left(\boldsymbol{f^k} + \boldsymbol{f^{k+1}}\right).$$

- *Método de Hermite-Simpson:*

$$\mathbf{0} = \boldsymbol{x^{k+1}} - \boldsymbol{x^k} - \frac{h_k}{6}\left(\boldsymbol{f^k} + 4\tilde{\boldsymbol{f}}^{\boldsymbol{k+1}} + \boldsymbol{f^{k+1}}\right),$$

onde:

$$\begin{aligned}
\tilde{\boldsymbol{f}}^{\boldsymbol{k+1}} &= \boldsymbol{f}\left(\tilde{\boldsymbol{x}}^{\boldsymbol{k+1}}, \boldsymbol{z}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{u}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{p}\right), \\
\tilde{\boldsymbol{x}}^{\boldsymbol{k+1}} &= \frac{1}{2}\left(\boldsymbol{x^k} + \boldsymbol{x^{k+1}}\right) + \frac{h_k}{8}\left(\boldsymbol{f^k} + \boldsymbol{f^{k+1}}\right), \\
\boldsymbol{z}^{\boldsymbol{k+\frac{1}{2}}} &\approx \boldsymbol{z}\left(\frac{1}{2}(t_k + t_{k+1})\right), \\
\boldsymbol{u}^{\boldsymbol{k+\frac{1}{2}}} &\approx \boldsymbol{u}\left(\frac{1}{2}(t_k + t_{k+1})\right).
\end{aligned}$$

- *Método de Runge-Kutta clásico:*

$$\mathbf{0} = \boldsymbol{x^{k+1}} - \boldsymbol{x^k} - \frac{h_k}{6}(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}),$$

onde:

$$\begin{aligned}
\mathbf{k_1} &= \boldsymbol{f^k}, \\
\mathbf{k_2} &= \boldsymbol{f^k}\left(\boldsymbol{x^k} + \frac{1}{2}\mathbf{k_1}, \boldsymbol{z}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{u}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{p}\right), \\
\mathbf{k_3} &= \boldsymbol{f^k}\left(\boldsymbol{x^k} + \frac{1}{2}\mathbf{k_2}, \boldsymbol{z}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{u}^{\boldsymbol{k+\frac{1}{2}}}, \boldsymbol{p}\right), \\
\mathbf{k_4} &= \boldsymbol{f^k}\left(\boldsymbol{x^k} + \mathbf{k_3}, \boldsymbol{z^{k+1}}, \boldsymbol{u^{k+1}}, \boldsymbol{p}\right).
\end{aligned}$$

Observación 3.7. No código do Capítulo 5, para facilitar a execución do método de Hermite-Simpson e reducir o número de variables, empréganse os puntos medios:

$$\boldsymbol{z}^{\boldsymbol{k+\frac{1}{2}}} = \frac{1}{2}\left(\boldsymbol{z^k} + \boldsymbol{z^{k+1}}\right), \quad \boldsymbol{u}^{\boldsymbol{k+\frac{1}{2}}} = \frac{1}{2}\left(\boldsymbol{u^k} + \boldsymbol{u^{k+1}}\right),$$

que, como en xeral as variables son continuas e os intervalos da malla pequenos, non supón unha gran diferencia.

Observación 3.8. Nótese que a partición temporal se pode definir tamén cando os extremos temporais son variables, é dicir, cando t_0 e t_f son susceptibles a modificacións. Neste caso, tomando constantes $\tau_k \in [0,1]$, basta definir $h_k = (\tau_{k+1} - \tau_k)(t_f - t_0)$ e $t_k = t_0 + \tau_k(t_f - t_0)$, de xeito que sempre se traballa coa mesma porción do intervalo total.

Exemplo 3.9 (Freada dun bloque)**.** Considérese outra vez o problema do Exemplo 1.6 e obsérvese como resulta a súa *collocation* a través das distintas opcións de discretización. Sexan a partición temporal $t_0 \leq t_1 \leq \cdots \leq t_f = t_{n_{col}}$ e as variables discretas x^k, z^k, u^k.

- *Método de Euler:*

$$\begin{aligned}
\min_{x^k,z^k,u^k} \quad & x^{n_{col}} \\
\text{s. a} \quad & x^{k+1} - x^k - h_k\left(-z^k - u^k\right) = 0, \quad k = 0, \dots, n_{col} - 1, \\
& \left(z^k\right)^2 + \left(u^k\right)^2 - 4 = 0, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \le x^k \le 3, \quad 0 \le z^k \le 2, \quad 0 \le u^k \le 2, \quad k = 0, \dots, n_{col}, \\
& x^0 = 3.
\end{aligned}$$

- *Método dos trapecios:*

$$\begin{aligned}
\min_{x^k,z^k,u^k} \quad & x^{n_{col}} \\
\text{s. a} \quad & x^{k+1} - x^k - \frac{h_k}{2}\left(-z^k - u^k - z^{k+1} - u^{k+1}\right) = 0, \quad k = 0, \dots, n_{col} - 1, \\
& \left(z^k\right)^2 + \left(u^k\right)^2 - 4 = 0, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \le x^k \le 3, \quad 0 \le z^k \le 2, \quad 0 \le u^k \le 2, \quad k = 0, \dots, n_{col}, \\
& x^0 = 3.
\end{aligned}$$

- *Método de Hermite-Simpson:*

$$\begin{aligned}
\min_{x^k,z^k,u^k} \quad & x^{n_{col}} \\
\text{s. a} \quad & x^{k+1} - x^k - \frac{h_k}{6}\left(-z^k - u^k + 4\tilde{f}^{k+1} - z^{k+1} - u^{k+1}\right) = 0, \quad k = 0, \dots, n_{col} - 1, \\
& \left(z^k\right)^2 + \left(u^k\right)^2 - 4 = 0, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \le x^k \le 3, \quad 0 \le z^k \le 2, \quad 0 \le u^k \le 2, \quad k = 0, \dots, n_{col}, \\
& x^0 = 3,
\end{aligned}$$

onde:

$$\tilde{f}^{k+1} = -z^{k+\frac{1}{2}} - u^{k+\frac{1}{2}}.$$

- *Método de Runge-Kutta clásico:*

$$\begin{aligned}
\min_{x^k,z^k,u^k} \quad & x^{n_{col}} \\
\text{s. a} \quad & x^{k+1} - x^k - \frac{h_k}{6}(\mathrm{k}_1 + 2\mathrm{k}_2 + 2\mathrm{k}_3 + \mathrm{k}_4) = 0, \quad k = 0, \dots, n_{col} - 1, \\
& \left(z^k\right)^2 + \left(u^k\right)^2 - 4 = 0, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \le x^k \le 3, \quad 0 \le z^k \le 2, \quad 0 \le u^k \le 2, \quad k = 0, \dots, n_{col}, \\
& x^0 = 3,
\end{aligned}$$

onde:

$$\mathrm{k}_1 = -z^k - u^k, \quad \mathrm{k}_2 = -z^{k+\frac{1}{2}} - u^{k+\frac{1}{2}}, \quad \mathrm{k}_3 = -z^{k+\frac{1}{2}} - u^{k+\frac{1}{2}}, \quad \mathrm{k}_4 = -z^{k+1} - u^{k+1}.$$

Exemplo 3.10 (Reactor descontinuo)**.** Considérese tamén o problema do Exemplo 1.7 e sexan a partición temporal $t_0 \leq t_1 \leq \cdots \leq t_f = t_{n_{col}}$ e as variables discretas x_1^k, x_2^k, u^k. Mediante o método dos trapecios resulta:

$$
\begin{aligned}
\min_{x_1^k, x_2^k, u^k} \quad & -x_2^{n_{col}} \\
\text{s. a} \quad & x_1^{k+1} - x_1^k - \frac{h_k}{2}\left(-x_1^k\left(u^k + \frac{\left(u^k\right)^2}{2}\right) - x_1^{k+1}\left(u^{k+1} + \frac{\left(u^{k+1}\right)^2}{2}\right)\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& x_2^{k+1} - x_2^k - \frac{h_k}{2}\left(x_1^k u^k + x_1^{k+1} u^{k+1}\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& 0 \leq u^k \leq 5 \quad k = 0, \ldots, n_{col}, \\
& x_1^0 = 1, \quad x_2^0 = 0.
\end{aligned}
$$

Exemplo 3.11 (Carro con péndulo - Mayer)**.** Do mesmo xeito, o Exemplo 1.10 resulta mediante o método dos trapecios en:

$$
\begin{aligned}
\min_{\boldsymbol{x^k}, u^k} \quad & x_5^{n_{col}} \\
\text{s. a} \quad & x_1^{k+1} - x_1^k - \frac{h_k}{2}\left(x_3^k + x_3^{k+1}\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& x_2^{k+1} - x_2^k - \frac{h_k}{2}\left(x_4^k + x_4^{k+1}\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& x_3^{k+1} - x_3^k - \frac{h_k}{2}\left(\frac{l m_2 \operatorname{sen}\left(x_2^k\right)\left(x_4^k\right)^2 + u^k + m_2 g \cos\left(x_2^k\right)\operatorname{sen}\left(x_2^k\right)}{m_1 + m_2\left(1 - \cos^2\left(x_2^k\right)\right)}\right. \\
& \qquad \left. + \frac{l m_2 \operatorname{sen}\left(x_2^{k+1}\right)\left(x_4^{k+1}\right)^2 + u^{k+1} + m_2 g \cos\left(x_2^{k+1}\right)\operatorname{sen}\left(x_2^{k+1}\right)}{m_1 + m_2\left(1 - \cos^2\left(x_2^{k+1}\right)\right)}\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& x_4^{k+1} - x_4^k + \frac{h_k}{2}\left(\frac{l m_2 \cos\left(x_2^k\right)\operatorname{sen}\left(x_2^k\right)\left(x_4^k\right)^2 + u^k \cos(x_2^k) + (m_1+m_2) g \operatorname{sen}(x_2^k)}{l m_1 + l m_2\left(1 - \cos^2(x_2^k)\right)}\right. \\
& \qquad + \frac{l m_2 \cos\left(x_2^{k+1}\right)\operatorname{sen}\left(x_2^{k+1}\right)\left(x_4^{k+1}\right)^2 + u^{k+1}\cos\left(x_2^{k+1}\right)}{l m_1 + l m_2\left(1 - \cos^2\left(x_2^{k+1}\right)\right)} \\
& \qquad \left. + \frac{(m_1+m_2) g \operatorname{sen}\left(x_2^{k+1}\right)}{l m_1 + l m_2\left(1 - \cos^2\left(x_2^{k+1}\right)\right)}\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& x_5^{k+1} - x_5^k - \frac{h_k}{2}\left(\left(u^k\right)^2 + \left(u^{k+1}\right)^2\right) = 0, \quad k = 0, \ldots, n_{col}-1, \\
& -d_{max} \leq x_1^k \leq d_{max}, \quad -u_{max} \leq q^k \leq u_{max}, \quad k = 0, \ldots, n_{col}, \\
& x_1^0 = 0, \quad x_2^0 = 0, \quad x_3^0 = 0, \quad x_4^0 = 0, \quad x_5^0 = 0, \\
& x_1^{n_{col}} = d, \quad x_2^{n_{col}} = \pi, \quad x_3^{n_{col}} = 0, \quad x_4^{n_{col}} = 0.
\end{aligned}
$$

3.3.2. Collocation con interpolación e puntos móbiles

Doutra banda, coma en [3], pódese escoller empregar interpolación con polinomios de Lagrange mediante puntos de Gauss ou Radau para obter os estados e controis en cada intervalo $I_k = [t_k, t_{k+1}]$.

Grao R	1	2	3	4	5
Gauss-Legendre	0.500000	0.211325	0.112702	0.069432	0.046910
		0.788675	0.500000	0.330009	0.230765
			0.887298	0.669991	0.500000
				0.930568	0.769235
					0.953090
Radau	1.000000	0.333333	0.155051	0.088588	0.057104
		1.000000	0.644949	0.409467	0.276843
			1.000000	0.787659	0.583590
				1.000000	0.860240
					1.000000

Táboa 3.1: Puntos móbiles de Gauss-Legendre e Radau como puntos da *collocation*.

Así, tómanse como variables:

$$\boldsymbol{x^{kj}} \approx \boldsymbol{x}(t_{kj}), \quad k = 0, \dots n_{col} - 1, \quad j = 0, \dots, R, \quad \boldsymbol{x^{n_{col}}} \approx \boldsymbol{x}(t_{n_{col}}),$$
$$\boldsymbol{z^{kj}} \approx \boldsymbol{z}(t_{kj}), \quad \boldsymbol{u^{kj}} \approx \boldsymbol{u}(t_{kj}), \quad k = 0, \dots n_{col} - 1, \quad j = 1, \dots, R,$$

onde $t_{kj} = t_k + \tau_j h_k$, con $\tau_0 = 0$ e con $0 < \tau_j \leq 1$, $j = 1, \dots, R$, puntos móbiles de Gauss ou Radau.

O cálculo destes puntos τ_j pódese atopar en [2], que fai un desenvolvemento similar ao de [3] xunto cunha explicación detallada da *collocation* con polinomios ortogonais de Gauss-Legendre ou Radau. Para o propósito do traballo, inclúese desta referencia a Táboa 3.1 cos distintos $\tau_j \in [0, 1]$ segundo os graos R máis habituais na interpolación.

Daquela, con $R + 1$ puntos de interpolación, os estados en cada I_k resultan en:

$$\tilde{\boldsymbol{x}}^{\boldsymbol{I_k}}(t) = \sum_{j=0}^{R} l_j(\tau) \boldsymbol{x^{kj}}, \quad l_j(\tau) = \prod_{r=0, r\neq j}^{R} \frac{\tau - \tau_r}{\tau_j - \tau_r}, \quad t = t_k + \tau h_k, \quad \tau \in [0, 1],$$

mentres que, en canto ás variables de estado alxébricas e de control, coas notación análogas $\boldsymbol{u^{kj}}$ e $\boldsymbol{z^{kj}}$, empréganse R puntos de interpolación para obter:

$$\tilde{\boldsymbol{z}}^{\boldsymbol{I_k}}(t) = \sum_{j=1}^{R} \bar{l}_j(\tau) \boldsymbol{z^{kj}}, \quad \tilde{\boldsymbol{u}}^{\boldsymbol{I_k}}(t) = \sum_{j=1}^{R} \bar{l}_j(\tau) \boldsymbol{u^{kj}}, \quad \bar{l}_j(\tau) = \prod_{r=1, r\neq j}^{R} \frac{\tau - \tau_r}{\tau_j - \tau_r}.$$

E entón, denotando $\dot{l}_j(\tau) = \frac{\mathrm{d}l_j}{\mathrm{d}t}(\tau)$, tense a discretización da ecuación (3.3) por:

$$\sum_{j=0}^{R} \dot{l}_j(\tau_r) \boldsymbol{x^{kj}} - h_k \boldsymbol{f}(\boldsymbol{x^{kr}}, \boldsymbol{z^{kr}}, \boldsymbol{u^{kr}}, \boldsymbol{p}) = \boldsymbol{0}, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1.$$

Ademais, se $n_{col} > 1$, cando se traballa con interpolación recoméndase impor restricións de continuidade nos extremos dos intervalos, razón pola que se incorporou τ_0 nos estados diferenciais:

$$\boldsymbol{x^{k+1,0}} = \sum_{j=0}^{R} l_j(1)\boldsymbol{x^{kj}}, \quad k = 0, \dots, n_{col} - 2,$$
$$\boldsymbol{x^{n_{col}}} = \sum_{j=0}^{R} l_j(1)\boldsymbol{x^{(n_{col}-1)j}}, \quad \boldsymbol{x^{0,0}} = \boldsymbol{x^0}.$$

Exemplo 3.12 (Freada dun bloque)**.** Neste caso, o problema do Exemplo 1.6 resultaría en:

$$\begin{aligned}
\min_{x^{kr}, z^{kr}, u^{kr}} \quad & x^{n_{col}} \\
\text{s. a} \quad & \sum_{j=0}^{R} \dot{l}_j(\tau_r) x^{kj} - h_k\left(-z^{kr} - u^{kr}\right) = 0, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& 4 = \left(z^{kr}\right)^2 + \left(u^{kr}\right)^2, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \leq x^{kr} \leq 3, \quad r = 0, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \leq z^{kr}, u^{kr} \leq 2, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& x^{k+1,0} = \sum_{j=0}^{R} l_j(1) x^{kj}, \quad k = 0, \dots, n_{col} - 2, \\
& x^{n_{col}} = \sum_{j=0}^{R} l_j(1) x^{(n_{col}-1),j}, \\
& x^{0,0} = 3.
\end{aligned}$$

Exemplo 3.13 (Reactor descontinuo)**.** O Exemplo 1.7 resultaría mediante interpolación en:

$$\begin{aligned}
\min_{\boldsymbol{x^{kr}}, u^{kr}} \quad & -x_2^{n_{col}} \\
\text{s. a} \quad & \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_1^{kj} - h_k\left(-x_1^{kr}\left(u^{kr} + \frac{\left(u^{kr}\right)^2}{2}\right)\right) = 0, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_2^{kj} - h_k x_1^{kr} u^{kr} = 0, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& 0 \leq u^{kr} \leq 5, \quad r = 1, \dots, R, \quad k = 0, \dots, n_{col} - 1, \\
& x_1^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_1^{kj}, \quad k = 0, \dots, n_{col} - 2, \\
& x_2^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_2^{kj}, \quad k = 0, \dots, n_{col} - 2, \\
& x_1^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_1^{(n_{col}-1),j}, \quad x_2^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_2^{(n_{col}-1),j}, \\
& x_1^{0,0} = 1, \quad x_2^{0,0} = 0.
\end{aligned}$$

Exemplo 3.14 (Carro con péndulo - Mayer)**.** Analogamente, o Exemplo 1.10 tamén se pode discretizar nesta sección como:

$$
\begin{aligned}
\min_{\boldsymbol{x^{kr}},u^{kr}} \quad & x_5^{n_{col}} \\
\text{s. a} \quad & \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_1^{kj} - h_k x_3^{kr} = 0, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_2^{kj} - h_k x_4^{kr} = 0, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_3^{kj} - h_k \left(\frac{l m_2 \operatorname{sen}\left(x_2^{kr}\right) \left(x_4^{kr}\right)^2 + u^{kr}}{m_1 + m_2\left(1 - \cos^2\left(x_2^{kr}\right)\right)} \right. \\
& \qquad \left. + \frac{m_2 g \cos\left(x_2^{kr}\right) \operatorname{sen}\left(x_2^{kr}\right)}{m_1 + m_2\left(1 - \cos^2\left(x_2^{kr}\right)\right)} \right) = 0, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_4^{kj} + h_k \left(\frac{l m_2 \cos\left(x_2^{kr}\right) \operatorname{sen}\left(x_2^{kr}\right) \left(x_4^{kr}\right)^2 + u^{kr} \cos\left(x_2^{kr}\right)}{l m_1 + l m_2\left(1 - \cos^2\left(x_2^{kr}\right)\right)} \right. \\
& \qquad \left. + \frac{(m_1+m_2) g \operatorname{sen}\left(x_2^{kr}\right)}{l m_1 + l m_2\left(1 - \cos^2\left(x_2^{kr}\right)\right)} \right) = 0, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r) x_5^{kj} - h_k \left(u^{kr}\right)^2 = 0, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& -d_{max} \le x_1^{kr} \le d_{max}, \quad r = 0,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& -u_{max} \le u^{kr} \le u_{max}, \quad r = 1,\dots,R, \quad k = 0,\dots,n_{col}-1, \\
& x_1^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_1^{kj}, \quad x_2^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_2^{kj}, \quad x_3^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_3^{kj}, \\
& \qquad\qquad x_4^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_4^{kj}, \quad x_5^{k+1,0} = \sum_{j=0}^{R} l_j(1) x_5^{kj}, \quad k = 0,\dots,n_{col}-2, \\
& x_1^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_1^{(n_{col}-1),j}, \quad x_2^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_2^{(n_{col}-1),j}, \quad x_3^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_3^{(n_{col}-1),j}, \\
& \qquad\qquad x_4^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_4^{(n_{col}-1),j}, \quad x_5^{n_{col}} = \sum_{j=0}^{R} l_j(1) x_5^{(n_{col}-1),j}, \\
& x_1^{0,0} = 0, \quad x_2^{0,0} = 0, \quad x_3^{0,0} = 0, \quad x_4^{0,0} = 0, \quad x_5^{0,0} = 0, \\
& x_1^{n_{col}} = d, \quad x_2^{n_{col}} = \pi, \quad x_3^{n_{col}} = 0, \quad x_4^{n_{col}} = 0.
\end{aligned}
$$

3.4. Programación non linear

Antes de continuar, obsérvase que os métodos directos se basean na transformación do problema orixinal nun de programación non linear (NLP), que pode contar só con variables independentes como na *collocation* ou tamén coas funcións de estado como variables dependentes nos métodos de *direct single shooting* e *multiple single shooting*. En calquera caso, preséntase a

continuación unha serie de coñecementos básicos para tratar os NLPs. Para elo, consultáronse [1] e [9].

3.4.1. Preliminares

Supóñase que se quere escoller unha serie de variables reais $\boldsymbol{\xi} = (\xi_1, \xi_2, \dots, \xi_n)$ para minimizar unha función escalar:

$$F(\boldsymbol{\xi})$$

suxeita a $m \leq n$ restricións:

$$\boldsymbol{c}(\boldsymbol{\xi}) = \mathbf{0}.$$

Obter un óptimo é equivalente a minimizar o Lagrangiano do problema:

$$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\lambda}) = F(\boldsymbol{\xi}) - \boldsymbol{\lambda}^T \boldsymbol{c}(\boldsymbol{\xi}),$$

onde as compoñentes do vector $\boldsymbol{\lambda}$ se coñecen como multiplicadores de Lagrange.

Deste xeito, nun punto óptimo $(\boldsymbol{\xi}^*, \boldsymbol{\lambda}^*)$ hanse de verificar as condicións necesarias:

$$\begin{aligned} \mathbf{0} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}^*, \boldsymbol{\lambda}^*) = \frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\xi}}(\boldsymbol{\xi}^*) - \left(\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{\xi}}(\boldsymbol{\xi}^*)\right)^T \boldsymbol{\lambda}^*, \\ \mathbf{0} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\xi}^*, \boldsymbol{\lambda}^*) = -\boldsymbol{c}(\boldsymbol{\xi}^*). \end{aligned}$$

Logo, aplicando o método iterativo de Newton para máis dunha dimensión (véxase [1]), partindo dun punto $(\boldsymbol{\xi}, \boldsymbol{\lambda})$ e querendo calcular $(\hat{\boldsymbol{\xi}}, \hat{\boldsymbol{\lambda}})$, chégase ao sistema:

$$\begin{aligned} -\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\xi}^2}(\boldsymbol{\xi}, \boldsymbol{\lambda})(\hat{\boldsymbol{\xi}} - \boldsymbol{\xi}) + \left(\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{\xi}}(\boldsymbol{\xi})\right)^T \hat{\boldsymbol{\lambda}} &= \frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{\xi}}(\boldsymbol{\xi}), \\ -\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{\xi}}(\boldsymbol{\xi})(\hat{\boldsymbol{\xi}} - \boldsymbol{\xi}) &= \boldsymbol{c}(\boldsymbol{\xi}). \end{aligned} \tag{3.4}$$

Cómpre salientar que no caso dunha función obxectivo F cuadrática e restricións $\boldsymbol{c}$ lineais, o algoritmo de Newton converxe nunha iteración, é dicir, as condicións anteriores devolven o óptimo.

3.4.2. Programación cuadrática

Para resolver os problemas NLP son de grande utilidade os coñecidos como problemas de programación cuadrática (*quadratic programming*, QP). A súa forma xeral é:

$$\min_{\boldsymbol{\xi}} F(\boldsymbol{\xi}) \tag{3.5a}$$

suxeito ás restricións lineais:

$$\mathbf{A}\boldsymbol{\xi} = \boldsymbol{a}, \tag{3.5b}$$

$$\mathbf{B}\boldsymbol{\xi} \geq \boldsymbol{b}, \tag{3.5c}$$

onde, sendo $\mathbf{H}$ unha matriz definida positiva, a función obxectivo é da forma cuadrática:

$$F(\boldsymbol{\xi}) = \boldsymbol{g}^T\boldsymbol{\xi} + \frac{1}{2}\boldsymbol{\xi}^T\mathbf{H}\boldsymbol{\xi},$$

con $\mathbf{A}$ e $\mathbf{B}$ matrices e $\boldsymbol{a}$, $\boldsymbol{b}$ e $\boldsymbol{g}$ vectores de números reais.

Así, para resolver numericamente (3.5) existe un método iterativo coñecido como o método de conxunto activo, que consiste en ir resolvendo iterativamente problemas como os da subsección 3.4.1. Para elo, introdúcese previamente a seguinte definición de restrición activa.

Definición 3.15 (Restrición activa)**.** Sexa o problema (3.5). Dise que unha restrición de (3.5c) é activa na solución $\boldsymbol{\xi}^*$ se verifica a igualdade, é dicir, $(\mathbf{B}\boldsymbol{\xi}^*)_i = \boldsymbol{b}_i$. O conxunto das restricións activas na solución $\boldsymbol{\xi}^*$ denomínase conxunto activo e denótase por $\mathcal{A}$.

Algoritmo 3.16 (Método do conxunto activo)**.** *Dada unha estimación do conxunto activo $\mathcal{A}^0$ e un punto inicial $\boldsymbol{\xi}^{\mathbf{0}}$ que verifique as restricións do problema, o algoritmo procede do seguinte xeito:*

1. *Dados $\boldsymbol{\xi}^{\boldsymbol{\nu}}$ e $\mathcal{A}^\nu$, calcúlase o mínimo do obxectivo cuadrático suxeito ás restricións* (3.5b) *e ás incluídas na estimación do conxunto activo. É dicir, resólvese con esas restricións o sistema* (3.4) *e obtense $\hat{\boldsymbol{\xi}}$.*

2. *Se $\boldsymbol{\Delta\xi} = \hat{\boldsymbol{\xi}} - \boldsymbol{\xi}^{\boldsymbol{\nu}}$, hai que tomar o maior $\alpha \in [0,1]$ tal que:*

$$\boldsymbol{\xi}^{\boldsymbol{\nu+1}} = \boldsymbol{\xi}^{\boldsymbol{\nu}} + \alpha\boldsymbol{\Delta\xi}$$

   respecta todas as inecuacións que non forman parte do conxunto de restricións activas actual.

3. - *Se $\alpha < 1$, tómase aquela restrición que limitou a escolla do α e engádese ao conxunto activo $\mathcal{A}^{\nu+1}$. Volta ao paso 1.*
   - *Se $\alpha = 1$, compróbase o signo dos multiplicadores de Lagrange asociados ao problema simplificado e:*
     - *Se todos os multiplicadores son non negativos, remátase o algoritmo.*
     - *En caso contrario, suprímese do conxunto activo a desigualdade co multiplicador máis negativo e tórnase ao paso 1.*

3.4.3. Programación cuadrática secuencial

Xa se está en condicións de tratar numericamente un problema de programación non linear da forma:

$$\begin{aligned} \min_{\boldsymbol{\xi}} \quad & F(\boldsymbol{\xi}) \\ \text{s. a} \quad & \boldsymbol{G}(\boldsymbol{\xi}) = \mathbf{0}, \\ & \boldsymbol{H}(\boldsymbol{\xi}) \leq \mathbf{0}. \end{aligned} \tag{3.6}$$

Para elo, considérase o Lagrangiano asociado a este problema:

$$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = F(\boldsymbol{\xi}) + \boldsymbol{\lambda}^T \boldsymbol{G}(\boldsymbol{\xi}) + \boldsymbol{\mu}^T \boldsymbol{H}(\boldsymbol{\xi}),$$

e procédese co coñecido como algoritmo de programación cuadrática secuencial (*sequential quadratic programming*, SQP). Este baséase en aproximar o problema por un cuadrático do tipo (3.5) e repetir o proceso até chegar a unha solución aceptable. En concreto, tense que decidir unha estratexia para escoller a lonxitude de paso α en cada iteración e outra para establecer unha condición terminal, como por exemplo:

$$\left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} \boldsymbol{\Delta\xi} \right\| \leq \varepsilon,$$

onde $\|\cdot\|$ denota unha norma do espazo euclidiano. Por exemplo, a norma dous ou a norma infinito. Unha descrición detallada atópase en [1].

Algoritmo 3.17 (SQP). *Dado o iterante* $(\boldsymbol{\xi}^{\boldsymbol{\nu}}, \boldsymbol{\lambda}^{\boldsymbol{\nu}}, \boldsymbol{\mu}^{\boldsymbol{\nu}})$*:*

1. *Resólvese o problema de minimizar a aproximación cuadrática do Lagrangiano baixo a aproximación linear das restricións. Se* $\boldsymbol{\Delta\xi} = \boldsymbol{\xi} - \boldsymbol{\xi}^{\boldsymbol{\nu}}$*, entón resólvese:*

$$\begin{aligned} \min_{\boldsymbol{\Delta\xi}} \quad & \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}^{\boldsymbol{\nu}}, \boldsymbol{\lambda}^{\boldsymbol{\nu}}, \boldsymbol{\mu}^{\boldsymbol{\nu}}) \boldsymbol{\Delta\xi} + \frac{1}{2} \boldsymbol{\Delta\xi}^T \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\xi}^2}(\boldsymbol{\xi}^{\boldsymbol{\nu}}, \boldsymbol{\lambda}^{\boldsymbol{\nu}}, \boldsymbol{\mu}^{\boldsymbol{\nu}}) \boldsymbol{\Delta\xi} \\ s.a \quad & \boldsymbol{G}(\boldsymbol{\xi}^{\boldsymbol{\nu}}) + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}^{\boldsymbol{\nu}}) \boldsymbol{\Delta\xi} = \mathbf{0}, \\ & \boldsymbol{H}(\boldsymbol{\xi}^{\boldsymbol{\nu}}) + \frac{\partial \boldsymbol{H}}{\partial \boldsymbol{\xi}}(\boldsymbol{\xi}^{\boldsymbol{\nu}}) \boldsymbol{\Delta\xi} \leq \mathbf{0}. \end{aligned}$$

2. *A partir de* $\boldsymbol{\Delta\xi}$ *e dos multiplicadores de Lagrange asociados* $\tilde{\boldsymbol{\lambda}}$ *e* $\tilde{\boldsymbol{\mu}}$*, calcúlanse os novos iterantes como:*

$$\begin{aligned} \boldsymbol{\xi}^{\boldsymbol{\nu+1}} &= \boldsymbol{\xi}^{\boldsymbol{\nu}} + \alpha \boldsymbol{\Delta\xi}, \\ \boldsymbol{\lambda}^{\boldsymbol{\nu+1}} &= \boldsymbol{\lambda}^{\boldsymbol{\nu}} + \alpha(\tilde{\boldsymbol{\lambda}} - \boldsymbol{\lambda}^{\boldsymbol{\nu}}), \\ \boldsymbol{\mu}^{\boldsymbol{\nu+1}} &= \boldsymbol{\mu}^{\boldsymbol{\nu}} + \alpha(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}^{\boldsymbol{\nu}}), \end{aligned}$$

con lonxitude de paso $\alpha \leq 1$*.*

3. Se a condición terminal non se verifica para o novo iterante, tórnase ao paso 1.

Por suposto, existen outras metodoloxías que tamén se poden empregar como o algoritmo de punto interior explicado en [1]. Porén, non se afondará en distintas estratexias xa que saen fóra da temática abordada neste traballo.

Capítulo 4

Refinamento da malla temporal

O proceso de resolución por un método directo do problema (1.4) consta de tres pasos: a transformación nun problema de dimensión finita, a correspondente resolución mediante técnicas de programación non linear e o refinamento da malla para repetir o proceso até obter un resultado con erro pequeno. Así pois, após unha presentación xeral dos métodos de discretización no Capítulo 3, afróntanse agora dúas estratexias con respecto ao refinamento da partición temporal para mellorar o resultado.

En primeiro lugar, tomando como referencia [1], enúnciase un algoritmo de refinamento da malla que se aplica despois de obter a solución do NLP discretizado. Este consiste en dividir a partición temporal naqueles intervalos nos que o erro é maior, para volver resolver o NLP coa malla máis fina. En particular, denominarase estratexia de elementos finitos fixos pois simplemente vanse engadindo máis puntos no canto de modificar os existentes. Ademais, para non dificultar a comprensión coa notación e por ser a técnica empregada en MATLAB no Capítulo 5, esta estratexia exponse baixo as notacións da *collocation* con esquemas de Runge-Kutta: variables $\boldsymbol{x^k}$, $\boldsymbol{z^k}$ e $\boldsymbol{u^k}$.

En segundo lugar, aínda que non incluír os elementos finitos h_k da malla como variables da optimización facilita a resolución do NLP, isto pode implicar unha converxencia máis lenta. Polo tanto, como se fai en [3], na sección 4.2 explícase un algoritmo que calcula a solución e malla óptimas ao mesmo tempo: elementos finitos movibles. Neste caso, por mor das restricións adicionais que convén engadir para controlar o erro, trabállase coa *collocation* con interpolación e puntos móbiles de Gauss ou Radau.

Agora ben, tanto en elementos finitos fixos coma movibles, o método de *collocation* pode ser escollido libremente se se levan a cabo as modificacións axeitadas. Por exemplo, é inmediato decatarse de que a subsección 4.1.1 pode ser enfocada dende a interpolación de Gauss-Radau se se traballa con esta estratexia.

4.1. Refinamento da malla con elementos finitos fixos

Tras aplicar unha discretización coma por exemplo as da subsección 3.3.1, cóntase con $n_{col}+1$ puntos de mallado e un problema de programación non linear. Logo, tras a aplicación dun algoritmo como o SQP, disponse dunha solución discreta do problema de programación non linear, formada polos valores nos puntos da malla $\boldsymbol{x^k}$, $\boldsymbol{z^k}$ e $\boldsymbol{u^k}$ e os parámetros $\tilde{\boldsymbol{p}}$.

4.1.1. Obtención dunha solución funcional

A intención é aproximar as funcións $\boldsymbol{x}(\cdot)$, $\boldsymbol{z}(\cdot)$ e $\boldsymbol{u}(\cdot)$ mediante interpolación con *splines*: $\tilde{\boldsymbol{x}}(\cdot)$, $\tilde{\boldsymbol{z}}(\cdot)$ e $\tilde{\boldsymbol{u}}(\cdot)$, respectivamente, cos que poder avaliar en calquera instante temporal. Deste xeito, para as variables de estado diferenciais, $\tilde{\boldsymbol{x}}(t)$ está determinado grazas ás imposicións:

$$\tilde{\boldsymbol{x}}(t_k) = \boldsymbol{x^k}, \quad \frac{\mathrm{d}\tilde{\boldsymbol{x}}}{\mathrm{d}t}(t_k) = \boldsymbol{f^k},$$

en todos os puntos t_k da malla, podéndose obter polinomios cúbicos a cachos de clase 1.

Do mesmo xeito, os estados alxébricos e controis represéntanse mediante $\tilde{\boldsymbol{z}}(t)$ e $\tilde{\boldsymbol{u}}(t)$, que quedan determinados grazas a:

$$\tilde{\boldsymbol{z}}(t_k) = \boldsymbol{z^k}, \quad \tilde{\boldsymbol{u}}(t_k) = \boldsymbol{u^k},$$

podéndose obter polinomios lineais a cachos continuos.

En [1] recoméndase o uso duns *splines* especiais, chamados *B-splines*, que basicamente conforman unha base do espazo de *splines* e son sinxelos de tratar computacionalmente. Non obstante, a idea de representación da solución é a mesma: construír unha función de tipo polinómico a cachos coa que representar unha solución continua. Ademais, como no código do Capítulo 5 só se emprega a función *spline* de MATLAB, non se afondará na definición de *B-splines* para só tratar con coñecementos do grao e non saírse da liña do traballo. Para unha explicación rigorosa dos *B-splines*, consúltese [4].

4.1.2. Estimación do erro

A continuación, preténdese estimar o erro da solución acadada. Para elo, unha estratexia empregada en [1] é supor que as variables de estado alxébricas, controis e parámetros $(\boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p})$ foron aproximadas correctamente e estímase a diferenza entre $\tilde{\boldsymbol{x}}(t)$ e $\boldsymbol{x}(t)$. Malia ser unha suposición necesaria para este algoritmo que ten boas propiedades, non é moi rigoroso pois o carácter óptimo desas variables non é comprobado. Ademais, trabállase con seguridade se $\boldsymbol{x}(t) \in \mathcal{C}^1$ e $\boldsymbol{z}(t), \boldsymbol{u}(t) \in \mathcal{C}^0$; noutro caso depende do problema.

Así, cando se considera unha ecuación diferencial ordinaria do tipo $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{u}, \boldsymbol{p})$ e resólvese mediante un esquema numérico, distínguese entre erro global e erro local. Dunha banda, o erro global no instante t_{k+1} defínese como a diferenza entre $\boldsymbol{x^{k+1}}$ e $\boldsymbol{x}(t_{k+1})$. Doutra banda, chámase erro local no instante t_{k+1} á diferenza entre $\boldsymbol{x^{k+1}}$ e a solución avaliada en t_{k+1} da ecuación diferencial que pasa por $\boldsymbol{x^k}$ en t_k. No caso de que a orde do método numérico sexa q, o erro local no intervalo k é da forma:

$$\varepsilon_k \approx \|\boldsymbol{c^{I_k}} h_k^{q+1}\|,$$

onde $\boldsymbol{c^{I_k}}$ adoita depender das derivadas parciais da función $\boldsymbol{f}$. Por exemplo, a discretización de tipo Hermite-Simpson é de orde $q = 4$ e a dos trapecios $q = 2$.

No entanto, cando se considera unha DAE, o método numérico pódese ver afectado por unha redución na orde, é dicir:

$$\varepsilon_k \approx \|\boldsymbol{c^{I_k}} h_k^{q-r+1}\|, \tag{4.1}$$

onde r é a redución da orde. E para estimar esta diminución, o primeiro é aproximar o erro.

Sexa entón o intervalo da malla $I_k = [t_k, t_k + h_k]$, con $\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{z}}(t), \tilde{\boldsymbol{u}}(t)$ e $\tilde{\boldsymbol{p}}$ as solucións obtidas en dito intervalo alén do NLP e da interpolación das funcións de t. Verifícase:

$$\boldsymbol{x}(t_k + h_k) = \boldsymbol{x}(t_k) + \int_{t_k}^{t_k+h_k} \dot{\boldsymbol{x}}(t)\mathrm{d}t = \boldsymbol{x}(t_k) + \int_{t_k}^{t_k+h_k} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{z}(t), \boldsymbol{u}(t), \boldsymbol{p})\mathrm{d}t.$$

Porén, para traballar con valores coñecidos:

$$\hat{\boldsymbol{x}}(t_k + h_k) = \boldsymbol{x}(t_k) + \int_{t_k}^{t_k+h_k} \boldsymbol{f}(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{z}}(t), \tilde{\boldsymbol{u}}(t), \tilde{\boldsymbol{p}})\mathrm{d}t,$$

que, denotando $\tilde{\boldsymbol{f}} = \boldsymbol{f}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{z}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{p}})$, pode modificarse:

$$\begin{aligned}
\hat{\boldsymbol{x}}^{\boldsymbol{k+1}} &= \boldsymbol{x^k} + \int_{t_k}^{t_k+h_k} \tilde{\boldsymbol{f}}\,\mathrm{d}t \\
&= \boldsymbol{x^k} + \int_{t_k}^{t_k+h_k} \dot{\tilde{\boldsymbol{x}}}\,\mathrm{d}t - \int_{t_k}^{t_k+h_k} \dot{\tilde{\boldsymbol{x}}}\,\mathrm{d}t + \int_{t_k}^{t_k+h_k} \tilde{\boldsymbol{f}}\,\mathrm{d}t \\
&= \boldsymbol{x^k} + \tilde{\boldsymbol{x}}^{\boldsymbol{k+1}} - \tilde{\boldsymbol{x}}^{\boldsymbol{k}} - \int_{t_k}^{t_k+h_k} \left(\dot{\tilde{\boldsymbol{x}}} - \tilde{\boldsymbol{f}}\right)\mathrm{d}t,
\end{aligned}$$

de xeito que, tendo en conta que por definición $\boldsymbol{x^k} = \tilde{\boldsymbol{x}}^{\boldsymbol{k}}$, resulta a seguinte expresión:

$$\tilde{\boldsymbol{x}}^{\boldsymbol{k+1}} - \hat{\boldsymbol{x}}^{\boldsymbol{k+1}} = \int_{t_k}^{t_k+h_k} \left(\dot{\tilde{\boldsymbol{x}}} - \tilde{\boldsymbol{f}}\right)\mathrm{d}t.$$

E mediante limitación modular en cada compoñente, obtense:

$$\left|\tilde{x}_i^{k+1} - \hat{x}_i^{k+1}\right| = \left|\int_{t_k}^{t_k+h_k} \left(\dot{\tilde{x}}_i - \tilde{f}_i\right)\mathrm{d}t\right| \leq \int_{t_k}^{t_k+h_k} \left|\dot{\tilde{x}}_i - \tilde{f}_i\right|\mathrm{d}t.$$

Daquela, defínese o erro local absoluto no intervalo k por compoñentes como:

$$\eta_i^{I_k} = \int_{t_k}^{t_k+h_k} \left| \dot{\tilde{x}}_i(t) - \tilde{f}_i(t) \right| \mathrm{d}t.$$

Nestes termos, pódese aproximar o erro local relativo no intervalo $[t_k, t_{k+1}]$ por:

$$\varepsilon_k \approx \max_i \frac{\eta_i^{I_k}}{(w_i+1)}, \quad \text{con} \quad w_i = \max_{k=0,\dots,n_{col}} [|\tilde{x}_i^k|, |\dot{\tilde{x}}_i^k|]. \tag{4.2}$$

Así pois, ε_k é o máximo erro integrado relativo no intervalo $[t_k, t_k + h_k]$ de todas as compoñentes nas ecuacións $\dot{\boldsymbol{x}} - \boldsymbol{f}$.

Cómpre facer fincapé en que o cálculo práctico de $\boldsymbol{\eta^{I_k}} = (\eta_1^{I_k}, \dots, \eta_{n_x}^{I_k})$ se debe facer mediante unha fórmula de cuadratura con alta precisión e cunha tolerancia preto do erro da máquina. Isto é por mor da súa relevancia á hora de estimar a redución da orde na seguinte sección.

4.1.3. Estimación da redución da orde

Seguindo o camiño de [1], proponse estimar a redución na orde do método numérico mediante a comparación de dúas mallas consecutivas. Para elo, suponse que a malla actual foi obtida engadindo N puntos á previa.

Analízase o comportamento dunha única variable x_i nun único intervalo I_k, de xeito que:

$$\theta = c\left(h_k^{q-r+1}\right)$$

representa o erro obtido na malla previa e:

$$\eta = c\left(\frac{h_k}{N+1}\right)^{q-r+1}$$

o erro na nova malla no mesmo intervalo. Se se asume que r e c non dependen da malla considerada, con r constante en todo I_k, despéxase:

$$\hat{r} = q + 1 - \frac{\log\left(\frac{\theta}{\eta}\right)}{\log(N+1)}. \tag{4.3}$$

Co cal, se nint denota o enteiro máis preto, pódese tomar como r o seguinte número enteiro non negativo:

$$r = \max[0, \min(\mathrm{nint}(\hat{r}), q)]. \tag{4.4}$$

Ollo, como xa se mencionou na subsección 4.1.2, esta aproximación é moi sensible aos erros calculados nas distintas mallas. Para máis detalles véxase [1].

Observación 4.1. Na práctica pretenderase calcular unha redución de orde r_k para cada intervalo, independentemente da compoñente x_i coa que se traballe. Neste traballo decidiuse tomar a media das expresións (4.3) antes de aplicar (4.4). Daquela, para cada $[t_k, t_{k+1}]$ terase un único r_k.

4.1.4. Construción da nova malla

Por último, unha vez computados os erros aproximados preténdese empregar esa información para refinar a partición temporal e conseguir unha solución máis axustada na seguinte iteración.

Unha vez obtida a solución e o erro na iteración $j-$ésima, emprégase a terminoloxía malla antiga para referise á da iteración $j-1$, malla actual para denotar a da iteración j xusto calculada, e malla nova para nomear a que se pretende obter. Logo, o obxectivo é deseñar a malla nova tal que se reduza o erro de discretización, mais engadindo un número axeitado de puntos para que o tamaño do NLP non medre estrepitosamente. A idea fundamental é ir engadindo puntos ás divisións que teñan máis erro.

Tendo en conta as expresións (4.1) e (4.2) para ε_k e denotando por r_k á redución da orde no intervalo $[t_k, t_k + h_k]$, despéxase:

$$\|\boldsymbol{c}^{\boldsymbol{I_k}}\| \approx \max_i \frac{\eta_i^{I_k}}{(w_i+1)h_k^{q-r_k+1}}.$$

Sexa agora $N_k \geq 0$ o número de puntos que se engaden ao interior do intervalo $[t_k, t_k + h_k]$. Tense a seguinte nova expresión para ε_k, que será unha aproximación para o erro en cada un dos novos $N_k + 1$ subintervalos:

$$\varepsilon_k \approx \|\boldsymbol{c}^{\boldsymbol{I_k}}\| \left(\frac{h_k}{N_k+1}\right)^{q-r_k+1} \approx \max_i \frac{\eta_i^{I_k}}{(w_i+1)} \left(\frac{1}{N_k+1}\right)^{q-r_k+1}. \tag{4.5}$$

Daquela, a malla nova pódese determinar mediante o seguinte problema de programación enteira:

$$\begin{aligned} \min_{N_k} \quad & \max_k \varepsilon_k \\ \text{s. a} \quad & \sum_k N_k \leq M-1, \\ & N_k \leq M_1, \quad \forall k. \end{aligned} \tag{4.6}$$

É dicir, deséxase atopar o conxunto de enteiros N_k para todo k que minimicen o máximo erro engadindo como moito $M-1$ puntos e, ademais, non se poden engadir máis de M_1 puntos a cada intervalo. Estas constantes M e M_1 fíxanse de antemán.

Nótese que hai dúas situacións ben diferenciadas cuxos resultados son intuitivos. Dunha banda, se o erro discretizado está distribuído de forma homoxénea, a malla nova dividirá cada un dos intervalos da actual. Neste caso defínese o erro medio como:

$$\bar{\varepsilon} = \frac{1}{n_{col}} \sum_{k=1}^{n_{col}} \varepsilon_k.$$

Doutra banda, se existe un intervalo $[t_a, t_a+1]$ cun erro moito máis elevado que o resto, engadiranse M_1 puntos a dito intervalo. Estas ideas están detrás da estratexia heurística complementaria que se presenta no próximo Algoritmo 4.2.

4.1.5. Algoritmo de refinamento da malla con elementos finitos fixos

En conclusión, a partir das seccións previas pódese enunciar un algoritmo de refinamento para incorporar á estratexia de resolución do problema. Ademais de seleccionar os novos puntos, tamén se ofrece a posibilidade de escoller a orde da nova discretización. Isto é, á hora de formular o problema coa nova partición temporal, decídese con que método numérico traballar. Por exemplo, o método dos trapecios é de orde 2 mentres que o de Hermite-Simpson é de orde 4. En xeral, adóitase comezar cun método de orde baixa para despois incrementalo nunha iteración avanzada.

Sexa j o índice da iteración do proceso no que se atopa a resolución e denótese por δ á tolerancia do erro escollida para limitar o erro da discretización. Disponse dos erros discretizados na malla actual ε_k para cada intervalo, sábese cal é a orde q do método de discretización empregado até o momento e inicialízase $N_k = 0$.

Algoritmo 4.2 (Refinamento da malla con elementos finitos fixos)**.**

1. ***Obtención dunha solución funcional.*** *Calcúlanse as representacións mediante splines* $\tilde{\boldsymbol{x}}(t)$, $\tilde{\boldsymbol{z}}(t)$ *e* $\tilde{\boldsymbol{u}}(t)$ *a partir da solución discreta.*

2. ***Estimación do erro.*** *Calcúlase a estimación do erro discretizado* ε_k *en cada intervalo da malla actual mediante* (4.2).

3. ***Selección da orde*** q ***para a nova malla***

   a) *Se* $q < 4$ *e* $\varepsilon_k \leq 2\bar{\varepsilon}$*, entón establécese* $q = 4$ *(por exemplo, Hermite-Simpson) e remátase o algoritmo.*

   b) *Se* $q < 4$ *e* $j > 2$*, entón* $q = 4$ *e remátase o algoritmo.*

4. ***Estimación da redución da orde.*** *Comparación das mallas vella e actual para establecer un valor* r_k *a partir de* (4.3) *e* (4.4).

5. ***Construción da nova malla.*** *Hai dúas opcións, ou ben resolver o problema* (4.6) *de programación enteira, ou ben aplicar o seguinte algoritmo heurístico:*

   a) *Calcúlase o intervalo* $[t_a, t_{a+1}]$ *co máximo erro, é dicir:*

   $$\varepsilon_a = \max_k \varepsilon_k.$$

   b) *Remátase o algoritmo se se cumpre algún dos seguintes criterios de parada:*

1) O erro satisfai a tolerancia, $\varepsilon_a \leq \delta$.

2) Foron engadidos $M-1$ *puntos.*

3) Foron engadidos M_1 *puntos a un só intervalo.*

c) *Engádese un punto ao intervalo* $[t_a, t_{a+1}]$, *é dicir,* $N_a = N_a + 1$.

d) *Actualízase a predición do erro para o intervalo* $[t_a, t_{a+1}]$ *en* (4.5).

e) *Tórnase ao paso* 5a.

Observación 4.3. É evidente que na práctica non se completa o Algoritmo 4.2 se as estimacións ε_k están por debaixo de certa tolerancia do erro. Neste caso, remataríase a resolución do problema e non procedería refinar a malla. Ademais, nótese que se pode ver o algoritmo presentado como un conxunto de algoritmos, no sentido de que hai varias posibilidades a seguir nos pasos 3 e 5.

4.2. Resolución e refinamento da malla con elementos finitos movibles

A diferenza do caso de elementos finitos fixos, o algoritmo aquí presentado involucra todo o proceso de resolución, non só o refinamento da malla entre cada NLP. Así pois, pártese dun problema de control óptimo que, como xa se comentou, decide atacarse dende a *collocation* con interpolación con puntos móbiles: $\tilde{\boldsymbol{x}}^{\boldsymbol{I_k}}$, $\tilde{\boldsymbol{z}}^{\boldsymbol{I_k}}$ e $\tilde{\boldsymbol{u}}^{\boldsymbol{I_k}}$.

4.2.1. Restricións sobre os elementos finitos

Para afrontar a resolución e refinamento conxuntamente, cómpre engadir restricións relativas ao erro e aos h_k. En primeiro lugar, traballando con n_{col} suficientemente grande e asumindo que os estados diferenciais son diferenciables até a orde precisa, o erro global $\boldsymbol{e}(t) = \boldsymbol{x}(t) - \tilde{\boldsymbol{x}}(t)$ pode ser representado como:

$$\max_{t\in[t_0,t_f]} \|\boldsymbol{e}(t)\| \leq C_1 \max_{k\in\{0,\ldots,n_{col}-1\}} \left(\max_{t\in[t_k,t_{k+1}]} \|\boldsymbol{T}^{\boldsymbol{I_k}}(t)\| \right) + O(h_k^{R+2}),$$

onde C_1 é unha constante coñecida, $\boldsymbol{T}^{\boldsymbol{I_k}}(t)$ obtense a partir da solución interpolada e $\|\cdot\|$ denota de novo unha norma do espazo euclidiano. En [3] apórtanse referencias que dan métodos para calcular dita limitación do erro. Mais en particular, denotando por $t_{k,nc} = t_k + h_k\tau_{nc}$, con $\tau_{nc} \in [0,1]$, un novo instante temporal non restrinxido na discretización de (3.3), tómase:

$$\boldsymbol{T}^{\boldsymbol{I_k}}(t_{k,nc}) = \begin{bmatrix} \dfrac{\mathrm{d}\tilde{\boldsymbol{x}}^{\boldsymbol{I_k}}(t_{k,nc})}{\mathrm{d}\tau} - h_k \boldsymbol{f}(\tilde{\boldsymbol{x}}^{\boldsymbol{I_k}}(t_{k,nc}), \tilde{\boldsymbol{z}}^{\boldsymbol{I_k}}(t_{k,nc}), \tilde{\boldsymbol{u}}^{\boldsymbol{I_k}}(t_{k,nc}), \boldsymbol{p}) \\ \boldsymbol{g}(\tilde{\boldsymbol{x}}^{\boldsymbol{I_k}}(t_{k,nc}), \tilde{\boldsymbol{z}}^{\boldsymbol{I_k}}(t_{k,nc}), \tilde{\boldsymbol{u}}^{\boldsymbol{I_k}}(t_{k,nc}), \boldsymbol{p}) \end{bmatrix},$$

e, definindo:

$$\bar{C} = \frac{1}{A}\int_0^{\tau_{nc}} \prod_{j=1}^{R}(s-\tau_j)\mathrm{d}s, \quad A = \prod_{j=1}^{R}(\tau_{nc}-\tau_j),$$

chégase a:

$$\|\boldsymbol{e}(t)\| \leq \bar{C}\|\boldsymbol{T}^{\boldsymbol{I_k}}(t_{k,nc})\|, \quad t \in [t_k, t_{k+1}].$$

Daquela, con esta terminoloxía e tomando unha tolerancia do erro $\delta > 0$, os valores das lonxitudes de paso h_k poden ser escollidos engadindo ao problema discretizado:

$$\begin{aligned} \sum_{k=0}^{n_{col}-1} h_k &= t_f - t_0, \\ h_k &\geq 0, \quad k = 0, \ldots, n_{col}-1, \\ \bar{C}\|\boldsymbol{T}^{\boldsymbol{I_k}}(t_{k,nc})\| &\leq \delta, \quad k = 0, \ldots, n_{col}-1. \end{aligned} \tag{4.7}$$

4.2.2. Restricións sobre a discretización dos controis

Supóñase que se teñen limitacións para os controis do tipo:

$$\boldsymbol{u_L} \leq \boldsymbol{u} \leq \boldsymbol{u_U}, \quad t \in [t_0, t_f],$$

como acontece habitualmente. Para evitar que estas non se verifiquen en puntos distintos dos da malla considerada, e facilitar o seu tratamento en compensación da dificultade engadida cos elementos finitos movibles, recoméndase traballar con aproximacións constantes ou lineais a cachos en cada intervalo. Isto é, se se opta por unha aproximación constante a anacos, defínese para $k = 0, \ldots, n_{col}-1$:

$$\boldsymbol{u^{kr}} = \boldsymbol{u^k}, \quad \boldsymbol{u_L} \leq \boldsymbol{u^k} \leq \boldsymbol{u_U}, \quad r = 1, \ldots, R, \tag{4.8}$$

e, se se opta por unha linear a anacos, tómanse valores no segmento que une $(t_k, \boldsymbol{u^{k,a}})$ e $(t_{k+1}, \boldsymbol{u^{k,b}})$, con valores $\boldsymbol{u^{k,a}}$ e $\boldsymbol{u^{k,b}}$ dentro das restricións, é dicir, para $k = 0, \ldots, n_{col}-1$:

$$\boldsymbol{u^{kr}} = \boldsymbol{u^{k,a}}(1-\tau_r) + \boldsymbol{u^{k,b}}\tau_r, \quad \boldsymbol{u_L} \leq \boldsymbol{u^{k,a}}, \boldsymbol{u^{k,b}} \leq \boldsymbol{u_U}, \quad r = 1, \ldots, R. \tag{4.9}$$

4.2.3. Algoritmo de elementos finitos movibles

Polo tanto, xa se está en condicións de presentar outro algoritmo de resolución dun problema de control óptimo. Para elo, primeiro hai que escoller unha malla inicial, fixando valores iniciais $\bar{\boldsymbol{u}}(t)$ e $\bar{\boldsymbol{p}}$ e resolvendo unha secuencia de NLPs ordenados comezando por $k = 0$. Tómase $\bar{h}_k$ a

solución de:

$$
\begin{aligned}
\max_{\boldsymbol{x^{kr}},\boldsymbol{z^{kr}},h_k} \quad & h_k \\
\text{s. a} \quad & \sum_{j=0}^{R} \dot{l}_j(\tau_r)\boldsymbol{x^{kj}} - h_k \boldsymbol{f}(\boldsymbol{x^{kr}},\boldsymbol{z^{kr}},\bar{\boldsymbol{u}}^{\boldsymbol{kr}},\bar{\boldsymbol{p}}) = \boldsymbol{0}, \quad r = 1,\dots,R, \\
& \boldsymbol{g}(\boldsymbol{x^{kr}},\boldsymbol{z^{kr}},\bar{\boldsymbol{u}}^{\boldsymbol{kr}},\bar{\boldsymbol{p}}) = \boldsymbol{0}, \quad r = 1,\dots,R, \\
& \bar{C}\|\boldsymbol{T}^{\boldsymbol{I_k}}(t_{k,nc})\| \leq \delta, \\
& 0 \leq h_k \leq t_f - t_0 - \sum_{k'=0}^{k-1} \bar{h}_{k'}.
\end{aligned}
\tag{4.10}
$$

Daquela, cando se acade a lonxitude $t_f - t_0$, xa se terán valores inicias para resolver o NLP discretizado coas restricións engadidas da subsección 4.2.1 e, se é necesario, da 4.2.2.

Exemplo 4.4 (Reactor descontinuo)**.** Ilústrase a continuación o problema obtido sobre o Exemplo 1.7:

$$
\begin{aligned}
\min_{\boldsymbol{x^{kr}},u^{kr},h_k} \quad & -x_2^{n_{col}} \\
\text{s. a} \quad & \sum_{j=0}^{R} \dot{l}_j(\tau_r)x_1^{kj} - h_k\left(-x_1^{kr}\left(u^{kr} + \frac{\left(u^{kr}\right)^2}{2}\right)\right) = 0, \quad r = 1,\dots,R,\ k = 0,\dots,n_{col}-1, \\
& \sum_{j=0}^{R} \dot{l}_j(\tau_r)x_2^{kj} - h_k x_1^{kr}u^{kr} = 0, \quad r = 1,\dots,R,\ k = 0,\dots,n_{col}-1, \\
& 0 \leq\ u^{kr} \leq 5, \quad r = 1,\dots,R,\ k = 0,\dots,n_{col}-1, \\
& x_1^{k+1,0} = \sum_{j=0}^{R} l_j(1)x_1^{kj}, \quad x_2^{k+1,0} = \sum_{j=0}^{R} l_j(1)x_2^{kj}, \quad k = 0,\dots,n_{col}-2, \\
& x_1^{n_{col}} = \sum_{j=0}^{R} l_j(1)x_1^{(n_{col}-1),j}, \quad x_2^{n_{col}} = \sum_{j=0}^{R} l_j(1)x_2^{(n_{col}-1),j}, \\
& x_1^{0,0} = 1, \quad x_2^{0,0} = 0, \\
& \sum_{k=0}^{n_{col}-1} h_k = t_f - t_0, \\
& h_k \geq 0, \quad k = 0,\dots,n_{col}-1, \\
& \bar{C}\|\boldsymbol{T}^{\boldsymbol{I_k}}(t_{k,nc})\| \leq \delta, \quad k = 0,\dots,n_{col}-1, \\
& u^{kr} = u^k, \quad u_L \leq u^k \leq u_U, \quad r = 1,\dots,R,\ k = 0,\dots n_{col}-1, \quad \text{ou} \\
& u^{kr} = u^{k,a}(1-\tau_r) + u^{k,b}\tau_r, \quad u_L \leq u^{k,a}, u^{k,b} \leq u_U, \quad r = 1,\dots,R,\ k = 0,\dots,n_{col}-1.
\end{aligned}
$$

Por último, a solución obtida será testada para ver se tamén o é do problema orixinal. En [3] proponse traballar cunha aproximación do Hamiltoniano do problema e observar se é constante.

Neste sentido, tomando como extensión da Definición 2.1 de Hamiltoniano:

$$\begin{aligned} H(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p},\boldsymbol{\lambda}(t),\boldsymbol{\mu}(t),\boldsymbol{\mu}(t)) =& (\boldsymbol{\lambda}(t))^T \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p}) \\ &+(\boldsymbol{\eta}(t))^T \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{z}(t),\boldsymbol{u}(t),\boldsymbol{p}) \\ &+(\boldsymbol{\mu}(t))^T \boldsymbol{c}(\boldsymbol{x}(t),\boldsymbol{u}(t)), \end{aligned}$$

pódese ver que ao longo da traxectoria óptima ten que ser constante (véxase [3]). Ademais, tamén se pode ver que os multiplicadores de Lagrange discretos obtidos na resolución do NLP, por exemplo mediante programación cuadrática secuencial (Algoritmo 3.17), serven para aproximar os multiplicadores do problema continuo. Daquela, proponse aproximar o Hamiltoniano nos nodos da malla mediante:

$$H^{kR+r} = \left(\boldsymbol{\lambda}^{\boldsymbol{kr}}\right)^T \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{kr}},\boldsymbol{z}^{\boldsymbol{kr}},\boldsymbol{u}^{\boldsymbol{kr}},\boldsymbol{p}) + \left(\boldsymbol{\eta}^{\boldsymbol{kr}}\right)^T \boldsymbol{g}(\boldsymbol{x}^{\boldsymbol{kr}},\boldsymbol{z}^{\boldsymbol{kr}},\boldsymbol{u}^{\boldsymbol{kr}},\boldsymbol{p}), \tag{4.11}$$

e comprobar o seu carácter constante.

En conclusión, obtense o algoritmo seguinte:

Algoritmo 4.5 (Algoritmo de elementos finitos movibles)**.**

1. *(Opcional) Divídese o intervalo de definición $[t_0,t_f]$ cunha partición pequena e resólvese o NLP resultante mediante collocation con* (4.8) *ou* (4.9) *para ter unha función inicial para $\boldsymbol{u}(t)$.*

2. *Dados uns valores iniciais para $\boldsymbol{u}(t)$ e $\boldsymbol{p}$, resólvese unha secuencia de NLPs* (4.10) *para $k=0,1,2,\ldots$ mentres $t_f - t_0 - \sum_{k'=0}^{k-1} \bar{h}_{k'} > 0$ co obxectivo de determinar o conxunto inicial de elementos finitos h_k.*

3. *Combínanse elementos finitos adxacentes se $\boldsymbol{u}(t)$ pode ser expresada coa mesma función de interpolación dentro dunha tolerancia pequena. Resólvese o NLP coas restricións* (4.7) *e, se son necesarias,* (4.8) *ou* (4.9)*. Compróbase a lonxitude dos elementos finitos e suprímense aqueles con $h_k \leq \delta_h(t_f - t_0)$, onde δ_h é unha tolerancia fixada. Resólvese de novo e aplícase a supresión de elementos finitos de lonxitude pequena até que non fique ningún.*

4. *Avalíase H^{kR+r} coa expresión* (4.11) *e estúdase se se cumpre en cada nodo:*

$$\left|H^{kR+r} - H^{kR+r-1}\right| \leq \text{máx}\left(\gamma_1 \left|H^{kR+r}\right|, \gamma_2\right),$$

onde γ_1 e γ_2 son constantes positivas pequenas. No caso de que se verifique o criterio, remátase o algoritmo. Noutro caso, se a condición é violada para o elemento I_k, divídese este en dous elementos do mesmo tamaño e tórnase ao paso 3.

Capítulo 5

Resolución con MATLAB

Chegados a este punto, procédese coa resolución a través de MATLAB ([6]) dos problemas de control óptimo enunciados, traballando co método directo da *collocation* con esquemas de tipo Runge-Kutta da subsección 3.3.1 e o Algoritmo 4.2 de refinamento da malla con elementos finitos fixos. A creación dun código que execute o Algoritmo 4.5 de elementos finitos movibles queda como traballo futuro para técnicas máis avanzadas de programación, mais en [3] obtivéronse con dita estratexia resultados similares aos dos Exemplos 5.2 e 5.4, dando validez á metodoloxía.

Así, deseñouse un código dende cero con axuda dalgunhas funcións de MATLAB para resolver o problema non linear que xorde en cada iteración e interpolar mediante *splines* os datos discretos. Deste xeito, a labor feita consistiu en construír un programa que transforme o problema funcional no discreto e, a partir da solución do NLP correspondente, elabore o refinamento da malla, repetindo o proceso até que o usuario o precise.

É un código que foi testado cos exemplos presentados no traballo e que está pensado para unha formulación do problema con función obxectivo de *tipo Mayer*. Cómpre salientar que a medida que medra o grao de refinamento, o programa demora a súa execución, razón pola que se adoita pedir a detención polo tamaño da malla antes que pola tolerancia do erro verificada.

En particular, para certos tamaños arbitrarios da malla, ilústranse a continuación algúns resultados obtidos. Nestes seguiuse o Algoritmo 4.2 escollendo entre as dúas opcións do paso 5, con constantes $M = 6$, $M_1 = 5$ e $\delta = 10^{-4}$, e substituíndo a función obxectivo do problema (4.6) pola suma dos erros estimados, pois en caso contrario non se conseguiu que o código engadise puntos para mallas elevadas. Ademais, tamén se alternou o paso 3 do algoritmo con empregar sempre a mesma orde de discretización.

Por último, na representación gráfica das funcións obtidas, inclúense os puntos da discretización, é dicir, a solución discreta obtida, e represéntanse sobre un *spline* de MATLAB que os une

para amosar unha función máis regular. Resulta interesante observar a colocación dos puntos da solución discreta, pois dá unha idea de onde foi necesario refinar para reducir o erro.

Para ver o código, pódese consultar o Anexo A. Ademais, tamén se inclúen os comandos para obter as representacións gráficas, se ben algunhas propiedades das imaxes foron modificadas na xanela gráfica de MATLAB para incluílas mellor no texto.

5.1. Resultados sobre os exemplos

Exemplo 5.1 (Freada dun bloque)**.** No Exemplo 1.6, a aceleración que o operario ten que transmitir vén representada na Figura 5.1, que provoca unha diminución da velocidade até 0.17157 no instante final. Esta solución obtívose con 5 elementos na malla tras resolver un único problema de programación non linear co método dos trapecios, resultando nun erro final estimado medio de 8.6383×10^{-15}. A Figura 5.1 tamén amosa a evolución da velocidade do bloque.

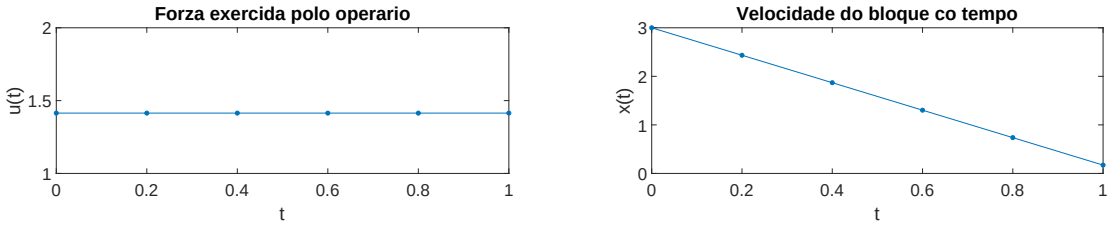

Figura 5.1: Forza exercida polo operario, $u(t)$, e velocidade, $x(t)$, do problema de freada dun bloque: obtención mediante discretización co método dos trapecios.

Exemplo 5.2 (Reactor descontinuo)**.** No Exemplo 1.7, mediante unha discretización de trapecios e algoritmo heurístico á hora de refinar a malla, obtívose o perfil de temperaturas da Figura 5.2 e a evolución do materiais da Figura 5.3. Deste xeito, coa temperatura descrita maximízase a cantidade do produto B até 0.57336 unidades e estímase un erro de 0.0013211 alén de resolver 5 problemas discretos, acadando os 26 nodos de mallado.

Doutra banda, incorporando o paso 3 do algoritmo, combinando discretización de trapecios e Hermite-Simpson, obtivéronse resultados similares: a cantidade final do produto B é 0.57037 e o erro estimado é 0.0012719. O número de puntos finais da malla é 26, resolvéronse 6 NLPs e refinouse en 4 ocasións. Os resultados gráficos son similares mais apréciase un comportamento distinto do algoritmo heurístico na colocación dos nodos. Véxanse as Figuras 5.4 e 5.5.

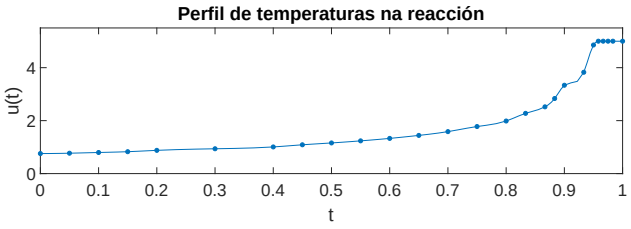

Figura 5.2: Perfil de temperaturas, $u(t)$, no problema do reactor descontinuo resolto mediante discretización de trapecios e algoritmo heurístico.

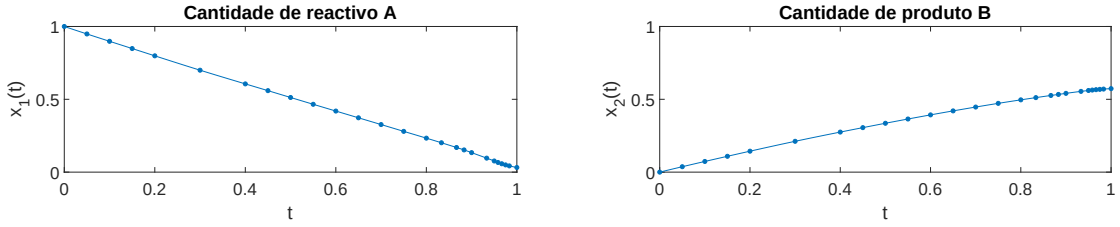

Figura 5.3: Cantidade do reactivo A, $x_1(t)$, e do produto B, $x_2(t)$, co paso do tempo no problema do reactor descontinuo resolto mediante discretización de trapecios e algoritmo heurístico.

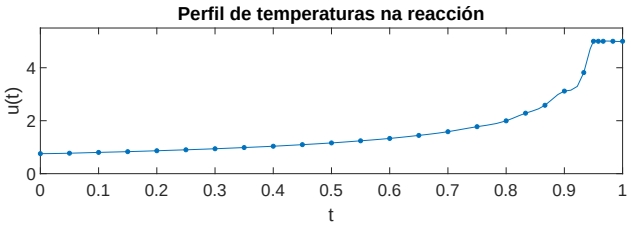

Figura 5.4: Perfil de temperaturas, $u(t)$, no problema do reactor descontinuo resolto co paso 3 do Algoritmo 4.2 e algoritmo heurístico.

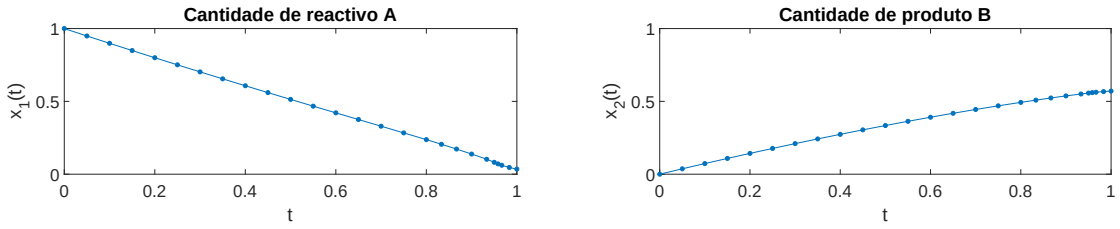

Figura 5.5: Cantidade do reactivo A, $x_1(t)$, e do produto B, $x_2(t)$, co paso do tempo no problema do reactor descontinuo resolto co paso 3 do Algoritmo 4.2 e algoritmo heurístico.

Por último, substituíndo no caso anterior o algoritmo heurístico polo problema de programación enteira (4.6), se se calcula até ter 26 nodos na discretización temporal, obtense unha cantidade final de produto B de 0.56971 e un erro estimado de 0.0031389. A representación gráfica amósase nas Figuras 5.6 e 5.7, nas que se nota que a distribución dos puntos da malla é distinta á do algoritmo heurístico, cos nodos acumulados á dereita.

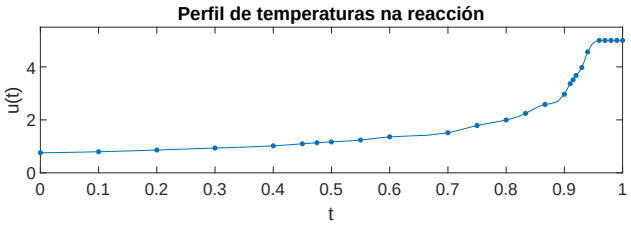

Figura 5.6: Perfil de temperaturas, $u(t)$, no problema do reactor descontinuo resolto co paso 3 do Algoritmo 4.2 e programación enteira.

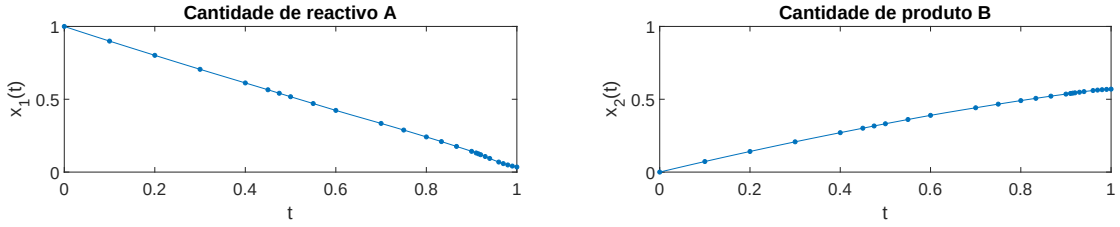

Figura 5.7: Cantidade do reactivo A, $x_1(t)$, e do produto B, $x_2(t)$, co paso do tempo no problema do reactor descontinuo resolto co paso 3 do Algoritmo 4.2 e programación enteira.

Así pois, obtéñense resultados similares nos tres casos, que se asemellan tamén ao valor da función obxectivo e representación gráfica da referencia [3], e que parecen aconsellar que o maior incremento da temperatura se dea arredor das 0.9 unidades temporais.

Exemplo 5.3 (Carro con péndulo)**.** No Exemplo 1.10, tomando $t_f = 2$, $m_1 = 1$, $m_2 = 0.3$, $d = 1$, $d_{max} = 2$, $u_{max} = 20$, $g = 9.81$ e $l = 0.5$ coas unidades correspondentes, unha iteración do proceso con 12 puntos da malla, é dicir, discretización (mediante trapecios) e resolución por programación cuadrática secuencial, devolve as traxectorias e forza das Figuras 5.8 e 5.9, respectivamente. Pódese comprobar que os resultados gráficos son similares aos de [7].

Obsérvase que o movemento obtido do carro non consiste en ir sempre cara adiante, senón que na metade do proceso a solución obtida indica que convén desprazar o carro uns instantes

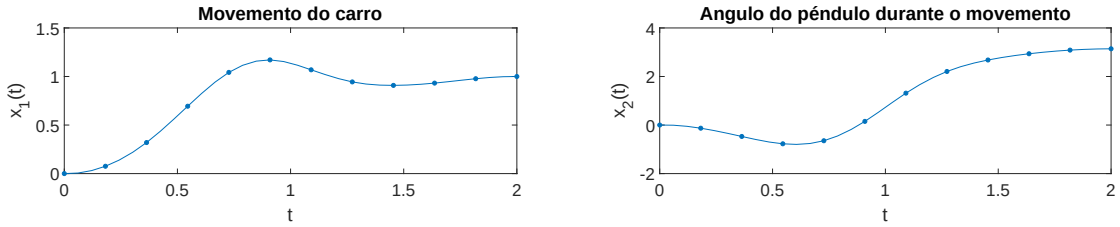

Figura 5.8: Posición do carro, $x_1(t)$, e ángulo que forma o péndulo coa vertical, $x_2(t)$, en cada instante no problema do carro con péndulo resolto mediante discretización de trapecios.

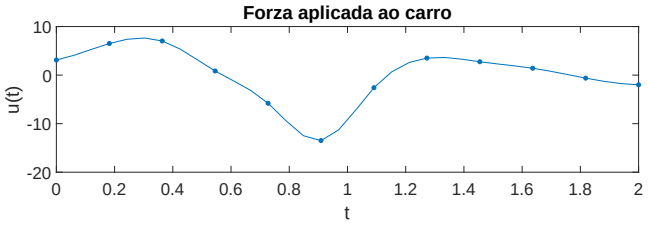

Figura 5.9: Forza, $u(t)$, que hai que aplicar ao carro no problema do carro con péndulo resolto mediante discretización de trapecios.

cara atrás, para finalmente rematar no punto final desexado.

Exemplo 5.4 (Problema de Rayleigh)**.** Finalmente, traballouse tamén con outro problema enunciado en [3], co obxectivo de ter unha referencia coa que comparar resultados. Así, o problema consiste nunha función obxectivo con *termo de Lagrange* suxeita a un sistema diferencial de primeira orde:

$$\begin{aligned}
\min_{\boldsymbol{x},u} \quad & \int_0^{2.5} (x_1(t))^2 + (u(t))^2 \mathrm{d}t \\
\text{s. a} \quad & \dot{x}_1(t) = x_2(t), \quad t \in [0, 2.5], \\
& \dot{x}_2(t) = -x_1(t) + \left(1.4 - 0.14(x_2(t))^2\right) x_2(t) + 4u(t), \quad t \in [0, 2.5], \\
& x_1(0) = -5, \quad x_2(0) = -5.
\end{aligned}$$

cuxa reformulación mediante *termo de Mayer* resulta en:

$$\begin{aligned}
\min_{\boldsymbol{x},u} \quad & x_3(t) \\
\text{s. a} \quad & \dot{x}_1(t) = x_2(t), \quad t \in [0, 2.5], \\
& \dot{x}_2(t) = -x_1(t) + \left(1.4 - 0.14(x_2(t))^2\right) x_2(t) + 4u(t), \quad t \in [0, 2.5], \\
& \dot{x}_3(t) = (x_1(t))^2 + (u(t))^2, \quad t \in [0, 2.5], \\
& x_1(0) = -5, \quad x_2(0) = -5, \quad x_3(0) = 0.
\end{aligned}$$

Á hora de mallar o intervalo temporal, incluíuse o instante temporal $t = 0.1$ como se fai en [3] para poder comparar resultados. Deste xeito, mediante *collocation* co método dos trapecios e algoritmo heurístico de refinamento, tras resolver 4 NLPs cunha malla final de 21 nodos obtívose un valor da función obxectivo de 29.4109 e un erro máximo estimado de 0.15154. O control resultante é o da Figura 5.10.

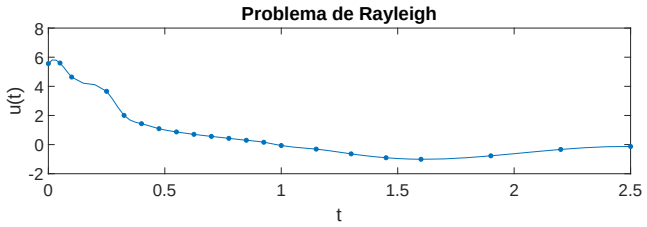

Figura 5.10: Control, $u(t)$, no problema de Rayleigh resolto mediante discretización de trapecios e algoritmo heurístico.

Se pola contra se decide incorporar o paso 3, resólvense 5 NLPs, cun obxectivo de 28.1442 e un erro estimado máximo de 0.139. Neste caso o control é o da Figura 5.11.

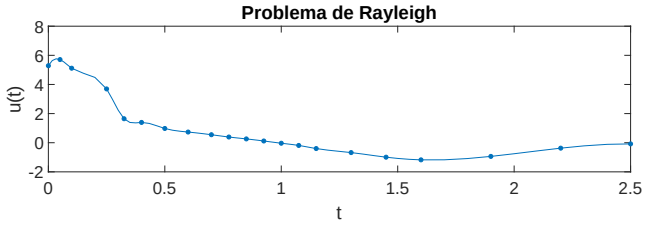

Figura 5.11: Control, $u(t)$, no problema de Rayleigh resolto co paso 3 do Algoritmo 4.2 e algoritmo heurístico.

Por último, se se decide empregar a estratexia de programación enteira, o valor acadado da función obxectivo é 28.062, obtido con 21 nodos da malla, 5 NLPs resoltos e un erro estimado máximo igual a 0.19007. Ilústrase a función $u(t)$ na Figura 5.12 e de novo pódese observar a distinta posición dalgúns puntos da malla con respecto ao algoritmo heurístico na Figura 5.11.

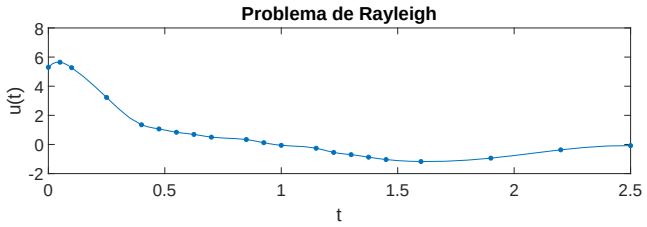

Figura 5.12: Control, $u(t)$, no problema de Rayleigh resolto co paso 3 do Algoritmo 4.2 e programación enteira.

5.2. Conclusións xerais sobre o código

En conclusión, cos exemplos obsérvase que as distintas variacións do Algoritmo 4.2 poden afectar lixeiramente ao resultado. Cómpre salientar que o erro obtido resolvendo con programación enteira é en xeral maior que aquel co algoritmo heurístico no paso 5 do algoritmo, malia que este último non ten garantías de atopar un óptimo. Esta casuística pódese deber a que ao final no problema de programación enteira estase a intentar minimizar unha estimación do erro, mais non se ten seguridade de que o erro real cumpra dito comportamento. Ademais, tamén hai que ter en conta que os cálculos da redución da orde e do erro estimado poden non ser boas aproximacións e que, por mor dos tempos de computación, estase a minimizar a suma dos erros estimados e non o seu máximo.

Adicionalmente, comprobouse que os tempos de execución medran considerablemente cando se emprega o paso 3 e aínda máis cando se substitúe o algoritmo heurístico polo problema de programación enteira. Por exemplo, no Exemplo 5.2, mediante unha discretización de trapecios e un algoritmo heurístico, o programa executouse en 38 segundos aproximadamente, mentres que co paso 3 do Algoritmo 4.2 en 96 segundos e, engadindo tamén o proceso de programación enteira, en 314 segundos.

Deste xeito, aínda que o esquema de Hermite-Simpson teña unha orde teórica maior que o dos trapecios e que resolver o problema de programación enteira (4.6) semelle unha opción máis fiable que o algoritmo heurístico, co código actual os resultados non o reflicten. En efecto, a discretización de Hermite-Simpson case non reduce o erro en comparación coa dos trapecios, e a programación enteira auméntao ao mesmo tempo que se incrementa o tempo de execución. Así pois, unha posible extensión futura desta análise sería a elaboración dun código máis sofisticado que conseguise reducir os tempos de traballo, para poder calcular mellores aproximacións e mellorar o comportamento do algoritmo cando se emprega o problema (4.6).

Anexo A

Código de MATLAB

Inclúese o anexo co código de MATLAB de elaboración propia empregado para obter os resultados do Capítulo 5. As funcións do programa principal amósanse en orde alfabética.

Ademais, co obxectivo de facilitar a súa execución ao lector, o código tamén se atopa no repositorio de GitHub de `https://github.com/IagoComesanha/Code_TFG_Iago_Comesanha_Solinho`.

A.1. Programa principal

Programa A.1: OCP_Refinamento.m

```
% Traballo fin de grao Iago Comesaña Soliño 2024.
%
% Programa para resolver o problema de control óptimo mediante a
% estratexia de collocation con esquemas Runge-Kutta e algoritmo de
% refinamento da malla.

tic
clc
clear
close all

% ELECCIÓN DO PROBLEMA: ficheiro de datos.
% data0 % Problema de freada dun bloque.
% data1 % Problema do reactor descontinuo.
data2 % Problema de Rayleigh.
% data3 % Problema do carro con péndulo.

% MALLA INICIAL.
% No caso do problema de Rayleigh (datos = 2), tómase un punto
% extra: 0.1, por recomendación da referencia da que se obtivo o
% problema.
if datos == 2 % Caso data2
```

```
23      th = [t0 linspace(0.1, tf, nv-1)];
24  else
25      th = linspace(t0, tf, nv);
26  end
27
28  %VALORES INICIAIS.
29  [xh, zh, uh, ph] = inic(variab, conx, conf, nv);
30
31  %Datos iniciais para os estados no exemplo do carro con péndulo,
32  %escollidos para obter resultados semellantes ao do artigo do
33  %problema.
34  if datos == 3 %Caso data3
35      xh(1,1:end) = 0.5*th;
36      xh(2,1:end) = 1.5*th;
37  end
38
39  %Inicialización do contador de refinamentos.
40  ref=0;
41
42  %Bucle de resolución.
43  for i=1:maxit
44
45      %TRANSFORMACIÓN EN NLP
46      %Variables discretas -> vector de variables \xi.
47      xih = discr2xi(variab, nv, xh, zh, uh, ph);
48      %Funcións OCP -> Restricións NLP.
49      [Fi, Gi, Hi, lbounds, ubounds] = discrfun2xifun(variab, nv, ...
50          th, Phi, f, g, c, conx, conf, discret, lboundsp, ...
51          lboundsu, uboundsp, uboundsu, lboundsx, uboundsx, ...
52          lboundsz, uboundsz);
53
54      %SOLUCIÓN DO NLP: SEQUENTIAL QUADRATIC PROGRAMMING.
55      xii = SQP(xih, Fi, Gi, Hi, lbounds, ubounds);
56
57      %INTERPOLACIÓN DE SOLUCIÓNS
58      %\xi -> var.discretas
59      [xh, zh, uh, ph] = xi2discr(variab, nv, xii);
60      %SPLINES.
61      %Mediante a función pp = spline(t,a) e ppval(pp, puntos).
62      [ppxh, ppzh, ppuh, ppxhder] = aproxsol(th, xh, zh, uh);
63
64      %ESTIMACIÓN DO ERRO
65      [errointegraNdo, etaerro, pesosw, erroestimado] = estimacionerro( ...
66          variab, th, ppxh, ppzh, ppuh, ph, f, ppxhder);
67
68      %TEST DE CONVERXENCIA
69      %Test de tolerancia.
70      if max(erroestimado) < tol
71          disp(['Iteración: ' num2str(i)])
72          disp(['Tolerancia superada: ' num2str(tol)])
73          disp(['Erro estimado máximo: ' num2str(max(erroestimado))])
74          disp(['Valor da función obxectivo: ' num2str(Phi(xh, ...
75                                                  zh,uh,ph))])
76          break
```

```
77      end
78      % Test de tamaño da malla.
79      if nv > maxnv
80          disp(['Iteración: ' num2str(i)])
81          disp(['Malla elevada, lonxitude: ' num2str(length(th))])
82          disp(['Erro estimado máximo: ' num2str(max(erroestimado))])
83          disp(['Valor da función obxectivo: ' num2str(Phi(xh, ...
84                                                          zh,uh,ph))])
85          break
86      end
87
88      %SELECCIÓN CAMBIO DE MÉTODO DE ORDE q OU NON
89      %E PROCESO DE REFINAMENTO DA MALLA.
90      if paso3 == 0
91          if ref == 0
92              % Argumentos descoñecidos no primeiro proceso de
93              % refinamento.
94              thetaerro = []; I=[]; thprev=[];
95          end
96          % Proceso de refinamento e actualización de variables.
97          [th, nv, I, xh, zh, uh, ref, thetaerro, thprev] = refinamento0( ...
98              ref, etaerro, thetaerro, I, errointegraNdo, th, nv, ...
99              erroestimado, q, estratexia, variab, ppxh, ppzh, ...
100             ppuh, thprev, tol);
101
102     elseif paso3 == 1
103         if ref == 0
104             % Argumentos descoñecidos no primeiro proceso de
105             % refinamento.
106             thetaerro = []; I=[]; thprev=[];
107         end
108         % Proceso de refinamento e actualización de variables.
109         [th, nv, I, xh, zh, uh, discret, ref, thetaerro, thprev] = refinamento1( ...
110             i, ref, discret, etaerro, thetaerro, I, ...
111             errointegraNdo, th, nv, erroestimado, q, ...
112             estratexia, variab, ppxh, ppzh, ppuh, thprev, tol);
113     end
114 end
115 toc
```

A.2. Datos para iniciar o programa

Programa A.2: data0.m

```
1 % Datos programa OCP_Refinamento.
2 % Problema de freada dun bloque.
3
4 % Intervalo temporal [t_0, t_f].
5 t0 = 0; tf=1;
6
7 % Número de variables do sistema.
```

```
8  variab = struct('ndif', 1, 'nalx', 1, 'ncon', 1, 'npar', 0);
9
10 %Función obectivo, funcional a minimizar. Definida segundo a
11 %discretización.
12 Phi =  @(xh,zh,uh,ph) xh(variab.ndif,end);
13
14 %Funcións do sistema de DAEs.
15 %dx = f(x,z,u,p); x(t0)=x_0;
16 %0 = g(x,z,u,p);
17
18 %Función f, o último elemento é o tamaño.
19 f = {@(x,z,u,p) -u(1)-z(1); 1};
20
21 %Función g, o último elemento é o tamaño.
22 g = {@(x,z,u,p) u(1).^2+z(1).^2-4; 1};
23
24 %Condicións inicias, x(t0)=x_0, o último elemento é o tamaño.
25 conx = {3; 1};
26
27 %Condicións finais, x(tf)=x_f, o último elemento é o tamaño.
28 conf = {0};
29
30 %Funcións das restricións, o último elemento é o tamaño.
31 c = {0};
32
33 %Límites das variables, o último elemento é o tamaño.
34 uboundsx = {3; 1}; uboundsz= {2; 1}; uboundsu = {2; 1};
35 uboundsp = {0};
36 lboundsx = {0; 1}; lboundsz = {0; 1}; lboundsu = {0; 1};
37 lboundsp = {0};
38
39 %Número de puntos iniciais da malla n_col + 1 = nv (tense en
40 %conta que Matlab comeza a contar en 1 e non en 0 como na
41 %formulación teórica). Número de elementos da malla:
42 %n_col = ncol.
43 ncol = 5; nv = 6;
44
45 %Tipo de discretización: trapecios [1], Hermite [2].
46 discret = 1;
47
48 %Orde teórica da discretización: trapecios [2], Hermite [4].
49 q = 2;
50
51 %Paso 3 do algoritmo: non [0], si [1].
52 paso3 = 0;
53
54 %Estratexia refinamento da malla: heurística [0], programación
55 %enteira [1].
56 estratexia = 0;
57
58 %Tolerancia do erro.
59 tol = 1.e-4;
60
61 %Número máximo de iteracións.
```

```
maxit = 10;

% Número máximo de nodos.
maxnv = 25;

% Datos que se toman, data(datos)
datos = 0;
```

Programa A.3: data1.m

```
% Datos programa OCP_Refinamento.
% Problema do reactor descontinuo.

% Intervalo temporal [t_0, t_f].
t0 = 0; tf=1;

% Número de variables do sistema.
variab = struct('ndif', 2, 'nalx', 0, 'ncon', 1, 'npar', 0);

% Función obectivo, funcional a minimizar. Definida segundo a
% discretización.
Phi =  @(xh,zh,uh,ph) -xh(variab.ndif,end);

% Funcións do sistema de DAEs.
% dx = f(x,z,u,p); x(t0)=x_0;
% 0 = g(x,z,u,p);

% Función f, o último elemento é o tamaño.
f = { @(x,z,u,p) -x(1).*(u(1) + (u(1).^2)./2); ...
      @(x,z,u,p) x(1).*u(1); ...
      2};

% Función g, o último elemento é o tamaño.
g = {0};

% Condicións inicias, x(t0)=x_0, o último elemento é o tamaño.
conx = { 1; 0; 2};

% Condicións finais, x(tf)=x_f, o último elemento é o tamaño.
conf={0};

% Funcións das restricións, o último elemento é o tamaño.
c = {0};

% Límites das variables, o último elemento é o tamaño.
uboundsx = {Inf; Inf; 2}; uboundsz={0}; uboundsu = {5; 1};
uboundsp = {0};
lboundsx = {-Inf; -Inf; 2}; lboundsz = {0}; lboundsu = {0; 1};
lboundsp = {0};

% Número de puntos iniciais da malla n_col + 1 = nv (tense en
% conta que Matlab comeza a contar en 1 e non en 0 como na
% formulación teórica). Número de elementos da malla:
% n_col = ncol.
```

```
ncol = 5; nv = 6;

%Tipo de discretización: trapecios [1], Hermite [2].
discret = 1;

%Orde teórica da discretización: trapecios [2], Hermite [4].
q = 2;

%Paso 3 do algoritmo: non [0], si [1].
paso3 = 1;

%Estratexia refinamento da malla: heurística [0], programación
%enteira [1].
estratexia = 1;

%Tolerancia do erro.
tol = 1.e-4;

%Número máximo de iteracións.
maxit = 10;

%Número máximo de nodos
maxnv = 25;

%Datos que se toman, data(datos)
datos = 1;
```

Programa A.4: data2.m

```
%Datos programa OCP_Refinamento.
%Problema de Rayleigh.

%Intervalo temporal [t_0, t_f].
t0 = 0; tf=2.5;

%Número de variables do sistema.
variab = struct('ndif', 3, 'nalx', 0, 'ncon', 1, 'npar', 0);

%Función obectivo, funcional a minimizar. Definida segundo a
%discretización.
Phi =  @(xh,zh,uh,ph) xh(3,end);

%Funcións do sistema de DAEs
%dx = f(x,z,u,p); x(t0)=x_0;
%0 = g(x,z,u,p);

%Función f, o último elemento é o tamaño.
f = {@(x,z,u,p) x(2);...
     @(x,z,u,p) -x(1) + (1.4 - 0.14.*(x(2).^2)).*x(2) + 4.*u(1);...
     @(x,z,u,p) x(1).^2 + u(1).^2;...
     3};

%Función g, o último elemento é o tamaño.
g = {0};
```

```
26
27 %Condicións inicias, x(t0)=x_0, o último elemento é o tamaño.
28 conx = { -5; -5; 0; 3};
29
30 %Condicións finais, x(tf)=x_f, o último elemento é o tamaño.
31 conf={0};
32
33 %Funcións das restricións, o último elemento é o tamaño.
34 c = {0};
35
36 %Límites das variables, o último elemento é o tamaño.
37 uboundsx = {Inf; Inf; Inf; 3}; uboundsz={0}; uboundsu = {Inf; 1};
38 uboundsp = {0};
39 lboundsx = {-Inf; -Inf; -Inf; 3}; lboundsz = {0};
40 lboundsu = {-Inf; 1}; lboundsp = {0};
41
42 %Número de puntos iniciais da malla n_col + 1 = nv (tense en
43 %conta que Matlab comeza a contar en 1 e non en 0 como na
44 %formulación teórica). Número de elementos da malla:
45 %n_col = ncol.
46 ncol = 5; nv = 6;
47
48 %Tipo de discretización: trapecios [1], Hermite [2].
49 discret = 1;
50
51 %Orde teórica da discretización: trapecios [2], Hermite [4].
52 q = 2;
53
54 %Paso 3 do algoritmo: non [0], si [1].
55 paso3 = 1;
56
57 %Estratexia refinamento da malla: heurística [0], programación
58 %enteira [1].
59 estratexia = 1;
60
61 %Tolerancia do erro.
62 tol = 1.e-4;
63
64 %Número máximo de iteracións.
65 maxit = 10;
66
67 %Número máximo de nodos
68 maxnv = 19;
69
70 %Datos que se toman, data(datos).
71 datos = 2;
```

Programa A.5: data3.m

```
1 %Datos programa OCP_Refinamento.
2 %Problema do carro con péndulo.
3
4 %Intervalo temporal [t_0, t_f].
5 t0 = 0; tf = 2;
```

```
6
7  % Número de variables do sistema.
8  variab = struct('ndif', 5, 'nalx', 0, 'ncon', 1, 'npar', 0);
9
10 % Función obectivo, funcional a minimizar. Definida segundo a
11 % discretización (desvantaxe do código).
12  Phi =  @(xh,zh,uh,ph) xh(5,end);
13
14 % Funcións do sistema de DAEs
15 % dx = f(x,z,u,p); x(t0)=x_0;
16 % 0 = g(x,z,u,p);
17
18 % Función f, o último elemento é o tamaño.
19 f = {@(x,z,u,p) x(3); ...
20      @(x,z,u,p) x(4); ...
21      @(x,z,u,p) ( 0.15*sin(x(2)).*(x(4).^2) + ...
22                u(1) + 0.3*9.81*cos(x(2)).*sin(x(2)) )./( 1 + ...
23                0.3*(1-(cos(x(2)).^2)) ) ;
24      @(x,z,u,p) -( 0.15*cos(x(2)).*sin(x(2)).*(x(4).^2) + ...
25                u(1).*cos(x(2)) + 1.3*9.81*sin(x(2)) )./( 0.5 + ...
26                0.15*(1-(cos(x(2)).^2)) );
27      @(x,z,u,p) u(1).^2;
28      5};
29
30 % Función g, o último elemento é o tamaño.
31 g = {0};
32
33 % Condicións inicias, x(t0)=x_0, o último elemento é o tamaño.
34 conx = { 0; 0; 0; 0; 0; 5};
35
36 % Condicións finais, x(tf)=x_f, o último elemento é o tamaño.
37 conf = {1; pi; 0; 0; 4};
38
39 % Funcións das restricións, o último elemento é o tamaño.
40 c = {0};
41
42 % Límites das variables, o último elemento é o tamaño.
43 uboundsx = {2; Inf; Inf; Inf; 800; 5}; uboundsz={0};
44 uboundsu = {20; 1}; uboundsp = {0};
45
46 lboundsx = {-2; -Inf; -Inf; -Inf; 0; 5}; lboundsz = {0};
47 lboundsu = {-20; 1}; lboundsp = {0};
48
49 % Número de puntos iniciais da malla n_col + 1 = nv (tense en
50 % conta que Matlab comeza a contar en 1 e non en 0 como na
51 % formulación teórica). Número de elementos da malla:
52 % n_col = ncol.
53 ncol = 11; nv = 12;
54
55 % Tipo de discretización: trapecios [1], Hermite [2].
56 discret = 1;
57
58 % Orde teórica da discretización: trapecios [2], Hermite [4].
59 q = 2;
```

```
60
61 % Paso 3 do algoritmo: non [0], si [1].
62 paso3 = 0;
63
64 % Estratexia refinamento da malla: heurística [0], programación
65 % enteira [1].
66 estratexia = 0;
67
68 % Tolerancia do erro.
69 tol = 1.e-4;
70
71 % Número máximo de iteracións.
72 maxit = 10;
73
74 % Número máximo de nodos.
75 maxnv = 8;
76
77 % datos que se toman, data(datos).
78 datos = 3;
```

A.3. Programas secundarios

Programa A.6: actualizacion.m

```
1  function [xh, zh, uh] = actualizacion(th, ppxh, ppzh, ppuh, variab)
2  % Actualización das variables nos nodos da malla mediante a
3  % avaliación dos splines nos novos puntos.
4  %
5  % DATOS DE ENTRADA:
6  % th: vector cos puntos da malla actual.
7  % ppxh: celda de variab.ndif filas e 1 columna que na fila
8  % i-ésima contén o spline que interpola os datos da variable
9  % diferencial i-ésima.
10 % ppzh: celda de variab.nalx filas e 1 columna que na fila
11 % i-ésima contén o spline que interpola os datos da variable
12 % alxébrica i-ésima.
13 % ppuh: celda de variab.ncon filas e 1 columna que na fila
14 % i-ésima contén o spline que interpola os datos da variable de
15 % control i-ésima.
16 % variab: estrutura co nº de variables do problema orixinal.
17 %
18 % DATOS DE SAÍDA:
19 % xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
20 % a variable diferencial i-ésima no nodo j-ésimo.
21 % zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
22 % a variable alxébrica i-ésima no nodo j-ésimo.
23 % uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
24 % a variable de control i-ésima no nodo j-ésimo.
25
26 % Número de nodos da malla.
27 nv = length(th);
```

```
28
29 %Variables de estado diferenciais.
30 xh = zeros(variab.ndif, nv);
31 for i = 1:variab.ndif
32     xh(i,:) = ppval(ppxh{i,1},th);
33 end
34
35 %Variables de estado alxébricas.
36 zh = zeros(variab.nalx, nv);
37 for i = 1:variab.nalx
38     zh(i,:) = ppval(ppzh{i,1},th);
39 end
40
41 %Variables de control.
42 uh = zeros(variab.ncon, nv);
43 for i = 1:variab.ncon
44     uh(i,:) = ppval(ppuh{i,1},th);
45 end
46 end
```

Programa A.7: aproxsol.m

```
1  function [ppxh, ppzh, ppuh, ppxhder] = aproxsol(th, xh, zh, uh)
2  %Función que constrúe celdas dunha columna para almacenar as
3  %aproximacións por splines correspondentes a cada variable do
4  %sistema. A interpolación realízase a partir dos datos nos puntos
5  %da malla mediante a función spline, que consegue unha estrutura
6  %capaz de ser avaliada.
7  %Opción de mellora para traballo futuro: incorporar a información
8  %das derivadas de x no seu spline como se indica no traballo.
9  %
10 %DATOS DE ENTRADA:
11 %th: puntos da malla.
12 %xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
13 %a variable diferencial i-ésima no nodo j-ésimo.
14 %zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
15 %a variable alxébrica i-ésima no nodo j-ésimo.
16 %uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
17 %a variable de control i-ésima no nodo j-ésimo.
18 %
19 %DATOS DE SAÍDA:
20 %ppxh: celda de variab.ndif filas e 1 columna que na fila i-ésima
21 %contén o spline que interpola os datos da variable diferencial
22 %i-ésima.
23 %ppzh: celda de variab.nalx filas e 1 columna que na fila i-ésima
24 %contén o spline que interpola os datos da variable alxébrica
25 %i-ésima.
26 %ppuh: celda de variab.ncon filas e 1 columna que na fila i-ésima
27 %contén o spline que interpola os datos da variable de control
28 %i-ésima.
29 %ppxhder : celda de variab.ndif filas e 1 columna que na fila
30 %i-ésima contén o spline derivada de ppxh{i,1}.
31
32 %Variables de estado diferenciais.
```

```
n = size(xh,1);
ppxh = cell(n,1);
for i = 1:n
    ppxh{i,1} = spline(th, xh(i,:));
end
%Cálculo da derivada de ppxh
ppxhder = cell(n,1);
for i = 1:n
    ppxhder{i,1} = fnder(ppxh{i,1},1);
end

%Variables de estado alxébricas.
n = size(zh,1);
ppzh = cell(n,1);
for i = 1:n
    ppzh{i,1} = spline(th, zh(i,:));
end

%Variables de control.
n = size(uh,1);
ppuh = cell(n,1);
for i = 1:n
    ppuh{i,1} = spline(th, uh(i,:));
end
end
```

Programa A.8: barmat.m

```
function v = barmat(funcion, x, nodo)
%Función para construir o argumento da función \bar{f}^{k+1}:
%devolve un vector do tipo de variable avaliado no nodo.
%
%DATOS DE ENTRADA:
%funcion: celda con cada función a avaliar, por ex barx.
%x: argumento de entrada de funcion.
%nodo: nodo da malla no que se pretende avaliar.
%
%DATOS DE SAÍDA:
%v: vector coa avaliación correspondente.

%Número de variables que se pasan como argumento.
n = size(funcion,1);
%Asignación.
v = zeros(n,1);
for i = 1:n
   v(i) = funcion{i,nodo}(x);
end




end
```

Programa A.9: cambiodeorde.m

```
function discretnew = cambiodeorde(i, erroestimado, discret)
% Función cambio de orde, decide se incrementar a orde do método de
% discretización.
%
%DATOS DE ENTRADA:
% i: iteración do método.
% erroestimado: vector coas estimacións do erro por intervalos.
% discret: tipo de discretización, [1] trapecios, [2] Hermite.
%
%DATOS DE SAÍDA:
% discretnew: novo tipo de discretización.

% \bar{\eps}: media do erro estimado
barerro = mean(erroestimado);
if ( all(erroestimado <= 2*barerro))
    % Auméntase a orde.
    discretnew = 2;
    return
elseif ( i > 2)
    % Auméntase orde.
    discretnew = 2;
    return
else
discretnew = discret;
end
end
```

Programa A.10: discr2xi.m

```
function xi = discr2xi(variab, nv, xh, zh, uh, ph)
% Función para converter o conxunto de variables orixinais
% avaliadas nos nodos temporais nun vector de variables \xi.
%
%DATOS DE ENTRADA:
% variab: estrutura co número de variables do problema.
% nv: número de nodos da malla considerada.
% xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
% a variable diferencial i-ésima no nodo j-ésimo.
% zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
% a variable alxébrica i-ésima no nodo j-ésimo.
% uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
% a variable de control i-ésima no nodo j-ésimo.
% ph: vector de parámetros (variab.npar x 1).
%
%DATOS DE SAÍDA:
% xi: vector de variables do problema discretizado, ordenadas por
% nodos e por tipo de variable, cos parámetros ao comezo. É dicir:
% xi = (p x^0 z^0 u^0 x^1 z^1 u^1 ...).

%Número de variables continuas do problema.
nvar = variab.ndif + variab.nalx + variab.ncon;

%Creación da matriz intermedia conxunta.
```

```
prexi = [xh; zh; uh];

%Creación do vector para almacenar todas as variables.
lonxi = nv*nvar + variab.npar;
xi = zeros(lonxi, 1);

%Asignación dos parámetros.
i0 = variab.npar;
xi(1:i0) = ph;

for i = 1:nv
    %Índices inicial e final.
    in = i0 + (i-1)*nvar + 1;
    fin = i0 + i*nvar;
    %Asignación columna da matriz de variables continuas.
    xi(in:fin) = prexi(:,i);
end
end
```

Programa A.11: discrfun2xifun.m

```
function [Fi, Gi, Hi, lbounds, ubounds] = discrfun2xifun(variab,...
    nv, th, Phi, f, g, c, conx, conf, discret, lboundsp, ...
    lboundsu, uboundsp, uboundsu, lboundsx, uboundsx, ...
    lboundsz, uboundsz)
%Función que constrúe a función obxectivo e as restricións do
%problema de programación non linear que hai que resolver en cada
%iteración do proceso.
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do problema.
%nv: nº de puntos da malla.
%th: puntos da malla.
%Phi: función obxectivo do problema orixinal.
%f: celda coas funcións das restricións de tipo x' = f.
%g: celda coas funcións das restricións alxébricas 0 = g.
%c: celda coas restricións de inecuación do problema orixinal.
%conx: celda coas condicións iniciais do sistema x' = f.
%conf: celda coas condicións finais das variables de estado.
%discret: tipo de método de discretización (trapecios [1],
%Hermite [2]).
%lbounds, ubounds (celdas coas limitacións das distintas
%variables)
%
%DATOS DE SAÍDA: (Funcións Vectoriais)
%Fi: función obxectivo do NLP.
%Gi: restricións de igualdade do NLP, Gi(xi) = 0.
%Hi: restricións do NLP do tipo Hi(xi) <= 0.
%lbounds: límites inferiores de xi.
%ubounds: límites superiores de xi.

if discret == 1
    [Fi, Gi, Hi, lbounds, ubounds] = TrapColl(variab, nv, th, ...
        Phi, f, g, c, conx, conf, lboundsp, lboundsu, uboundsp, ...
```

```
        uboundsu, lboundsx, uboundsx, lboundsz, uboundsz);
elseif discret == 2
    [Fi, Gi, Hi, lbounds, ubounds] = HermiteColl(variab, nv, th,...
        Phi, f, g, c, conx, conf, lboundsp, lboundsu, uboundsp, ...
        uboundsu, lboundsx, uboundsx, lboundsz, uboundsz);
end
end
```

Programa A.12: engadir.m

```
function thi = engadir(I, th)
%Función que constrúe a nova malla a partir da anterior e do nº
%de puntos que hai que engadir por intervalo. Engádense os
%puntos de xeito uniforme.
%
%DATOS DE ENTRADA:
%I: puntos engadidos en cada intervalo.
%th: malla actual.
%
%DATOS DE SAÍDA:
%thi: nova malla.

%Número de elementos da malla previa.
n = length(I);

thi = [];
for i=1:n
    if I(i)==0
        %Escríbese só o punto t_i.
        thi = [thi th(i)];
    end
    if I(i)>=1
        %Divídese equitativamente.
        z = linspace(th(i), th(i+1), I(i)+2);
        %Non se engade o punto t_i+1.
        thi = [thi z(1:end-1)];
    end
end
%Engádese t_f.
thi = [thi th(end)];
end
```

Programa A.13: epsk.m

```
function erroestimado = epsk(etaerro, pesosw)
%Función que calcula definitivamente o erro estimado para
%comprobar a validez da solución obtida.
%
%DATOS DE ENTRADA:
%etaerro: matriz con nv-1 filas e variab.ndif columnas. En {k,i}
%contén o erro integrado do intervalo k-ésimo para a compoñente
%i-esima.
%pesosw: vector de pesos cuxa coordenada i-ésima é o máximo do
%valorabsoluto de x_i nos nodos da malla e da súa derivada
```

```
%
%DATOS DE SAÍDA:
%erroestimado: vector que en cada compoñente contén o erro
%estimado no intervalo k-ésimo, que é o máximo do erro relativo
%por compoñentes no intervalo k-ésimo.

%Número de elementos da malla.
M = size(etaerro, 1);
%Número de variables diferenciais.
n = size(etaerro, 2);
%Erro estimado útil.
erroestimado = zeros(M,1);
for k = 1:M
    cociente = zeros(n,1);
    for i = 1:n
        cociente(i) = etaerro(k,i)/(1 + pesosw(i));
    end
    erroestimado(k) = max(cociente);
end
end
```

Programa A.14: erroabs.m

```
function errointegrando = erroabs(variab, ppxh, ppzh, ppuh, ph, ...
                                                          f, ppxhder)
%Cálculo da función de erro absoluto eps dentro da integral.
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do problema.
%ppxh: celda de variab.ndif filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable diferencial
%i-ésima.
%ppzh: celda de variab.nalx filas e 1 columna que na fila i-ésima
%contén ospline que interpola os datos da variable alxébrica
%i-ésima.
%ppuh: celda de variab.ncon filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable de control
%i-ésima.
%ph: vector de parámetros
%f: celda coas funcións das restricións x' = f.
%ppxhder: celda de variab.ndif filas e 1 columna que na fila
%i-ésima contén o spline derivada de ppxh{i,1}.
%
%DATOS DE SAÍDA:
%errointegrando: celda con variab.ndif filas e 1 columna que
%contén a función de erro por compoñentes.

%Cálculo argumentos de fchapeu.
xchapeu = @(t) [];
for j = 1:variab.ndif
    xchapeu = @(t) [xchapeu(t) ppval(ppxh{j,1},t)];
end
zchapeu = @(t) [];
for j = 1:variab.nalx
```

```
    zchapeu = @(t) [zchapeu(t) ppval(ppzh{j,1},t)];
end
uchapeu = @(t) [];
for j = 1:variab.ncon
    uchapeu = @(t) [uchapeu(t) ppval(ppuh{j,1},t)];
end

%Definición da función de erro no integrando: |dx - f|.
errointegrando = cell(f{end,1},1);
for i =1:f{end,1}
    errointegrando{i,1} = @(t) abs( ppval(ppxhder{i,1},t) - ...
                    f{i,1}(xchapeu(t),zchapeu(t),uchapeu(t),ph) );
end
end
```

Programa A.15: errointegrado.m

```
function etaerro = errointegrado(th, errointegraNdo)
%Función que obtén o erro absoluto integrado por intervalos.
%
%DATOS DE ENTRADA:
%th: nodos da malla.
%errointegraNdo: celda con variab.ndif filas e 1 columna que
%contén a función de erro por compoñentes.
%
%DATOS DE SAÍDA:
%etaerro: matriz con length(th)-1 filas e size(errointegraNdo,1)
%columnas. En (k,i) contén o erro integrado do intervalo k-ésimo
%para a compoñente i-esima.

%Número de nodos da malla.
nv = length(th);
%Número de variables de estado diferenciais.
ndif = size(errointegraNdo, 1);
%Creación da matriz de erro integrado
etaerro = zeros(nv-1, ndif);
for k = 1:(nv-1)
    for i = 1:ndif
        etaerro(k,i) = integral(errointegraNdo{i,1}, th(k), ...
                        th(k+1), 'RelTol', 1e-10, 'AbsTol', 1e-15);
    end
end
end
```

Programa A.16: estimacionerro.m

```
function [errointegraNdo, etaerro, pesosw, erroestimado] = estimacionerro( ...
    variab, th, ppxh, ppzh, ppuh, ph, f, ppxhder)
%Función para calcular a estimación do erro do algoritmo de
%refinamento da malla.
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do problema.
%th: vector cos nodos da malla temporal.
```

```
%ppxh: celda de variab.ndif filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable diferencial
%i-ésima.
%ppzh: celda de variab.nalx filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable alxébrica
%i-ésima.
%ppuh: celda de variab.ncon filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable de control
%i-ésima.
%ph: vector de parámetros
%f: celda coas funcións das restricións x' = f.
%ppxhder: celda de variab.ndif filas e 1 columna que na fila
%i-ésima contén o spline derivada de ppxh{i,1}.
%
%DATOS DE SAÍDA:
%errointegrando: celda con variab.ndif filas e 1 columna que
%contén a función de erro por compoñentes.

%Cálculo do integrando como función.
errointegraNdo = erroabs(variab, ppxh, ppzh, ppuh, ph, f, ppxhder);

%Cálculo da integral de dentro.
etaerro = errointegrado(th, errointegraNdo);

%Cálculo dos pesos w.
pesosw = peso(th, ppxh, ppxhder);

%Cálculo erro aproximado epsilon.
erroestimado = epsk(etaerro, pesosw);
end
```

Programa A.17: HermiteColl.m

```
function [Fi, Gi, Hi, lbounds, ubounds] = HermiteColl(variab, ...
    nv, th, Phi, f, g, c, conx, conf, lboundsp, lboundsu, ...
    uboundsp, uboundsu, lboundsx, uboundsx, lboundsz, uboundsz)
%Función para obter a función obxectivo e restricións do problema
%NLP mediante un esquema de discretización de Hermite-Simpson.
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do problema.
%nv: nº de nodos da malla.
%th: puntos da malla.
%Phi: función obxectivo do problema orixinal.
%f: celda coas funcións das restricións de tipo x' = f.
%g: celda coas funcións das restricións alxébricas 0 = g.
%c: celda coas restricións de inecuación do problema orixinal.
%conx: celda coas condicións iniciais do sistema x' = f.
%conf: celda coas condicións finais das variables de estado.
%lbounds, ubounds (celdas cos límites das distintas variables)
%
%DATOS DE SAÍDA: (Funcións Vectoriais)
%Fi: función obxectivo do NLP.
%Gi: restricións de igualdade do NLP, Gi(xi) = 0.
```

```
22 %Hi: restricións do NLP do tipo Hi(xi) <= 0.
23 %lbounds: límites inferiores de xi.
24 %ubounds: límites superiores de xi.
25
26 %Número de variables continuas do sistema.
27 nvar = variab.ndif + variab.nalx + variab.ncon;
28 %Parámetros
29 i0 = variab.npar;
30 indp = 1:i0;
31
32 %f(x(t_i-1)), g(x(t_i-1)), c(x(t_i-1))
33 fi = cell(f{end,1}, nv); gi = cell(g{end,1}, nv);
34 ci = cell(c{end,1}, nv);
35 for i = 1:nv
36     %Índices respectivos no vector xi para conseguir as variables
37     %avaliadas en t_i-1.
38     indx = (i0 + (i-1)*nvar + 1):(i0 + (i-1)*nvar + variab.ndif);
39     indz = (i0+ (i-1)*nvar + variab.ndif + 1):(i0 + ...
40                         (i-1)*nvar + variab.ndif + variab.nalx);
41     indu = (i0 + (i-1)*nvar + variab.ndif + ...
42                                     variab.nalx + 1):(i0 + i*nvar);
43     %Funcións f nos nodos da malla.
44     for j = 1:f{end,1}
45         fi{j,i} = @(xi) f{j,1}(xi(indx), xi(indz), xi(indu), ...
46                                                         xi(indp));
47     end
48     %Funcións g nos nodos da malla.
49     for j = 1:g{end,1}
50         gi{j,i} = @(xi) g{j,1}(xi(indx), xi(indz), xi(indu), ...
51                                                         xi(indp));
52     end
53     %Funcións c nos nodos da malla.
54     for j = 1:c{end,1}
55         ci{j,i} = @(xi) c{j,1}(xi(indx), xi(indz), xi(indu), ...
56                                                         xi(indp));
57     end
58 end
59
60 %Lonxitude dos elementos da malla.
61 h = th(2:end) - th(1:end-1);
62
63 %\bar{x}^{k+1}
64 barx = cell(variab.ndif, nv);
65 for j = 1:variab.ndif
66     barx{j,1} = @(xi) xi(i0 + j);
67     for i = 2:nv
68         indk = i0 + (i-2)*nvar + j;
69         indk1 = i0 + (i-1)*nvar + j;
70         barx{j,i} = @(xi) (1/2)*(xi(indk) + xi(indk1)) + ...
71                         (h(i-1)/8)* (fi{j,i-1}(xi) + fi{j,i}(xi));
72     end
73 end
74
75 %\bar{z}^{k+1/2};
```

```
76  zmedio = cell(variab.nalx, nv);
77  for j = 1:variab.ncon
78      zmedio{j,1} = @(xi) xi(i0 + j);
79      for i = 2:nv
80          indk = i0 + (i-2)*nvar + variab.ndif + j;
81          indk1 = i0 + (i-1)*nvar + variab.ndif + j;
82          zmedio{j,i} = @(xi) (1/2)*(xi(indk) + xi(indk1));
83      end
84  end
85
86  %\bar{u}^{k+1/2}
87  sumar = variab.ndif + variab.nalx;
88  umedio = cell(variab.ncon, nv);
89  for j = 1:variab.ncon
90      umedio{j,1} = @(xi) xi(i0 + sumar + j);
91      for i = 2:nv
92          indk = i0 + (i-2)*nvar + sumar + j;
93          indk1 = i0 + (i-1)*nvar + sumar + j;
94          umedio{j,i} = @(xi) (1/2)*(xi(indk) + xi(indk1));
95      end
96  end
97
98  %\bar{f}^{k+1}
99  barf = cell(f{end,1}, nv);
100 for j = 1:f{end,1}
101     barf{j,1} = @(xi) 0;
102     for i = 2:nv
103         barf{j,i} = @(xi) f{j,1}(barmat(barx,xi,i), ...
104               barmat(zmedio,xi,i), barmat(umedio,xi,i), xi(1:i0));
105     end
106 end
107
108 %Función obxectivo con argumento \xi.
109 Fi = @(xi) Phi(matrix(xi,variab,nv,1), matrix(xi,variab,nv,2), ...
110                matrix(xi,variab,nv,3), xi(indp) );
111
112 %FUNCIÓNS G(xi) = 0 e H(xi)<=0.
113 %Restricións en celdas.
114 preG = cell(variab.ndif,nv);
115 for j = 1:variab.ndif
116     %Condicións iniciais.
117     preG{j,1} = @(xi) xi(i0 + j) - conx{j,1};
118     %Esquema de tipo H-S.
119     for i = 2:nv
120         %Índices i, i-1.
121         ind1 = i0 + (i-1)*nvar + j;
122         ind = i0+ (i-2)*nvar + j;
123         %Restrición correspondente.
124         preG{j,i} = @(xi) xi(ind1) - xi(ind) - ...
125          h(i-1)*(fi{j,i-1}(xi) + 4.*barf{j,i}(xi) + fi{j,i}(xi))/6;
126     end
127 end
128 %Condicións finais.
129 preGf = cell(conf{end,1}, 1);
```

```
130 for j=1:conf{end,1}
131     preGf{j,1}=@(xi) xi(end-nvar+j) - conf{j,1};
132 end
133
134 % Función vectorial G e H.
135 Gi = @(x)[]; Hi = @(x)[];
136 for i =1:nv
137     for j = 1:variab.ndif
138         Gi = @(x) [Gi(x); preG{j,i}(x)];
139     end
140     for j = 1:g{end,1}
141         Gi = @(x) [Gi(x); gi{j,i}(x)];
142     end
143     for j = 1:c{end,1}
144         Hi = @(x) [Hi(x); ci{j,i}(x)];
145     end
146 end
147 for j = 1:conf{end,1}
148     Gi = @(xi) [Gi(xi); preGf{j,1}(xi)];
149 end
150
151 % LÍMITES DAS VARIABLES
152 % Límites inferiores e superiores de parámetros
153 lboundspvec = zeros(variab.npar,1);
154 uboundspvec = zeros(variab.npar,1);
155 for i = 1:variab.npar
156     lboundspvec(i) = lboundsp{i,1};
157     uboundspvec(i) = uboundsp{i,1};
158 end
159 % Límites inferiores e superiores de variables continuas
160 lboundsvec=zeros(nvar,1);
161 uboundsvec=zeros(nvar,1);
162 for i = 1:variab.ndif
163     lboundsvec(i) = lboundsx{i,1};
164     uboundsvec(i) = uboundsx{i,1};
165 end
166 for i = 1:variab.nalx
167     lboundsvec(variab.ndif + i) = lboundsz{i,1};
168     uboundsvec(variab.ndif + i) = uboundsz{i,1};
169 end
170 for i = 1:variab.ncon
171     lboundsvec(variab.ndif + variab.nalx + i) = lboundsu{i,1};
172     uboundsvec(variab.ndif + variab.nalx + i) = uboundsu{i,1};
173 end
174 % Límites inferiores e superiores totais.
175 lbounds = zeros(i0 + nv*nvar,1);
176 ubounds = zeros(i0 + nv*nvar,1);
177 lbounds(indp) = lboundspvec;
178 ubounds(indp) = uboundspvec;
179 for i = 1:nv
180        lbounds((i0 + (i-1)*nvar + 1): (i0 + i*nvar)) = lboundsvec;
181        ubounds((i0 + (i-1)*nvar + 1): (i0 + i*nvar)) = uboundsvec;
182 end
183 end
```

Programa A.18: inic.m

```
function [xh, zh, uh, ph] = inic(variab, conx, conf, nv)
%Función de inicialización das variables do sistema avaliadas na
%malla. Inicialízanse todas a vector de uns, e engádense ás
%diferenciais as condicións iniciais, e as finais se as hai.
%(Ollo: ter coidado se as variables non poden tomar o valor 1,
%asignar outros valores manualmente)
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do sistema.
%conx: celda coas condicións iniciais do problema.
%conf: celda coas condicións finais do problema.
%nv: nº de puntos da malla.
%
%DATOS DE SAÍDA:
%xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
%a variable diferencial i-ésima no nodo j-ésimo.
%zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
%a variable alxébrica i-ésima no nodo j-ésimo.
%uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
%a variable de control i-ésima no nodo j-ésimo.
%ph: vector cos parámetros do sistema (valores inciais): de
%lonxitude variab.npar.

%Variables de estado diferenciais.
xh = ones(variab.ndif, nv);
%Condicións iniciais.
for i = 1:conx{end,1}
    xh(i,1) = conx{i,1};
end
%Condicións finais.
for i = 1:conf{end,1}
    xh(i,nv) = conf{i,1};
end

%Variables de estado alxébricas.
zh = ones(variab.nalx, nv);

%Variables de control.
uh = ones(variab.ncon, nv);

%Parámetros.
ph = ones(variab.npar, 1);

end
```

Programa A.19: intuitivo.m

```
function I = intuitivo(erroestimado, nel, q ,r, tol)
%Función para decidir o nº de puntos que se van engadir no
%proceso de  refinamento da malla de xeito intuitivo: vaise
%engadindo ao intervalo que maior erro ten.
%
%DATOS DE ENTRADA:
```

```
7  %erroestimado: vector coas estimacións do erro por intervalos
8  %da malla.
9  %nel: nº de elementos/intervalos da malla.
10 %q: orde do método.
11 %r: redución da orde, vector por intervalos.
12 %tol: tolerancia do erro.
13 %
14 %DATOS DE SAÍDA:
15 %I: vector co nº de puntos que se engaden a cada intervalo.
16
17 %Inicialización vector de número de puntos.
18 I = zeros(nel,1);
19
20 %Erro futuro.
21 erroestimadonovo = erroestimado;
22
23 %Bucle: como moito 5 puntos en total.
24 for k=1:5
25     %Erro máximo e índice correspondente.
26     [erroa, ind] = max(erroestimadonovo);
27     if I(ind) == 5 %Xa se engadiron 5 puntos a un intervalo.
28         return
29     elseif erroa <= tol %O erro máximo é pequeno.
30         return
31     else %Engadirase un punto ao elemento correspondente.
32         I(ind) = I(ind) + 1;
33     end
34     %Estimación do erro futuro.
35     erroestimadonovo(ind) = erroestimado(ind)*((1/(I(ind) ...
36                                             +1))^(q-r(ind)+1));
37 end
38 end
```

Programa A.20: matrix.m

```
1  function A = matrix(xi, variab, nv, ind)
2  %Función para pasar como argumentos as matrices xh, zh ou uh no
3  %canto do vector xi.
4  %
5  %DATOS DE ENTRADA:
6  %xi: vector coas variables, avaliación das funcións orixinais nos
7  %nodos da malla.
8  %variab: estrutura coas dimensións do problema orixinal.
9  %nv: nº de nodos da malla.
10 %ind: 1 xh, 2 zh, 3 uh.
11 %
12 %DATOS DE SAÍDA:
13 %A: matriz a partir de xi segundo que variable se requira.
14
15 if ind == 1 %Variables x.
16     [A, ~, ~, ~] = xi2discr(variab, nv, xi);
17 elseif ind == 2 %Variables z.
18     [~, A, ~, ~] = xi2discr(variab, nv, xi);
19 elseif ind == 3 %Variables u.
```

```
    [~, ~, A, ~] = xi2discr(variab, nv, xi);
end
end
```

Programa A.21: novamalla.m

```
function [thi, I] = novamalla(th, erroestimado, r, q, ...
    estratexia, tol)
% Cálculo dos novos puntos da malla.
%
%DATOS DE ENTRADA:
%th: malla actual.
%erroestimado: vector de \eps_k, que son os erros estimados na
%solución actual e que se pretenden diminuir.
%r: vector de r_k's coas reducións estimadas da orde en cada
%intervalo.
%q: orde do método empregado para discretizar.
%estratexia: 0 calcula os puntos de xeito intuitivo mentres que 1
%calcula resolvendo un problema de programación enteira.
%tol: tolerancia do erro.
%
%DATOS DE SAÍDA:
%thi: nova malla.
%I: puntos engadidos en cada intervalo.

%Número de elementos da malla actual.
nel = length(th)-1;

if estratexia == 1
    %ESTRATEXIA PROBLEMA DE PROGRAMACIÓN ENTEIRA.
    %Definición da función obxectivo.
    funk = @(x) erroestimado(1)*((1/(x(1)+1))^(q-r(1)+1));
    for k = 2:nel
       funk = @(x) [funk(x); ...
                    erroestimado(k)*((1/(x(k)+1))^(q-r(k)+1))];
    end
    fun = @(x) sum(funk(x));
    %fun = @(x) max(funk(x)); Con esta opción chega un momento no
    %que non engade puntos mais todavía hai erro.

    %Chamada para ga. Función de Matlab que resolve o problema.
    %M = 6, M_1 = 5.
    I = probint(fun, nel, 6, 5);

elseif estratexia == 0
    %Cálculo con algoritmo heurístico.
    I = intuitivo(erroestimado, nel, q, r, tol);
end

%Construción dos novos puntos da malla a partir do número de
%puntos que se engaden.
thi = engadir(I, th);
end
```

Programa A.22: peso.m

```
function pesosw = peso(th, ppxh, ppxhder)
% Cálculo dos pesos w que se empregan para obter un erro relativo.
%
%DATOS DE ENTRADA:
%th: nodos da malla.
%ppxh: celda de variab.ndif filas e 1 columna que na fila i-ésima
%contén o spline que interpola os datos da variable diferencial
%i-ésima.
%ppxhder: celda de variab.ndif filas e 1 columna que na fila
%i-ésima contén o spline derivada de ppxh{i,1}.
%
%DATOS DE SAÍDA:
%pesosw: vector de pesos cuxa coordenada i-ésima é o máximo do
%valor absoluto de x_i nos nodos da malla e da súa derivada.

%Número de variables de estado diferenciais.
ndif = size(ppxh, 1);
%Vector de pesos.
pesosw = ones(ndif, 1);
for i = 1:ndif
    pesosw(i) = max( [abs( ppval(ppxh{i,1},th) ), ...
                        abs( ppval(ppxhder{i,1},th) )] );
end
end
```

Programa A.23: probint.m

```
function I = probint(fun, nel, M, M1)
%Función para resolver o problema de programación enteira que
%pretender atopar o nº de puntos que se engade a cada subintervalo
%co obxectivo de mellorar o erro estimado.
%
%DATOS DE ENTRADA:
%fun: función obxectivo (sum_k \eps_k).
%nel: nº de elementos da malla, nv - 1.
%M-1: nº máximo de puntos a engadir en total.
%M1: nº máximo de puntos a engadir nun intervalo.
%
%DATOS DE SAÍDA:
%I: vector co nº de puntos a engadir en cada intervalo.

%Restricións do problema.
A = ones(1, nel);
b = M-1;
Aeq = []; beq = [];
%Límites.
lb = zeros(1, nel);
ub = M1*ones(1, nel);

nonlcon=[];

%Número de variables enteiras.
intcon = 1:nel;
```

```
27
28 options = optimoptions('ga', 'Display', 'off', ...
29                                     'FunctionTolerance', 1.e-30);
30
31 % Resolución.
32 I = ga(fun, nel, A, b, Aeq, beq, lb, ub, nonlcon, intcon, options);
33 end
```

Programa A.24: reducion.m

```
1  function r = reducion(etaerro, thetaerro, q, I, errointegraNdo, ...
2      thprev)
3  % Función para o vector de r_k's onde cada r_k é a estimación da
4  % redución da orde sufrida no intervalo k-ésimo da malla actual.
5  % Tomarase como estimación k-ésima a media das estimacións no
6  % intervalo k de cada compoñente dos estados.
7  %
8  % DATOS DE ENTRADA:
9  % etaerro: erro integrado estimado na malla actual.
10 % thetaerro: erro integrado estimado na malla previa.
11 % q: orde do método de discretización.
12 % I: vector co nº de puntos engadidos á malla antiga.
13 % errointegraNdo: celda con f{end,1} (ou variab.ndif) filas e
14 % 1 columna que contén a función de erro por compoñentes.
15 % thprev: nodos da malla previa.
16 %
17 % DATOS DE SAÍDA:
18 % r: vector coa redución de orde por intervalos.
19
20 % Número de elementos da malla.
21 rtam = size(etaerro, 1);
22 % Número de variables de estado diferenciais.
23 comp = size(etaerro, 2);
24 % Inicialización vector r de reducións de orde.
25 r = zeros(rtam, 1);
26 % Número de elementos da malla previa.
27 inter = size(thetaerro, 1);
28 % Vectores auxiliares intermedios.
29 prerhat = zeros(inter, 1);
30 prer = zeros(inter, 1);
31
32 for k = 1:inter
33    % r_k
34    % Argumento do logaritmo do denominador.
35    denom = 1 + I(k);
36    % Erro do elemento k para cada compoñente: vectores.
37    etavec = zeros(comp, 1);
38    thetavec = zeros(comp, 1);
39    for i = 1:comp
40        etavec(i) = integral(errointegraNdo{i,1}, thprev(k), ...
41                    thprev(k+1), 'RelTol', 1e-10, 'AbsTol', 1e-15);
42        thetavec(i) = thetaerro(k,i);
43    end
44    % Vector de redución da orde nos elementos previos.
```

```
    prerhat(k) = mean(q + 1 - log(thetavec./etavec)/log(denom));
    prer(k) = max(0, min(round(prerhat(k)),q));
end

%Reparto da redución da orde polos elementos que proveñen do
%mesmo previo.
j = 1;
for k = 1:inter
    %Puntos engadidos ao elemento k.
    numpuntos = I(k);
    if numpuntos == 0
        %Mantense un único intervalo.
        r(j) = prer(k);
        j = j+1;
    elseif numpuntos >= 1
        %Quedan numpuntos + 1 intervalos.
        r(j:(j+numpuntos)) = prer(k);
        j = j + numpuntos + 1;
    end
end
end
```

Programa A.25: refinamento0.m

```
function [th, nv, I, xh, zh, uh, ref, thetaerro, thprev] = refinamento0( ...
    ref, etaerro, thetaerro, I, errointegraNdo, th, nv,...
    erroestimado, q, estratexia, variab, ppxh, ppzh, ppuh, ...
    thprev, tol)
%Función para refinar a malla coa que se está a traballar e
%actualizar as variables no caso de que non se inclúa o paso 3
%do algoritmo de refinamento (é dicir, non se muda a
%discretización).
%
%DATOS DE ENTRADA:
%ref: nº de refinamentos realizados da malla até o momento.
%etaerro: erro integrado estimado na malla actual.
%thetaerro: erro integrado estimado na malla previa.
%I: vector co nº de puntos engadidos á malla antiga.
%errointegraNdo: celda con f{end,1} (ou variab.ndif) filas e 1
%columna que contén a función de erro por compoñentes.
%th: malla actual.
%nv: nº de puntos da malla.
%erroestimado: vector de \eps_k, que son os erros estimados na
%solución actual e que se pretenden diminuir.
%q: orde do método empregado para discretizar.
%estratexia: 0 calcula os puntos de xeito intuitivo mentres que
%1 calcula resolvendo un problema de programación enteira.
%variab: estrutura co nº de variables do problema orixinal.
%ppxh: celda de variab.ndif filas e 1 columna que na fila
%i-ésima contén o spline que interpola os datos da variable
%diferencial i-ésima.
%ppzh: celda de variab.nalx filas e 1 columna que na fila
%i-ésima contén o spline que interpola os datos da variable
%alxébrica i-ésima.
```

```
%ppuh: celda de variab.ncon filas e 1 columna que na fila
%i-ésima contén o spline que interpola os datos da variable de
%control i-ésima.
%thprev: nodos da malla previa.
%tol: tolerancia do erro pedida.
%
%DATOS DE SAÍDA: (actualización)
%th: malla temporal, con nv elementos.
%nv: nº de puntos da malla.
%I: vector co nº de puntos engadidos a cada intervalo da malla
%previa.
%xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
%a variable diferencial i-ésima no nodo j-ésimo.
%zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
%a variable alxébrica i-ésima no nodo j-ésimo.
%uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
%a variable de control i-ésima no nodo j-ésimo.
%ref: nº de refinamentos realizados da malla até o momento.
%thetaerro: erro integrado estimado na malla previa.
%thprev: nodos da malla previa.

%ESTIMACIÓN DA REDUCIÓN DA ORDE
if ref >= 1
    %Xa se conta con información dunha primeira malla (hai dous
    %erros).
    r = reducion(etaerro, thetaerro, q, I, errointegraNdo, ...
                                                    thprev);
else
    %r para o primeiro refinamento.
    r = zeros(nv-1, 1);
end

%CONSTRUCIÓN DA NOVA MALLA
[thi, I] = novamalla(th, erroestimado, r, q, estratexia, tol);

%ACTUALIZACIÓN VARIABLES
%Almacenaxe malla previa.
thprev = th;
%Nova malla.
th = thi; nv = length(thi);
%Aumentou o nº de refinamentos realizados.
ref = ref + 1;
%Almacenaxe erro da malla previa.
thetaerro = etaerro;
%Actualización das variables mediante avaliación dos splines.
[xh, zh, uh] = actualizacion(th, ppxh, ppzh, ppuh, variab);
end
```

Programa A.26: refinamento1.m

```
function [th, nv, I, xh, zh, uh, discret, ref, thetaerro,thprev] = refinamento1( ...
    i, ref, discret, etaerro, thetaerro, I, errointegraNdo, th, ...
    nv, erroestimado, q, estratexia, variab, ppxh, ppzh, ppuh, ...
    thprev, tol)
```

```
5  %Función para refinar a malla coa que se está a traballar e
6  %actualizar as variables no caso de que se inclúa o paso 3 do
7  %algoritmo de refinamento (é dicir, decídese se mudar a
8  %discretización).
9  %
10 %DATOS DE ENTRADA:
11 %i: iteración do algoritmo.
12 %ref: nº de refinamentos realizados da malla até o momento.
13 %discret: tipo de discretización empregada para os esquemas
14 %Runge-Kutta, trapecios [1], Hermite-Simpson [2].
15 %etaerro: erro integrado estimado na malla actual.
16 %thetaerro: erro integrado estimado na malla previa.
17 %I: vector co nº de puntos engadidos á malla antiga.
18 %errointegraNdo: celda con f{end,1} (ou variab.ndif) filas e
19 %1 columna que contén a función de erro por compoñentes.
20 %th: malla actual.
21 %erroestimado: vector de \eps_k, que son os erros estimados na
22 %solución actual e que se pretenden diminuir.
23 %q: orde do método empregado para discretizar.
24 %estratexia: 0 calcula os puntos de xeito intuitivo mentres que
25 %1 calcula resolvendo un problema de programación enteira.
26 %variab: estrutura co nº de variables do problema orixinal.
27 %ppxh: celda de variab.ndif filas e 1 columna que na fila
28 %i-ésima contén o spline que interpola os datos da variable
29 %diferencial i-ésima.
30 %ppzh: celda de variab.nalx filas e 1 columna que na fila
31 %i-ésima contén o spline que interpola os datos da variable
32 %alxébrica i-ésima.
33 %ppuh: celda de variab.ncon filas e 1 columna que na fila
34 %i-ésima contén o spline que interpola os datos da variable de
35 %control i-ésima.
36 %thprev: nodos da malla previa.
37 %tol: tolerancia do erro.
38 %
39 %DATOS DE SAÍDA: (actualización)
40 %th: malla temporal, con nv elementos.
41 %nv: nº de puntos da malla.
42 %I: vector co nº de puntos engadidos a cada intervalo da
43 %malla previa.
44 %xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
45 %a variable diferencial i-ésima no nodo j-ésimo.
46 %zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
47 %a variable alxébrica i-ésima no nodo j-ésimo.
48 %uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
49 %a variable de control i-ésima no nodo j-ésimo.
50 %discret: método de discretización que se vai empregar a partir
51 %de agora na construción dos esquemas Runge-Kutta.
52 %ref: nº de refinamentos realizados da malla até o momento.
53 %thetaerro: erro integrado estimado na malla previa.
54 %thprev: nodos da malla previa.
55
56 %Elección da nova orde
57 if discret == 1
58     discretnew = cambiodeorde(i, erroestimado, discret);
```

```
elseif discret == 2
    discretnew = 2;
end
% Actualización da orde: orde correcta teórica coa que hai que
% traballar.
if discretnew == 2
    q = 4;
end

% Se non se decidiu cambiar de método de discretización,
% continúase co algoritmo de refinamento.
if discretnew == discret
    % ESTIMACIÓN DA REDUCIÓN DA ORDE
    if ref >= 1
        % Xa se conta con información dunha primeira malla
        % (hai dous erros).
        r = reducion(etaerro, thetaerro, q, I, ...
                        errointegraNdo, thprev);
    else
        % r para o primeiro refinamento.
        r = zeros(nv-1, 1);
    end

    % CONSTRUCIÓN DA NOVA MALLA
    [thi, I] = novamalla(th, erroestimado, r, q, estratexia, tol);

    % ACTUALIZACIÓN VARIABLES
    % Almacenaxe malla previa.
    thprev = th;
    % Nova malla.
    th = thi; nv = length(thi);
    % Aumentou o nº de refinamentos realizados.
    ref = ref + 1;
    % Almacenaxe erro da malla previa.
    thetaerro = etaerro;
end

% Actualización das variables mediante avaliación dos splines.
[xh, zh, uh] = actualizacion(th, ppxh, ppzh, ppuh, variab);
% Actualízase a discretización.
discret = discretnew;
end
```

Programa A.27: SQP.m

```
function xii = SQP(xi0, Fi, Gi, Hi, lbounds, ubounds)
% Función para a resolución dun NLP mediante Sequential Quadratic
% Programming. Constrúese a estrutura problem e chámase a fmincon.
%
% DATOS DE ENTRADA:
% xi0: iterante inicial.
% Fi: función obxectivo.
% Gi: Gi(xi) = 0.
% Hi: Hi(xi) <= 0.
```

```
%lbounds: límites inferiores.
%ubounds: límites superiores.
%
%DATOS DE SAÍDA:
%xii: solución do NLP.

problem.objective = Fi;
problem.x0 = xi0;
problem.Aineq = []; problem.bineq = [];
problem.Aeq = []; problem.beq = [];
problem.lb = lbounds; problem.ub = ubounds;
problem.nonlcon = @nonlinear;
problem.solver = 'fmincon';
problem.options = optimoptions('fmincon', 'Algorithm', 'sqp', ...
                    'MaxIterations', 50, 'Display', 'off');
xii = fmincon(problem);

function [c ceq] = nonlinear(x)
c = Hi(x);
ceq = Gi(x);
end
end
```

Programa A.28: TrapColl.m

```
function [Fi, Gi, Hi,lbounds, ubounds] = TrapColl(variab, nv, ...
    th, Phi, f, g, c, conx, conf, lboundsp, lboundsu, uboundsp, ...
    uboundsu, lboundsx, uboundsx, lboundsz, uboundsz)
%Función para obter a función obxectivo e restricións do problema
%NLP mediante un esquema de discretización de trapecios.
%
%DATOS DE ENTRADA:
%variab: estrutura co nº de variables do problema.
%nv: nº de puntos da malla.
%th: puntos da malla.
%Phi: función obxectivo do problema orixinal.
%f: celda coas funcións das restricións de tipo x' = f.
%g: celda coas funcións das restricións alxébricas 0 = g.
%c: celda coas restricións de inecuación do problema orixinal.
%conx: celda coas condicións iniciais do sistema x' = f.
%conf: celda coas condicións finais das variables de estado.
%lbounds, ubounds (celdas cos límites das distintas variables)
%
%DATOS DE SAÍDA: (Funcións Vectoriais)
%Fi: función obxectivo do NLP.
%Gi: restricións de igualdade do NLP, Gi(xi) = 0.
%Hi: restricións do NLP do tipo Hi(xi) <= 0.
%lbounds: límites inferiores de xi.
%ubounds: límites superiores de xi.

%Número de variables continuas do sistema.
nvar = variab.ndif + variab.nalx + variab.ncon;
%Parámetros
i0 = variab.npar;
```

```
indp = 1:i0;

%f(x(t_i)), g(x(t_i)), c(x(t_i))
fi = cell(f{end,1}, nv); gi = cell(g{end,1}, nv);
ci = cell(c{end,1}, nv);
for i = 1:nv
    %Índices respectivos no vector xi para conseguir as
    %variables avaliadas en t_i-1.
    indx = (i0 + (i-1)*nvar + 1):(i0 + (i-1)*nvar + variab.ndif);
    indz = (i0+ (i-1)*nvar + variab.ndif + 1):(i0 + (i-1)*nvar +...
                                        variab.ndif + variab.nalx);
    indu = (i0 + (i-1)*nvar + variab.ndif + variab.nalx ...
                                                + 1):(i0 + i*nvar);
    %Funcións f nos nodos da malla.
    for j = 1:f{end,1}
        fi{j,i} = @(xi) f{j,1}(xi(indx), xi(indz), xi(indu), ...
                                                        xi(indp));
    end
    %Funcións g nos nodos da malla.
    for j = 1:g{end,1}
        gi{j,i} = @(xi) g{j,1}(xi(indx), xi(indz), xi(indu), ...
                                                        xi(indp));
    end
    %Funcións c nos nodos da malla.
    for j = 1:c{end,1}
        ci{j,i} = @(xi) c{j,1}(xi(indx), xi(indz), xi(indu), ...
                                                        xi(indp));
    end
end

%Lonxitude dos elementos da malla.
h = th(2:end) - th(1:end-1);

%Función obxectivo con argumento \xi.
Fi = @(xi) Phi(matrix(xi,variab,nv,1), matrix(xi,variab,nv,2), ...
                matrix(xi,variab,nv,3), xi(indp));

%FUNCIÓNS G(xi) = 0 e H(xi) <= 0.
%Restricións en celdas.
preG = cell(variab.ndif,nv);
for j = 1:variab.ndif
    %Condicións iniciais.
    preG{j,1} = @(xi) xi(i0 + j) - conx{j,1};
    %Esquema de tipo trapecios.
    for i = 2:nv
        %Índices i, i-1.
        ind1 = i0 + (i-1)*nvar + j;
        ind = i0+ (i-2)*nvar + j;
        %Restrición correspondente.
        preG{j,i} = @(xi) xi(ind1) - xi(ind) - ...
                        h(i-1)*(fi{j,i-1}(xi) + fi{j,i}(xi))/2 ;
    end
end
%Condicións finais.
```

```
84  preGf = cell(conf{end,1}, 1);
85  for j = 1:conf{end,1}
86      preGf{j,1} = @(xi) xi(end-nvar+j) - conf{j,1};
87  end
88
89  %Función vectorial G e H.
90  Gi = @(x)[]; Hi = @(x)[];
91  for i =1:nv
92      for j = 1:variab.ndif
93          Gi = @(x) [Gi(x); preG{j,i}(x)];
94      end
95      for j = 1:g{end,1}
96          Gi = @(x) [Gi(x); gi{j,i}(x)];
97      end
98      for j = 1:c{end,1}
99          Hi = @(x) [Hi(x); ci{j,i}(x)];
100     end
101 end
102 for j = 1:conf{end,1}
103     Gi = @(xi) [Gi(xi); preGf{j,1}(xi)];
104 end
105
106 %LÍMITES DAS VARIABLES.
107 %Límites inferiores e superiores de parámetros.
108 lboundspvec = zeros(variab.npar,1);
109 uboundspvec = zeros(variab.npar,1);
110 for i = 1:variab.npar
111     lboundspvec(i) = lboundsp{i,1};
112     uboundspvec(i) = uboundsp{i,1};
113 end
114 %Límites inferiores e superiores de variables continuas.
115 lboundsvec = zeros(nvar,1);
116 uboundsvec = zeros(nvar,1);
117 for i = 1:variab.ndif
118     lboundsvec(i) = lboundsx{i,1};
119     uboundsvec(i) = uboundsx{i,1};
120 end
121 for i = 1:variab.nalx
122     lboundsvec(variab.ndif + i) = lboundsz{i,1};
123     uboundsvec(variab.ndif + i) = uboundsz{i,1};
124 end
125 for i = 1:variab.ncon
126     lboundsvec(variab.ndif + variab.nalx + i) = lboundsu{i,1};
127     uboundsvec(variab.ndif + variab.nalx + i) = uboundsu{i,1};
128 end
129 %Límites inferiores e superiores totais.
130 lbounds = zeros(i0 + nv*nvar,1);
131 ubounds = zeros(i0 + nv*nvar,1);
132 lbounds(indp) = lboundspvec;
133 ubounds(indp) = uboundspvec;
134 for i = 1:nv
135       lbounds((i0 + (i-1)*nvar + 1): (i0 + i*nvar)) = lboundsvec;
136       ubounds((i0 + (i-1)*nvar + 1): (i0 + i*nvar)) = uboundsvec;
137 end
```

```
138 end
```

Programa A.29: xi2discr.m

```
1  function [xh, zh, uh, ph] = xi2discr(variab, nv, xi)
2  %Función que pasa do vector xi ás variables distinguidas xh, zh,
3  %uh, ph.
4  %
5  %DATOS DE ENTRADA:
6  %variab: estrutura co nº de variables do problema.
7  %nv: nº de nodos da malla.
8  %xi: vector de variables.
9  %
10 %DATOS DE SAÍDA:
11 %xh: matriz (variab.ndif x nv) onde o elemento (i,j) representa
12 %a variable diferencial i-ésima no nodo j-ésimo.
13 %zh: matriz (variab.nalx x nv) onde o elemento (i,j) representa
14 %a variable alxébrica i-ésima no nodo j-ésimo.
15 %uh: matriz (variab.ncon x nv) onde o elemento (i,j) representa
16 %a variable de control i-ésima no nodo j-ésimo.
17 %ph: vector (variab.npar x 1) de parámetros.
18
19 %Variables continuas do problema.
20 nvar = variab.ndif + variab.nalx + variab.ncon;
21 %Número de parámetros.
22 i0 = variab.npar;
23
24 %Matriz intermedia, coas variables por filas, nodos por columnas.
25 prexi = zeros(nvar,nv);
26 for i = 1:nv
27     in = i0 + (i-1)*nvar + 1;
28     fin = i0 + i*nvar;
29     prexi(:,i) = xi(in:fin);
30 end
31
32 %Variables nos nodos
33 in = 1; fin = variab.ndif;
34 xh = prexi(in:fin, :);
35
36 in = in + variab.ndif; fin = fin + variab.nalx;
37 zh = prexi(in:fin, :);
38
39 in = in + variab.nalx; fin = fin + variab.ncon;
40 uh = prexi(in:fin, :);
41
42 ph = xi(1:i0);
43 end
```

A.4. Representación gráfica

Programa A.30: graficos.m

```
% Código para obter as representacións gráficas do traballo.

% Problema de freada dun bloque.
thnew =[];
for i = 1:length(th)-1
    v = linspace(th(i),th(i+1),4);
    thnew = [thnew v(1:end-1)];
end
thnew = [thnew th(end)];
subplot(1,2,1)
yy = spline(th,uh,thnew);
plot(thnew,yy)
ylim([1,2])
xlabel('t'); ylabel('u(t)');
title('Forza exercida polo operario');
hold on
plot(th,uh,'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
            'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off
subplot(1,2,2)
yy = spline(th,xh,thnew);
plot(thnew,yy)
plot(th,xh)
xlabel('t'); ylabel('x(t)');
title('Velocidade do bloque co tempo');
hold on
plot(th,xh,'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
            'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off

% Problema do reactor descontinuo.
thnew =[];
for i = 1:length(th)-1
    v = linspace(th(i),th(i+1),4);
    thnew = [thnew v(1:end-1)];
end
thnew = [thnew th(end)];
yy = spline(th,uh,thnew);
plot(thnew,yy)
ylim([0,5.5])
xlabel('t');ylabel('u(t)');
title('Perfil de temperaturas na reacción');
hold on
plot(th,uh,'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
        'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off

subplot(1,2,1)
yy = spline(th,xh(1,:),thnew);
plot(thnew,yy)
```

```
xlabel('t'); ylabel('x_1(t)'); title('Cantidade de reactivo A');
ylim([0,1])
hold on
plot(th,xh(1,:),'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
        'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off
subplot(1,2,2)
yy = spline(th,xh(2,:),thnew);
plot(thnew,yy)
xlabel('t'); ylabel('x_2(t)'); title('Cantidade de produto B')
ylim([0 1])
hold on
plot(th,xh(2,:),'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
             'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off

%Problema do carro con péndulo.
thnew =[];
for i = 1:length(th)-1
    v = linspace(th(i),th(i+1),4);
    thnew = [thnew v(1:end-1)];
end
thnew = [thnew th(end)];
subplot(1,2,1)
yy = spline(th,xh(1,:),thnew);
plot(thnew,yy)
xlabel('t'); ylabel('x_1(t)'); title('Movemento do carro');
ylim([0,1.5])
hold on
plot(th,xh(1,:),'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
        'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off
subplot(1,2,2)
yy = spline(th, xh(2,:), thnew);
plot(thnew,yy)
xlabel('t'); ylabel('x_2(t)');
title('Ángulo do péndulo durante o movemento')
ylim([-2,4])
hold on
plot(th,xh(2,:),'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
          'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off

subplot(1,1,1)
yy = spline(th,uh,thnew);
plot(thnew,yy)
ylim([-20,10])
xlabel('t'); ylabel('u(t)'); title('Forza aplicada ao carro');
hold on
plot(th,uh,'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
    'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
hold off

%Problema de Rayleigh
```

```
105 thnew =[];
106 for i = 1:length(th)-1
107     v = linspace(th(i),th(i+1),4);
108     thnew = [thnew v(1:end-1)];
109 end
110 thnew = [thnew th(end)];
111 yy = spline(th,uh,thnew);
112 plot(thnew,yy)
113 ylim([-2,8])
114 xlabel('t'); ylabel('u(t)'); title('Problema de Rayleigh');
115 hold on
116 plot(th,uh,'o','MarkerEdgeColor',[0 0.4470 0.7410], ...
117         'MarkerFaceColor',[0 0.4470 0.7410],'MarkerSize',2)
118 hold off
```

Bibliografía

[1] John T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*, second ed., Society for Industrial and Applied Mathematics, 2010.

[2] Lorenz T. Biegler, *Nonlinear programming: Concepts, algorithms, and applications to chemical processes*, Society for Industrial and Applied Mathematics, USA, 2010.

[3] Lorenz T. Biegler, Stephen L. Campbell e Volker Mehrmann, *Control and optimization with differential-algebraic constraints*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2012.

[4] Carl de Boor, *A practical guide to splines*, Springer-Verlag, New York, 1987.

[5] Arthur E. Bryson e Ho Yu-Chi, *Applied optimal control: optimization, estimation and control*, John Wiley & Sons, New York, 1975.

[6] The MathWorks Inc., *MATLAB*, Versión: 24.1.0.2568132 (R2024a), Software, 2024.

[7] Matthew Kelly, *An introduction to trajectory optimization: How to do your own direct collocation*, SIAM Review **59** (2017), no. 4, 849-904.

[8] Lev S. Pontryagin, *The mathematical theory of optimal processes*, John Wiley & Sons, New York, 1965.

[9] Sebastian Sager, *Numerical methods for mixed–integer optimal control problems*, Ph.D. thesis, Universität Heidelberg, 2006.