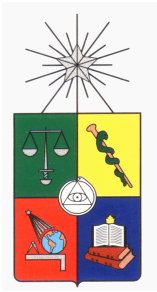

UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

IMPLEMENTACIÓN DE UN MÉTODO DE PROGRAMACIÓN SEMIDEFINIDA USANDO COMPUTACIÓN PARALELA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

OSCAR FRANCISCO PEREDO ANDRADE

PROFESOR GUÍA:
SR. HÉCTOR ARIEL RAMIREZ CABRERA

MIEMBROS DE LA COMISIÓN:
SR. WALTER GÓMEZ BOFILL
SR. GONZALO JAVIER HERNÁNDEZ OLIVA

SANTIAGO DE CHILE
ABRIL 2010

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO
POR: OSCAR FRANCISCO PEREDO ANDRADE
FECHA: 20/04/2010
PROF. GUÍA: HÉCTOR ARIEL RAMIREZ CABRERA

IMPLEMENTACIÓN DE UN MÉTODO DE PROGRAMACIÓN SEMIDEFINIDA USANDO COMPUTACIÓN PARALELA

En el presente trabajo se estudió y rediseño una implementación existente del algoritmo **Filter-SDP**, el cual resuelve problemas de programación semidefinida no lineal de la forma:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.a} & h(x) = 0 \\ & G(x) \preceq 0 \end{array} \tag{1}$$

donde $f : \mathbb{R} \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ y $G : \mathbb{R}^n \to \mathbb{S}^m$ son funciones de clase C^2 y $\mathbb{S}^m$ denota el espacio lineal de las matrices simétricas de $m \times m$ dotado del producto interno $A \cdot B = \sum_{i,j=1}^{m} A_{ij}B_{ij}$. El algoritmo resuelve en cada iteración una aproximación local representada por un problema de programación semidefinida lineal, y adicionalmente se utiliza un esquema de penalización multi-objetivo, en el cual se minimiza la función objetivo y una función de mérito utilizando un *filtro*.

Se estudió la introducción de cálculo paralelo en partes específicas del algoritmo, con el objetivo de disminuír el tiempo de ejecución. Se reimplementó el algoritmo utilizando el lenguaje C y la librería de cálculo paralelo MPI. Esa nueva implementación se comparó con un desarrollo ya existente, realizado sobre la plataforma MATLAB, y se midió su speedup en los problemas más pesados de una batería de tests escogida. Como apoyo al desarrollo del algoritmo, se diseñaron nuevas fases de restauración sobre la plataforma MATLAB, con el objetivo de mejorar la calidad de las soluciones obtenidas. Se diseñaron 4 nuevos métodos para la fase de restauración del algoritmo, cuyas principales áreas de influencia son la restauración inexacta, el diseño de controladores retroalimentados de salida estática y el posicionamiento de polos.

Dentro de los resultados obtenidos, se logró visualizar las ventajas de la nueva implementación con respecto al desarrollo ya existente, así como demostrar el beneficio que se obtiene en el speedup para problemas pesados. También se realizó una comparación entre los métodos diseñados para la fase de restauración, con la cual se llegó a conclusiones que pueden abrir nuevas áreas de investigación y trabajo a futuro.

Finalmente, se aprendió a utilizar una herramienta de álgebra lineal que funciona sobre ambientes de cálculo paralelo, ScaLAPACK, y se perfeccionó el proceso de desarrollo de software que ya se tenía sobre este tipo de plataformas.

AGRADECIMIENTOS

Agradezco a mi familia por su apoyo incondicional a lo largo de este proceso. A mi madre Flora y a mis hermanas Paulina y Valentina, y a mi padre Oscar que desde lejos me entrega su apoyo.

También le agradezco a mis compañeros de la cantina por su apoyo y buena onda: Gustavo, Cristóbal, Ignacio, Manuel, Nicolás, Raúl y Felipe. Y también a mis amigos computines: Álvaro y Daniel.

Te agradezco Natalia por entregarme amor y ánimo hasta el final de este trabajo. Significa mucho para mí.

Héctor, gracias por aguantar a este memorista tan especial que te ha tocado... el proceso demoró más de lo que yo hubiera querido, pero afortunadamente llegamos a buen término y ojalá sigamos trabajando juntos en proyectos futuros.

Julián, gracias por darme acceso al servidor HPC en el laboratorio ALGES. Sin ese acceso no habría podido realizar las pruebas necesarias para este trabajo. También agradezco al equipo de sistemas del DIM y a los sysadmin del cluster syntagma, por aguantar a este humilde usuario.

Eterin, gracias por ayudarme en todos los procesos administrativos, sin tu ayuda no hubiera logrado sacar adelante este trabajo.

Para tí, Tía Cristina,
en el lugar que estés...
te extrañamos

Índice general

Índice de figuras

Índice de algoritmos

Índice de códigos

Capítulo 1

Introducción

El tema central del presente trabajo consiste en estudiar la implementación existente de un algoritmo de programación semidefinida no lineal y proponer una nueva aplicación que utiliza cálculo paralelo. Se escogió el algoritmo **Filter-SDP**, desarrollado en [GR06], el cual resuelve el siguiente problema de programación semidefinida no lineal:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & f(x) \\ \text{s.a} & h(x) = 0 \\ & G(x) \preceq 0 \end{array} \tag{1.1}$$

donde $f : \mathbb{R} \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ y $G : \mathbb{R}^n \to \mathbb{S}^m$ son funciones de clase C^2 y $\mathbb{S}^m$ denota el espacio lineal de las matrices simétricas de $m \times m$ dotado del producto interno $A \cdot B = \sum_{i,j=1}^{m} A_{ij}B_{ij}$.

El algoritmo **Filter-SDP** resuelve en cada iteración una aproximación local representada por un problema de *programación semidefinida lineal* (LSDP), muy parecido a lo que ocurre en la programación no lineal con el algoritmo de *programación cuadrática sucesiva* (SQP). Adicionalmente se utiliza un esquema de penalización multi-objetivo, el cual minimiza la función objetivo y una función de mérito (la cual es igual a cero si el punto evaluado es factible y mayor que cero en caso contrario) de manera separada utilizando un *filtro* $\mathcal{F}$, que permite discriminar puntos a ser evaluados en cada etapa por el método subyacente, revisando si un punto candidato aumenta o disminuye las funciones objetivo y de penalización, y escogiendo aquellos que mejoran ambas simultáneamente.

El algoritmo escogido realiza diversas operaciones en las cuales se estudia la aplicación de cálculo paralelo con el objetivo de disminuír el tiempo de ejecución y además se exploran nuevos métodos para algunos pasos del algoritmo **Filter-SDP**, en particular, nuevas fases de restauración.

La implementación desarrollada aborda el siguiente problema, proveniente del área de

control de sistemas:

$$\begin{array}{cccc} \min_{F\in\mathbb{R}^{p\times r},Q,V\in\mathbb{S}^n} & Tr((C_1+D_{12}FC)Q(C_1+D_{12}FC)^T) & & \\ \text{s.a} & (A+BFC)Q+Q(A+BFC)^T+B_1B_1^T & = & 0 \\ & (A+BFC)V+V(A+BFC)^T+I & = & 0 \\ & V & \succ & 0 \end{array} \tag{1.2}$$

Como tema secundario se plantea la exploración de una metodología de trabajo para transformar algoritmos desarrollados en sistemas no orientados al cálculo paralelo, en particular MATLAB © [Inc97], hacia sistemas netamente orientados, en particular ANSI C [KR78] y MPI [Pac96]. Esta metodología se describe utilizando como base los comentarios de desarrolladores expertos y la experiencia personal adquirida.

1.1. Contexto

Los principales conceptos utilizados en este trabajo son la programación semidefinida, los métodos de filtro y la computación paralela. A continuación se presenta una introducción breve de cada una junto con su contexto histórico.

1.1.1. Programación Semidefinida

La *programación semidefinida* (SDP) es una rama de la programación matemática cuyo objetivo es minimizar la función lineal $c^{\mathrm{T}}x$ donde $c, x \in \mathbb{R}^m$, con c constante y x variable, bajo la restricción:

$$F_0 + \sum_{i=1}^{m} x_i F_i \succeq 0$$

donde $F_0, \ldots, F_n \in \mathbb{S}^n$ son matrices reales simétricas constantes. El problema SDP lineal primal se plantea de la forma:

$$\begin{array}{cc} \underset{x\in\mathbb{R}^m}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & F_0+\sum_{i=1}^m x_iF_i \succeq 0 \end{array} \tag{1.3}$$

Ésta área de investigación se mantuvo sin llamar mayormente la atención hasta finales de la década de los 80's y principios de los 90's, época en la que se introdujeron métodos teóricamente eficientes y que funcionaban muy bien en la práctica.

Junto al crecimiento explosivo de algoritmos eficientes para su resolución, el modelamiento de diversos problemas provenientes de una gran cantidad de áreas contribuyó de manera significativa a la investigación en programación semidefinida. Algunas de esas áreas son la optimización combinatorial ([Lov79], [GW95], [Ali95]), la teoría de control

de sistemas ([BEFB93]), el diseño estructural ([BTe93]) y la estadística ([BW80], [Fle81], [Fle85], [Sha82]).

Ejemplos, aplicaciones y algoritmos de resolución para la programación semidefinida se revisarán con mayor detalle en el capítulo Antecedentes.

1.1.2. Métodos de Filtro

El problema general de la programación no lineal general se plantea de la forma

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.a} & c(x) \leq 0 \end{array} \tag{1.4}$$

con $f : \mathbb{R}^n \to \mathbb{R}$ y $c : \mathbb{R}^n \to \mathbb{R}$ de clase $\mathcal{C}^2$. La mayoría de los métodos que resuelven (1.4) están basados en el método de Newton y son iterativos, es decir, dado un punto x_k, se resuelve una aproximación lineal o cuadrática de (1.4) para obtener un nuevo punto x_{k+1}. Cuando se está lo suficientemente cerca del óptimo x^*, la convergencia está garantizada, pero en caso contrario la sucesión $\{x_k\}$ puede no converger.

Para mejorar lo anterior se pueden utilizar funciones de penalización que se contruyen como combinación lineal de la función objetivo y una función de mérito que mide la factibilidad del punto x, por ejemplo, $f(x) + \mu h(c(x))$, con $h(c(x))$ igual a cero si x satisface $c(x) \leq 0$ y mayor que cero en caso contrario.

Otro enfoque posible consiste en considerar la minimización de ambas funciones $f(x)$ y $h(c(x))$ por separado utilizando un enfoque multi-objetivo, con énfasis en la minimización de $h(c(x))$ (para garantizar la factibilidad). Un *filtro* $\mathcal{F}$ se define como una colección de puntos $\{(f(x_k), h(c(x_k)))\}_{k=1}^N$ donde ningún elemento *domina* a otro. Un punto $(f(x_k), h(c(x_k)))$ *domina* a un punto $(f(x_l), h(c(x_l)))$ sí y sólo sí

$$f(x_k) \leq f(x_l) \text{ y } h(c(x_k)) \leq h(c(x_l))$$

Observando la figura 1.1 se puede ver que un punto domina a todos aquellos que están *más arriba* siguiendo por el eje de $f(x)$ y *más a la derecha* siguiendo por el eje de $h(c(x))$.

El filtro es una herramienta que permite discriminar puntos x_k antes de ser utilizados en alguna fase interna del algoritmo de minimización subyacente. Sólo se escogen los puntos que mejoran la función objetivo o la función de mérito, comparando con la colección de puntos que ya se encuentran almacenados en el filtro.

La idea central de los métodos de filtro fue introducida por primera vez en [FL02] por Fletcher y Leyffer, donde se utilizó como herramienta para probar la convergencia global de algoritmos de resolución para (1.4), pero prontamente fue utilizada en distintas ideas para programación no lineal. Por ejemplo, se establecieron resultados de convergencia global para métodos de filtro en SQP utilizando una región de confianza ([FGL$^+$02, FLT02]), convergencia local para métodos de filtro particulares en SQP con

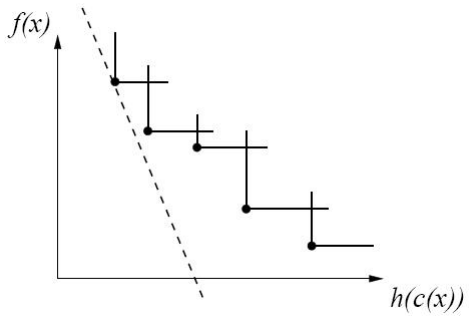

Figura 1.1: Ejemplo de un filtro asociado a una función de mérito ([FL02])

región de confianza que evitan el efecto Maratos ([Ulb04]), convergencia global y local en algoritmos de filtro en SQP usando búsqueda lineal ([WB05b, WB05a]) y también han sido utilizados con éxito en métodos de punto interior ([BSV02, UUV04]).

En el capítulo Antecedentes se detallan algoritmos que utilizan filtros en reemplazo de los métodos de penalización clásica y se verá su adaptación al algoritmo **Filter-SDP**.

1.1.3. Computación Paralela

Los conceptos de computación paralela o *cálculo paralelo* se refieren a la utilización de uno o varios procesadores destinados a resolver una tarea con la esperanza de reducir el tiempo de cómputo original. En este trabajo se estudiaron los principales avances en la introducción de cálculo paralelo dentro de la programación semidefinida, donde se utilizan conceptos bastante interesantes desde el punto de vista tanto teórico como práctico. La principal implementación estudiada fue CSDP ([Bor99]), debido a su versatilidad y a que su código es abierto.

En este trabajo se revisó el enfoque orientado a la memoria distribuída. En el capítulo Antecedentes se explicará con mayor detalle este enfoque y en que partes del algoritmo **Filter-SDP** se utilizará.

1.2. Motivación

La principal motivación para desarrollar este trabajo fue estudiar y experimentar el proceso de desarrollo de una aplicación científica desde su diseño, realizado en MATLAB ©, hasta su implementación en un sistema de cálculo paralelo, realizado en C y MPI.

MATLAB © es uno de los software más populares utilizado para el diseño de algoritmos, sin embargo, trasladar la implementación realizada en ese lenguaje a un sistema de producción o desarrollo, como lo es un sistema de cálculo paralelo, es una tarea difícil.

Típicamente se diseña el algoritmo utilizando MATLAB © y luego manualmente se convierte a C u otro lenguaje para ponerlo en producción. Este proceso es largo y propenso a errores, y a pesar de que existen herramientas de conversión automática de código entre ambos sistemas, la utilización de cálculo paralelo hace aún más compleja esa conversión, por lo cual se debe realizar de forma manual.

Otro elemento que motivó este trabajo fue la investigación de sistemas que utilicen computación paralela para resolver problemas de programación semidefinida. Debido a la reciente expansión de los sistemas de cálculo paralelo dentro del Centro de Modelamiento Matemático de la Universidad de Chile, se optó por investigar un área poco explorada, y realizar un aporte para futuros proyectos e investigaciones realizadas.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo general de este trabajo consiste en estudiar y reimplementar el algoritmo **Filter-SDP** utilizando elementos de la computación paralela. Se revisarán las oportunidades de paralelización del algoritmo y se diseñarán nuevos métodos para fases específicas del algoritmo, en particular, para la fase de restauración. Finalmente se realizarán pruebas numéricas de las nuevas fases diseñadas utilizando una batería de problemas provenientes del área de control de sistemas.

1.3.2. Objetivos Específicos

Los objetivos específicos de este trabajo son los siguientes:

1. Estudio de herramientas de cálculo paralelo aplicables en el algoritmo **Filter-SDP**.
2. Estudio del algoritmo **Filter-SDP** y reimplementación utilizando C y MPI.
3. Diseño e implementación de nuevas fases de restauración para el algoritmo **Filter-SDP**.
4. Realización de pruebas numéricas para las nuevas fases de restauración utilizando la batería de problemas *COMPleib* [Lei04].

Adicionalmente, se plantea como objetivo secundario generar guías de uso y manuales sobre la instalación y utilización de distintos sistemas de cálculo paralelo asociados a la programación semidefinida y el álgebra lineal.

Capítulo 2

Antecedentes

2.1. Programación semidefinida

La programación semidefinida ha unificado varios problemas en programación matemática, los cuales pueden ser expresados como casos particulares de SDP. También ha permitido modelar problemas aplicados provenientes de diversas áreas de la ingeniería. En esta sección se revisarán primero sus principales aplicaciones, luego algunas nociones básicas sobre dualidad y posteriormente dos algoritmos de resolución para SDP: punto interior y paquete espectral, debido a que presentan implementaciones que utilizan computación paralela, lo cual se estudiará en la sección 2.4.

2.1.1. Aplicaciones

Programación lineal

El problema de **programación lineal** con $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ y $A = [a_1|\dots|a_n] \in \mathbb{R}^{m\times n}$ se escribe de la forma

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & Ax + b \geq 0 \end{array} \tag{2.1}$$

y utilizando la siguiente propiedad

Propiedad 2.1.1. *Para $v \in \mathbb{R}^n$, $v \geq 0$ sí y sólo sí diag$(v) = \begin{bmatrix} v_1 & & \\ & \ddots & \\ & & v_n \end{bmatrix} \succeq 0$.*

el problema se puede formular como un problema SDP:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & \text{diag}(Ax + b) \succeq 0 \end{array} \tag{2.2}$$

o equivalentemente

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & c^{\mathrm{T}}x \\ \text{s.a} & \underbrace{\operatorname{diag}(b)}_{F_0} + \sum_{i=1}^{n} x_i \underbrace{\operatorname{diag}(a_i)}_{F_i} \succeq 0 \end{array} \tag{2.3}$$

Programación cuadrática

Los problemas de **programación cuadrática** también se pueden expresar como un SDP. Para $c \in \mathbb{R}^n, b, g \in \mathbb{R}^m, d \in \mathbb{R}, A, H \in \mathbb{R}^{m\times n}$, con $H = [h_1|\dots|h_n]$, el problema se escribe

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & (Ax+b)^{\mathrm{T}}(Ax+b) - c^{\mathrm{T}}x - d \\ \text{s.a} & Hx + g \geq 0 \end{array} \tag{2.4}$$

acotando la función objetivo por una variable auxiliar t se obtiene

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n, t\in\mathbb{R}} & t \\ \text{s.a} & (Ax+b)^{\mathrm{T}}(Ax+b) - c^{\mathrm{T}}x - d \leq t \\ & Hx + g \geq 0 \end{array} \tag{2.5}$$

y utilizando la siguiente propiedad

Propiedad 2.1.2. *Para* $U = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$ *con* A, C *simétricas y* $A \succ 0$*, se tiene que* $U \succeq 0$ *sí y sólo sí* $C - B^TA^{-1}B \succeq 0$*. A la matríz* $C - B^TA^{-1}B$ *se le llama complemento de Schur de* A *en* U*.*

el problema se puede formular como un problema SDP:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n, t\in\mathbb{R}} & t \\ \text{s.a} & \begin{bmatrix} I_{m\times m} & Ax+b \\ (Ax+b)^{\mathrm{T}} & c^{\mathrm{T}}x + d + t \end{bmatrix} \succeq 0 \\ & \operatorname{diag}(Hx+g) \succeq 0 \end{array} \tag{2.6}$$

Para agrupar las dos restricciones semidefinidas se utiliza la siguiente propiedad

Propiedad 2.1.3 (Criterio de Sylvester)**.** $A \succeq 0$ *sí y sólo sí todo* menor principal *(diagonal) de* A *(submatríz que se obtiene como resultado de eliminar filas de índices* I *y columnas de índices* J *a la matríz* A*, con* $I = J$*) es semidefinido positivo.*

con lo cual el problema queda de la forma

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n, t\in\mathbb{R}} & t \\ \text{s.a} & \begin{bmatrix} I_{m\times m} & Ax+b & \\ (Ax+b)^{\mathrm{T}} & c^{\mathrm{T}}x + d + t & \\ & & \operatorname{diag}(Hx+g) \end{bmatrix} \succeq 0 \end{array} \tag{2.7}$$

En este caso,

$$\begin{aligned} F_0 &= \begin{bmatrix} I_{m\times m} & b & \\ b^{\mathrm{T}} & d & \\ & & \operatorname{diag}(g) \end{bmatrix} \\ F_1 &= \begin{bmatrix} 0_{m\times m} & 0_{m\times 1} & \\ 0_{1\times m} & 1 & \\ & & 0_{m\times m} \end{bmatrix} \\ F_{i+1} &= \begin{bmatrix} 0_{m\times m} & a_i & \\ a_i^{\mathrm{T}} & c_i & \\ & & \operatorname{diag}(h_i) \end{bmatrix}, i = 1, \ldots, n \end{aligned}$$

donde F_0 es la matríz constante, F_1 está asociada a la variable t y F_{i+1} está asociada a x_i, con $i = 1, \ldots, n$.

Programación de conos de segundo orden

Otro ejemplo se refiere a la **programación de conos de segundo orden**. Para $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}$, el problema se puede escribir de la forma

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{n+1}}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & Ax = b \\ & \|\overline{x}\| \leq x_0 \end{array} \tag{2.8}$$

donde $\overline{x} = (x_1, \ldots, x_n)$ y $x = (x_0, \overline{x})$. Usando la siguiente propiedad

Propiedad 2.1.4. *$\|\overline{x}\| \leq x_0$ sí y sólo sí* $\begin{bmatrix} x_0 & \overline{x}^T \\ \overline{x} & x_0 I_{n\times n} \end{bmatrix} \succeq 0$.

el problema se puede formular como un problema SDP:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{n+1}}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & Ax = b \\ & \begin{bmatrix} x_0 & \overline{x}^{\mathrm{T}} \\ \overline{x} & x_0 I_{n\times n} \end{bmatrix} \succeq 0 \end{array} \tag{2.9}$$

Transformando la restricción $Ax = b$ en $\begin{bmatrix} -A \\ A \end{bmatrix} x + \begin{pmatrix} b \\ -b \end{pmatrix} \geq 0$, el problema queda de la forma

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{n+1}}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & \begin{bmatrix} -A \\ A \end{bmatrix} x + \begin{pmatrix} b \\ -b \end{pmatrix} \geq 0 \\ & \begin{bmatrix} x_0 & \overline{x}^{\mathrm{T}} \\ \overline{x} & x_0 I_{n\times n} \end{bmatrix} \succeq 0 \end{array} \tag{2.10}$$

o equivalentemente

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{n+1}}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & \text{diag}\left(\begin{bmatrix} -A \\ A \end{bmatrix} x + \begin{pmatrix} b \\ -b \end{pmatrix}\right) \succeq 0 \\ & \begin{bmatrix} x_0 & \overline{x}^{\mathrm{T}} \\ \overline{x} & x_0 I_{n\times n} \end{bmatrix} \succeq 0 \end{array} \tag{2.11}$$

y utilizando la propiedad 2.1.3 se tiene que

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{n+1}}{\text{mín}} & c^{\mathrm{T}}x \\ \text{s.a} & \begin{bmatrix} \text{diag}\left(\begin{bmatrix} -A \\ A \end{bmatrix} x + \begin{pmatrix} b \\ -b \end{pmatrix}\right) & \\ & \begin{matrix} x_0 & \overline{x}^{\mathrm{T}} \\ \overline{x} & x_0 I_{n\times n} \end{matrix} \end{bmatrix} \succeq 0 \end{array} \tag{2.12}$$

Y en este caso,

$$\begin{aligned} F_0 &= \begin{bmatrix} \text{diag}\begin{pmatrix} b \\ -b \end{pmatrix} & \\ & 0_{n+1\times n+1} \end{bmatrix} \\ F_1 &= \begin{bmatrix} \text{diag}\begin{pmatrix} -a_1 \\ a_1 \end{pmatrix} & \\ & \begin{matrix} 1 & 0_{1\times n} \\ 0_{n\times 1} & I_{n\times n} \end{matrix} \end{bmatrix} \\ F_i &= \begin{bmatrix} \text{diag}\begin{pmatrix} -a_{i+1} \\ a_{i+1} \end{pmatrix} & \\ & \begin{matrix} 0 & e_i^{\mathrm{T}} \\ e_i & 0_{n\times n} \end{matrix} \end{bmatrix}, \; i = 1, \ldots, n \end{aligned}$$

con $e_i \in \mathbb{R}^n$ el vector de ceros con un 1 en la posición i-ésima, y donde F_0 es la matríz constante, F_1 está asociada a la variable x_0 y F_{i+1} está asociada a x_i, con $i = 1, \ldots, n$.

Optimización combinatorial y no-convexa

El problema de asignación cuadrática para variables 1 y -1 se puede plantear de la forma:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & x^{\mathrm{T}}Ax + 2b^{\mathrm{T}}x \\ \text{s.a} & x_i \in \{-1, 1\},\; i = 1, \ldots, n \end{array} \tag{2.13}$$

Este problema es NP-duro, por lo tanto la obtención de cotas inferiores para su valor óptimo es de utilidad para métodos de tipo ramificación y acotamiento. Utilizando programación semidefinida se puede construir una relajación que entrega como resultado una cota inferior de (2.13).

La restricción $x_i \in \{-1, 1\}$ se puede escribir como $x_i^2 = 1$ e introduciendo la matríz de variables $X = X^{\mathrm{T}} = xx^{\mathrm{T}} \in \mathbb{S}^n$ de tal manera que

$$x^{\mathrm{T}}Ax = \mathrm{Tr}(XA)$$

el problema (2.13) se puede formular de la siguiente manera

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & \mathrm{Tr}(XA) + 2b^{\mathrm{T}}x \\ \text{s.a} & X_{ii} = 1,\ i = 1, \ldots, n \\ & X = xx^{\mathrm{T}} \end{array} \tag{2.14}$$

El problema anterior posee la restricción $X = xx^{\mathrm{T}}$ la cual se puede relajar considerando a la matríz X como una variable independiente de x e imponiendo $X \succeq xx^{\mathrm{T}}$. Bajo esa suposición podemos utilizar el complemento de Schur visto en la propiedad 2.1.2 considerando $A = 1$, $B = x$ y $C = X$, con lo cual el problema se plantea de la forma

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n, X\in\mathbb{S}^n} & \mathrm{Tr}(XA) + 2b^{\mathrm{T}}x \\ \text{s.a} & X_{ii} = 1,\ i = 1, \ldots, n \\ & \begin{bmatrix} 1 & x^{\mathrm{T}} \\ x & X \end{bmatrix} \succeq 0 \end{array} \tag{2.15}$$

Acotando la función objetivo por una variable auxiliar, transformando la restricción de igualdad en una restricción semidefinida y juntando todas las restricciones en una sola gran restricción semidefinida se obtiene la formulación SDP que entrega cotas inferiores para el problema (2.13).

Se puede generalizar el resultado visto para el problema de asignación cuadrática, de la siguiente forma. Supongamos que el problema a resolver es el siguiente:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & x^{\mathrm{T}}A_0x + 2b_0^{\mathrm{T}}x + c_0 \\ \text{s.a} & x^{\mathrm{T}}A_ix + 2b_i^{\mathrm{T}}x + c_i \leq 0,\ i = 1, \ldots, n \end{array} \tag{2.16}$$

donde las matrices $A_i, i = 0, \ldots, n$ no necesariamente son definidas positivas. Este problema es no convexo y NP-duro, e incluye a los problemas de optimización con función objetivo polinomial y restricciones polinomiales. El problema (2.16) se puede escribir como un SDP:

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & \mathrm{Tr}(XA_0) + 2b_0^{\mathrm{T}}x + c_0 \\ \text{s.a} & \mathrm{Tr}(XA_i) + 2b_i^{\mathrm{T}}x + c_i \leq 0,\ i = 1, \ldots, n \\ & X = xx^T \end{array} \tag{2.17}$$

y realizando la misma relajación del problema (2.15), el problema (2.16) relajado que entrega cotas inferiores queda de la forma

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n, X\in\mathbb{S}^n} & \mathrm{Tr}(XA_0) + 2b_0^{\mathrm{T}}x + c_0 \\ \text{s.a} & \mathrm{Tr}(XA_i) + 2b_i^{\mathrm{T}}x + c_i \leq 0,\ i = 1, \ldots, n \\ & X \succeq xx^T \end{array} \tag{2.18}$$

o equivalentemente:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n, X\in\mathbb{S}^n}{\text{mín}} & \text{Tr}(XA_0) + 2b_0^{\text{T}}x + c_0 \\ \text{s.a} & \text{Tr}(XA_i) + 2b_i^{\text{T}}x + c_i \leq 0,\ i = 1,\ldots,n \\ & \begin{bmatrix} X & x \\ x^{\text{T}} & 1 \end{bmatrix} \succeq 0 \end{array} \tag{2.19}$$

Valores propios

Diversos problemas asociados a la minimización o maximización de funciones de valores propios de matrices simétricas se pueden representar como SDP's.

Por ejemplo, para minimizar el máximo valor propio de una matríz $X \in \mathbb{S}^n$, $\lambda_{\text{máx}}(X)$, primero consideremos la siguiente restricción con $t \in \mathbb{R}$:

$$tI_n - X \succeq 0$$

Los valores propios de la matríz $tI_n - X$ serán exactamente t menos los valores propios de X y al imponer que $tI_n - X \succeq 0$ se tiene que t mayora a todos los valores propios de X y además $t - \lambda_{\text{máx}}(x) \geq 0$, luego, el problema que minimiza el máximo valor propio de X es

$$\begin{array}{ll} \underset{t\in\mathbb{R}}{\text{mín}} & t \\ \text{s.a} & tI_n - X \succeq 0 \end{array} \tag{2.20}$$

Análogamente, si se desea maximizar el valor propio mínimo, la formulación es la siguiente

$$\begin{array}{ll} \underset{t\in\mathbb{R}}{\text{máx}} & t \\ \text{s.a} & X - tI_n \succeq 0 \end{array} \tag{2.21}$$

Un resultado similar se obtiene para un caso particular de valores propios generalizados, aquellos valores $\lambda \in \mathbb{R}$ que satisfacen $Ax = \lambda Bx$, donde $A, B \in \mathbb{S}^n$ y $B \succ 0$. Utilizando la raíz cuadrada de B, de la forma $B = B^{1/2}B^{1/2}$ y resolviendo el polinomio carácteristico de $\lambda B - A$ se obtiene

$$\begin{aligned} det(\lambda B - A) = 0 &\leftrightarrow det(B^{1/2}(\lambda I_n - B^{-1/2}AB^{-1/2})B^{1/2}) = 0 \\ &\leftrightarrow det(B^{1/2})det(\lambda I_n - B^{-1/2}AB^{-1/2})det(B^{1/2}) = 0 \\ &\leftrightarrow det(\lambda I_n - B^{-1/2}AB^{-1/2}) = 0 \end{aligned}$$

Con lo anterior, se tiene que los valores propios generalizados de $Ax = \lambda Bx$ corresponden exactamente a los valores propios de la matríz $B^{-1/2}AB^{-1/2}$ y utilizando las formulaciones (2.20) y (2.21) se puede minimizar el valor propio máximo o maximizar el valor propio mínimo.

Si se tiene que

$$\lambda_1(X) \geq \lambda_2(X) \geq \ldots \lambda_n(X)$$

la suma de los primeros k valores propios de una matríz $X \in \mathbb{S}^n$, $S_k(X) = \sum_{i=1}^{k} \lambda_i(X)$, se puede representar como una restricción semidefinida positiva para las variables $Z \in \mathbb{S}^n$ y $s \in \mathbb{R}$:

$$\begin{aligned} t - ks - \text{Tr}(Z) &\geq 0 && (2.22)\\ Z &\succeq 0 && (2.23)\\ Z - X + sI_n &\succeq 0 && (2.24) \end{aligned}$$

donde $t \in \mathbb{R}$ acota a la suma de la forma $S_k(X) \leq t$. Lo anterior se demuestra con la siguiente propiedad:

Propiedad 2.1.5. *Dado un par* $(X, t) \in \mathbb{S}^n \times \mathbb{R}$*, entonces existen* $(Z, s) \in \mathbb{S}^n \times \mathbb{R}$ *solución de* (2.22)-(2.23)-(2.24) *sí y sólo sí* $S_k(X) \leq t$*.*

Suponiendo que (X, t, Z, s) es solución de (2.22)-(2.23)-(2.24), se tiene que $X \preceq Z + sI_n$ por (2.24). Recordando que el vector de valores propios $\lambda(X) = (\lambda_1(X), \ldots, \lambda_n(X))^{\text{T}}$ es $\succeq$-monótono, es decir, si $X, X' \in \mathbb{S}^n$, entonces $X \succeq X' \rightarrow \lambda(X) \geq \lambda(X')$, se obtiene

$$\begin{aligned} \lambda(X) &\leq \lambda(Z + sI_n)\\ &= \lambda(Z) + s \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \end{aligned}$$

y por lo tanto

$$S_k(X) \leq S_k(Z) + sk$$

Como $Z \succeq 0$ por (2.23), se tiene que $S_k(X) \leq \text{Tr}(Z) + sk$ y por (2.22) se obtiene $S_k(X) \leq t$. Ahora, suponiendo $S_k(X) \leq t$, sea $s = \lambda_k(X)$. Entonces la matríz $X - sI_n$ tendrá a los mayores k valores propios no negativos y a los $n - k$ restantes no positivos, es decir, $\lambda_i(X - sI_n) \geq 0$ para $i \leq k$ y $\lambda_i(X - sI_n) \leq 0$ para $i > k$. Si $X = EDE^{-1}$, sea $Z \in \mathbb{S}^n$ de la forma

$$Z = E \begin{pmatrix} \lambda_1(X - sI_n) & & & & & \\ & \ddots & & & & \\ & & \lambda_k(X - sI_n) & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} E^{-1}$$

Con lo anterior, Z y s satisfacen (2.23) y (2.24) pues por construcción $Z \succeq 0$ y $Z - X + sI_n \succeq 0$. Además se satisface (2.22) pues $\text{Tr}(Z) = S_k(X) - sk$ luego $t - sk - \text{Tr}(Z) = t - S_k(X) \geq 0$.

2.1.2. Dualidad

El problema dual de (1.3) es el siguiente:

$$\begin{array}{ll} \underset{Z\in\mathbb{S}^n}{\text{máx}} & -\text{Tr}(F_0 Z) \\ \text{s.a} & Tr(F_i Z) = c_i,\ i = 1, ..., m \\ & Z \succeq 0 \end{array} \tag{2.25}$$

Para ejemplificar, recordemos que la formulación SDP para el problema de programación lineal (2.1) es:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^m}{\text{mín}} & c^{\text{T}} x \\ \text{s.a} & \text{diag}(b) + \sum_{i=1}^m x_i \text{diag}(a_i) \succeq 0 \end{array} \tag{2.26}$$

Luego, el problema dual es de la forma:

$$\begin{array}{ll} \underset{Z\in\mathbb{S}^n}{\text{máx}} & -\text{Tr}(\text{diag}(b) Z) \\ \text{s.a} & \text{Tr}(\text{diag}(a_i) Z) = c_i,\ i = 1, ..., m \\ & Z \succeq 0 \end{array} \tag{2.27}$$

Como $\text{diag}(a_i)Z$ y $\text{diag}(b)Z$ son matrices diagonales de la forma $\text{diag}(l_i z_{ii} : i = 1, ..., m)$, podemos considerar Z como una matríz diagonal, con lo cual el problema coincide con el dual de la programación lineal:

$$\begin{array}{ll} \underset{z\in\mathbb{R}^m}{\text{máx}} & -b^T z \\ \text{s.a} & a_i^T z = c_i,\ i = 1, ..., m \\ & z \geq 0 \end{array} \tag{2.28}$$

Un concepto importante en programación matemática es la caracterización del *salto de dualidad* o *gap de dualidad* entre x y Z. Sean x primal factible y Z dual factible, el salto de dualidad η se caracteriza de la forma:

$$\begin{aligned} \eta &= c^{\text{T}} x - (-\text{Tr}(F_0 Z)) \\ &= \sum_{i=1}^m c_i x_i + \text{Tr}(F_0 Z) \\ &= \sum_{i=1}^m \text{Tr}(F_i Z) x_i + \text{Tr}(F_0 Z) \\ &= \text{Tr}(F(x) Z) \end{aligned}$$

donde $F(x) = F_0 + \sum_{i=1}^m x_i F_i$. Utilizando la siguiente propiedad:

Propiedad 2.1.6. *Si A, B son matríces simétricas semidefinidas positivas, entonces Tr$(AB) \geq 0$.*

se tiene que $\eta \geq 0$.

En programación lineal los valores optimales del problema primal y dual siempre eran iguales, excepto en el caso donde ambos eran infactibles. En el caso SDP, pueden ocurrir situaciones donde no existan soluciones óptimas aún cuando los valores óptimos sean finitos, o donde ambos valores óptimos son finitos pero el salto de dualidad es mayor que cero.

Dos teoremas importantes asociados a los valores óptimos primal y dual, donde se caracterizan las particularidades de SDP, son los siguientes:

Teorema 2.1.7 (Dualidad débil)**.** *Si p^* y d^* son valores óptimos para el problema primal y dual, se tiene que $p^* \geq d^*$ o equivalentemente el salto de dualidad η satisface $\eta \geq 0$.*

Teorema 2.1.8 (Dualidad fuerte)**.** *Si p^* y d^* son valores óptimos para el problema primal y dual, se tiene que $p^* = d^*$ si se satisfacen:*

1. *$\exists x$ tal que $F(x) \succ 0$*
2. *$\exists Z$ tal que $Z = Z^T \succ 0$ y $Tr(F_i Z) = c_i, i = 1, ..., m$*

Por ejemplo, el siguiente problema no respeta las condiciones para que se cumpla la dualidad fuerte:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^3}{\text{mín}} & x_1 \\ \text{s.a} & \begin{bmatrix} 0 & x_1 & 0 \\ x_1 & x_2 & 0 \\ 0 & 0 & x_1+1 \end{bmatrix} \succeq 0 \end{array} \tag{2.29}$$

El conjunto de soluciones factibles es $\{(x_1, x_2) : x_1 = 0, x_2 \geq 0\}$, por lo tanto $p^* = 0$. El dual es el siguiente:

$$\begin{array}{ll} \underset{Z\in\mathbb{S}^3}{\text{máx}} & -z_{33} \\ \text{s.a} & z_{21} + z_{12} + z_{33} = 1 \\ & z_{22} = 0 \\ & Z \succeq 0 \end{array} \tag{2.30}$$

o equivalentemente:

$$\begin{array}{ll} \underset{Z\in\mathbb{S}^3}{\text{máx}} & -z_{33} \\ \text{s.a} & \underbrace{\begin{bmatrix} z_{11} & \frac{1-z_{33}}{2} & z_{13} \\ \frac{1-z_{33}}{2} & 0 & z_{23} \\ z_{31} & z_{32} & z_{33} \end{bmatrix}}_{Z} \succeq 0 \end{array} \tag{2.31}$$

La restricción anterior es equivalente a que los menores principales (diagonales) sean semidefinidos positivos, es decir:

$$\begin{array}{lll} \underset{Z\in\mathbb{S}^3}{\text{máx}} & -z_{33} & \\ \text{s.a} & \left(\frac{1-z_{33}}{2}\right)^2 & \geq 0 \\ & z_{11}z_{33} - z_{13}z_{31} & \geq 0 \\ & z_{23}z_{32} & \geq 0 \end{array} \tag{2.32}$$

Como Z es simétrica:

$$\begin{array}{lll} \underset{Z\in\mathbb{S}^3}{\max} & -z_{33} & \\ \text{s.a} & \left(\frac{1-z_{33}}{2}\right)^2 & \geq 0 \\ & z_{11}z_{33} - z_{13}^2 & \geq 0 \\ & z_{23}^2 & \geq 0 \end{array} \tag{2.33}$$

luego $d^* = 1$, pues $z_{33} = -1$ maximiza la función objetivo respetando la restricción $\left(\frac{1-z_{33}}{2}\right)^2 \geq 0$. En este caso $p^* \neq d^*$, y esto ocurre pues no se cumplen las condiciones de dualidad fuerte.

Suponiendo que existen x, Z factibles tales que $c^{\mathrm{T}}x = p^* = -\mathrm{Tr}(F_0 Z) = d^*$, el salto de dualidad en este caso es $\eta = 0$, luego, $\mathrm{Tr}(F(x)Z) = 0$. Recordando que $F(x) \succeq 0$ y $Z \succeq 0$, se tiene que

$$F(x)Z = 0$$

Este resultado se denomina *condición de holgura complementaria*.

Para caracterizar los puntos óptimales del problema primal y dual, se tiene el siguiente resultado:

Proposición 2.1.9. *El punto $x \in \mathbb{R}^n$ es primal-óptimo sí y sólo sí existe $Z \in \mathbb{S}^n$ tal que*

$$\begin{aligned} F(x) &\succeq 0 \\ Z &\succeq 0 \\ \mathrm{Tr}(F_i Z) &= c_i,\ i = 1, ..., m \\ F(x)Z &= 0 \end{aligned}$$

2.1.3. Algoritmos de resolución

Punto interior

Los métodos de punto interior se basan en las condiciones optimales descritas en el teorema (2.1.9), reemplazando la condición de holgura complementaria por una condición perturbada. Usando una variable de holgura $S \in \mathbb{S}^n$, las condiciones de optimalidad de la proposición 2.1.9 se pueden escribir de la forma

$$\begin{aligned} F(x) + S = 0 \qquad & S \succeq 0 \\ \mathrm{Tr}(F_i Z) = c_i,\ i = 1, ..., m \qquad & Z \succeq 0 \\ SZ = 0 & \end{aligned}$$

Al suponer que se satisfacen las condiciones de dualidad fuerte (factibilidad estricta), el siguiente sistema perturbado tiene solución única para cada $\mu > 0$:

$$\begin{aligned} F(x) + S = 0 \qquad & S \succ 0 \\ \mathrm{Tr}(F_i Z) = c_i,\ i = 1, ..., m \qquad & Z \succ 0 \\ SZ = \mu I_n & \end{aligned}$$

Es posible probar que el conjunto $\{(x_\mu, S_\mu, Z_\mu) : \mu > 0\}$ define una curva regular parametrizada por μ, la cual usualmente se denomina *camino central*.

El objetivo es calcular $(\Delta x, \Delta S, \Delta Z)$ de manera que $S + \Delta S \succeq 0$ y $Z + \Delta Z \succeq 0$. Ignorando las restricciones $S, Z \succ 0$ y denotando

$$F^*(Z) = \begin{pmatrix} \text{Tr}(F_1 Z) \\ \vdots \\ \text{Tr}(F_n Z) \end{pmatrix}$$

, para obtener una solución del sistema perturbado se debe resolver el sistema

$$T_\mu(x, S, Z) := \begin{pmatrix} F(x) + S \\ F^*(Z) - c \\ SZ - \mu I_n \end{pmatrix} = 0 \tag{2.34}$$

Utilizando el método de Newton-Raphson se construye el sistema

$$J_{T_\mu}(x, S, Z)(\Delta x, \Delta S, \Delta Z) = -T_\mu(x, S, Z) \tag{2.35}$$

donde

$$J_{T_\mu}(x, S, Z)(\Delta x, \Delta S, \Delta Z) := \begin{pmatrix} \sum_{i=1}^n \Delta x_i F_i + \Delta S \\ F^*(\Delta Z) \\ \Delta S Z + S \Delta Z \end{pmatrix} \tag{2.36}$$

con lo cual el sistema (2.35) queda de la forma:

$$\begin{pmatrix} \sum_{i=1}^n \Delta x_i F_i + \Delta S \\ F^*(\Delta Z) \\ \Delta S Z + S \Delta Z \end{pmatrix} = \begin{pmatrix} -F(x) - S \\ c - F^*(Z) \\ \mu I_n - SZ \end{pmatrix} \tag{2.37}$$

En este sistema, se requiere que las matrices ΔS y ΔZ sean simétricas, pues se utilizaran como direcciones de descenso. En el caso de ΔS, por la primera ecuación de (2.37), se tiene la simetría, pues $\sum_{i=1}^m \Delta x_i F_i + F(x) + S$ es simétrica. El caso de ΔZ no es directo. Para solucionar lo anterior, en [Zha98] se propone realizar una simetrización, cambiando la tercera ecuación de (2.37) por

$$H_P(\Delta S Z + S \Delta Z) = \mu I_n - H_P(SZ)$$

donde H_P es una transformación lineal definida de la forma

$$H_P(N) = \frac{1}{2}\left(PNP^{-1} + (PNP^{-1})^{\text{T}}\right) \tag{2.38}$$

para una matríz $N \in \mathbb{R}^{n \times n}$ cualquiera y una matríz no singular $P \in \mathbb{R}^{n \times n}$. Las elecciones más populares para la matríz P son 3:

- *AHO*: $P = I$ ([AHO96])

- *HKM*: $P = Z^{1/2}$ ([HRVW96, KSH97, Mon97])
- *NT*: $P = \left(X^{-1/2}(X^{1/2}ZX^{1/2})^{1/2}X^{-1/2}\right)^{1/2}$ ([NT97])

En resumen, la dirección $(\Delta x, \Delta S, \Delta Z)$ en el punto (x, S, Z) satisface

$$\begin{array}{rcl} \sum_{i=1}^m \Delta x_i F_i + \Delta S & = & R \\ \mathrm{Tr}(F_i \Delta Z) & = & r_i,\ i = 1, \ldots, m \\ H_P(\Delta S Z) + H_P(S \Delta Z) & = & H \end{array} \tag{2.39}$$

donde R y r_i se definen de la forma

$$\begin{aligned} R &= -F_0 - \sum_{i=1}^m x_i F_i - S \\ r_i &= c_i - \mathrm{Tr}(F_i Z),\ i = 1, \ldots, m \\ H &= \mu I_n - H_P(SZ) \end{aligned}$$

y definiendo los operadores lineales $\mathcal{E} : \mathbb{S}^n \to \mathbb{S}^n$ y $\mathcal{F} : \mathbb{S}^n \to \mathbb{S}^n$ como

$$\mathcal{E}(E) = H_P(EZ), \qquad \mathcal{F}(E) = H_P(SE) \tag{2.40}$$

, se puede reescribir el sistema (2.39):

$$\begin{array}{rcl} \sum_{i=1}^m \Delta x_i F_i + \Delta S & = & R \\ \mathrm{Tr}(F_i \Delta Z) & = & r_i,\ i = 1, \ldots, m \\ \mathcal{E}(\Delta S) + \mathcal{F}(\Delta Z) & = & H \end{array} \tag{2.41}$$

Usando el siguiente lema ([MZ99]) se construye el sistema final que debe ser resuelto en cada iteración:

Lema 2.1.10. *Sean $\mathcal{E} : \mathbb{S}^n \to \mathbb{S}^n$ y $\mathcal{F} : \mathbb{S}^n \to \mathbb{S}^n$ operadores lineales definidos en* (2.40) *y tal que $\mathcal{E}$ es invertible y sean $(r, R, H) \in \mathbb{R}^n \times \mathbb{S}^n \times \mathbb{S}^n$ y $F_i \in \mathbb{S}^n, i = 1, \ldots, m$ dados. Sean*

$$\begin{aligned} V_j &= \mathcal{E}^{-1}(\mathcal{F}(F_j)),\ j = 1, \ldots, m && (2.42) \\ V &= \mathcal{E}^{-1}(\mathcal{F}(R) - H) && (2.43) \\ M_{ij} &= \mathit{Tr}(F_i V_j),\ i, j = 1, \ldots, m && (2.44) \\ h_i &= r_i + \mathit{Tr}(F_i V),\ i = 1, \ldots, m && (2.45) \end{aligned}$$

Entonces $(\Delta x, \Delta S, \Delta Z)$ satisface el sistema (2.41) *sí y sólo sí Δx satisface el sistema*

$$M \Delta x = h \tag{2.46}$$

y $(\Delta S, \Delta Z)$ están dados por

$$\Delta S = \sum_{j=1}^m \Delta x_j V_j - V \tag{2.47}$$

$$\Delta Z = R - \sum_{i=1}^m \Delta x_i F_i \tag{2.48}$$

En particular, si la matríz M es invertible, el sistema (2.41) *tiene solución única.*

Teniendo en consideración este esquema, a continuación se explicará un método primal-dual predictor-corrector desarrollado por Mehrotra en [Meh92]. Este método es de especial importancia pues es el que se encuentra implementado en las herramientas computacionales utilizadas en este trabajo para resolver problemas de programación semidefinida lineal (CSDP, [Bor99]).

En el esquema predictor-corrector se calculan 2 direcciones en cada iteración, una dirección de *escalamiento afín* $(\Delta x^a, \Delta S^a, \Delta Z^a)$ solución del sistema (2.41) con $\mu = 0$, es decir, $H = -H_P(SZ)$. Luego se calcula un parámetro $\sigma \in [0, 1)$ de la forma

$$\sigma := \left[\frac{\text{Tr}\left((Z + \alpha_D^a \Delta Z^a)(S + \alpha_P^a \Delta S^a)\right)}{\text{Tr}(ZS)} \right]^2 \tag{2.49}$$

con

$$\alpha_P^a = \text{mín}\{1, \text{máx}\{\alpha \geq 0 : S + \alpha \Delta S^a \succeq 0\}\} \tag{2.50}$$
$$\alpha_D^a = \text{mín}\{1, \text{máx}\{\alpha \geq 0 : Z + \alpha \Delta Z^a \succeq 0\}\} \tag{2.51}$$

Este parámetro se utiliza para calcular la dirección de *corrección* $(\Delta x^c, \Delta S^c, \Delta Z^c)$ resolviendo el sistema

$$\begin{array}{rcl} \sum_{i=1}^m \Delta x_i^c F_i + \Delta S^c & = & 0 \\ \text{Tr}(F_i \Delta Z^c) & = & 0,\ i = 1, \ldots, m \\ \mathcal{E}(\Delta S^c) + \mathcal{F}(\Delta Z^c) & = & \sigma \mu I_n - H_P(\Delta Z^a \Delta S^a) \end{array} \tag{2.52}$$

con $\mu := \frac{\text{Tr}(ZS)}{n}$. Esta solución es de la forma:

$$\Delta Z^c = -\sum_{i=1}^m \Delta x^c F_i \tag{2.53}$$
$$\Delta S^c = -V^c - \mathcal{E}^{-1}(\mathcal{F}(\Delta Z^c)) \tag{2.54}$$

con $V^c = \mathcal{E}^{-1}(H_P(\Delta S^a \Delta Z^a) - \sigma \mu I)$. Utilizando el lema (2.1.10) y considerando

$$h_i^c = r_i + \text{Tr}(F_i V^c),\ i = 1, \ldots, m \tag{2.55}$$

se deben resolver dos sistemas $M \Delta x^a = h$ y $M \Delta x^c = h^c$ para obtener las direcciones $(\Delta Z^a, \Delta S^a)$ y $(\Delta Z^c, \Delta S^c)$. La dirección final se contruye sumando ambas direcciones y ponderando por escalares de la forma

$$\hat{x} = x + \alpha_P(\Delta x^a + \Delta x^c) \tag{2.56}$$
$$\hat{S} = S + \alpha_P(\Delta S^a + \Delta S^c) \tag{2.57}$$
$$\hat{Z} = Z + \alpha_D(\Delta Z^a + \Delta Z^c) \tag{2.58}$$

con

$$\alpha_P = \text{mín}\{1, \text{máx}\{\alpha \geq 0 : S + \alpha(\Delta S^a + \Delta S^c) \succeq 0\}\} \tag{2.59}$$
$$\alpha_D = \text{mín}\{1, \text{máx}\{\alpha \geq 0 : Z + \alpha(\Delta Z^a + \Delta Z^c) \succeq 0\}\} \tag{2.60}$$

Se repite el proceso para el nuevo punto $(\hat{x}, \hat{S}, \hat{Z})$ tomando como condición de parada $\text{Tr}(SZ) \leq \epsilon$ para ϵ suficientemente pequeño. Todos los pasos del método se pueden ver en el algoritmo 1.

Algoritmo 1 Punto Interior Primal-Dual Predictor-Corrector

1: **(INPUT)** Punto inicial $(0, S_0, Z_0)$, con $S_0, Z_0 \succ 0$; tolerancia $\epsilon > 0$.
2: **(INICIALIZACIÓN)** $k = 0$.
3: **while** $\text{Tr}(Z_k S_k) > \epsilon$ **do**
4: Para $\mu = 0$, construir dirección de escalamiento afín:

- Calcular M y h descritas en (2.44) y (2.45).
- Resolver $M\Delta x^a = h$.
- Calcular ΔZ^a y ΔS^a usando (2.48) y (2.47).

5: Calcular σ y los valores α_P^a y α_D^a usando (2.49), (2.50) y (2.51).
6: Para $\mu = \frac{\text{Tr}(Z_k S_k)}{n}$, construir dirección de corrección:

- Calcular h^c descrita en (2.55).
- Resolver $M\Delta x^c = h^c$.
- Calcular ΔZ^c y ΔS^c usando (2.53) y (2.54).

7: Calcular los valores α_P^c y α_D^c usando las fórmulas (2.50) y (2.51) con ΔS^c y ΔZ^c.
8: Sumar las direcciones de escalamiento afín y de corrección para obtener $(\Delta x, \Delta S, \Delta Z)$.
9: Actualización:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_P \Delta x \\ S_{k+1} &= S_k + \alpha_P \Delta S \\ Z_{k+1} &= Z_k + \alpha_D \Delta Z \end{aligned}$$

con α_P y α_D calculados segun (2.59) y (2.60).
10: $k \leftarrow k + 1$
11: **end while**

Paquete espectral

Otro método desarrollado para la resolución de SDP's trata directamente con el problema (2.25) agregando una restricción sobre la traza de las variables. En esta parte se utilizará la notación definida en [HR00] para conservar la limpieza de los resultados.

Para $C, A_i \in \mathbb{S}^n, i = 1, \ldots, n$ y $b \in \mathbb{R}^m$, el problema primal con traza constante $\text{Tr}(X) = 1$ se escribe de la forma:

$$\begin{array}{ll} \underset{X \in \mathbb{S}^n}{\text{máx}} & \langle C, X \rangle \\ \text{s.a} & \mathcal{A}X = b \\ & \langle X, I \rangle = 1 \\ & X \succeq 0 \end{array} \tag{2.61}$$

donde $\langle C, X \rangle = \text{Tr}(CX)$ y $\mathcal{A}X = \begin{bmatrix} \langle A_1, X \rangle \\ \vdots \\ \langle A_m, X \rangle \end{bmatrix}$. El problema dual asociado es

$$\begin{array}{ll} \underset{Z \in \mathbb{S}^n, y \in \mathbb{R}^m, y_0 \in \mathbb{R}}{\text{mín}} & y_0 + b^{\text{T}} y \\ \text{s.a} & Z = y_0 I + \mathcal{A}^{\text{T}} y - C \\ & Z \succeq 0 \end{array} \tag{2.62}$$

donde $\mathcal{A}^{\text{T}} y = \sum_{i=1}^m y_i A_i$. De la condición de holgura complementaria, las soluciones primal y dual satisfacen $XZ = 0$, por lo tanto conmutan y son simultáneamente diagonalizables, es decir, $X = P \Lambda_X P^{\text{T}}$ y $Z = P \Lambda_Z P^{\text{T}}$. Si se impone que $X \neq 0$, luego,

$$\lambda_{\text{mín}}(\Lambda_Z) = 0 = \lambda_{\text{mín}}(Z)$$

Entonces:

$$\begin{array}{rcl} 0 & = & \lambda_{\text{mín}}(Z) \\ 0 & = & \lambda_{\text{mín}}(y_0 I + \mathcal{A}^{\text{T}} y - C) \\ 0 & = & y_0 + \lambda_{\text{mín}}(\mathcal{A}^{\text{T}} y - C) \\ y_0 & = & -\lambda_{\text{mín}}(\mathcal{A}^{\text{T}} y - C) \\ y_0 & = & \lambda_{\text{máx}}(C - \mathcal{A}^{\text{T}} y) \end{array}$$

El problema dual se puede plantear de forma irrestricta de la siguiente manera:

$$\underset{y \in \mathbb{R}^m}{\text{mín}} \quad \lambda_{\text{máx}}(C - \mathcal{A}^{\text{T}} y) + b^{\text{T}} y \tag{2.63}$$

sin perdida de generalidad, al cambiar los signos de C, A_i y b, se tiene:

$$\underset{y \in \mathbb{R}^m}{\text{mín}} \quad \lambda_{\text{máx}}(\mathcal{A}^{\text{T}} y - C) - b^{\text{T}} y \tag{2.64}$$

La idea a continuación es minimizar la función $f(y) = \lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) - b^{\text{T}}y$ utilizando el método del *paquete proximal*. En la iteración k-ésima de este método, se debe calcular un subgradiente g_k de f evaluado en un punto x_k, es decir, $g_k \in \partial f(y_k)$, el cual se almacena en un *paquete* $G_k = \{g_k\} \cup G_{k-1}$ y se utiliza en la resolución de un subproblema:

$$\min_{y \in \mathbb{R}^m} \underbrace{\left\{ \max_{g \in G^k} f(y_k) + \langle g, y - y_k \rangle \right\}}_{\hat{f}(y_k)} + \frac{\rho^k}{2} \|y - y_k\|^2 \tag{2.65}$$

donde $\rho^k > \rho_{\text{mín}}$, con $\rho_{\text{mín}}$ un parámetro fijo. Las soluciones de (2.65), $\overline{y}$ y $\overline{g}$, se utilizan para actualizar el *paquete* y verificar el criterio de parada $f(y_k) - \hat{f}(\overline{y}) \leq \delta(|f(y_k)| + 1)$ para δ suficientemente pequeño.

El primer paso consiste en calcular un subgradiente de $\lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) - b^{\text{T}}y$. Para ello, conviene usar la siguiente caracterización:

$$\begin{aligned} \lambda_{\text{máx}}(Z) &= \text{máx}\{\langle q, Zq \rangle : \|q\| = 1\} && (2.66) \\ &= \text{máx}\{\langle qq^{\text{T}}, Z \rangle : \|q\| = 1\} && (2.67) \\ &= \text{máx}\{\langle U, Z \rangle : U \in \mathbb{S}^n, \text{Tr}(U) = 1, U \succeq 0\} && (2.68) \end{aligned}$$

La última igualdad se debe al siguiente resultado ([Ove92]):

$$conv\{ww^{\text{T}} : w \in \mathbb{R}^n, \|w\| = 1\} = \{U : U \in \mathbb{S}^n, \text{Tr}(U) = 1, U \succeq 0\} \tag{2.69}$$

donde $conv\{A\}$ representa la envoltura convexa del conjunto A.

El subdiferencial $\partial\lambda_{\text{máx}}(Z)$ se caracteriza de la forma ([HUY95], teorema 3.1):

$$\partial\lambda_{\text{máx}}(Z) = \{W \in \mathbb{S}^n_+ : \text{Tr}(W) = 1, \langle W, Z \rangle = \lambda_{\text{máx}}(Z)\} \tag{2.70}$$

En términos numéricos, puede resultar difícil obtener un subgradiente, por lo cual el ϵ-subdiferencial $\partial_\epsilon\lambda_{\text{máx}}(Z)$ es de mayor utilidad al entregar estabilidad a los cálculos realizados:

$$\partial_\epsilon\lambda_{\text{máx}}(Z) = \{W \in \mathbb{S}^n_+ : \text{Tr}(W) = 1, \langle W, Z \rangle \geq \lambda_{\text{máx}}(Z) - \epsilon\} \tag{2.71}$$

Si ϵ es suficientemente pequeño, el $\epsilon-$subdiferencial aproxima al subdiferencial. Para calcular el $\epsilon-$subdiferencial de $f(y) = \lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) - b^{\text{T}}y$, se procede como en [HR00], usando [HUL96], proposición XI.1.3.1:

$$\begin{aligned} \partial_\epsilon \left(\lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) - b^{\text{T}}y \right) &= \mathcal{A}\left(\partial_\epsilon \lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) \right) - \{b\} \\ &= \{\mathcal{A}W - b : W \in \mathbb{S}^n, \text{Tr}(W) = 1, \\ &\quad \left\langle W, \mathcal{A}^{\text{T}}y - C \right\rangle \geq \lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C) - \epsilon\} \end{aligned}$$

Para calcular un elemento del $\epsilon-$subdiferencial, tomando $v \in \mathbb{R}^n$, $\|v\| = 1$ tal que

$$(\mathcal{A}^{\text{T}}y - C)v = \lambda_{\text{máx}}(\mathcal{A}^{\text{T}}y - C)v$$

se tiene que $W = vv^{\mathrm{T}}$ satisface $vv^{\mathrm{T}} \in \mathbb{S}^n_+$, $\mathrm{Tr}(vv^{\mathrm{T}}) = \|v\| = 1$ y además:

$$\begin{aligned}\left\langle vv^{\mathrm{T}}, \mathcal{A}^{\mathrm{T}}y - C\right\rangle &= \left\langle v, (\mathcal{A}^{\mathrm{T}}y - C)v\right\rangle \\ &= \left\langle v, \lambda_{\max}(\mathcal{A}^{\mathrm{T}}y - C)v\right\rangle \\ &= \lambda_{\max}(\mathcal{A}^{\mathrm{T}}y - C) \\ &\geq \lambda_{\max}(\mathcal{A}^{\mathrm{T}}y - C) - \epsilon\end{aligned}$$

luego, $\mathcal{A}vv^{\mathrm{T}} - b \in \partial_\epsilon \left(\lambda_{\max}(\mathcal{A}^{\mathrm{T}}y - C) - b^{\mathrm{T}}y\right)$.

De esta manera, en cada iteración del método se debe encontrar un par (θ, v), valor propio y vector propio maximal asociado de la matríz $\mathcal{A}^{\mathrm{T}}y_k - C$, con y_k el punto asociado a la iteración k-ésima. Como se utiliza el $\epsilon-$subdiferencial, se desea encontrar un par (θ, v) lo suficientemente *maximales*, es decir, con ϵ suficientemente pequeño. En la práctica, basta con que la pareja (θ, v) estén cercanos a los maximales, y para ello se puede utilizar el método de Lanczos ([Par87]), el cual resuelve precisamente este problema, y con un buen comportamiento numérico si A_i y C son ralas.

Todos los pasos del método se pueden ver en el algoritmo 2.

Algoritmo 2 Paquete Espectral

1: **(INPUT)** Función objetivo $f(y) = \lambda_{\max}(\mathcal{A}^{\mathrm{T}}y - C) - b^{\mathrm{T}}y$; punto inicial y_0; parámetro de mejoramiento $m_L \in (0, 1/2)$; parámetro de término $\delta > 0$; peso $\rho^0 \geq \rho_{min} > 0$.
2: **(INICIALIZACIÓN)** $k = 0$; $g^0 \in \partial f(y_0)$; $G^0 = \{g^0\}$.
3: **while** $k = 0, 1, 2, ...$ **do**
4: **(SUBPROBLEMA)** Resolver

$$\min_{y \in \mathbb{R}^m} \underbrace{\left\{ \max_{g \in G^k} f(y_k) + < g, y - y_k > \right\}}_{\hat{f}(y_k)} + \frac{\rho^k}{2}\|y - y_k\|^2$$

y obtener $\overline{y}$, $\overline{g}$ óptimos.
5: **if** $f(\overline{y}) \leq f(y_k) + m_L(\hat{f}(\overline{y}) - f(y_k))$ **then**
6: $y_{k+1} = \overline{y}$ (paso en serio)
7: **else**
8: $y_{k+1} = y_k$ (paso nulo)
9: **end if**
10: **if** $f(y_k) - \hat{f}(\overline{y}) \leq \delta(|f(y_k)| + 1)$ **then**
11: **(OUTPUT)** Retornar $\tilde{y} = y_k$ como la solución aproximada de $\min\{y \in \mathbb{R}^m : f(y)\}$. **PARAR**.
12: **end if**
13: **(UPDATE)** Agregar $\overline{g}$ y un nuevo elemento g de $\partial f(\overline{y})$ a G^k para obtener G^{k+1}. Actualizar ρ^k para obtener $\rho^{k+1} > \rho_{min}$. $k \leftarrow k + 1$.
14: **end while**

2.2. Métodos de Filtro

Como se mencionó en el capítulo introductorio, los métodos de filtro tiene por objetivo realizar una minimización multi-objetivo de la función f (función objetivo) y de una función de mérito h (medida de la infactibilidad). En esta sección se describe formalmente el concepto de *filtro* y se verá una aplicación a la programación no lineal, utilizando como referencia el artículo de Fletcher, Leyffer y Toint ([FLT02]).

Primero algunas definiciones básicas:

Definición 2.2.1 (Dominancia)**.** *Un punto* $(f(x_k), h(c(x_k)))$ domina *a un punto* $(f(x_l), h(c(x_l)))$ *si se satisface*

$$f(x_k) \leq f(x_l) \text{ y } h(c(x_k)) \leq h(c(x_l))$$

Definición 2.2.2 (Filtro)**.** *Un filtro* $\mathcal{F}$ *se define como una colección de puntos*

$$\{(f(x_k), h(c(x_k)))\}_{k=1}^{N}$$

, con $N \geq 1$*, donde ningún elemento domina a otro.*

Definición 2.2.3 (Aceptabilidad)**.** *Un punto* $\overline{x}$ *se dice* aceptable *por un filtro* $\mathcal{F} = \{(f_j, h_j)\}_{j=1}^{N}$ *si para el par* $(f, h) := (f(\overline{x}), h(c(\overline{x})))$ *se satisface para todo* $j = 1, \ldots, N$*:*

$$h \leq \beta h_j \text{ ó } f + \gamma h \leq f_j \tag{2.72}$$

donde $0 < \gamma < \beta < 1$*. El par* (f, h) *también se dice* aceptable *por el filtro* $\mathcal{F}$*.*

Definición 2.2.4 (Actualización del filtro)**.** *Sea* (f, h) *aceptable por un filtro* $\mathcal{F}$*. Si se agrega* (f, h) *al filtro, se dice que se realiza una* actualización del filtro $\mathcal{F}$ *al eliminar de él todos los pares* (f_j, h_j) *que son dominados por el nuevo elemento* (f, h)*.*

Como primer resultado, se presenta el lema fundamental y su corolario aplicado al filtro, cuyas demostraciones se pueden revisar en [FLT02], asociado a la convergencia de la función de mérito en un filtro:

Lema 2.2.5. *Sean las sucesiones* $\{h_k\}$ *y* $\{f_k\}$ *tales que* $h_k \geq 0$ *y* f_k *es monótona decreciente e acotada inferiormente. Sean* β *y* γ *tales que* $0 < \gamma < \beta < 1$*. Si para todo* k *se satisface*

$$h_{k+1} \leq \beta h_k \text{ ó } f_k - f_{k+1} \geq \gamma h_{k+1} \tag{2.73}$$

, entonces $h_k \to 0$*.*

Corolario 2.2.6. *Sea* $\{(h_k, f_k)\}_{k \in \mathbb{N}}$ *una sucesión de pares añadidos a un filtro* $\mathcal{F}$*, donde* $h_k > 0$ *y* $\{f_k\}$ *esta inferiormente acotada. Entonces* $h_k \to 0$*.*

El corolario anterior garantiza la convergencia de la función de mérito a cero lo que significa que para una suceción de puntos $\{x_k\}_{k\in\mathbb{N}}$ aceptables por un filtro, en el límite se satisface la factibilidad del problema (2.74).

Para ejemplificar la utilización de estos métodos, en esta sección se revisará su aplicación en el algoritmo *Filter-SQP* descrito en [FLT02], el cual resuelve el problema de programación no lineal con restricciones de igualdad y desigualdad:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & f(x) \\ \text{s.a} & c_i(x) = 0,\ i \in \mathcal{E} \\ & c_i(x) \leq 0,\ i \in \mathcal{I} \end{array} \tag{2.74}$$

donde $\mathcal{E}$ y $\mathcal{I}$ son los índices de las restricciones respectivas. Este algoritmo sirve como base para el desarrollo del algoritmo **Filter-SDP**, el cual se explicará en detalle en la sección 2.3.

Primero conviene definir el problema cuadrático QP(x_k, ρ):

$$\begin{array}{ll} \underset{d\in\mathbb{R}^n}{\text{mín}} & q(d) := \nabla f(x_k)^{\mathrm{T}} d + \frac{1}{2} d^{\mathrm{T}} B_k d \\ \text{s.a} & c_i(x_k) + \nabla c_i(x_k)^{\mathrm{T}} d = 0,\ i \in \mathcal{E} \\ & c_i(x_k) + \nabla c_i(x_k)^{\mathrm{T}} d \leq 0,\ i \in \mathcal{I} \\ & \|d\|_\infty \leq \rho \end{array} \tag{2.75}$$

donde $B_k \in \mathbb{S}^n$ y la función de mérito $h(c(x))$:

$$h(c(x)) = \sum_{i\in\mathcal{I}} \text{máx}\{0, c_i(x)\} + \sum_{i\in\mathcal{E}} c_i^2(x) \tag{2.76}$$

La idea principal consiste en resolver en cada iteración el problema QP(x_k, ρ) para $\rho > 0$, obteniendo una solución $d_k \in \mathbb{R}^n$ o verificando que no hay factibilidad. En caso de no haber factibilidad se ingresa el punto $(h(c(x_k)), f(x_k))$ al filtro $\mathcal{F}$, se actualiza el filtro $\mathcal{F}$ y se inicia una nueva iteración. En caso de existir solución, se verifica si $d_k = 0$, en ese caso se satisfacen las condiciones KKT para el punto x_k, por lo tanto se reporta esa solución y se detiene el algoritmo. Si $d_k \neq 0$, se *ajusta* esa solución hasta que se satisfagan condiciones que garantizan la convergencia y además $x_k + d_k$ sea aceptable para el filtro $\mathcal{F}$.

Previo a la resolución de QP(x_k, ρ), se debe verificar que el punto x_k sea aceptable por el filtro $\mathcal{F}$, por lo tanto se tienen 2 condiciones fundamentales para el éxito del algoritmo:

- x_k es aceptable para $\mathcal{F}$
- QP(x_k, ρ) es factible para algún $\rho \geq \rho_o$

El proceso para obtener un punto x_k que satisface las restricciones anteriores se denomina *fase de restauración* y tiene un papel fundamental en el rendimiento del algoritmo (implementación y desempeño) así como su convergencia.

Para simplificar la notación, se denota:

$$\begin{aligned} \Delta q_k &:= q(0) - q(d_k) = -\nabla f(x_k)^{\mathrm{T}} d_k - \frac{1}{2} d_k^{\mathrm{T}} B_k d_k \\ \Delta f_k &:= f(x_k) - f(x_k + d_k) \end{aligned}$$

El algoritmo 3 describe los pasos del algoritmo *Filter-SQP*.

La convergencia del algoritmo se demuestra en el siguiente teorema, cuya demostración se detalla en [FLT02]:

Teorema 2.2.7 (Convergencia Filter-SQP)**.** *Si se cumplen las siguiente suposiciones:*

1. *Todos los puntos x_k obtenidos por el algoritmo pertenecen a un conjunto X no vacío, cerrado y acotado.*
2. *Las funciones $f(x)$ y $c_i(x), i \in \mathcal{I} \cup \mathcal{E}$ son de clase $\mathcal{C}^2$ en un abierto que contiene a X.*
3. *Existe $M > 0$ tal que las matrices B_k satisfacen $\|B_k\|_2 \leq M$ para todo k.*

Entonces una de las siguientes afirmaciones es verdadera:

(A) *La fase de restauración falla al no encontrar un punto x que sea aceptable por el filtro y factible para QP(x, ρ) para algún $\rho \geq \rho_o$.*

(B) *Se encuentra un punto KKT del problema* (2.74) *($d = 0$ resuelve QP(x_k, ρ) para algún k).*

(C) *Existe un punto de acumulación factible para* (2.74) *el cual puede ser un punto KKT o bien no satisfacer las condiciones de Mangasarian-Fromovitz.*

Algoritmo 3 Filter-SQP

(INICIALIZACIÓN) $k \leftarrow 1$, $\mathcal{F}^0 = \{(u, -\infty)\}$, $d_k \leftarrow \infty^n$, $\beta \in (0,1), \gamma \in (0,\beta), u > 0, \sigma \in (0,1), \overline{\rho} > 0$, $\rho_{inicial} > \overline{\rho}$, max_iteraciones > 1.
while $k <$ max_iteraciones **do**
  (FASE RESTAURACIÓN) Encontrar x_k y $\rho_{inicial} \geq \tilde{\rho} > \overline{\rho}$ tales que:

  A1 $(h(c(x_k)), f(x_k))$ es aceptable para $\mathcal{F}^{k-1}$.

  B1 $QP(x_k, \tilde{\rho})$ es factible.

  $\rho \leftarrow \tilde{\rho}$.
  (PROBLEMA TANGENCIAL) Resolver $QP(x_k, \rho)$.
  if $\|d_k\| < +\infty$ ($QP(x_k, \rho)$ es factible) **then**
    if $\|d_k\| < \epsilon$ **then**
      Fin del algoritmo. Solución: x_k.
    end if
    if $(h(c(x_k + d_k)), f(x_k + d_k))$ no es aceptable por $\mathcal{F}^{k-1} \cup \{(h(c(x_k)), f(x_k))\}$ **then**
      $\rho \leftarrow \frac{\rho}{2}$.
      Ir a **PROBLEMA TANGENCIAL**.
    else
      if $\Delta f_k < \sigma \Delta q_k$ y $\Delta q_k > 0$ **then**
        $\rho \leftarrow \frac{\rho}{2}$.
        Ir a **PROBLEMA TANGENCIAL**.
      else
        if $\Delta q_k \geq 0$ **then**
          $\mathcal{F}^k \leftarrow \text{Add}((h(c(x_k)), f(x_k)), \mathcal{F}^{k-1})$ **Iteración tipo** h
        else
          $\mathcal{F}^k \leftarrow \mathcal{F}^{k-1}$ **Iteración tipo** f
        end if
        $x_{k+1} \leftarrow x_k + d_k$, $k \leftarrow k + 1$.
        $\rho \leftarrow \rho_{inicial}$
        Ir a **PROBLEMA TANGENCIAL**.
      end if
    end if
  else
    $\mathcal{F}^k \leftarrow \text{Add}((h(c(x_k)), f(x_k)), \mathcal{F}^{k-1})$ **Iteración tipo** h
    $k \leftarrow k + 1$
    Ir a **FASE RESTAURACIÓN**.
  end if
end while

2.3. Filter-SDP

2.3.1. Algoritmo

En esta sección se describe en detalle el algoritmo **Filter-SDP** desarrollado en [GR06]. Como se mencionó en el capítulo introductorio, este algoritmo resuelve problemas del siguiente tipo:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{mín}} & f(x) \\ \text{s.a} & h(x) = 0 \\ & G(x) \preceq 0 \end{array} \tag{2.77}$$

donde $f : \mathbb{R} \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ y $G : \mathbb{R}^n \to \mathbb{S}^m$ son funciones de clase C^2 y $\mathbb{S}^m$ denota el espacio lineal de las matrices simétricas de $m \times m$ dotado del producto interno $A \cdot B = \text{Tr}(AB) = \sum_{i,j=1}^{m} A_{ij}B_{ij}$.

En la sección anterior se describe el algoritmo *Filter-SQP*, aplicado a problemas de programación no lineal, el cual sirve como base para desarrollar **Filter-SDP**, pues se utiliza un filtro para prejuiciar los puntos a ser evaluados, una fase de restauración y un subproblema lineal con una región de confianza.

En esta parte, se denota $Dh(x) := \left[\frac{\partial h_i(x)}{\partial x_j}\right]_{ij}$ como la matríz Jacobiana de tamaño $p \times n$ asociada a h. Además, para operadores lineales $Ay := \sum_{i=1}^{m} y_i A_i$ con $A_i \in \mathbb{S}^m$, como es el caso de $DG(x)$, se define su operador adjunto $A^\top$:

$$A^\top Z = (A_1 \cdot Z, \ldots, A_n \cdot Z)^\top, \quad \forall Z \in \mathbb{S}^m \tag{2.78}$$

Primero se deben escribir condiciones de optimalidad para este problema. Se dirá que $(\overline{\lambda}, \overline{Y})$ es un multiplicador de Lagrange asociado al punto $\overline{x}$, si $(\overline{x}, \overline{\lambda}, \overline{Y}) \in \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{S}^m$ satisface las siguientes condiciones de Karush-Kuhn-Tucker (KKT):

$$\nabla_x L(\overline{x}, \overline{\lambda}, \overline{Y})^\top = \nabla f(\overline{x})^\top + Dh(\overline{x})^\top \overline{\lambda} + DG(\overline{x})^\top \overline{Y} = 0, \tag{2.79a}$$

$$G(\overline{x})\overline{Y} = 0, \tag{2.79b}$$

$$G(\overline{x}) \preceq 0, h(\overline{x}) = 0, \overline{Y} \succeq 0, \tag{2.79c}$$

donde $L : \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{S}^m \to \mathbb{R}$ es el Lagrangiano del problema (2.77):

$$L(x, \lambda, Y) := f(x) + h(x)^\top \lambda + Y \cdot G(x). \tag{2.80}$$

Tambien se debe establecer una calificación de las restricciones. Para ello se utiliza la calificación de *Robinson* [Rob76] definida en un punto factible $\overline{x}$ de (2.77) como:

$$0 \in \text{Int}\left\{ \begin{pmatrix} G(\overline{x}) \\ h(\overline{x}) \end{pmatrix} + \begin{pmatrix} DG(\overline{x}) \\ Dh(\overline{x}) \end{pmatrix} \mathbb{R}^n - \begin{pmatrix} \mathbb{S}^m_- \\ \{0\} \end{pmatrix} \right\}, \tag{2.81}$$

donde $\text{Int}(C)$ denota el interior topologico del conjunto C, y $\mathbb{S}^m_- = \{A \in \mathbb{S}^m \mid A \preceq 0\}$.

Se puede probar que la calificación de Robinson (2.81) es equivalente a la calificación de *Mangasarian-Fromovitz*:

$$\{\nabla h_j(\overline{x})\}_{j=1}^{p} \quad \text{son linelmente independientes, y} \tag{2.82a}$$

$$\exists \bar{d} \in \mathbb{R}^n \text{ s. a.} \quad \begin{cases} Dh(\overline{x})\bar{d} = 0 \\ \text{and } G(\overline{x}) + DG(\overline{x})\bar{d} \prec 0. \end{cases} \tag{2.82b}$$

Además, se puede mostrar que bajo la calificación (2.81) el conjunto de multiplicadores de Lagrange $\Lambda(\overline{x})$ es no vacío y acotado [Kur76]. Más aún, cuando $\overline{x}$ es una solución local de (2.77), la calificación (2.81) es equivalente a que $\Lambda(\overline{x})$ sea no vacío y acotado.

El siguiente lema relaciona las condiciones de optimalidad de Karush-Kuhn-Tucker y la calificación de restricciones de Mangasarian-Fromovitz:

Lema 2.3.1. *Sea $\bar{x}$ un punto factible para* (2.77) *y que satisface las condiciones de Mangasarian-Fromovitz (2.82). Si $\bar{x}$ no es punto crítico* (2.77)*, entonces existe un vector unitario $\bar{s} \in \mathbb{R}^p$ y un real $\bar{\eta} > 0$ tales que para todo $\eta \in (0, \bar{\eta}]$ se cumple*

$$\nabla f(\overline{x})\bar{s} < 0, \tag{2.83a}$$

$$Dh(\overline{x})\bar{s} = 0, \tag{2.83b}$$

$$G(\overline{x}) + \eta DG(\overline{x})\bar{s} \prec 0. \tag{2.83c}$$

El lema anterior permite construir un subproblema cuya solución entrega información sobre las condiciones de optimalidad de punto evaluado. En el algoritmo se resuelven iterativamente aproximaciones semidefinidas locales (*trust region local semidefinite approximation*):

$$\begin{aligned} QP(x,\rho): \quad \min_{d\in\mathbb{R}^n} \quad & \nabla f(x)d + \frac{1}{2}d^T B d \\ s.a. \quad & h(x) + Dh(x)d = 0 \\ & G(x) + DG(x)d \preceq 0 \\ & \|d\|_\infty \leq \rho \end{aligned}$$

donde la matríz B contiene información de segundo orden para el problema 2.77. En el problema $QP(x,\rho)$ se obtiene como solución un vector d, el cual puede ser distinto de cero si el punto x_k no es punto crítico, y será igual a cero si lo es. Adicionalmente se mantiene un filtro $\mathcal{F}$ de puntos $(\theta(x), f(x))$, donde $\theta(x)$ cuantifica la factibilidad del punto x de la siguiente forma:

$$\theta(x) \quad = \quad \|h(x)\|_2 + \max\{0, \lambda_1(G(x))\} \tag{2.84}$$

donde $\lambda_1(A)$ es el máximo valor propio de la matríz A. Si $\theta(x) = 0$, entonces x es factible. Para que el filtro $\mathcal{F} = \{(\theta_i, f_i)\}_{i=1}^{N}$ acepte un nuevo punto $(\overline{\theta}, \overline{f})$, se debe satisfacer alguna (o ambas) de las siguientes condiciones:

$$\overline{\theta} \quad \leq \quad \beta\theta_i \tag{2.85}$$

$$\overline{f} + \gamma\overline{\theta} \quad \leq \quad f_i \tag{2.86}$$

con $\beta \in (0,1)$ y $\gamma \in (0,\beta)$. Al aceptar un nuevo punto, se debe actualizar el filtro, como se mencionó en la definición 2.2.4.

Todos los pasos del método **Filter-SDP** se pueden ver en el algoritmo 4.

Con respecto a la convergencia global del método, primero se debe suponer que se satisfacen las siguientes hipótesis:

1. Los puntos generados por el algoritmo pertenecen a un conjunto $X \subset \mathbb{R}^n$ compacto no vacío.
2. Se satisfacen las condiciones de Mangasarian-Fromovitz en cada punto factible de (2.77) que pertenece al conjunto X.
3. Existe una constante $M > 0$ tal que $\|B^k\|_2 \leq M$ para todo k.

Este conjunto de hipótesis se denotará (H). Sin pérdida de generalidad se puede suponer que las cantidades $\|\nabla f(x)\|_2$, $\|D^2 f(x)\|_2$, $\|Dh_i(x)\|_2$, $\|D^2 h_i(x)\|_2$, $\|DG(x)\|_{Fr}$ y $\|D^2 G(x)\|_{Fr}$ están acotadas por M en un convexo compacto suficientemente *grande* contenido en X.

El principal resultado que garantiza la convergencia global es el siguiente, cuya demostración se puede revisar en [GR06]:

Teorema 2.3.2. *Si se tiene que las hipótesis (H) son verdaderas, entonces dada una sucesión $\{x_k\}$ generada por el algoritmo **Filter-SDP**, una y sólo una de las siguientes situaciones puede ocurrir:*

(A) *La fase de restauración falla en encontrar un punto x_k que satisfaga (A1) y (B1).*

(B) *Un punto crítico de* (2.77) *es encontrado, es decir, $d = 0$ resuelve el problema auxiliar $QP(x_k, \rho)$ para algún k.*

(C) *Existe un punto de acumulación de la sucesión $\{x_k\}$ que es un punto crítico de* (2.77).

Como se mencionó en la sección 2.2, las etapas más importantes del algoritmo son la fase de restauración y el subproblema $QP(x, \rho)$. Ambas partes se estudian en el capítulo Trabajo realizado, poniendo enfasis en su resolución secuencial y utilizando cálculo paralelo.

2.3.2. Implementación en MATLAB ©

La implementación existente fue realizada el año 2006 por Gabriela Briones, para el artículo [GR06], y en la cual se utilizan los toolbox SeDuMi [Stu99] y YALMIP [LÖ04]. En esta parte se revisará a grandes rasgos la implementación y sus puntos principales.

Algoritmo 4 Filter-SDP

(INICIALIZACIÓN) $k \leftarrow 1$, $\mathcal{F}^0 = \{(u, -\infty)\}$, $d_k \leftarrow \infty^n$, $\beta \in (0,1), \gamma \in (0,\beta), u > 0, \sigma \in (0,1), \overline{\rho} > 0$, $\rho_{inicial} > \overline{\rho}$, max_iteraciones > 1.
while $k <$ max_iteraciones **do**
  (FASE RESTAURACIÓN) Encontrar x_k y $\rho_{inicial} \geq \tilde{\rho} > \overline{\rho}$ tales que:

  A1 $(\theta(x_k), f(x_k))$ es aceptable para $\mathcal{F}^{k-1}$.

  B1 $QP(x_k, \tilde{\rho})$ es factible.

  $\rho \leftarrow \tilde{\rho}$.
  (PROBLEMA TANGENCIAL) Resolver $QP(x_k, \rho)$.
  if $\|d_k\| < +\infty$ ($QP(x_k, \rho)$ es factible) **then**
    if $\|d_k\| < \epsilon$ **then**
      Fin del algoritmo. Solución: x_k.
    end if
    if $(\theta(x_k + d_k), f(x_k + d_k))$ no es aceptable por $\mathcal{F}^{k-1} \cup \{(\theta(x_k), f(x_k))\}$ **then**
      $\rho \leftarrow \frac{\rho}{2}$.
      Ir a **PROBLEMA TANGENCIAL**.
    else
      if $\nabla f(x_k)^T d_k + \frac{1}{2} d_k^T B d_k < 0$ y $f(x_k) + \sigma(\nabla f(x_k)^T d_k + \frac{1}{2} d_k^T B d_k) < f(x_k + d_k)$ **then**
        $\rho \leftarrow \frac{\rho}{2}$.
        Ir a **PROBLEMA TANGENCIAL**.
      else
        if $\nabla f(x_k)^T d_k + \frac{1}{2} d_k^T B d_k \geq 0$ **then**
          $\mathcal{F}^k \leftarrow \text{Add}((\theta(x_k), f(x_k)), \mathcal{F}^{k-1})$ **Iteración tipo** θ
        else
          $\mathcal{F}^k \leftarrow \mathcal{F}^{k-1}$ **Iteración tipo** f
        end if
        $x_{k+1} \leftarrow x_k + d_k$, $k \leftarrow k + 1$.
        $\rho \leftarrow \rho_{inicial}$
        Ir a **PROBLEMA TANGENCIAL**.
      end if
    end if
  else
    $\mathcal{F}^k \leftarrow \text{Add}((\theta(x_k), f(x_k)), \mathcal{F}^{k-1})$ **Iteración tipo** θ
    $k \leftarrow k + 1$
    Ir a **FASE RESTAURACIÓN**.
  end if
end while

Para esta revisión, sólo se utilizó el problema de la forma (1.2), y en este caso, la variable x se representa de la forma $x = (F, Q, V)$ con $F \in \mathbb{R}^{p\times r}$ y $Q, V \in \mathbb{S}^n$, y se utilizarán las siguientes funciones:

$$\begin{aligned} f(F,Q,V) &= \text{Tr}(C_F Q C_F^{\text{T}}) && (2.87)\\ \theta(F,Q,V) &= \|A_F Q + Q A_F^{\text{T}} + B_1 B_1^{\text{T}}\| && (2.88)\\ &\quad + \|A_F V + V A_F^{\text{T}} + I\| && (2.89)\\ &\quad + \text{máx}\{\lambda_1(-V), 0\} && (2.90) \end{aligned}$$

donde $A_F = A + BFC$, $C_F = C_1 + D_{12}FC$, $\lambda_1(A) = \text{máx}_i\, \lambda_i(A)$ y las matríces $A, B, B_1, C, C_1, D_{12}$ representan a la planta LTI:

$$\begin{aligned} x'(t) &= Ax(t) + B_1 w(t) + Bu(t)\\ z(t) &= C_1 x(t) + D_{12} u(t)\\ y(t) &= Cx(t) \end{aligned} \qquad (2.91)$$

con $x(t)$ el estado, $u(t)$ el control de entrada, $y(t)$ la observación de salida, $z(t)$ la salida regulada y $w(t)$ el ruido de entrada. La descripción de este problema y su derivación para ser formulado en el formato de **Filter-SDP** se revisará en el capítulo Resultados.

En la figura 2.1 se puede ver el grafo de dependencias de cada función. En él se puede observar que la función raíz corresponde a `corredor.m`, el cual contiene las instrucciones básicas para su ejecución.

La aplicación se encuentra instalada actualmente en el servidor `euler` del Departamento de Ingeniería Matemática. El modo de uso es el siguiente:

```
[18:33 0.07][euler] % matlab08 -nodisplay -nosplash -r "corredor('REA1');"
```

En este caso, `REA1` corresponde al código del problema a resolver, proveniente de la librería COMPleib, descrita en el capítulo 4.

A continuación se describe cada función:

- `corredor.m`: realiza la carga de datos desde la librería COMPleib y selecciona el tipo de problema a resolver, de la forma (1.2). La carga de datos desde la librería COMPleib se realiza de la siguiente manera, desde MATLAB:

  ```
  >> [A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('REA1');
  ```

  De esta forma se almacenan las matrices y dimensiones asociados al problema (2.91), donde $A \in \mathbb{R}^{nx\times nx}$, $B_1 \in \mathbb{R}^{nx\times nw}$, $B \in \mathbb{R}^{nx\times nu}$, $C_1 \in \mathbb{R}^{nz\times nx}$, $D_{11} \in \mathbb{R}^{nz\times nw}$, $D_{12} \in \mathbb{R}^{nz\times nu}$, $C \in \mathbb{R}^{ny\times nz}$ y $D_{21} \in \mathbb{R}^{ny\times nw}$.

- `algmodificado.m`: implementa los pasos del algoritmo. Es el archivo principal pues todos los pasos del algoritmo se encuentran implementados aquí.

- `theta_ff.m`: calcula la función $\theta(x_k)$ descrita en (2.88).

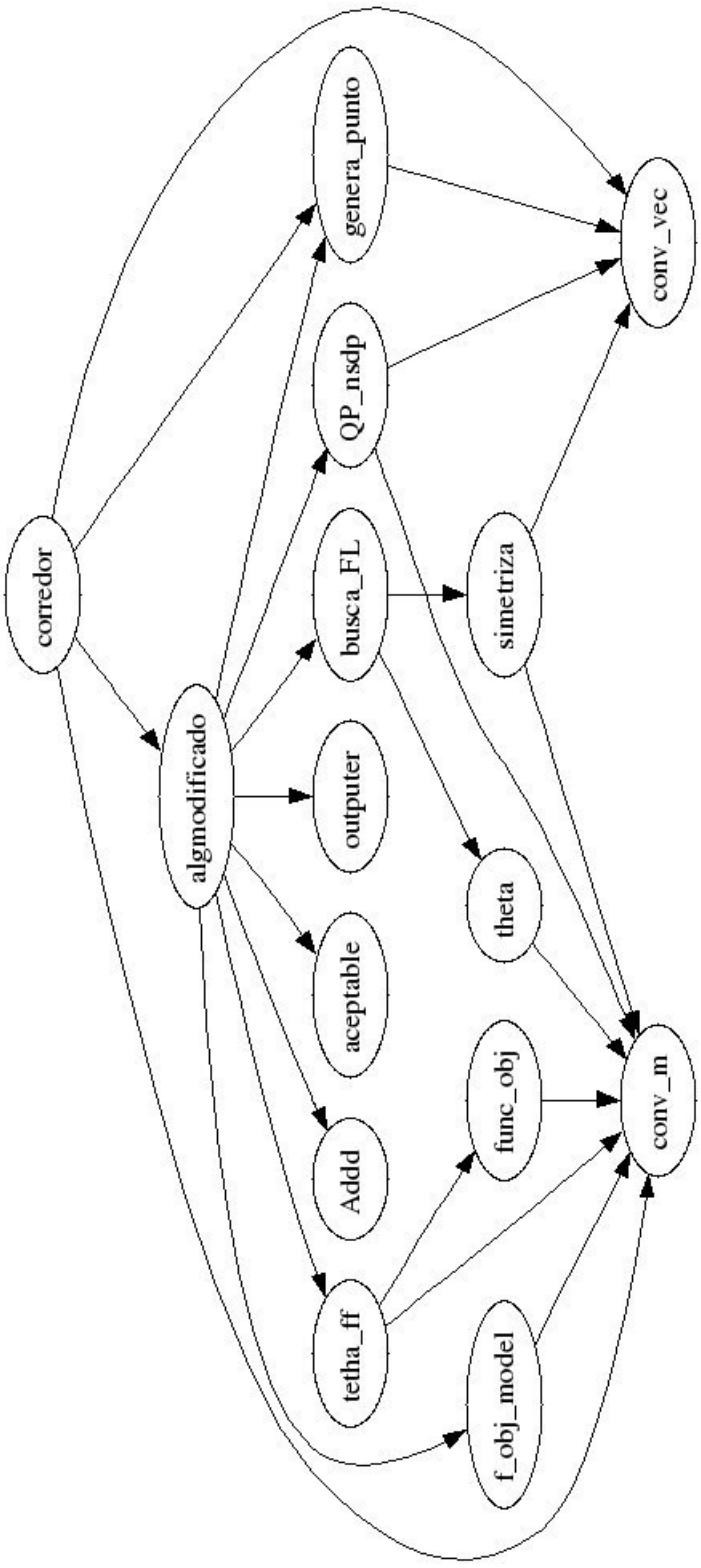

Figura 2.1: Funciones en MATLAB © para implementación de **Filter-SDP**

- `Addd.m`: agrega al filtro $\mathcal{F}_k$ un punto x_k, realizando las actualizaciones internas necesarias.
- `aceptable.m`: verifica que un punto x_k sea aceptable por el filtro $\mathcal{F}_k$, según las reglas (2.72).
- `outputer.m`: escribe en pantalla la evolución del algoritmo.
- `busca_FL.m`: busca un punto que minimice la función de mérito $\theta(x_k)$, partiendo de algún punto incial entregado como input.
- `QP_nsdp.m`: resuelve el problema $QP(x_k, \rho)$ descrito en (D.8).
- `genera_punto.m`: entrega un punto inicial para el algoritmo.
- `func_obj.m`: calcula la función $f(x_k)$ descrita en (2.87).
- `simetriza.m`: función auxiliar que realiza la simetrización de una matríz, es decir, $\frac{1}{2}(A + A^{\mathrm{T}})$.
- `conv_m.m`: función auxiliar que convierte un vector en matríz.
- `conv_vec.m`: función auxiliar que convierte una matríz en un vector.

2.4. Computación Paralela

En esta sección se presenta un resumen de las principales arquitecturas asociadas a la computación paralela. El término computación paralela (o cálculo paralelo) se refiere a la situación en la que al menos 2 procesadores cooperan intercambiando información mientras trabajan en diferentes partes de uno o más problemas. Existen diferentes clasificaciones de computadores que permiten realizar cálculo paralelo, dependiendo del número de procesadores, el acceso de los procesadores a la memoria, las redes que comunican a los procesadores entre sí, etc. Mayores detalles sobre las arquitecturas de cálculo paralelo se pueden revisar en [Dun90].

2.4.1. Memoria Compartida

Las arquitecturas de memoria compartida se pueden representar mediante un esquema como se presenta en la figura 2.2. En este esquema, se cuenta con N procesadores conectados mediante un *bus de datos*, el cual permite el envío de datos hacia/desde la memoria central y entre los procesadores. Cuando un procesador almacena un dato en la memoria, al ser única, ningún otro procesador puede utilizar ese espacio para almacenamiento (memoria local), pero sí puede hacer uso del dato ya almacenado por el otro procesador.

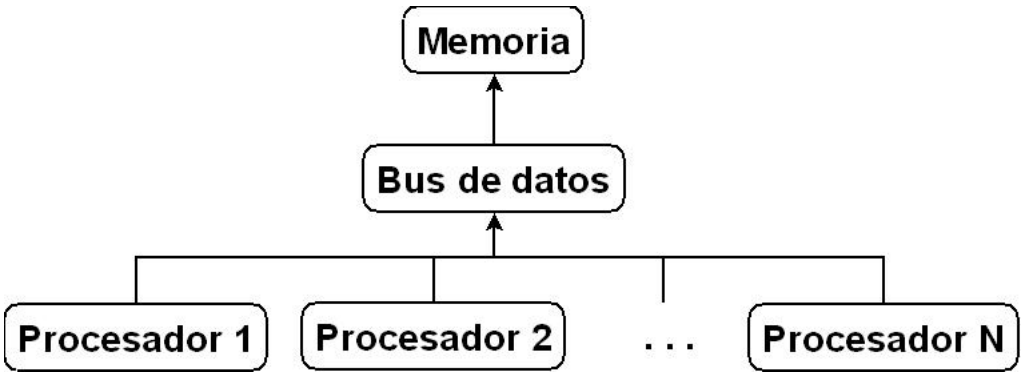

Figura 2.2: Esquema de computador con memoria compartida

Lo anterior permite una comunicación mucho más rápida entre los procesadores, y si además se cuenta con un bus de datos eficiente o diseñado especialmente para la arquitectura utilizada, hace que la computación usando memoria compartida sea la opción más rápida a utilizar.

Sus desventajas radican en que si se desea escalar (aumentar el número de procesadores) se debe rediseñar la arquitectura o comprar un nuevo sistema independiente del que ya se tiene, pues en muchos casos no es posible modificar el sistema, debido a las optimizaciones de fábrica con las que vienen los sistemas.

Los sistemas de memoria compartida usualmente se utilizan para resolver problemas donde la distribución de la información es uniforme y hay una mínima cantidad de ella que debe ser compartida. Aplicaciones de lo anterior son multiplicaciones y descomposiciones de matrices, resolución de sistemas lineales, aplicaciones de procesamiento gráfico, entre otras.

Una librería para programar en sistemas de memoria compartida se llama OpenMP [ope]. En ella se utilizan *pragmas* para definir partes del código que se distribuirán entre los procesadores (concurrencia). Se basa en la creación de threads de ejecución paralelos compartiendo las variables del proceso padre que los crea.

En el código 2.1 se observa un ejemplo de utilización de la librería. Se puede observar la utilización de pragmas para definir variables privadas para cada proceso (línea 6), asi como variables compartidas y la utilización de sincronización (línea 10).

Código 2.1: Hola Mundo para OpenMP

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main (int argc, char *argv[]) {
5    int th_id, nthreads;
6    #pragma omp parallel private(th_id)
7    {
8      th_id = omp_get_thread_num();
9      printf("Hello World from thread %d\n", th_id);
10     #pragma omp barrier
11     if ( th_id == 0 ) {
12       nthreads = omp_get_num_threads();
13       printf("There are %d threads\n",nthreads);
14     }
15   }
16   return 0;
17 }
```

2.4.2. Memoria Distribuída

En este enfoque, se utilizan procesadores que no comparten una memoria central, como se muestra en el esquema de la figura 2.3, cada uno tiene su propia memoria conectada, por lo tanto es necesario enviar mensajes entre los procesos para coordinar la tareas a través de una red. La comunicación entre los procesos se puede realizar mediante diversos protocolos o librerías (MPI [For94], PVM [pvm], entre otros), pero todos tienen en común a la coordinación de las comunicaciones de los procesos como eje central.

La principal desventaja de esta arquitectura radica en la utilización de una *red de comunicación* entre los procesadores. Si bien existen redes de comunicación de alta velocidad y baja latencia, como Myri-10G [myr] o Infiniband [inf], es difícil competir con las velocidades de los buses de datos instalados en el hardware, como ocurre con los sistemas de memoria compartida.

Su ventaja más importante es la posibilidad de escalar. Si se desea aumentar el número de procesadores en el sistema, el rendimiento de las aplicaciones escalará de manera directa. Al estar conectados a través de una red (LAN), se pueden agregar procesadores según se necesiten, siempre que no se superen umbrales que perjudiquen la comunicación permitida por la red. En los sistemas de memoria compartida sólo es posible escalar si se conecta por red otro sistema (en ese caso sería un sistema híbrido de memoria compartida y distribuída, lo cual requiere de mayor sincronización al aumentar la complejidad de las comunicaciones) o se cambia el conjunto de procesadores existente por otro con un mayor número. En este último caso pueden producirse problemas de capacidad física de la placa utilizada o de distribución de energía dentro del sistema (cuando se agrega un número grande de nuevos procesadores).

Una de las librerías para programar en sistemas de memoria distribuída estudiadas en este trabajo se llama MPI [For94, Pac96]. Esta librería también sirve para sistemas de memoría compartida pero inicialmente se diseño para sistemas distribuídos. En ella

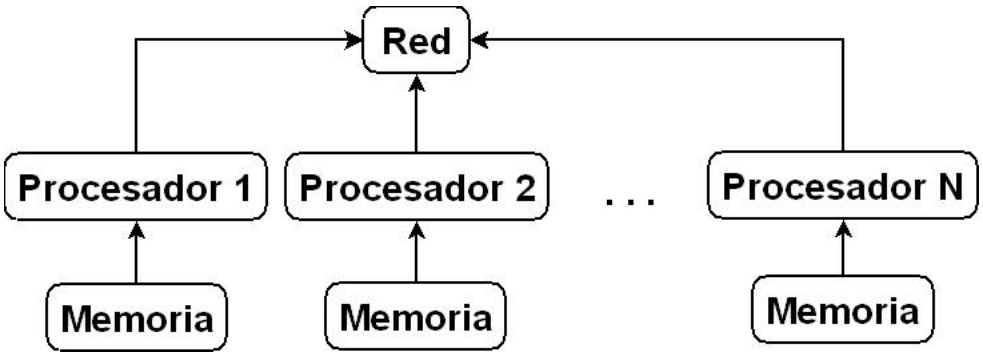

Figura 2.3: Esquema de computador con memoria distribuida

se utilizan *mensajes* para comunicar a los procesos que se encuentren trabajando. Las comunicaciones básicas son:

- `MPI_Send`: Permite enviar un mensaje ubicado en la dirección `buf` de tamaño `count` y de tipo `datatype`, al proceso `dest` a través del comunicador `comm` con la etiqueta `tag`.

```
int MPI_Send( void *buf,
              int count,
              MPI_Datatype datatype,
              int dest,
              int tag,
              MPI_Comm comm );
```

- `MPI_Recv`: Permite recibir un mensaje en la dirección `buf` de tamaño `count` y de tipo `datatype` enviado por el proceso `source` a través del comunicador `comm` con la etiqueta `tag` guardando en `status` el estado de la recepción.

```
int MPI_Recv( void *buf,
              int count,
              MPI_Datatype datatype,
              int source,
              int tag,
              MPI_Comm comm,
              MPI_Status *status );
```

- `MPI_Bcast`: Permite enviar un mensaje ubicado en la dirección `buffer` de tamaño `count` y tipo `datatype`, desde el proceso `root` hacia el resto de los procesos a través del comunicador `comm`.

Código 2.2: Hola Mundo para MPI

```
1  #include <stdio.h>
2  #include "mpi.h"
3
4  int main(int argc, char **argv){
5    int my_rank;
6    int p;
7    MPI_Init(&argc,&argv);
8    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
9    MPI_Comm_size(MPI_COMM_WORLD,&p);
10   printf("Hello! I am process %d of %d processes\n",my_rank,p);
11   MPI_Finalize();
12 }
```

```
int MPI_Bcast( void *buffer,
               int count,
               MPI_Datatype datatype,
               int root,
               MPI_Comm comm );
```

- `MPI_Barrier`: Bloquea a los procesos a través del comunicador `comm` hasta que todos hayan llegado al punto donde se realiza la llamada a la instrucción.

```
int MPI_Barrier( MPI_Comm comm );
```

En el código 2.2 se observa un ejemplo de utilización de la librería. Se puede observar la inicialización del esquema de paso de mensajes (línea 7), la obtención de variables globales (líneas 8 y 9) y la finalización del esquema (línea 11).

2.4.3. Speedup

Con las principales arquitecturas de cálculo paralelo ya mencionadas, conviene mencionar una definición del concepto llamado *speedup*, extraída de [Pac96]:

Definición 2.4.1 (Speedup)**.** *El speedup de una aplicación se define como la razón entre el tiempo secuencial que ésta demora y el tiempo en paralelo, utilizando múltiples procesos. Más precisamente, si T_σ es el tiempo secuencial y $r \in [0,1]$ corresponde a la fracción de la aplicación que se puede paralelizar de manera óptima, con lo cual el tiempo en paralelo utilizando P procesos es $T_\pi = (1-r)T_\sigma + \frac{rT_\sigma}{P}$, entonces el speedup utilizando P procesos es*

$$S = \frac{T_\sigma}{T_\pi} \tag{2.92}$$

$$= \frac{T_\sigma}{(1-r)T_\sigma + \frac{rT_\sigma}{P}} \tag{2.93}$$

En términos prácticos, mientras mayor sea el speedup logrado por una aplicación que corre sobre múltiples procesos, menor será el tiempo que demora su ejecución. Lo que se busca son aplicaciones que posean un speedup creciente como función del número de procesos involucrados.

Estudiar el speedup teórico de una aplicación tiene una importancia crucial debido a que existen aplicaciones que no son susceptibles a ser paralelizadas o que tienen una cota superior en términos del número de procesos que la pueden abordar, y poder detectar a tiempo una aplicación así ayuda a enfocar los esfuerzos en otras alternativas o no destinar más recursos a la paralelización.

La Ley de Amdahl, según [Pac96], entrega una cota superior para el speedup de una aplicación. Si el speedup (visto como función del número de procesos P) está dado por

$$S(P) = \frac{T_\sigma}{(1-r)T_\sigma + \frac{rT_\sigma}{P}} \tag{2.94}$$

con $r \in [0,1]$, eliminando T_σ queda

$$S(P) = \frac{1}{(1-r) + \frac{r}{P}} \tag{2.95}$$

Derivando con respecto a P se obtiene

$$\frac{dS(P)}{dP} = \frac{r}{((1-r)P + r)^2} \tag{2.96}$$

La fórmula anterior es positiva para todo P, luego $S(P)$ es una función creciente cuyo límite cuando P tiende a ∞ es

$$S(P) \rightarrow \frac{1}{1-r} \tag{2.97}$$

Debido a esto, existe una cota superior para el speedup que se puede obtener, incluso utilizando cientos de miles de procesos. Por ello es importante calcular una cota apriori del speedup (siempre que sea posible) para ver si conviene invertir en la incorporación de más procesos en el cálculo de la aplicación.

2.4.4. Cálculo paralelo en SDP

En esta parte se revisarán las principales áreas de paralelización del algoritmo de punto interior presentado en la sección 2.1.3. Como se vió en el lema (2.1.10), siguiendo el esquema de Mehrotra, en cada iteración se deben resolver 2 sistemas de la forma $M\Delta x = h$ (ver ecuación (2.46)) para las direcciones de escalamiento afín y corrección.

Construcción de la matríz M

Una primera tarea paralelizable consiste en calcular de manera distribuída la matríz M descrita en (2.46) donde

$$M_{ij} = \text{Tr}(F_i \mathcal{E}^{-1}(\mathcal{F}(F_j))),\ i,j = 1,\ldots,m \tag{2.98}$$

donde los operadores lineales $\mathcal{E} : \mathbb{S}^n \to \mathbb{S}^n$ y $\mathcal{F} : \mathbb{S}^n \to \mathbb{S}^n$ se definen como

$$\mathcal{E}(E) = H_P(EZ), \qquad \mathcal{F}(E) = H_P(SE) \tag{2.99}$$

con

$$H_P(N) = \frac{1}{2}\left(PNP^{-1} + (PNP^{-1})^{\text{T}}\right) \tag{2.100}$$

para una matríz $N \in \mathbb{R}^{n\times n}$ cualquiera y una matríz no singular $P \in \mathbb{R}^{n\times n}$. Se revisará el caso ***HKM*** donde $P = Z^{1/2}$, es decir,

$$H_{Z^{1/2}}(N) = \frac{1}{2}\left(Z^{1/2}NZ^{-1/2} + (Z^{1/2}NZ^{-1/2})^{\text{T}}\right) \tag{2.101}$$

con lo cual

$$M_{ij} = \text{Tr}(F_i H^{-1}_{Z^{1/2}}(H_{Z^{1/2}}(SF_j))Z),\ i,j = 1,\ldots,m \tag{2.102}$$

y se puede probar, simplificando términos, que las entradas de la matríz se calculan de la forma:

$$M_{ij} = \text{Tr}(F_i S^{-1} F_j Z),\ i,j = 1,\ldots,m \tag{2.103}$$

Se debe observar que la matríz M es simétrica pues F_i, S^{-1}, F_j y Z son simétricas y tienen las mismas dimesiones, por lo tanto las multiplicaciones internas conmutan, obteniendose la simetría.

Suponiendo que se cuenta con N procesos, denotados $P_0,\ldots,P_{N-1}$, se revisarán 3 distribuciones para el cálculo de la matríz M (solamente basta con calcular la parte triangular inferior, pues es simétrica), *distribución unidimensional cíclica por filas*, *distribución unidimensional cíclica por bloques de filas* y *distribución bidimensional cíclica por bloques de filas*.

En la *distribución unidimensional cíclica por filas*, cada proceso calcula una fila de la parte triangular inferior de M, de manera separada. Cuando todos los procesos ya se han asignado (cuando se han calculado $N-1$ filas), los procesos se vuelven a asignar de igual manera, es decir, al proceso P_0 se le asigna la fila N, al P_1 se le asigna la fila $N+1$, etc. En la figura 2.4 se puede observar la distribución de filas por cada proceso. La fórmula para determinar que proceso debe calcular cada fila es la siguiente:

$$\text{fila } i \to \text{ proceso } P_{i \bmod N} \tag{2.104}$$

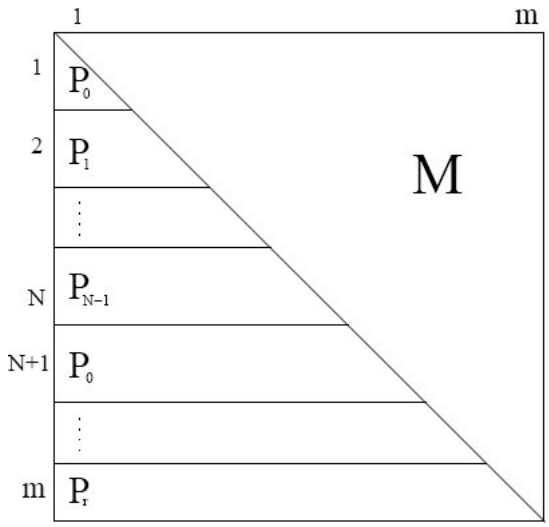

Figura 2.4: Distribución unidimensional cíclica por filas para cálculo de matríz M

En la *distribución unidimensional cíclica por bloques de filas*, cada proceso calcula un bloque de filas de tamaño $n_b < m$ de la parte triangular inferior de M, de manera separada. Cuando todos los procesos ya se han asignado a sus respectivos bloques, análogo al caso anterior, se vuelven a asignar de igual manera, es decir, al proceso P_0 se le asigna un nuevo bloque ubicado entre las filas $(N-1)n_b + 1$ y Nn_b, al P_1 se le asigna el bloque ubicado entre las filas $Nn_b + 1$ y $(N+1)n_b$, etc. Se debe observar que en este caso, el último bloque puede tener un tamaño diferente al de los bloques anteriores,

$$m - \underbrace{\left\lfloor \frac{m}{n_b} \right\rfloor}_{k} n_b$$

En la figura 2.5 se puede observar la distribución de bloques de filas por cada proceso. La fórmula para determinar que proceso debe calcular cada fila es la siguiente:

$$\text{fila } i \rightarrow \text{ proceso } P_{\left\lfloor \frac{i}{n_b} \right\rfloor \bmod N} \tag{2.105}$$

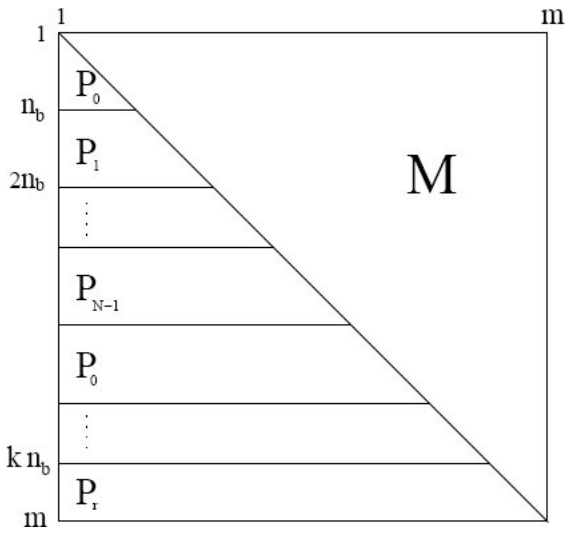

Figura 2.5: Distribución unidimensional cíclica por bloques de filas para cálculo de matríz M

Una vez distribuída la información necesaria en cada proceso, se procede a calcular los términos M_{ij} para cada fila $i = 1, \ldots, m$ de la matríz según la fórmula (2.103) y realizando los siguientes pasos:

1. Calcular $F_i S^{-1}$ una sola vez.
2. Para $j = 1, \ldots, i$, calcular $F_j Z$.
3. Para $j = 1, \ldots, i$, calcular $\text{Tr}(F_i S^{-1} F_j Z)$.

Otro enfoque equivalente es el siguiente (utilizando la propiedad de la traza descrita en (D.2.1)):

1. Calcular $Z F_i S^{-1}$ una sola vez.
2. Para $j = 1, \ldots, i$, calcular $\text{Tr}((Z F_i S^{-1}) F_j)$.

Se puede ver que cada proceso trabaja de manera independiente y no necesita sincronizar su información con el resto.

El software CSDP [Bor99] utiliza una distribución unidimensional cíclica por bloques de filas y los dos enfoques de cálculo para los valores de M_{ij}, intercambiando dinamicamente dependiendo de la densidad de las matrices F_i involucradas.

Descomposición de Cholesky de la matríz M

El siguiente paso consiste en realizar una descomposición de Cholesky de la matríz M. Recordemos que la descomposición de Cholesky dice que si $A \in \mathbb{S}^n$ es una matríz simétrica y definida positiva existe una matríz $L \in \mathbb{R}^{n \times n}$ triangular inferior con valores estrictamente positivos en la diagonal tal que $A = LL^{\mathrm{T}}$. Se revisarán 2 implementaciones de la descomposición: *gaxpy* y *producto externo* ([GL96]). Estas implementaciones se utilizan en el enfoque utilizado por el software CSDP, donde se utiliza la función `PDPOTRF` de la librería ScaLAPACK [BCC+96], la cual implementa una descomposición *hacia la derecha* utilizando una partición por bloques de la matríz.

Las implementaciones *gaxpy* y *producto externo* se pueden ver en los algoritmos 5 y 6. En ambos algoritmos se sobreescribe el factor de Cholesky L en la parte triangular inferior de la matríz A.

Algoritmo 5 Gaxpy-Cholesky

1: **for** $j = 1, \ldots, n$ **do**
2: **if** $j > 1$ **then**
3: $A(j:n, j) \leftarrow A(j:n, j) - A(j:n, 1:(j-1))A(j, 1:(j-1))^{\mathrm{T}}$
4: **end if**
5: $A(j:n, j) \leftarrow \frac{1}{\sqrt{A(j,j)}} A(j:n, j)$
6: **end for**

Algoritmo 6 Producto Externo-Cholesky

1: **for** $k = 1, \ldots, n$ **do**
2: $A(k, k) \leftarrow \sqrt{A(k, k)}$
3: $A((k+1):n, k) \leftarrow \frac{1}{A(k,k)} A((k+1):n, k)$
4: **if** $j = k+1, \ldots, n$ **then**
5: $A(j:n, j) \leftarrow A(j:n, j) - A(j:n, k)A(j, k)$
6: **end if**
7: **end for**

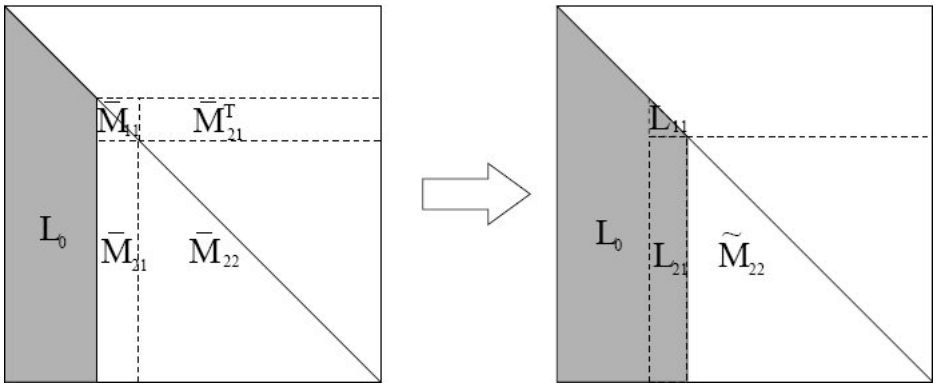

Figura 2.6: Descomposición de Cholesky *hacia la derecha* de la matríz M utilizando una partición por bloques

Ahora, con respecto a la descomposición hacia la derecha usando una partición por bloques, asumiendo que se está en el paso k-ésimo de la descomposición, la matríz M está particionada como en la figura 2.6. En la figura, L_0 es la parte del factor L que ya se ha calculado previamente y $\overline{M}$ es el bloque simétrico de $m_k \times m_k$ que resta por factorizar. Si $\overline{M}$ se particiona de la forma:

$$\overline{M} = \begin{bmatrix} \overline{M}_{11} & \overline{M}_{21}^{\mathrm{T}} \\ \overline{M}_{21} & \overline{M}_{22} \end{bmatrix} \tag{2.106}$$

$$= \begin{bmatrix} \overline{L}_{11} & 0 \\ \overline{L}_{21} & \overline{L}_{22} \end{bmatrix} \begin{bmatrix} \overline{L}_{11}^{\mathrm{T}} & \overline{L}_{21}^{\mathrm{T}} \\ 0 & \overline{L}_{22}^{\mathrm{T}} \end{bmatrix} \tag{2.107}$$

$$= \begin{bmatrix} \overline{L}_{11}\overline{L}_{11}^{\mathrm{T}} & \overline{L}_{11}\overline{L}_{21}^{\mathrm{T}} \\ \overline{L}_{21}\overline{L}_{11}^{\mathrm{T}} & \overline{L}_{21}\overline{L}_{21}^{\mathrm{T}} + \overline{L}_{22}\overline{L}_{22}^{\mathrm{T}} \end{bmatrix} \tag{2.108}$$

donde las matrices $\overline{M}_{11}$ y L_{11} son de tamaño $m_b \times m_b$, $\overline{M}_{21}$ y L_{21} de tamaño $(m_k - m_b) \times m_b$ y $\overline{M}_{22}$ de tamaño $(m_k - m_b) \times (m_k - m_b)$, con $m_b > 1$ el número de columnas del bloque.

Las operaciones a realizar en la k-ésima iteración de la descomposición son:

1. Calcular la descomposición de Cholesky del bloque diagonal $\overline{M}_{11}$ usando las estrategias *gaxpy* o *producto externo* descritas anteriormente. Se obtiene L_{11} como resultado.
2. Calcular L_{21} de la forma: $L_{21} = \overline{M}_{21}(L_{11}^{\mathrm{T}})^{-1}$.
3. Actualizar el resto de la matríz $\overline{M}$ hacia la derecha: $\tilde{M}_{22} = \overline{M}_{22} - L_{21}L_{21}^{\mathrm{T}}$.

En este caso, la distribución de los datos sobre los procesos que se requiere se llama *distribución bidimensional cíclica por bloques*, como se muestra en la figura 2.7, y es fundamental para el óptimo rendimiento del esquema utilizado.

En la *distribución bidimensional cíclica por bloques de filas* los P procesos se distribuyen en una grilla rectangular de $P_r \times P_c$ celdas, indexadas por (p_r, p_c) con $p_r \in \{0, \ldots, P_r - 1\}, p_c \in \{0, \ldots, P_c - 1\}$. Si las dimensiones de los bloques rectangulares son r filas por c columnas, y se tienen M elementos, indexados por $0 \leq m < M$, el m-ésimo elemento se almacenará en la i-ésima locación en el bloque b a través del proceso p donde:

$$\langle p, b, i \rangle := \left\langle \left\lfloor \frac{m}{r} \right\rfloor \bmod P, \left\lfloor \frac{\left\lfloor \frac{m}{r} \right\rfloor}{P} \right\rfloor, m \bmod r \right\rangle \tag{2.109}$$

Ahora bien, si se desea distribuir una matríz de datos, el elemento (m, n)-ésimo se almacenará en la (i, j)-ésima locación en el bloque (b, d) a través del proceso (p, q) donde:

$$\begin{aligned} \langle (p, q), (b, d), (i, j) \rangle \quad := \quad & \left\langle \left(\left\lfloor \tfrac{m}{r} \right\rfloor \bmod P_r, \left\lfloor \tfrac{n}{c} \right\rfloor \bmod P_c \right), \right. \\ & \left. \left(\left\lfloor \frac{\left\lfloor \frac{m}{r} \right\rfloor}{P_r} \right\rfloor, \left\lfloor \frac{\left\lfloor \frac{n}{c} \right\rfloor}{P_c} \right\rfloor \right), (m \bmod r, n \bmod c) \right\rangle \end{aligned} \tag{2.110}$$

Por ejemplo, en la figura 2.7, los parámtros son $P_r = P_c = r = c = 2$ y el elemento a_{52}, cuyos índices serían $m = 4$ y $n = 1$, está indexado de la siguiente manera:

$$\begin{aligned} (p, q) &= \left(\left\lfloor \frac{4}{2} \right\rfloor \bmod 2, \left\lfloor \frac{1}{2} \right\rfloor \bmod 2 \right) && (2.111) \\ &= (0, 0) && (2.112) \\ (b, d) &= \left(\left\lfloor \frac{\left\lfloor \frac{4}{2} \right\rfloor}{2} \right\rfloor, \left\lfloor \frac{\left\lfloor \frac{1}{2} \right\rfloor}{2} \right\rfloor \right) && (2.113) \\ &= (1, 0) && (2.114) \\ (i, j) &= (4 \bmod 2, 1 \bmod 2) && (2.115) \\ &= (0, 1) && (2.116) \end{aligned}$$

Es decir, al elemento a_{52} de la matríz se le asigna la ubicación $(1, 2)$ en el bloque $(2, 1)$ para el proceso $(0, 0)$ como se muestra en la figura.

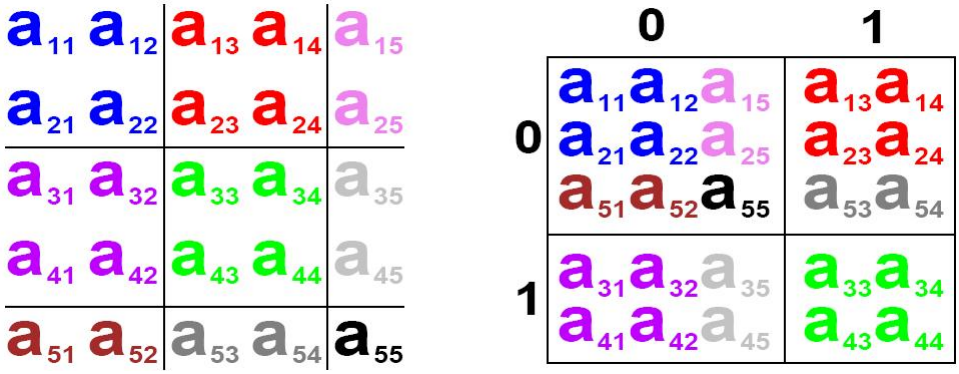

Figura 2.7: Distribución bidimensional cíclica por bloques para descomposición de Cholesky de matríz M (a la izquierda la distribución de los datos en los bloques, a la derecha la distribución de los datos en los procesos)

Un problema recurrente que surge en este punto se refiere a la distribución de los datos a utilizar. Como se mencionó anteriormente, para calcular la matríz M, el software CSDP utiliza una distribución unidimensional cíclica por bloques de filas. Ahora, si se quiere utilizar la función `PDPOTRF` de la librería ScaLAPACK, se deben redistribuir los datos y utilizar una distribución bidimensional cíclica. Esta redistribución de los datos puede generar cuellos de botella en el rendimiento y perjudicar el tiempo de ejecución de la aplicación en estudio. Debido a esto, en [IK07] se plantean diversos esquemas para mapear el conjunto de procesos desde una distribución hacia otra de manera biyectiva. La implementación realizada en [IK07], llamada PCSDP, sirve como base para la utilización de CSDP en un ambiente de computación con memoria distribuída.

Resolución del sistema $M\Delta x = h$

Para resolver el sistema $M\Delta x = h$ se utiliza la descomposición de Cholesky $M = LL^{\mathrm{T}}$ y se resuelven 2 sistemas triangulares de manera directa:

$$L\tilde{\Delta x} = h \tag{2.117}$$
$$L^{\mathrm{T}}\Delta x = \tilde{\Delta x} \tag{2.118}$$

La función que realiza esta tarea se llama `PDPOTRS` de la librería ScaLAPACK, la cual recibe como inputs a la matríz M y al vector h e internamente realiza la descomposición de Cholesky con la función `PDPOTRF`, y resuelve los sistemas (2.117) y (2.118).

2.4.5. Cálculo paralelo en Filter-SDP

En esta parte se presentan las principales áreas de posible paralelización del algoritmo **Filter-SDP** presentado en la sección 2.3. Estas áreas son solamente *potenciales*, pues este

trabajo se trata sobre el estudio de ellas y su evaluación. A continuación se detallan los pasos y las posibles estrategias para paralelizar:

Cálculo de $\theta(x_k)$

Un punto donde se puede introducir cálculo paralelo es la obtención de $\theta(x)$, pues requiere calcular $\lambda_1(G(x))$, lo cual puede ser costoso si n y m son grandes. Una implementación posible se encuentra en la librería ScaLAPACK [BCC+96], bajo la rutina `PDSYEVX`, la cual calcula el valor propio máximo realizando primero una reducción a una forma tridiagonal, luego se encuentran los valores propios y finalmente se transforman inversamente esos valores. La inclusión de esta aplicación requiere aprender a utilizar ScaLAPACK de forma eficiente junto con la conexión correcta con otras aplicaciones.

Resolución de $QP(x_k, \rho)$

Como el problema $Q(x_k, \rho)$ se puede expresar de manera semidefinida lineal (función objetivo lineal y restricciones semidefinidas lineales), se puede utilizar la librería CSDP [Bor99] y sus versiones paralelas implementadas con OpenMP [BY07] y MPI [IK07], donde se realiza una implementación de un método de punto interior primal-dual en paralelo. El desafío consiste en la inclusión óptima de esta implementación en el marco de este trabajo.

Operaciones algebraicas

Para reducir el tiempo de cálculo de las operaciones algebraicas realizadas, se propone utilizar ScaLAPACK [BCC+96] junto a las librerías de Intel MKL [mkl].

Capítulo 3

Trabajo realizado

En este capítulo se revisará el trabajo realizado, el cual se puede dividir en 3 partes:

- Estudio de sistemas de cálculo paralelo.
- Implementación de algoritmo **Filter-SDP** utilizando cálculo paralelo.
- Diseño e implementación de distintas fases de restauración.

Cada parte intenta reflejar el proceso experimentado, pues al comienzo del trabajo se estudiaron las herramientas de cálculo paralelo y se comenzó a implementar la aplicación en C y MPI, sin embargo, había un aspecto del algoritmo que impedía su funcionamiento correcto, la fase de restauración. Esta fase no se encontraba plenamente estudiada para los problemas particulares que se pretendía resolver (COMPleib), por lo tanto, se decidió diseñar e implementar diferentes fases en MATLAB ©, con el objetivo de mejorar la convergencia del algoritmo y definir cual sería la fase definitiva a implementar en el sistema de cálculo paralelo. Cuando la fase de restauración estuviera bien definida, se procedería a implementarla en C y MPI, sin embargo por motivos de tiempo, sólo se implementó la fase de restauración básica, la cual utiliza el método de Nelder-Mead simplex (análogo a la implementación desarrollada en MATLAB), lo que no impide seguir trabajando en esa área a futuro.

A continuación se revisará cada área en detalle.

3.1. Estudio de sistemas de cálculo paralelo

El primer paso para la construcción de un sistema de cálculo paralelo para el algoritmo **Filter-SDP**, consistió en estudiar en profundidad dos herramientas, las librerías de álgebra lineal para alto desempeño y la versión paralelizada del solver de programación semidefinida CSDP, PCSDP [IK07] (memoria distribuída).

3.1.1. Álgebra lineal para alto desempeño

A continuación se detallan las librerías estudiadas para la utilización de rutinas de álgebra lineal sobre plataformas de cálculo de alto desempeño. Se detallan las librerías básicas más conocidas y referenciadas en artículos del área, las cuales son BLAS, LAPACK, BLACS y ScaLAPACK. Adicionalmente se incluyó un apéndice donde se explica la utilización de cada una de ellas, mediante ejemplos simples.

BLAS

Basic Linear Algebra Subprograms (BLAS) es la API de facto utilizada para publicar librerías que desarrollan operaciones de algebra lineal tales como multiplicación de vectores y matrices. Los subprogramas se publicaron por primera vez en 1979 [LHKK79], y han sido utilizados para construir librerías más grandes tales como LAPACK. Su utilización en el área del HPC es inmensa e implementaciones optimizadas de BLAS han sido desarrolladas por empresas vendedoras de hardware tales como Intel o AMD.

Los subprogramas se dividen en 3 niveles:

- Nivel 1: operaciones vectoriales de la forma:

  $$\boldsymbol{y} \leftarrow \alpha\boldsymbol{x} + \boldsymbol{y}$$

  así como distintas versiones de productos punto y normas vectoriales, entro otros.

- Nivel 2: operaciones entre vectores y matrices de la forma:

  $$\boldsymbol{y} \leftarrow \alpha A\boldsymbol{x} + \beta\boldsymbol{y}$$

  así como operaciones para resolver $T\boldsymbol{x} = \boldsymbol{y}$ para $\boldsymbol{x}$ con T triangular, entre otros.

- Nivel 3: operaciones entre matrices de la forma:

  $$C \leftarrow \alpha AB + \beta C$$

  así como operaciones para resolver $B \leftarrow \alpha T^{-1}B$ para matrices T triangulares, entre otros. Este nivel contiene una operación ampliamente usada llamada *General Matrix Multiply*.

En A.2 se puede observar un ejemplo completo para la utilización de la rutina `DGEMM`, la cual realiza la multiplicación entre 2 matrices.

LAPACK

Linear Algebra PACKage (LAPACK) es la librería de software para álgebra lineal más utilizada en aplicaciones científicas. Provee rutinas para resolver sistemas de ecuaciones lineales y mínimos cuadrados lineales, problemas de valores y vectores propios, y descomposiciones a valores singulares. También incluye rutinas para realizar factorizaciones de matrices, como LU, QR, Cholesky y descomposición de Schur. LAPACK fue escrita originalmente para el lenguaje FORTRAN 77 y actualmente para Fortran 90 [ABD+90].

LAPACK se puede entender como el sucesor de las rutinas de resolución de sistemas lineales y mínimos cuadrados LINPACK [DMBS79] y de las rutinas de resolución de problemas de valores propios EISPACK [SBD+76, GBDM77]. LINPACK fue diseñado para correr sobre computadores vectoriales con memoria compartido, modernos para su época. LAPACK, en cambio, depende de la linrería BLAS para explotar las capacidades de las arquitecturas actuales, y de esa forma correr a magnitudes mayores que LINPACK, dada una implementación eficiente de BLAS. LAPACK también ha sido extendido para correr sobre sistemas computacionales de memoria compartida en paquetes posteriores, llamados ScaLAPACK y PLAPACK.

En A.3 se puede observar un ejemplo completo para la utilización de la rutina `DPOTRF`, la cual realiza la descomposición de Cholesky para una matríz definida positiva.

BLACS

Basic Linear Algebra Communication Subprograms (BLACS) [DW95] es una librería de paso de mensajes entre procesadores con memoria distribuída orientada a problemas de álgebra lineal. Funciona como una capa intermedia entre el sistema local de comunicación entre procesadores y las aplicaciones de álgebra lineal que corran sobre ese sistema. Permite desarrollar aplicaciones de manera más fácil y con mayor portabilidad. Por ejemplo, la librería ScaLAPACK [BCC+96] utiliza BLACS para realizar la conexión entre MPI [For94] y las librerías de álgebra lineal bases de ScaLAPACK.

ScaLAPACK

Scalable Linear Algebra Package (ScaLAPACK) [BCC+96] es una librería con rutinas para resolución de problemas de álgebra lineal diseñada para máquinas con memoria distribuída con soporte para PVM [pvm] y MPI [For94]. Es la continuación del proyecto LAPACK [ABD+90], el cual funciona en máquinas con memoria compartida (mono o multi procesador). Ambas librerías contienen rutinas para resolver sistemas lineales de ecuaciones, problemas de mínimos cuadrados y problemas cálculo de valores y vectores propios. Ambos proyectos tienen como caracteristicas su eficiencia, escalabilidad, confiabilidad, portabilidad, flexibilidad y facilidad de uso.

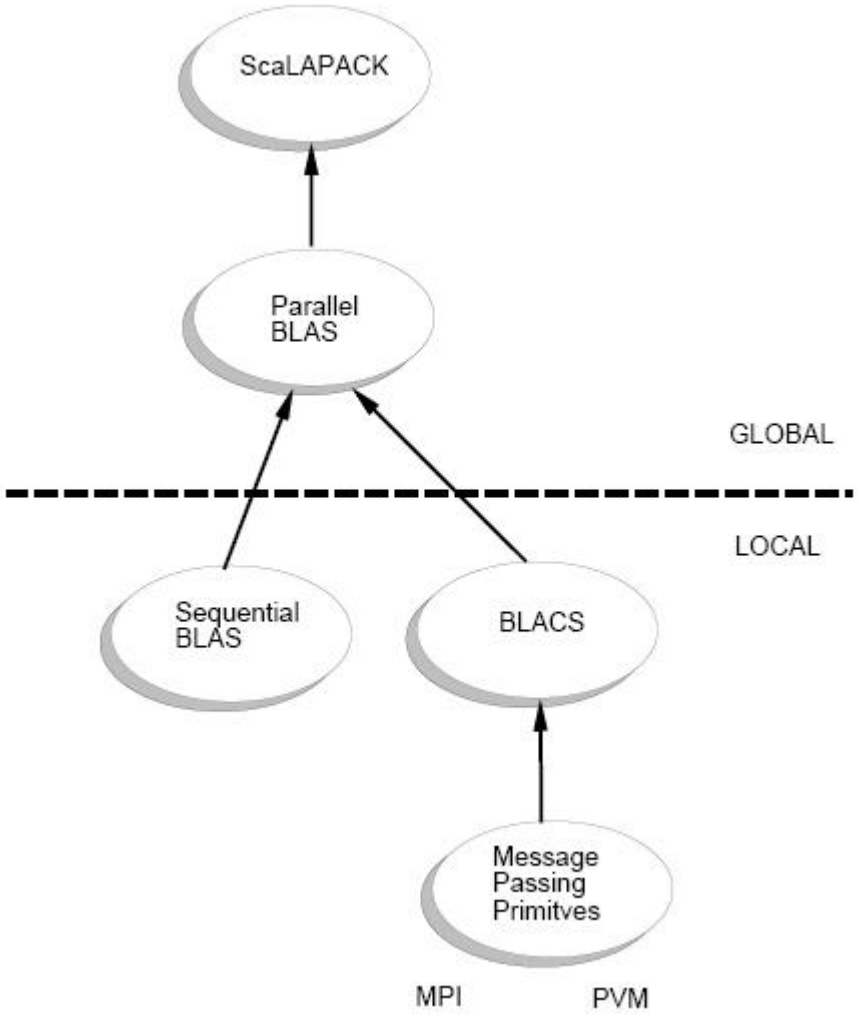

Figura 3.1: Esquema de librería ScaLAPACK

Para la utilización de estas librerías, se requiere tener instaladas las librerías BLAS y BLACS, las cuales son un estandar utilizado en diversas aplicaciones numéricas. LAPACK se puede utilizar en cualquier máquina con BLAS instalado, y ScaLAPACK en cualquier máquina con BLAS y BLACS instalados. El diagrama de la estructura de las librerías utilizadas por ScaLAPACK se puede ver en la figura 3.1.

En A.4 se puede observar dos ejemplos completos para la utilización de las rutina `PDPSYEV` y `PDSYEVX`, las cuales calculan la totalidad y el mayor de los valores propios de una matríz real respectivamente.

3.1.2. CSDP para memoria distribuída

CSDP [Bor99] es un software de código abierto licenciado bajo Common Public License Versión 1.0 diseñado para la resolución de problemas de programación semidefinida (paquete Debian: `coinor-csdp`). El algoritmo que utiliza es una versión predictor-corrector del método primal-dual propuesto por Helmberg, Rendl, Vanderbei y Wolkowicz en [HRVW96, KSH97, Mon97]. Su descripción se puede ver en la sección 2.1.3.

El lenguaje utilizado para su implementación es C, debido a su eficiencia y portabilidad. En sistemas con múltiples procesadores (memoria compartida y distribuída), CSDP puede correr en paralelo debido a que existen versiones que utilizan OpenMP en su código con el objetivo de paralelizar diversos loops y rutinas de cálculo, y MPI con el objetivo de paralelizar operaciones algebraicas. El código fue diseñado de tal manera que las librerías BLAS, LAPACK y ScaLAPACK se puedan utilizar de manera fácil y eficiente.

Algunos aspectos de CSDP hacen que su uso sea particularmente flexible. Puede manejar matrices generales o simétricas con relativa simplicidad, definiendo estructuras de bloques dentro de las matrices. Tambien maneja las restricciones lineales semidefinidas de forma estructurada. Este diseño hace que la resolución de problemas se desarrolle de manera eficiente y se haya escogido utilizar como herramienta central, frente a otros sistemas similares, como SDPA [YFF+05], desarrollado en C++ y cuyo código no se encontraba abierto al momento de comenzar este trabajo. Además de lo anterior, se realizó una búsqueda de benchmarks para ambos sistemas encontrandose que el tiempo de CPU utilizado para resolver los problemas test de SDPLIB [Bor98] fue de aproximadamente 155000 y 99000 segundos para SDPA y CSDP respectivamente, y el mejor tiempo obtenido en SDPA fue 3 veces más rápido que el más lento, mientras que en CSDP fue 15 veces más rápido que el más lento (revisar [Bor99]).

El sistema resuelve problemas del tipo:

$$\begin{array}{rrcl} \text{máx} & \text{Tr}(CX) & & \\ & A(X) & = & a \\ & X & \succeq & 0 \end{array} \tag{3.1}$$

donde

$$A(X) = \begin{bmatrix} \text{Tr}(A_1X) \\ \text{Tr}(A_2X) \\ \cdots \\ \text{Tr}(A_mX) \end{bmatrix}. \tag{3.2}$$

donde todas las matrices A_i, X y C se asumen reales y simétricas. El dual del problema anterior es

$$\begin{array}{rrcl} \text{mín} & a^{\text{T}}y & & \\ & A^{\text{T}}(y) - C & = & Z \\ & Z & \succeq & 0 \end{array} \tag{3.3}$$

donde

$$A^{\text{T}}(y) = \sum_{i=1}^{m} y_i A_i. \tag{3.4}$$

El formato de entrada de un problema es mediante un archivo basado en el sistema SDPA, llamado SDPA *sparse*. Por ejemplo se desea resolver el siguiente problema:

$$\begin{array}{rrcl} \text{máx} & \text{tr}\,(CX) & & \\ & \text{tr}\,A_1X & = & 1 \\ & \text{tr}\,A_2X & = & 2 \\ & X & \succeq & 0 \end{array} \tag{3.5}$$

donde

$$C = \begin{bmatrix} 2 & 1 & & & & & \\ 1 & 2 & & & & & \\ & & 3 & 0 & 1 & & \\ & & 0 & 2 & 0 & & \\ & & 1 & 0 & 3 & & \\ & & & & & 0 & \\ & & & & & & 0 \end{bmatrix} \tag{3.6}$$

$$A_1 = \begin{bmatrix} 3 & 1 & & & & & \\ 1 & 3 & & & & & \\ & & 0 & 0 & 0 & & \\ & & 0 & 0 & 0 & & \\ & & 0 & 0 & 0 & & \\ & & & & & 1 & \\ & & & & & & 0 \end{bmatrix} \tag{3.7}$$

$$A_2 = \begin{bmatrix} 0 & 0 & & & & & \\ 0 & 0 & & & & & \\ & & 3 & 0 & 1 & & \\ & & 0 & 4 & 0 & & \\ & & 1 & 0 & 5 & & \\ & & & & & 0 & \\ & & & & & & 1 \end{bmatrix}. \tag{3.8}$$

En este problema, X, Z, A_1, A_2 y C son matrices de 3 bloques. El primer bloque es una matríz de 2×2, el segundo bloque es una matríz de 3×3 y el tercero es un bloque diagonal de 2 entradas. El problema escrito en formato SDPA *sparse* es el siguiente:

```
2
3
2 3 -2
1.0000000000000000e+00 2.0000000000000000e+00
0 1 1 1 2.0000000000000000e+00
0 1 1 2 1.0000000000000000e+00
0 1 2 2 2.0000000000000000e+00
0 2 1 1 3.0000000000000000e+00
```

```
0 2 1 3 1.0000000000000000e+00
0 2 2 2 2.0000000000000000e+00
0 2 3 3 3.0000000000000000e+00
1 1 1 1 3.0000000000000000e+00
1 1 1 2 1.0000000000000000e+00
1 1 2 2 3.0000000000000000e+00
1 3 1 1 1.0000000000000000e+00
2 2 1 1 3.0000000000000000e+00
2 2 2 2 4.0000000000000000e+00
2 2 3 3 5.0000000000000000e+00
2 2 1 3 1.0000000000000000e+00
2 3 2 2 1.0000000000000000e+00
```

El 2 en la primera línea indica que el problema tiene 2 restricciones. El 3 en la segunda línea indica que hay 3 bloques en las matrices X y Z. La tercera línea entrega el tamaño de cada bloque (negativo si es un bloque diagonal). La cuarta línea indica los valores del lado derecho para cada restricción. Las líneas restantes contienen los valores de las celdas para las matrices C, A_1 y A_2. El primer número indica el número de la matríz, con C igual a 0. El segundo indica el bloque al cual pertenece, el tercero y cuarto son su fila y columnas con respecto a ese bloque y el quinto número es el valor de la celda.

Despues de resolver el problema, la solución se puede almacenar en un archivo de la siguiente forma:

```
7.4999999967481123e-01 9.9999999573633946e-01
1 1 1 1 2.5000001871068356e-01
1 1 1 2 -2.5000000032518932e-01
1 1 2 2 2.5000001871068356e-01
1 2 1 1 6.8952728511491658e-10
1 2 1 3 -4.2636602512977484e-10
1 2 2 2 2.0000000026316105e+00
1 2 3 3 1.9999999983679522e+00
1 3 1 1 7.5000001936105942e-01
1 3 2 2 1.0000000154225877e+00
2 1 1 1 1.2500000146708257e-01
2 1 1 2 1.2499999266458176e-01
2 1 2 2 1.2500000146708257e-01
2 2 1 1 6.6666967082089057e-01
2 2 1 3 -4.5183348114451421e-07
2 2 2 2 2.0062933863723688e-10
2 2 3 3 2.0063510893323200e-10
2 3 1 1 5.8683415560354947e-10
2 3 2 2 4.0125847850854105e-10
```

En este caso, en la primera fila están los valores de y. Las líneas que empiezan con 1, corresponden a los valores de Z, y las líneas que empiezan con 2 corresponden a X.

Como las matrices son simétricas, sólo se necesita almacenar la parte triangular superior de la matríz. De esta forma, la solución óptima del problema es (redondeando al tercer decimal):

$$y = \begin{bmatrix} 0{,}75 & 1{,}00 \end{bmatrix} \tag{3.9}$$

$$Z = \begin{bmatrix} 0{,}25 & -0{,}25 & & & & & & \\ -0{,}25 & 0{,}25 & & & & & & \\ & & 0 & 0 & 0 & & & \\ & & 0 & 2{,}00 & 0 & & & \\ & & 0 & 0 & 2{,}00 & & & \\ & & & & & 0{,}75 & & \\ & & & & & & 1{,}00 & \end{bmatrix} \tag{3.10}$$

$$X = \begin{bmatrix} 0{,}125 & 0{,}125 & & & & & & \\ 0{,}125 & 0{,}125 & & & & & & \\ & & 0{,}667 & 0 & 0 & & & \\ & & 0 & 0 & 0 & & & \\ & & 0 & 0 & 0 & & & \\ & & & & & 0 & & \\ & & & & & & 0 & \end{bmatrix} \tag{3.11}$$

Este software posee una implementación para sistemas de memoria distribuída llamada PCSDP [IK07], la cual utiliza la librerías MPI, BLAS, LAPACK y ScaLAPACK, y está basada en la versión 5.0 de CSDP. Las partes paralelizadas de este sistema se pueden ver en detalle en 2.4.4, las cuales incluyen la obtención de la matríz complemento de Schur M, su descomposición de Cholesky y la resolución de un sistema lineal, dividiendo los cálculos por proceso y utilizando las rutinas de ScaLAPACK `PDPOTRF` y `PDPOTRS` respectivamente.

En B se puede ver la instalación, ejecución y pruebas realizadas con este sistema.

3.2. Implementación utilizando cálculo paralelo

La implementación realizada, llamada `fnlsdp`, se basó en un trabajo previo desarrollado en MATLAB © para el articulo [GR06]. Para este trabajo se estudió esa implementación, se desarrollaron funciones equivalentes a las desarrolladas en MATLAB © (valor propio máximo y cálculo del problema QP), se implementó una fase de restauración y se implementó una versión en C del algoritmo, con el fin de servir como base para una aplicación de alto desempeño. En esta sección se revisarán los principales aspectos con respecto a la implementación realizada, se detallan las experiencias adquiridas, los problemas que se presentaron y la manera en que se resolvieron.

3.2.1. Consideraciones básicas

El primer paso para el desarrollo de la aplicación era la extracción de los datos de COMPleib desde MATLAB © hacia archivos de texto, para ser leídos desde C. El script desarrollado en lenguaje Perl se puede ver en el código 3.1.

La idea consiste en recorrer un listado de códigos de problemas, `codes.txt`, de la forma:

```
AC1
AC2
AC3
AC4
AC5
AC6
AC7
AC8
AC9
...
HF2D14
HF2D15
HF2D16
HF2D17
HF2D18
```

La lista completa se puede ver en [Lei04] y su descripción en 4. De esta forma, se leen los códigos de los problemas y se genera un script de MATLAB que realiza la extracción de los datos y su posterior escritura en un archivo de texto. El script generado se puede ver en el código 3.2, y la ejecución completa de esta parte se realiza de la siguiente forma:

```
[22:30 0.00][euler] % perl extractNames.pl > scriptCOMP.m
[22:30 0.01][euler] % matlab -nodisplay -nosplash -nodesktop -r "scriptCOMP;quit;"
```

El resultado se guarda en un directorio llamado `compleib_data`, donde se almacenan los archivos en el siguiente formato:

```
<code>_A.dat
<code>_B1.dat
<code>_B.dat
<code>_C1.dat
<code>_C.dat
<code>_D11.dat
<code>_D12.dat
<code>_D21.dat
<code>_dims.dat
```

donde `<codes>` es el código de cada problema que aparece en `codes.txt`.

3.2.2. Estructura de la aplicación

La estructura de la aplicación, siguió el patrón de la estructura del software CSDP, utilizado como base, y es la siguiente:

```
fnlsdp-1.0.1
|-- Makefile
|-- data
|   |-- compleib_data
|   `-- initial_points
|-- doc
|-- include
|-- lib
|-- solver
`-- src
```

El directorio `data/compleib_data` contiene a los archivos generados en la parte anterior y el directorio `data/initial_points` contiene a los puntos iniciales utilizados por el algoritmo, en el mismo formato que las matrices de `compleib_data`. Por ejemplo, para el problema `REA1`, las matrices $F_0 \in \mathbb{R}^{nu \times ny}$, $Q_0 \in \mathbb{R}^{nx \times nx}$ y $V_0 \in \mathbb{R}^{nx \times nx}$ iniciales para el algoritmo, se almacenan de la forma:

```
[operedo@syntagma initial_points]$ ls
REA1_01_F0.dat  REA1_01_Q0.dat  REA1_01_V0.dat
[operedo@syntagma initial_points]$ cat REA1_01_F0.dat
1.3473
2.9842
-1.7793
1.1992
0.1184
```

Código 3.1: Script en Perl de extracción de datos COMPleib

```
#!/usr/bin/perl
%vars=( "A" => "A",
  "B1" => "B1",
  "B" => "B",
  "C1" => "C1",
  "C" => "C",
  "D11" => "D11",
  "D12" => "D12",
  "D21" => "D21",
  "dims" => "nx nw nu nz ny");

open(IN,"<codes.txt");
$actuallines=0;
$numlines=0;
while(<IN>){
  $numlines=$numlines+1;
}
close(IN);

open(IN,"<codes.txt");
print "import java.io.PrintWriter;\n";
print "import java.io.FileWriter;\n";
while(<IN>){
  $actuallines=$actuallines+1;
  $code=$_;
  chomp($code);
  $code=~s/ //g;
  print "%saving problem ",$code," in compleib_data directory\n";
  print "disp(['saving ",$code,"..... ',num2str(100*$actuallines/$numlines),'/100']);\n";
  print "[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('",$code,"');\n";
  foreach $key (sort keys %vars){
    if($key eq "dims"){
      print "pw=java.io.PrintWriter(
          java.io.FileWriter('compleib_data/",$code,"_",$key,".dat')
          );\n";
      print "line=num2str(nx);\n";print "pw.println(line);\n";
      print "line=num2str(nw);\n";print "pw.println(line);\n";
      print "line=num2str(nu);\n";print "pw.println(line);\n";
      print "line=num2str(nz);\n";print "pw.println(line);\n";
      print "line=num2str(ny);\n";print "pw.println(line);\n";
      #print "end\n";
      print "pw.flush();\n";
      print "pw.close();\n"}
    else{
      print "pw=java.io.PrintWriter(
          java.io.FileWriter('compleib_data/",$code,"_",$key,".dat')
          );\n";
      print "for index=1:size(",$vars{$key},",1)\n";
      print "\tline=num2str(full(",$vars{$key},"(index,:)));\n";
      print "\tpw.println(line);\n";
      print "end\n";
      print "pw.flush();\n";
      print "pw.close();\n"
    }
  }
}
close(IN);
```

Código 3.2: Script generado para MATLAB de extracción y escritura de datos COMPleib

```
import java.io.PrintWriter;
import java.io.FileWriter;
%saving problem AC1 in compleib_data directory
disp(['saving problem AC1 ....... ',num2str(100*1/169),'/100']);
[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('AC1');
pw=java.io.PrintWriter(java.io.FileWriter('compleib_data/AC1_A.dat'));
for index=1:size(A,1)
        line=num2str(full(A(index,:)));
        pw.println(line);
end
pw.flush();
pw.close();
pw=java.io.PrintWriter(java.io.FileWriter('compleib_data/AC1_B.dat'));
for index=1:size(B,1)
        line=num2str(full(B(index,:)));
        pw.println(line);
end
pw.flush();
pw.close();
...
...
pw=java.io.PrintWriter(java.io.FileWriter('compleib_data/AC1_D21.dat'));
for index=1:size(D21,1)
        line=num2str(full(D21(index,:)));
        pw.println(line);
end
pw.flush();
pw.close();
pw=java.io.PrintWriter(java.io.FileWriter('compleib_data/AC1_dims.dat'));
line=num2str(nx);
pw.println(line);
line=num2str(nw);
pw.println(line);
line=num2str(nu);
pw.println(line);
line=num2str(nz);
pw.println(line);
line=num2str(ny);
pw.println(line);
pw.flush();
pw.close();
%saving problem AC2 in compleib_data directory
disp(['saving problem AC2 ....... ',num2str(100*2/169),'/100']);
[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('AC2');
pw=java.io.PrintWriter(java.io.FileWriter('compleib_data/AC2_A.dat'));
for index=1:size(A,1)
        line=num2str(full(A(index,:)));
        pw.println(line);
end
pw.flush();
pw.close();
...
```

```
-0.3040
[operedo@syntagma initial_points]$ cat REA1_01_Q0.dat
0.9557
0.116
-0.1798
0.09
0.116
0.9673
-0.0742
0.0985
-0.1798
-0.0742
1.0771
0.2301
0.09
0.0985
0.2301
0.6671
[operedo@syntagma initial_points]$ cat REA1_01_V0.dat
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
1
```

En este caso, $nu = 2$, $ny = 3$ y $nx = 4$ y las matrices son respectivamente:

$$F_0 = \begin{bmatrix} 1{,}3473 & -1{,}7793 & 0{,}1184 \\ 2{,}9842 & 1{,}1992 & -0{,}3040 \end{bmatrix} \tag{3.12}$$

$$Q_0 = \begin{bmatrix} 0{,}9557 & 0{,}116 & -0{,}1798 & 0{,}09 \\ 0{,}116 & 0{,}9673 & -0{,}0742 & 0{,}0985 \\ -0{,}1798 & -0{,}0742 & 1{,}0771 & 0{,}2301 \\ 0{,}09 & 0{,}0985 & 0{,}2301 & 0{,}6671 \end{bmatrix} \tag{3.13}$$

$$V_0 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}. \tag{3.14}$$

En el directorio `doc` se encuentra la documentación de PCSDP y la generada con Doxygen de las funciones para `fnlsdp`. En el directorio `include` se encuentran los siguiente archivos:

```
[operedo@syntagma fnlsdp-1.0.1]$ ls -lh include/
total 280K
-rwxrwxr-x  1 operedo operedo 2.0K Feb  1  2007 blockmat.h
-rwxrwxr-x  1 operedo operedo 8.7K Mar 12  2009 declarations.h
-rwxrwxr-x  1 operedo operedo 1.2K Feb  1  2007 index.h
-rwxrwxr-x  1 operedo operedo  836 Feb  1  2007 parameters.h
-rwxrwxr-x  1 operedo operedo  29K Mar 12  2009 scalapack.h
-rw-r-----  1 operedo operedo  18K Mar 12  2009 PBblacs.h
-rw-r-----  1 operedo operedo  29K Mar 11  2009 PBblas.h
-rw-r-----  1 operedo operedo  21K Mar 11  2009 pblas.h
-rw-r-----  1 operedo operedo  53K Mar 11  2009 PBpblas.h
-rw-r-----  1 operedo operedo  83K Mar 11  2009 PBtools.h
```

Los archivos `PBblacs.h`, `PBblas.h`, `PBpblas.h`, `PBtools.h` y `pblas.h` poseen las declaraciones de las rutinas de las librerías BLAS y BLACS en su versión paralela y se descargan siguiendo las instrucciones descritas en el archivo `INSTALL` ubicado en la carpeta inicial. Los archivos `blockmat.h`, `index.h` y `parameters.h` provienen de la aplicación PCSDP y no se deben modificar. El archivo `declarations.h` posee las declaraciones de todas las funciones de PCSDP y `fnlsdp`, por lo tanto cada nueva función que se desee agregar se debe actualizar en este archivo de encabezado.

El directorio `src` contiene a todos los archivos con las respectivas funciones de PCSDP y `fnlsdp`. En este directorio se realiza la programación de las funciones y su compilación como librería llamable, bajo el nombre de `libfnlsdp.a`.

El directorio `solver` contiene al archivo `fnlsdp.c` donde se realiza la selección del problema a resolver y el punto inicial utilizado. También se encuentra un script de ejemplo para ejecutar el programa usando un esquema SGE con encolamiento.

3.2.3. Resolución de $QP(x_k, \rho)$

Para implementar la función `QP_nsdp.m` se utilizó la aplicación PCSDP que resuelve en paralelo un problema SDP lineal. Los principales inconvenientes para utilizar PCSDP fueron la incompatibilidad de formatos entre CSDP y COMPleib, y la ejecución utilizando múltiples procesos.

Un desafío mayor fue la construcción de un conector entre el formato de entrada COMPleib y la formulación de $QP(x_k, \rho)$, hacia un formato CSDP de la forma (3.1). En el apéndice D se detalla el proceso de conversión de formato, junto a sus justificaciones y propiedades utilizadas.

Para realizar la paralelización de esta función, se utilizó la misma base de código que para PCSDP, es decir, se desarrollo `fnlsdp` sobre PCSDP, utilizando las mismas funciones originales de PCSDP.

Se implementaron los siguientes archivos:

- `fnlsdp_read_data.c`: realiza la lectura de datos desde el directorio `data` y contiene a las funciones que realizan lectura de matrices y datos en general.
- `fnlsdp_build_mats.c`: construye las matrices C y $A_1, \ldots, A_m$ descritas en (3.1), a partir de la formulación (D.8) basada en las matrices que definen a la planta LTI definida en (2.91). También contiene las funciones asociadas a la multiplicación de matrices.
- `fnlsdp_print_mats.c`: imprime información relativa a las matrices cargadas y construídas en el código.
- `fnlsdp_solve_qp.c`: calcula la solución del problema $QP(x_k, \rho)$ definido en (D.8).

Internamente, PCSDP utiliza principalmente dos rutinas de la librería ScaLAPACK para su funcionamiento: `PDPORTF` y `PDPOTRS`. Para ver su documentación completa, revisar:

```
http://www.netlib.org/scalapack/html/src/pdpotrf.f
http://www.netlib.org/scalapack/double/pdpotrs.f
```

El estudio del speedup obtenido de esta parte del código se puede revisar en el capítulo Resultados. El detalle de la implementación se puede revisar en el apéndice E.

3.2.4. Cálculo de $\theta(x_k)$

El cálculo de la función de mérito $\theta(x_k)$ en el caso particular de la formulación (1.2), descrita en (2.88), requiere calcular la norma matricial de dos términos y el cálculo del valor propio máximo de una matríz simétrica.

Para implementar la función `theta_ff.m`, la cual implementa la función $\theta(x_k)$ en MATLAB , se utilizó la rutina `PDSYEVX` proveniente de la librería ScaLAPACK. El principal incoveniente para utilizar esta rutina fue la dificultad en la utilización de la librería. Debido a la distribución y balance de carga que deben tener los procesos con respecto a las celdas de la matríz, este punto fue de especial dificultad.

La rutina `PDSYEVX` calcula el valor propio máximo de una matríz simétrica. En el apéndice A.4 se muestra un ejemplo de ejecución de la rutina `PDSYEVX` utilizando 4 procesos.

Para ver su documentación completa, revisar:

`http://www.netlib.org/scalapack/double/pdsyevx.f`

Se implementaron los siguientes archivos:

- `fnlsdp_objective_theta.c`: realiza el cálculo de la función (2.88) asociada a la formulación (1.2). También se realiza el cálculo de la función (2.87).

El estudio del speedup obtenido de esta parte del código se puede revisar en el capítulo Resultados. El detalle de la implementación se puede revisar en el apéndice E.

3.2.5. Fase de restauración

La implementación de una fase de restauración se encapsuló en una sola función, con el objetivo de probar diferentes alternativas. Por motivos de tiempo no se terminó de implementar todas las alternativas de fase de restauración, pero quedará como trabajo a futuro.

Para recordar qué se busca en la fase de restauración, se tiene que un punto x_k y un real ρ es válido para la fase de restauración si:

A1 $(\theta(x_k), f(x_k))$ *es aceptable para* $\mathcal{F}^{k-1}$

B1 $QP(x_k, \rho)$ *es factible*

El algoritmo utilizado es el mismo que utiliza la implementación MATLAB ©, implementado en la función nativa `fminsearch`, la cual implementa una versión del algoritmo *Nelder-Mead simplex* (optimización sin derivadas) descrito en [NM65]. En este algoritmo, se realiza una búsqueda en el espacio utilizando un *simplex*, como se ve en la figura 3.2 y los pasos del algoritmo se pueden ver en 7.

Se implementaron los siguientes archivos:

- `fnlsdp_filter.c`: contiene las funciones asociadas al manejo de los filtros $\mathcal{F}_k$.
- `fnlsdp_retoration.c`: contiene las funciones asociadas a la obtención de un punto aceptable por un filtro $\mathcal{F}_k$ y factible para el problema $QP(x_k, \rho)$. En particular, contiene la implementación de método de Nelder-Mead simplex.

En esta parte del desarrollo no se utilizó cálculo paralelo. El detalle de la implementación se puede revisar en el apéndice E.

Algoritmo 7 Algoritmo de Nelder-Mead simplex ó `Downhill`

1: $k \leftarrow 0$
2: Escoger un punto inicial $\overline{x}_k \in \mathbb{R}^N$.
3: **while** $k = 0, 1, 2, \ldots$ ó condición de parada **do**
4: Escoger $N + 1$ puntos de $\mathbb{R}^N$ alrededor de $\overline{x}_k$, formando un *simplex*: $x_1, x_2, \ldots, x_{N+1}$.
5: Calcular los valores de la function f en esos puntos, reindexando los puntos según el orden: $f(x_1) \leq f(x_2) \leq ... \leq f(x_{N+1})$.
6: Calcular x_0, centre de gravidad de todos los points salvo x_{N+1}.
7: Calcular $x_r = x_0 + (x_0 - x_{N+1})$ (reflexión de x_{N+1} por encima de x_0).
8: **if** $f(x_r) < f(x_1)$ **then**
9: Calcular $x_e = x_0 + 2(x_0 - x_{N+1})$ (estiramiento del simplex).
10: **if** $f(x_e) < f(x_r)$ **then**
11: Reemplazar x_{N+1} por x_e
12: **else**
13: Reemplazar x_{N+1} por x_r.
14: **end if**
15: Ir al paso 4.
16: **end if**
17: **if** $f(x_n) < f(x_r)$ **then**
18: Calcular $x_c = x_{N+1} + \frac{1}{2}(x_0 - x_{N+1})$ (contracción del simplex).
19: **if** $f(x_c) \leq f(x_r)$ **then**
20: Reemplazar x_{N+1} por x_c.
21: Ir al paso 4.
22: **end if**
23: **end if**
24: **if** Se alcanza un equilibrio en el orden de los puntos ($f(x_1) = \cdots = f(x_{n+1})$) **then**
25: $\overline{x}_k \leftarrow x_0 + \frac{1}{2}(\overline{x}_k - x_1)$.
26: $k \leftarrow k + 1$
27: Ir al paso 4.
28: **end if**
29: **end while**

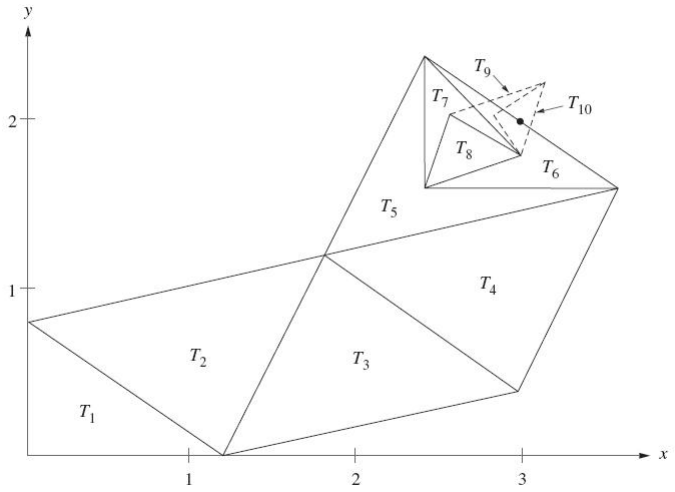

Figura 3.2: Esquema del algoritmo Nelder-Mead simplex

3.2.6. Operaciones algebraicas

Todas las operaciones algebraicas utilizan las funciones de BLAS y LAPACK descritas en 3.1. En particular, la principal rutina utilizada es `DGEMM`, pertenenciente al nivel 3 de BLAS y la cual realiza la operación:

$$C \leftarrow \alpha AB + \beta C$$

Para utilizar cálculo paralelo, se debe utilizar la rutina `PDGEMM`, la cual realiza la multiplicación distribuyendo la carga de las filas y columnas según las distribuciones descritas en 2.4.4.

En esta parte del desarrollo no se utilizó cálculo paralelo. La documentación de la rutina `PDGEMM` se puede ver en:

`http://www.netlib.org/scalapack/pblas_qref.html#PvGEMM`

3.2.7. Compilación y ejecución

Para la compilación de la aplicación, primero se debe editar el archivo `solver/Makefile` de la siguiente manera. Primero se deben ubicar las líneas:

```
MKLLIB=/opt/intel/mkl/10.0.3.020/lib/64
BLAS= -lmkl_lapack -lmkl_ipf -lguide -lpthread -lgfortran
```

Se puede observar que se está utilizando la versión 10.0.3.020 de la librería *Intel MKL* en su modalidad para arquitecturas IA-64. Para la modalidad para arquitecturas Intel 64 se deben modificar las líneas de la siguiente forma:

```
MKLLIB=/opt/intel/mkl/10.0.3.020/lib/em64t
BLAS= -lmkl_lapack -lmkl_em64t -lguide -lpthread -lgfortran
```

La compilación se realiza ejecutando el `Makefile` de la forma:

```
[operedo@syntagma fnlsdp-1.0.1]$ make
```

Para la ejecución de la aplicación, se debe especificar en archivo `solver/params.fnlsdp` el código del problema y el identificador de los puntos iniciales a utilizar (pueden haber diferentes puntos inciales a utilizar, generados con distintos métodos). Por ejemplo, para ejecutar la aplicación con 4 procesos:

```
[operedo@syntagma fnlsdp-1.0.1/solver]$ mpirun -np 4 ./fnlsdp
```

Si se está utilizando una sistema de colas tipo SGE, se debe revisar el archivo `script.sh` para revisar el modo de uso. Su ejecución en este caso es:

```
[operedo@syntagma fnlsdp-1.0.1/solver]$ qsub script.sh
```

Y la revisión de su estado (todos los procesos que están encolados y en ejecución dentro del servidor) se realiza con `qstat`:

```
[operedo@syntagma fnlsdp-1.0.1/solver]$ qstat
job-ID  prior   name       user   state submit/start at  queue             slots
-----------------------------------------------------------------------------
 152035 0.55500 fcab-test  fcab   r     02/25/2010 17:02 all.q@compute-1-1 2
 152037 0.55500 fcab-test  fcab   r     02/25/2010 17:57 all.q@compute-1-3 2
```

3.3. Diseño de distintas fases de restauración

Debido a que la implementación en MATLAB © no siempre convergía, se decidió estudiar en profundidad una parte crítica del algoritmo, la fase de restauración, y proponer algún método para reemplazar la fase existente.

Se estudiaron diferentes fases de restauración para el algoritmo **Filter-SDP**. Conviene recordar que la fase de restauración del algoritmo es la siguiente:

Encontrar x_k *y* $\rho > \overline{\rho}$ *tales que:*

A1 $(\theta(x_k), f(x_k))$ *es aceptable para* $\mathcal{F}^{k-1}$

B1 $QP(x_k, \rho)$ *es factible*

Se revisararán 3 enfoques, además del enfoque original utilizado en la implementación existente.

Conviene recordar que el problema a resolver tiene la forma (1.2) y la variable x se representa de la forma $x = (F, Q, V)$ con $F \in \mathbb{R}^{p\times r}$ y $Q, V \in \mathbb{S}^n$, y se utilizarán las funciones $f(x)$ y $\theta(x)$ descritas en (2.87) y (2.88).

3.3.1. Enfoque original

El enfoque original que se encontraba implementado consistía en obtener un punto x_k minimizando la función de mérito $\theta(x)$ con un algoritmo nativo de MATLAB ©. Una vez minimizada esa función se verificaba si la solución obtenida admitía la factibilidad del problema $QP(x_k, \rho)$. El algoritmo se puede ver en 8 y el código asociado en C.1.

Algoritmo 8 Fase de restauración original

$N \leftarrow$ número de veces que se realiza la búsqueda
$\rho_{\text{máx}} \leftarrow$ radio máximo de la región de confianza de $QP(x_k, \rho)$
$\rho \leftarrow (0, \rho_{\text{máx}})$
$x_k \leftarrow$ punto inicial o proveniente de la iteración $k-1$
while ((x_k no es aceptable para $\mathcal{F}_{k-1}$) $\vee$ ($QP(x_k, \rho)$ no es factible)) $\wedge$ paso $\leq N$ **do**
    $x_k \leftarrow$ `fminsearch`$(\theta(\cdot), x_k)$
    if x_k es aceptable para $\mathcal{F}_{k-1}$ **then**
        while ($QP(x_k, \rho)$ no es factible) $\wedge$ ($\rho < \rho_{\text{máx}}$) **do**
            $\rho \leftarrow 2 * \rho$
            $d_k \leftarrow QP(x_k, \rho)$ (*si* $QP(x_k, \rho)$ *no es factible,* d_k *queda indefinido*)
        end while
    end if
    paso $\leftarrow$ paso $+ 1$
end while

La función nativa de MATLAB ©, `fminsearch`$(\theta(\cdot), x_k)$, implementa una versión del algoritmo *Nelder-Mead simplex* (optimización sin derivadas) descrito en [NM65].

La principal desventaja de esta implementación es que en el caso de no encontrar un punto aceptable por el filtro y factible para $QP(x_k, \rho)$, solamente se evalua una nueva minimización con `fminsearch` partiendo del mismo punto. Al estar realizando una nueva minimización partiendo del mismo punto, lo más probable es que se obtenga como solución ese mismo punto, quedando estancado el algoritmo. Lo que se hizo en el desarrollo anterior fue obtener puntos factibles para el problema, utilizando métodos externos,

perturbarlos y utilizarlos como puntos iniciales en el algoritmo. De esta forma, siempre se partía de puntos cercanos a algún óptimo local.

Lo anterior motivó la investigación de otros métodos donde se exploran diferentes puntos en caso de llegar a estancamientos. En las siguiente partes se detallan esos métodos.

3.3.2. Retauración inexacta

En [SM08] se detalla una fase de restauración para el problema de programación no lineal

$$\begin{array}{ll} \text{mín} & f(x) \\ & c_i(x) \leq 0,\ i = 1, \ldots, m \end{array} \tag{3.15}$$

El algoritmo propuesto se compone de 2 fases, una de factibilidad y otra de optimalidad. La fase de factibilidad resuelve el siguiente problema para un punto x_k, factible o no factible:

$$\begin{array}{lll} LP(x_k): & \min\limits_{d \in \mathbb{R}^n} & \sum_{i \in J} \nabla c_i(x_k)^{\mathrm{T}} d \\ & s.a. & \nabla c_i(x_k)^{\mathrm{T}} d + c_i(x_k) \leq 0,\ i \in J^* \end{array} \tag{3.16}$$

donde J son las restricciones no satisfechas y J^* las satisfechas. La solución d se utiliza como dirección para definir un punto $z_k = x_k + \alpha d$ el cual debe satisfacer una condición de *mayor factibilidad* que el punto x_k, en el sentido de minimización de la función de mérito $h(x) = \| \sum_{i=1}^{m} \text{máx}\{0, c_i(x)\} \|$:

$$h(z_k) \leq (1 - \gamma) h(x_k) \tag{3.17}$$

y además se exige que el punto z_k sea aceptable por el filtro $\mathcal{F}_k$. En caso contrario, se reduce el valor de α hasta que se llega a una cierta tolerancia o hasta que el punto z_k es aceptado por el filtro y satisface (3.17).

Si el punto z_k pasó con exito la fase de factibilidad anterior, el paso siguiente consiste en resolver una fase de optimalidad para z_k:

$$\begin{array}{lll} QP(z_k): & \min\limits_{d \in \mathbb{R}^n} & \nabla f(z_k) d + \frac{1}{2} d^T \nabla^2 L(z_k, \lambda_k) d \\ & s.a. & \nabla c_i(x_k)^{\mathrm{T}} d + c_i(x_k) \leq 0 \end{array} \tag{3.18}$$

La solución d se utiliza para definir el punto $x_{k+1} = z_k + \alpha d$, el cual debe satisfacer una condición de *mayor optimalidad* que el punto z_k, minimizando la función objetivo $f(x)$:

$$f(x_{k+1}) \leq (1 - \gamma) f(z_k) \tag{3.19}$$

y además se exige que el punto x_{k+1} sea aceptable por el filtro $\mathcal{F}_k$.

La adaptación realizada para el problema (3.20):

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & f(x) \\ \text{s.a} & h(x) = 0 \\ & G(x) \preceq 0 \end{array} \tag{3.20}$$

consiste en definir las restricciones pertenecientes a J y J^*, primero revisando $h(x)$. Si $\|h_i(x)\| < \epsilon_{TOL}$, entonces $i \in J^*$, y si $\|h_i(x)\| \geq \epsilon_{TOL}$, entonces $i \in J$. Luego se analizan los valores propios de $G(x)$. Suponiendo que $G(x) = PDP^{-\mathrm{T}}$, con P la matríz de vectores propios y D valores propios, ordenados de manera descendente, se escoge $i \in J^*$ si $\lambda_i(G(x)) \leq 0$. De esta manera, todos los valores propios de $G(x)$ son negativos, positivos o existe $\bar{i}$ tal que $i \leq \bar{i}$ implica $i \in J^*$ y $i > \bar{i}$ implica $i \in J$. Llamando a ambos conjuntos de índices J_G^- y J_G^+ respectivamente, el problema $LP(x)$ descrito en (3.16) se escribe de la forma:

$$\begin{array}{lll} LP(x_k): & \min\limits_{d\in\mathbb{R}^n} & \sum_{i=1}^{|J_G^+|} \lambda_i(G(x_k) + DG(x_k)d) \\ & & +\sigma \sum_{j\in J} \|h_j(x_k) + Dh_j(x_k)d\|^2 \\ & \text{s.a} & h_j(x_k) + Dh_j(x_k)d = 0,\ j \in J^* \\ & & E^{\mathrm{T}}(G(x_k) + DG(x_k)d)E \preceq 0 \end{array} \tag{3.21}$$

donde E es la matríz cuyas columnas son los vectores propios asociados a los índices J_G^-.

Para resolver el problema (3.21), se utiliza el siguiente programa semidefinido, cuya construcción se detalla en los algoritmos 10 y 11:

$$\begin{array}{lll} \min\limits_{t_1,t_2,s\in\mathbb{R},d\in\mathbb{R}^n,Z\in\mathbb{S}^n} & t_1 + \sigma t_2 & \\ \text{s.a} & h_j(x_k) + Dh_j(x_k)d = 0,\ j \in J^* \\ & E^{\mathrm{T}}(G(x_k) + DG(x_k)d)E \preceq 0 \\ & t_1 - rs - \mathrm{Tr}(Z) \geq 0 \\ & Z - (G(x_k) + DG(x_k)d) + sI \succeq 0 \\ & t_2 - \|(h_j(x_k) + Dh_j(x_k)d)_{j\in J}\| \geq 0 \\ & Z \succeq 0 \end{array} \tag{3.22}$$

donde r es el índice asociado a J_G^+, es decir, $r = |J_G^+|$. Para la deducción de este programa se utilizaron resultados asociados a valores propios vistos en 2.1.1. Los códigos asociados se pueden ver en C.2, C.3, C.4 y C.5.

Con esto, se tiene un método para minimizar la función de mérito partiendo de un punto x_k. La solución del programa (3.22) se utiliza de la forma $z_k = x_k + \alpha d$, analogamente a lo que se describe para el caso no lineal. Este método pretende ser una alternativa a la utilización de la función `fminsearch`, sin embargo, no produjo buenos resultados debido a que tambien se produce estancamiento si es que no se cuenta con un método para revisar distintas áreas de la región factible (o cercanas a la región factible), utilizando puntos iniciales cada vez que el método se quede estancado. Debido a esto, se estudiaron métodos de generación de puntos iniciales para el problema SOF, el cual se detalla en el capítulo Resultados.

Algoritmo 9 Fase de restauración inexacta

$\beta \in (0, 1)$
$\gamma \in (0, \beta)$
$N \leftarrow$ número de veces que se realiza la búsqueda
$\rho_{\text{máx}} \leftarrow$ radio máximo de la región de confianza de $QP(x_k, \rho)$
$\rho \leftarrow (0, \rho_{\text{máx}})$
$x_k \leftarrow$ punto inicial o proveniente de la iteración $k - 1$
while ((x_k no es aceptable para $\mathcal{F}_{k-1}$) $\vee$ ($QP(x_k, \rho)$ no es factible)) $\wedge$ paso $\leq N$ **do**
    $x_k \leftarrow \texttt{lsdp}(\theta(\cdot), x_k, \mathcal{F}_k, \beta, \gamma)$
    if x_k es aceptable para $\mathcal{F}_{k-1}$ **then**
        while ($QP(x_k, \rho)$ no es factible) $\wedge$ ($\rho < \rho_{\text{máx}}$) **do**
            $\rho \leftarrow 2 * \rho$
            $d_k \leftarrow QP(x_k, \rho)$ (*si* $QP(x_k, \rho)$ *no es factible,* d_k *queda indefinido*)
        end while
    end if
    paso $\leftarrow$ paso $+ 1$
end while

Algoritmo 10 Función `lsdp` para restauración inexacta (versión 1)

INPUT: $x_k, \mathcal{F}_{k-1}, \beta, \gamma$
$\alpha \leftarrow \pm 1$ (elegir el signo del descenso)
$ac \leftarrow 0$
$\epsilon \leftarrow 10^{-4}$
Contruir problema (3.22) para el punto x_k.
Obtener d solución de (3.22).
while $ac = 0 \wedge |\alpha| \geq \epsilon$ **do**
    $z = x_k + \alpha d$
    if z es aceptable por $\mathcal{F}_{k-1} \wedge \theta(x_k) > \theta(z)$ **then**
        $ac \leftarrow 1$
    else
        $\alpha \leftarrow \frac{\alpha}{2}$
    end if
end while

Algoritmo 11 Función `lsdp` para restauración inexacta (versión 2)

1: INPUT: $x_k, \mathcal{F}_{k-1}, \beta, \gamma$
2: $\alpha \leftarrow \pm 1$ (elegir el signo del descenso)
3: $ac \leftarrow 0$
4: $\epsilon \leftarrow 10^{-4}$
5: Contruir problema (3.22) para el punto x_k.
6: Agregar restricciones asociadas al filtro para $d = (F_d, Q_d, V_d) \in \mathbb{R}^{p\times r} \times \mathbb{S}^n \times \mathbb{S}^n$:

$$t - \frac{1}{3}\overline{\theta} \geq 0 \tag{3.23}$$

$$tI_{n\times n} + V_d \succeq 0_{n\times n} \tag{3.24}$$

donde $\overline{\theta}$ se define como:

$$\overline{\theta} = \min_{(\theta_i, f_i)\in\mathcal{F}_{k-1}} \theta_i$$

7: Obtener d solución de (3.22).
8: **while** $ac = 0 \wedge |\alpha| \geq \epsilon$ **do**
9: $z = x_k + \alpha d$
10: **if** z es aceptable por $\mathcal{F}_{k-1} \wedge \theta(x_k) > \theta(z)$ **then**
11: $ac \leftarrow 1$
12: **else**
13: $\alpha \leftarrow \frac{\alpha}{2}$
14: **end if**
15: **end while**

3.3.3. Soluciones SOF suboptimales

Un enfoque diferente se utilizó tomando como base el trabajo realizado en [Mos08], donde se detalla un método para encontrar soluciones suboptimales del problema asociado a los controladores de salida estática con retroalimentación (*Static-Output Feedback*). Este tipo de problemas consiste en encontrar una matríz F tal que $u(t) = Fy(t)$, que minimiza el funcional $J(F)$:

$$J(F) := \mathbb{E}\left(\int_0^\infty (x(t)^{\mathrm{T}} Q x(t) + u(t)^{\mathrm{T}} R u(t)) dt\right) \tag{3.25}$$

con $Q \succeq 0$ y $R \succ 0$, sujeto a la siguiente dinámica:

$$\begin{aligned} x'(t) &= Ax(t) + Bu(t) && (3.26)\\ x(0) &= x_0 && (3.27)\\ y(t) &= Cx(t) && (3.28) \end{aligned}$$

Usando (3.28) y $u(t) = Fy(t)$, se puede reemplazar $u(t)$ por $FCx(t)$ en (3.26), y suponiendo que F permite que el sistema sea asintóticamente estable, es decir

$$\alpha(A + BFC) := \max_i\{\mathrm{Re}(\lambda_i(A + BFC))\} < 0 \tag{3.29}$$

la solución del sistema $x'(t) = (A + BFC)x(t), x(0) = x_0$ será $x(t) = e^{(A+BFC)t}$, y se puede reemplazar dentro del funcional $J(F)$ obteniendose:

$$J(F) := \mathrm{Tr}\left(P \underbrace{\int_0^\infty e^{(A+BFC)^{\mathrm{T}}t}(C^{\mathrm{T}}F^{\mathrm{T}}RFC + Q)e^{(A+BFC)t}dt}_{L(F)}\right) \tag{3.30}$$

con $P = \mathbb{E}(x_0 x_0^{\mathrm{T}})$. Recordando el teorema que caracteriza a la solución de la ecuación de Lyapunov (revisar demostración en [Che70]):

Teorema 3.3.1. *Si $\alpha(A) < 0$, entonces para toda matríz N existe una única matríz M que satisface la ecuación*

$$A^T M + MA + N = 0 \tag{3.31}$$

y además

$$M = \int_0^\infty e^{A^T t} N e^{At} dt \tag{3.32}$$

, se tiene que la matríz $L(F)$ es solución de la ecuación de Lyapunov siguiente:

$$L(F)(A + BFC) + (A + BFC)^{\mathrm{T}} L(F) + C^{\mathrm{T}}F^{\mathrm{T}}RFC + Q = 0 \tag{3.33}$$

y considerando a la variable $L(F)$ independiente de F, el problema (equivalente) que se resuelve entonces es:

$$\begin{array}{cc} \min\limits_{(F,L)\in S_F\times\mathbb{R}^{n\times n}} & \text{Tr}(PL) \\ \text{s.a} & L(A+BFC)+(A+BFC)^{\text{T}}L+C^{\text{T}}F^{\text{T}}RFC+Q=0 \end{array} \tag{3.34}$$

donde $S_F = \{F : \alpha(A+BFC) < 0\}$.

El método propuesto en [Mos08] minimiza el siguiente problema perturbado:

$$\begin{array}{cc} \min\limits_{(F,\mu)\in S_F^{\mu}} & \text{Tr}(PL(F,\mu))+\sigma\mu^2 \\ \text{s.a} & L(F,\mu)\bar{A}_{\mu}+\bar{A}_{\mu}^{\text{T}}L(F,\mu)+C^{\text{T}}F^{\text{T}}RFC+Q=0 \end{array} \tag{3.35}$$

donde $\bar{A}_{\mu} = A-\mu I+BFC$, $S_F^{\mu} = \{(F,\mu) : \alpha(\bar{A}_{\mu}) < 0\}$ y $\sigma > 0$ es un parámetro de penalización. La idea es realizar una minimización irrestricta reduciendo progresivamente σ.

En [Mos08] se demuestra que las condiciones de primer orden para la función

$$J_F^{\sigma}(F,\mu) = \text{Tr}(PL(F,\mu))+\sigma\mu^2 \tag{3.36}$$

son las siguientes:

$$\nabla_K J_F^{\sigma}(F,\mu) := 2(B^{\text{T}}L(F,\mu)+RFC)K(F,\mu)C^{\text{T}} = 0 \tag{3.37}$$

$$\nabla_{\mu} J_F^{\sigma}(F,\mu) := 2(\text{Tr}(L(F,\mu)K(F,\mu))+\sigma\mu) = 0 \tag{3.38}$$

$$L(F,\mu)\bar{A}_{F,\mu}+\bar{A}_{F,\mu}^{\text{T}}L(F,\mu)+C^{\text{T}}F^{\text{T}}RFC+Q = 0 \tag{3.39}$$

$$\bar{A}_{F,\mu}K(F,\mu)+K(F,\mu)\bar{A}_{F,\mu}^{\text{T}}+P = 0 \tag{3.40}$$

donde $K(F,\mu)$ es una variable auxiliar, solución de (3.40). Además, se demuestra una caracterización para la matríz F, conociendo L y K:

Lema 3.3.2. *Sean $(F,\mu) \in S_F^{\mu}$ solución del problema* (3.35)*. Suponiendo que la matríz C es de rango completo y que S_F^{μ} es no vacío, entonces*

$$F(L,K) = -R^{-1}B^{T}LKC^{T}(CKC^{T})^{-1} \tag{3.41}$$

Utilizando $P = Q = I_{n\times n}$ y $R = I_{p\times p}$, se obtiene una solución para la ecuación:

$$\bar{A}_{F,\mu}K+K\bar{A}_{F,\mu}^{\text{T}}+I_{n\times n} = 0 \tag{3.42}$$

la cual proviene de la ecuación (3.40). Esta ecuación coincide con la tercera restricción de la formulación (1.2), por lo tanto el método permite encontrar una matríz factible para esa restricción. Para encontrar la matríz que satisface la primera restricción de (**??**), se resuelve independientemente una ecuación de Lyapunov de la forma:

$$A_F L+LA_F^{\text{T}}+B_1B_1^{\text{T}} = 0 \tag{3.43}$$

De esta forma, se intenta obtener una solución factible para el problema (1.2), la cual servirá como punto inicial para el algoritmo **Filter-SDP**.

Con las consideraciones anteriores, el algoritmo en detalle se puede ver en 12 y su código en C.6. Lo que se espera de este método es obtener puntos iniciales cuya función de mérito θ (ver (2.88)) sea menor (*mayor* factibilidad) que con el enfoque original, el cual usaba puntos a priori cercanos a un óptimo local. En este caso, partiendo de un punto cualquiera, se espera llegar a un punto inicial de mayor factibilidad para comenzar a realizar las iteraciones del algoritmo global.

Algoritmo 12 Cálculo de controladores SOF suboptimales usando penalización

1: Calcular μ_0 tal que $(F_0, \mu_0) = (0, \mu_0) \in S_F^{\mu}$ y resolver (3.39) y (3.40) para obtener L_0 y K_0.
2: Escogen parámetros iniciales $0 < \tau < \frac{1}{2}$, $a > 1$, $b > 0$, $\gamma \in (0, 1)$, $\epsilon_{stab} < 1$, $\epsilon_0 > 0$ y $\sigma_0 > 0$.
3: **for** $j = 0, 1, \ldots$ **do**
4: $k = 0$
5: **while** $\min\{\|\nabla_F J_F^{\sigma_j}(F_k, \mu_k)\|, \|\nabla_\mu J_F^{\sigma_j}(F_k, \mu_k)\|\} \geq \epsilon_j$ **do**
6: Calcular $\mu_{k+1}^{\sigma_j} = -\frac{1}{\sigma_j}\text{Tr}(L_k K_k)$.
7: Calcular $F_{k+1} = -R^{-1}B^{\text{T}}L_k M_k C^{\text{T}}(CM_kC^{\text{T}})^{-1}$.
8: $i = 0$
9: **while** $\alpha(\bar{A}_{F_{k+1}, \mu_{k+1}^{\sigma_j}}) \geq 0$ **do**
10: $F_{k+1} = F_k + \gamma^i \tau (F_{k+1} - F_k)$
11: **if** $\gamma^i \tau \leq \epsilon_{stab}$ **then**
12: Parar.
13: **end if**
14: $i = i + 1$
15: **end while**
16: Dados F_{k+1} y $\mu_{k+1}^{\sigma_j}$ resolver las ecuaciones (3.39) y (3.40) para obtener L_{k+1} y M_{k+1}.
17: $k = k + 1$
18: **end while**
19: Escoger $\sigma_{j+1} \in (0, a\sigma_j)$ y $\epsilon_{j+1} \in (0, \epsilon_j)$.
20: $F_{j+1} := F_k$, $\mu_{j+1} := \mu_k$.
21: **end for**

3.3.4. Posicionamiento de polos

El método de soluciones SOF suboptimales se inicializa con F_0 igual a la matríz cero con dimensiones apropiadas (etapa 1 del algoritmo 12). Para mejorar el desempeño del algoritmo 12, se buscó algún método que entregara una matríz F dando como entrada

Algoritmo 13 Fase de restauración SOF

$N \leftarrow$ número de veces que se realiza la búsqueda
$\rho_{\text{máx}} \leftarrow$ radio máximo de la región de confianza de $QP(x_k, \rho)$
$\rho \leftarrow (0, \rho_{\text{máx}})$
if $k = 0$ **then**
  $x_k \leftarrow$ `pen`
else
  $x_k \leftarrow$ punto inicial o proveniente de la iteración $k - 1$
end if
while ((x_k no es aceptable para $\mathcal{F}_{k-1}$) $\vee$ ($QP(x_k, \rho)$ no es factible)) $\wedge$ paso $\leq N$ **do**
  $x_k \leftarrow$ `fminsearch`$(\theta(\cdot), x_k)$ ó $x_k \leftarrow$ `lsdp`$(\theta(\cdot), x_k, \mathcal{F}_k, \beta, \gamma)$
  if x_k es aceptable para $\mathcal{F}_{k-1}$ **then**
    while ($QP(x_k, \rho)$ no es factible) $\wedge$ ($\rho < \rho_{\text{máx}}$) **do**
      $\rho \leftarrow 2 * \rho$
      $d_k \leftarrow QP(x_k, \rho)$ (*si* $QP(x_k, \rho)$ *no es factible,* d_k *queda indefinido*)
    end while
  end if
  paso $\leftarrow$ paso $+ 1$
end while

los valores propios que debe tener la matríz $A + BFC$. Este problema, llamado *posicionamiento de polos*, se describe de la siguiente forma: para las matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ y $C \in \mathbb{R}^{p\times n}$, y el vector $\lambda^D \in \mathbb{C}^n$, encontrar la matríz $F \in \mathbb{R}^{m\times p}$ tal que $\lambda(A + BFC) = \lambda^D$, con $\lambda(A)$ el vector de valores propios asociado a la matríz A.

La idea de este método es utilizar una matríz $A+BF_kC$ de tal forma que al calcular las ecuaciones de Lyapuov del método SOF de la parte anterior, (3.39) y (3.40), haya mayor chance de obtener soluciones L_{k+1} y M_{k+1}, pues el sistema estará mejor condicionado, en el sentido que se obtiene una estabilidad asintótica del sistema con $\alpha(A + BF_kC) < 0$ y se satisface la condición del teorema (3.3.1) desde las primeras iteraciones del método de soluciones SOF suboptimales.

El algoritmo estudiado se describe en [YO07]. En él se resuelve un problema de mínimos cuadrados de la siguiente forma:

$$\text{mín}_{F\in\mathbb{R}^{m\times p}} \quad f(F) := \tfrac{1}{2}\|\lambda(A + BFC) - \lambda^D\|_2^2 \tag{3.44}$$

La función $f(F)$ se puede escribir de la forma:

$$f(F) = \frac{1}{2}\sum_{i=1}^{n} \underbrace{\left(\lambda_i(A + BFC) - \lambda_i^D\right)^*}_{r_i^*(F)} \underbrace{\left(\lambda_i(A + BFC) - \lambda_i^D\right)}_{r_i(F)} \tag{3.45}$$

Las condiciones necesarias para que exista una solución a este problema son la (A, B)-controlabilidad y la (A, C)-observabilidad. Una condicion suficiente es la siguiente: $mp >$

n ([Wan92]).

Para resolver (3.44), en [YO07] se utiliza un método de región de confianza que minimiza iterativamente una aproximación local cuadrática:

$$\begin{array}{rl} \min_{p\in\mathbb{R}^n} & m_k(p) := f(x_k) + \nabla f(x_k)^{\mathrm{T}} p + p^{\mathrm{T}} B_k p \\ \text{s.a} & \|p\|_2 \leq \Delta_k \end{array} \tag{3.46}$$

La matríz B_k se puede construir usando información de primer o segundo orden. Si se usa información de primer orden, el método se llama *Levenberg-Marquardt* [Mar63]:

$$B_k = J(x_k)^{\mathrm{T}} J(x_k) \tag{3.47}$$

con $J(x)$ la matríz Jacobiana de la función $r(x) = (r_1(x), \ldots, r_n(x))^{\mathrm{T}}$, asociada a la función objetivo $f(x) = \frac{1}{2}\sum_{i=1}^{n} r_i^*(x) r_i(x)$. Si se usa información de segundo orden, el método recibe el nombre de *región de confianza de Newton* [NW99]:

$$B_k = J(x_k)^{\mathrm{T}} J(x_k) + \sum_{i=1}^{n} r_i(x_k) \nabla^2 r_i(x_k) \tag{3.48}$$

Ahora, como la función involucra valores propios, conviene recordar el siguiente resultado ([Kat82], sección 2.5.7), relativo a las derivadas de los valores propios de una matríz:

Teorema 3.3.3. *Considere una función matricial $A : \mathbb{R}^N \to \mathbb{R}^{n\times n}$. Si $A(x)$ es k veces continuamente diferenciable en una vecindad Ω de x, y además en cada punto $\tilde{x}$ de Ω $A(\tilde{x})$ tiene distintos valores propios, entonces los valores propios de $A(x)$ son k veces continuamente diferenciables en Ω.*

Con este teorema en mente, si λ_i denota el i-ésimo valor propio de $A(x)$, $D = (\lambda_1, \ldots, \lambda_n)^{\mathrm{T}}$ y $X \in \mathbb{C}^{n\times n}$ tal que $A(x)X = XD$, se tiene que:

$$\frac{\partial \lambda_i}{\partial x_k} = \left(X^{-1} \frac{\partial A(x)}{\partial x_k} X \right)_{ii} \tag{3.49}$$

$$\frac{\partial^2 \lambda_i}{\partial x_k \partial x_l} = \left(X^{-1} \frac{\partial^2 A(x)}{\partial x_k \partial x_l} X \right)_{ii} + \sum_{j=1, j\neq i}^{n} \frac{P_{ij}Q_{ji} + P_{ji}Q_{ij}}{\lambda_i - \lambda_j} \tag{3.50}$$

donde $P = X^{-1}\frac{\partial A(x)}{\partial x_k}X$ y $Q = X^{-1}\frac{\partial A(x)}{\partial x_l}X$. En este caso, la función objetivo es (3.45), al derivar se obtiene:

$$\frac{\partial f(F)}{\partial F_{kl}} = \mathrm{Re}\left\{ \sum_{i=1}^{n} (\lambda_i(A + BFC) - \lambda_i^D)^* \frac{\partial \lambda_i(A + BFC)}{\partial F_{kl}} \right\} \tag{3.51}$$

$$\frac{\partial^2 f(F)}{\partial F_{kl} \partial F_{pq}} = \mathrm{Re}\left\{ \sum_{i=1}^{n} \left(\frac{\partial \lambda_i(A + BFC)}{\partial F_{kl}} \right)^* \left(\frac{\partial \lambda_i(A + BFC)}{\partial F_{pq}} \right) \right. \tag{3.52}$$

$$\left. + \sum_{i=1}^{n} (\lambda_i(A + BFC) - \lambda_i^D)^* \frac{\partial^2 \lambda_i(A + BFC)}{\partial F_{kl} \partial F_{pq}} \right\} \tag{3.53}$$

$$\tag{3.54}$$

En este caso, $A(x) = A + BFC$, luego:

$$\frac{\partial(A + BFC)}{\partial F_{kl}} = B_{\cdot,k}C_{l,\cdot} \tag{3.55}$$

Con los cálculos anteriores, el algoritmo se puede ver en 15 y su código combinado con el enfoque SOF en C.7. Las funciones originales descritas en [YO07] se pueden ver en los códigos C.8, C.9, C.10, C.11 y C.12.

En el algoritmo 16 se puede observar que en el caso de que no se encuentren puntos aceptables y factibles, se realiza una nueva búsqueda de un punto incial con el algoritmo de posicionamiento de polos (líneas 19 y 20), ya que utiliza una componente aleatoria que permite explorar nuevas áreas de la región factible, con la esperanza de encontrar nuevos puntos que sean soluciones de la fase de restauración. Esa componente aleatoria se describe en el algoritmo 14, en él se calcula el tamaño de la matríz A mediante una función de su norma (la ráiz de su norma en este caso) y se construye un vector λ^D ajustado a las dimensiones del problema. Posteriormente se escala, restandole a cada componente de λ^D el factor $\max_i(\text{Re}\{\lambda^D\})$ para obtener sólo valores negativos y además se les resta un valor δ que corresponde a un valor input asociado al número de intentos por encontrar un nuevo punto aceptable y factible. La idea es que mientras más intentos se han realizado, más negativos serán los polos en los cuales estoy posicionando la matríz F_k, con el objetivo de forzar la extracción de puntos *más factibles* (para obtener matrices $A + BF_kC$ que satisfagan $\alpha(A + BF_kC) < 0$).

Algoritmo 14 Aleatoriedad del algoritmo de Posicionamiento de polos

1: $\delta = 0{,}75 \cdot (\text{\# de intentos}) + 0{,}1$
2: $\text{tam} = \sqrt{\|A\|_F}$
3: $\lambda^D = \lambda(-\text{tam} \cdot 1_{n\times n} + (2 \cdot \text{tam}) \cdot \text{randn}_{n\times n})$
4: $\lambda^D \leftarrow \lambda^D - \max_i(\text{Re}\{\lambda^D\}) \cdot 1_{n\times 1} - \delta \cdot 1_{n\times 1}$

Algoritmo 15 Posicionamiento de polos

$\tilde{\Delta} > 0$, $\eta \in [0, 1/4)$
$k \leftarrow 0$
$F_k \leftarrow$ random (matríz aleatoria)
$\Delta_k \in (0, \tilde{\Delta})$
for $k = 0, 1, 2, \ldots$ **do**
  Obtener p_k resolviendo (aproximadamente o exactamente) (3.46) con el parámetro Δ_k y donde p_k eventualmente es una matríz de las mismas dimensiones que F_k.
  Evaluar $\rho_k = \frac{f(F_k) - f(F_k + p_k)}{m_k(0) - m_k(p_k)}$.
  if $\rho_k < \frac{1}{4}$ **then**
    $\Delta_{k+1} = \frac{1}{4}\Delta_k$
  else
    if $\rho_k > \frac{3}{4} \wedge \|p_k\|_2 = \Delta_k$ **then**
      $\Delta_{k+1} = \text{mín}\{2\Delta_k, \tilde{\Delta}\}$
    else
      $\Delta_{k+1} = \Delta_k$
    end if
  end if
  if $\rho_k > \eta$ **then**
    $F_{k+1} \leftarrow F_k + p_k$
  else
    $F_{k+1} \leftarrow F_k$
  end if
end for

Algoritmo 16 Fase de restauración Posicionamiento de polos

1: $N \leftarrow$ número de veces que se realiza la búsqueda
2: $\rho_{\text{máx}} \leftarrow$ radio máximo de la región de confianza de $QP(x_k, \rho)$
3: $\rho \leftarrow (0, \rho_{\text{máx}})$
4: **if** $k = 0$ **then**
5: $x_k \leftarrow \texttt{polos}(k)$
6: **else**
7: $x_k \leftarrow$ punto inicial o proveniente de la iteración $k - 1$
8: **end if**
9: **while** ((x_k no es aceptable para $\mathcal{F}_{k-1}$) $\vee$ ($QP(x_k, \rho)$ no es factible)) $\wedge$ paso $\leq N$ **do**
10: $x_k \leftarrow \texttt{fminsearch}(\theta(\cdot), x_k)$ ó $x_k \leftarrow \texttt{lsdp}(\theta(\cdot), x_k, \mathcal{F}_k, \beta, \gamma)$
11: **if** x_k es aceptable para $\mathcal{F}_{k-1}$ **then**
12: **while** ($QP(x_k, \rho)$ no es factible) $\wedge$ ($\rho < \rho_{\text{máx}}$) **do**
13: $\rho \leftarrow 2 * \rho$
14: $d_k \leftarrow QP(x_k, \rho)$ (*si* $QP(x_k, \rho)$ *no es factible,* d_k *queda indefinido*)
15: **end while**
16: **end if**
17: paso $\leftarrow$ paso $+ 1$
18: **end while**
19: **if** paso $> N$ **then**
20: $x_k \leftarrow \texttt{polos}(k)$
21: Ir al paso 9.
22: **end if**

Capítulo 4

Resultados

4.1. COMPleib

Para realizar una comparación del desempenõ de las fases de restauración propuestas y de las implementaciones existentes, se utilizará la batería de problemas $COMPl_eib$, *COnstrained Matrix-optimization Problem library*, ([Lei04], [LL04],[LL03]). Los problemas contenidos en $COMPl_eib$ pertenecen a las siguientes áreas:

- Programas semidefinidos lineales
- Programas semidefinidos no lineales
- Minimización de funciones espectrales
- Ecuaciones de Lyapunov y Riccati
- Problemas de diseño de sistemas de control y matrices relacionadas

La versión 1.0 posee 124 problemas y la versión 1.1, 168 problemas. Las categorías de problemas y los distintos tipos de aplicaciones de donde provienen son las siguientes:

- Static Output Feedback
- Flujo de calor 2D
- Modelos de segundo orden
- Control de orden reducido

En las siguiente sección se revisarán brevemente cada categoría, definiendo el tipo de problema que se intenta resolver.

4.1.1. Categorías de problemas

Static Output Feedback

Una planta LTI se define como un sistema para $A \in \mathbb{R}^{nx\times nx}$, $B_1 \in \mathbb{R}^{nx\times nw}$, $B \in \mathbb{R}^{nx\times nu}$, $C_1 \in \mathbb{R}^{nz\times nx}$, $D_{11} \in \mathbb{R}^{nz\times nw}$, $D_{12} \in \mathbb{R}^{nz\times nu}$, $C \in \mathbb{R}^{ny\times nz}$ y $D_{21} \in \mathbb{R}^{ny\times nw}$, de la forma:

$$\begin{array}{rcl} \dot{x}(t) & = & Ax(t) + B_1 w(t) + Bu(t) \\ z(t) & = & C_1 x(t) + D_{11} w(t) + D_{12} u(t) \\ y(t) & = & Cx(t) + D_{21} w(t) \end{array} \tag{4.1}$$

con x estado, u control de entrada, y salida observada, z salida regulada y w ruido de entrada. El control retroalimentado por la salida observada se define:

$$u(t) = Fy(t) \tag{4.2}$$

con F una matríz de ganancia desconocida. De (4.1) y (4.2) se deduce la siguiente relación:

$$u(t) = F(Cx(t) + D_{21} w(t)) \tag{4.3}$$

Reemplazando (4.3) en (4.1), se obtiene:

$$\sum_{cl} : \left\{ \begin{array}{rcl} \dot{x}(t) & = & (A + BFC)x(t) + (B_1 + BFD_{21})w(t) \\ z(t) & = & (C_1 + D_{12}FC)x(t) + (D_{11} + D_{12}FD_{21})w(t) \end{array} \right. \tag{4.4}$$

El objetivo de este tipo de problemas es calcular F tal que el sistema $\sum_{cl}$ sea asintóticamente estable y además se minimice la norma $\mathcal{H}_2$ o $\mathcal{H}_\infty$ del sistema. Estas normas se definen de la forma:

- $\| \sum_{cl} \|_{\mathcal{H}_2}^2 = \text{Tr}\left(\frac{1}{2\pi}\int_{-\infty}^{\infty} H(jw)H(jw)^* dw\right)$, donde

  $$H(s) = C(sI - A)^{-1}B + D$$

  es la función de transferencia para el sistema

  $$\begin{array}{rcl} \dot{x}(t) & = & Ax(t) + Bw(t) \\ z(t) & = & Cx(t) + Dw(t) \end{array}$$

- $\| \sum_{cl} \|_{\mathcal{H}_\infty} = \text{máx}_w\, \sigma_{\text{máx}}(H(jw))$, donde $\sigma_{\text{máx}}(A)$ representa el valor singular máximo de la matríz A.

Para construir la formulación semidefinida, conviene escribir el teorema de Lyapunov:

Teorema 4.1.1 (Lyapunov)**.** *Son equivalentes:*

1. *Existe F tal que $A(F) = A + BFC$ es Hurwitz, es decir, $\Re(\lambda_i(A(F))) < 0$.*

2. *Para cada W, existe F tal que la ecuación de Lyapunov tiene solución única X:*

$$A(F)^T X + XA(F) + W = 0$$

si $W \succ 0 (W \succeq 0)$ entonces $X \succ 0 (X \succeq 0)$.

3. *Existe F matríz y V matríz simétrica tales que*

$$A(F)^T V + VA(F) + I = 0, V \succ 0$$

Con este teorema, las formulaciones que se presentan son las que se plantean a continuación.

Formulación $\mathcal{H}_2$

Problema 4.1.2. *Suponga que $D_{11} = 0$ y $D_{21} = 0$. Dadas las matrices reales A, B, C, B_1, C_1, D_{12} y un entero $0 \leq n_c < n$, encontrar una ganancia F de orden n_c tal que la matríz $A(F) = A + BFC$ es Hurwitz y la norma $\mathcal{H}_2$ del sistema $\sum_{cl}$ es minimal.*

La norma $\mathcal{H}_2$ de $\sum_{cl}$ se puede escribir equivalentemente como:

$$\| \sum_{cl} \|^2_{\mathcal{H}_2} = Tr(C(F)QC(F)^T) \tag{4.5}$$

donde $C(F) = C_1 + D_{12}FC$ y Q simétrica satisface la ecuación de Lyapunov,

$$A(F)Q + QA(F)^T + B_1 B_1^T = 0$$

Juntando la tercera equivalencia del Teorema de Lyapunov y la definición de la norma $\mathcal{H}_2$ según (4.5), se tiene el siguiente problema NLSDP:

Formulación 4.1.3.

$$\begin{array}{rrcl} min_{F,Q,V} & Tr((C_1 + D_{12}FC)Q(C_1 + D_{12}FC)^T) & & \\ s.a & (A + BFC)Q + Q(A + BFC)^T + B_1 B_1^T & = & 0 \\ & (A + BFC)V + V(A + BFC)^T + I & = & 0 \\ & V & \succ & 0 \end{array} \tag{4.6}$$

Formulación $\mathcal{H}_\infty$

Problema 4.1.4. *Dadas las matrices reales A, B, C, B_1, C_1, D_{11}, D_{12}, D_{21} y el entero $0 \leq n_c < n_x$, encontrar una ganancia F de orden n_c, una matríz simétrica P y $\gamma > 0$ tal que para un γ mínimo, la tripleta (F, P, γ) satisface la ecuación de Riccati*

$$\begin{aligned} A(F)^T P + PA(F) + \frac{1}{\gamma} C(F)^T C(F) & \\ + \frac{1}{\gamma} M(F, P, \gamma) R(F, \gamma)^{-1} M(F, P, \gamma)^T & = 0 \end{aligned}$$

donde

$$\begin{aligned} R(F,\gamma) &= I - \gamma^{-2} D(F)^T D(F) \\ M(F,P,\gamma) &= PB(F) + \gamma^{-1} C(F)^T C(F) \end{aligned}$$

también se cumple

$$R(F,\gamma) \succ 0, P \succeq 0$$

y además $\tilde{A}(F,P,\gamma) = A(F) + \gamma^{-1} B(F) R(F,\gamma)^{-1} M(F,P,\gamma)^T$ ***es Hurwitz.***

La formulación SDP para este problema es la siguiente:

Formulación 4.1.5.

$$\begin{array}{lrcl} \textit{min}_{F,P,W,\gamma} & \gamma & & \\ \textit{s.a} & A(F)^T P + PA(F) + \gamma^{-1} C(F)^T C(F) + & & \\ & \gamma^{-1} M(F,P,\gamma) R(F,\gamma)^{-1} M(F,P,\gamma)^T & = & 0 \\ & \tilde{A}(F,P,\gamma) V + V \tilde{A}(F,P,\gamma)^T + I & = & 0 \\ & R(F,\gamma) & \succ & 0 \\ & W & \succ & 0 \\ & P & \succeq & 0 \\ & \gamma & > & 0 \end{array} \tag{4.7}$$

Flujo de calor 2D

Se quiere controlar un modelo de flujo de calor 2D. Usando diferencias finitas, se obtiene un gran sistema de control, donde las matrices son ralas (E invertible):

$$\begin{aligned} E\dot{x}(t) &= (A + \delta A)x(t) + G(x(t)) + B_1 w(t) + Bu(t) \\ z(t) &= C_1 x(t) + D_{12} u(t) \\ y(t) &= Cx(t) \\ u(t) &= Fy(t) \\ x(0) &= x_0 \end{aligned} \tag{4.8}$$

En este modelo, x es la aproximación de la temperatura, u es el control de entrada, y denota la observación, w es el ruido de entrada y z la salida regulada.

Si se considera $G(x(t)) = 0$, el modelo es el siguiente:

$$\begin{aligned} \dot{x}(t) &= \underbrace{E^{-1}(A + \delta A)}_{A} x(t) + \underbrace{E^{-1} B_1}_{B_1} w(t) + \underbrace{E^{-1} B}_{B} u(t) \\ z(t) &= C_1 x(t) + D_{12} u(t) \\ y(t) &= Cx(t) \\ u(t) &= Fy(t) \\ x(0) &= x_0 \end{aligned} \tag{4.9}$$

La formulación del problema es la correspondiente a SOF-$\mathcal{H}_2$, (4.6).

Modelos de segundo orden

El modelo es de la forma:

$$M\ddot{q} + D\dot{q} + Sq \;=\; \hat{B}u \tag{4.10}$$

con M masa, D amortiguación y S rigidez.
Utilizando los siguientes cambios, se obtiene un sistema de primer orden:

$$x := \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, A := \begin{bmatrix} 0 & I \\ -M^{-1}D & -M^{-1}S \end{bmatrix}, B := \begin{bmatrix} 0 \\ M^{-1}\hat{B} \end{bmatrix}$$

La formulación del problema es la correspondiente a SOF-$\mathcal{H}_2$, (4.6).

Control de orden reducido

Estos problemas no son directamente estabilizables por un control SOF, sin embargo se pueden replantear utilizando una técnica de aumento de las dimensiones del sistema:

$$\begin{aligned}
\begin{bmatrix} \dot{x}(t) \\ \dot{x}_c(t) \end{bmatrix} &= \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ x_c(t) \end{bmatrix} + \begin{bmatrix} B_1 \\ 0 \end{bmatrix} w(t) + \begin{bmatrix} 0 & B \\ I & 0 \end{bmatrix}\begin{bmatrix} \dot{x}_c(t) \\ u(t) \end{bmatrix} \\
z(t) &= \begin{bmatrix} C_1 & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ x_c(t) \end{bmatrix} + D_{11}w(t) + \begin{bmatrix} 0 & D_{12} \end{bmatrix}\begin{bmatrix} \dot{x}_c(t) \\ u(t) \end{bmatrix} \\
\begin{bmatrix} x_c(t) \\ y(t) \end{bmatrix} &= \begin{bmatrix} 0 & I \\ C & 0 \end{bmatrix}\begin{bmatrix} x(t) \\ x_c(t) \end{bmatrix} + \begin{bmatrix} 0 \\ D_{21} \end{bmatrix} w(t) \\
\begin{bmatrix} \dot{x}_c(t) \\ u(t) \end{bmatrix} &= F\begin{bmatrix} x_c(t) \\ y(t) \end{bmatrix}
\end{aligned}$$

La formulación del problema es la correspondiente a SOF-$\mathcal{H}_2$, (4.6).

4.1.2. Formato de entrada

Desde MATLAB, se llama a la función `COMPleib`, pasando como argumento el identificador del problema:

```
[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] =
                COMPleib('AC1');
```

La manera de extraer los datos desde MATLAB es la siguiente, en este caso, para el problema de código AC1:

```
>> [A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('AC1');
```

Código 4.1: Script en Perl de chequeo de controlabilidad y observabilidad de datos COMPleib

```
#!/usr/bin/perl
%vars=( "A" => "A",
  "B1" => "B1",
  "B" => "B",
  "C1" => "C1",
  "C" => "C",
  "D11" => "D11",
  "D12" => "D12",
  "D21" => "D21",
  "dims" => "nx nw nu nz ny");
open(IN,"<codes.txt");
$actuallines=0;
$numlines=0;
while(<IN>){
  $numlines=$numlines+1;
}
close(IN);

open(IN,"<codes.txt");
while(<IN>){
  $actuallines=$actuallines+1;
  $code=$_;
  chomp($code);
  $code=~s/ //g;
  print "%checking problem ",$code," in compleib_data directory\n";
  print "[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('",$code,"');\n";
  print "r=rank(ctrb(A,B));\n";
  print "r2=rank(obsv(A,C));\n";
  print "if r==nx && r2==nx; disp(['>>>>>>>",$code," is asymp. stable!']);end;\n";
}
close(IN);
```

4.1.3. Selección de problemas

Para realizar la comparación, se decidió revisar la estabilidad de cada problema, revisando la (A, B)-controlabilidad y la (A, C)-observabilidad. Si un problema no es asintóticamente estable (condición necesaria es que sea (A, B)-controlable y (A, C)-observable), entonces no se utilizará para la comparación.

Para realizar ese chequeo, se utilizó un script en lenguaje Perl, cuyo código se puede ver en 4.1, y que genera un script de MATLAB © que entrega los problemas que cumplen las condiciones. El script generado se puede ver en el código 4.2, y la ejecución completa de esta parte se realiza de la siguiente forma:

```
[22:30 0.00][euler] % perl checkctrbobsv.pl > scriptCTRBOBSV.m
[22:30 0.01][euler] % matlab -nodisplay -nosplash -r "scriptCTRBOBSV;quit;"
```

La lista de los problemas (54 en total) a revisar se puede ver en 4.1 y 4.2.

Código 4.2: Script generado para MATLAB de chequeo de controlabilidad y observabilidad de datos COMPleib

```
[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('AC1');
r=rank(ctrb(A,B));
r2=rank(obsv(A,C));
if r==nx && r2==nx; disp(['>>>>>>>HF2D17 is asymp. stable!']);end;
...
[A,B1,B,C1,C,D11,D12,D21,nx,nw,nu,nz,ny] = COMPleib('HF2D18');
r=rank(ctrb(A,B));
r2=rank(obsv(A,C));
if r==nx && r2==nx; disp(['>>>>>>>HF2D18 is asymp. stable!']);end;
```

Código	n_x	n_u	n_y	Estructura de A	Categoría
AC1	5	3	3	densa	SOF
AC2	5	3	3	densa	SOF
AC3	5	2	4	densa	SOF
AC5	4	2	2	densa	SOF
AC6	7	2	4	densa	SOF
AC11	5	2	4	densa	SOF
AC12	4	3	4	densa	SOF
AC15	4	2	3	densa	SOF
AC16	4	2	4	densa	SOF
AC17	4	2	2	densa	SOF
AC18	10	2	2	densa	SOF
HE1	4	2	1	densa	SOF
HE2	4	2	2	densa	SOF
HE3	8	4	6	densa	SOF
HE4	8	4	6	densa	SOF
HE5	8	4	2	densa	SOF
REA1	4	2	3	densa	SOF
REA2	4	2	2	densa	SOF
DIS1	8	4	4	densa	SOF
DIS2	3	2	2	densa	SOF
DIS3	6	4	4	densa	SOF
DIS4	6	4	6	densa	SOF
DIS5	4	2	2	densa	SOF

Cuadro 4.1: Problemas pertenecientes a COMPleib para realizar la comparación (parte 1)

Código	n_x	n_u	n_y	Estructura de A	Categoría
BDT1	11	3	3	rala	SOF
MFP	4	3	2	densa	SOF
EB1	10	1	1	rala	SOF
EB2	10	1	1	rala	SOF
EB3	10	1	1	rala	SOF
TF1	7	2	4	densa	SOF
PSM	7	2	3	densa	SOF
NN1	3	1	2	densa	SOF
NN2	2	1	1	densa	SOF
NN3	4	1	1	densa	SOF
NN4	4	2	3	densa	SOF
NN5	7	1	2	densa	SOF
NN8	3	2	2	densa	SOF
NN9	5	3	2	densa	SOF
NN10	8	3	3	densa	SOF
NN12	6	2	2	densa	SOF
NN13	6	2	2	densa	SOF
NN14	6	2	2	densa	SOF
NN15	3	2	2	densa	SOF
NN16	8	4	4	densa	SOF
NN17	3	2	1	densa	SOF
TMD	4	2	2	densa	Segundo orden
FS	5	1	3	densa	Segundo orden
DLR1	10	2	2	densa	Segundo orden
ROC1	9	2	2	densa	Orden reducido
ROC3	11	4	4	densa	Orden reducido
ROC4	9	2	2	densa	Orden reducido
ROC6	5	3	3	densa	Orden reducido
ROC7	5	2	3	densa	Orden reducido
ROC8	9	4	4	densa	Orden reducido
ROC9	6	3	3	densa	Orden reducido

Cuadro 4.2: Problemas pertenecientes a COMPleib para realizar la comparación (parte 2)

4.2. Comparación entre `fnlsdp` e implementación MATLAB

Para comparar el rendimiento de ambos sistemas, se corrieron los tests pertenecientes a la batería de problemas COMPleib descritos en la sección anterior, para el problema tipo (1.2) y utilizando el método SOF más posicionamiento de polos, descrito en 3.3.4 para generar los puntos iniciales (ambos sistemas parten del mismo punto incial).

Los resultados obtenidos en ambos sistemas se pueden ver en las tablas 4.3, 4.4 y 4.5, donde $h_1(x)$, $h_2(x)$ y $\lambda_1(V)$ representan la primera, segunda y tercera restricciones en (1.2).

Para analizar los resultados, se calculó el porcentaje de ocaciones en las que alguno de los métodos tenía mejor desempeño que el otro. Para ello, se contaron sólo los problemas donde alguno de los métodos entregó resultados. Si uno entregó resultados (se identifica con un *) y el otro no, se considera a todos los valores del que no entregó resultados como $+\infty$. La información obtenida es la siguiente:

- En el 78 % de los casos el tiempo de ejecución entregado por `fnlsdp` tuvo un valor menor al entregado por la implementación MATLAB.
- En el 68 % de los casos, el punto entregado por `fnlsdp` tuvo un valor de $\|dx\|$ menor al entregado por la implementación MATLAB.
- En el 83 % de los casos, el valor de la función objetivo $f(x)$ entregada por `fnlsdp` tuvo un valor menor al entregado por la implementación MATLAB.
- En el 81 %, 79 % y 69 % de los casos, los valores entregados por `fnlsdp` de $\|h_1(x)\|$, $\|h_2\|$ y $\lambda_1(V)$ respectivamente, fueron menores a los valores entregados por MATLAB.
- El número de problemas donde $\|dx\| \leq 0{,}1$ fue de 5 para MATLAB y 1 para `fnlsdp`. Esto indica que el número de problemas resueltos adecuadamente fue muy pequeño en ambos métodos, con MATLAB obteniendo una ligera ventaja. Esto se puede deber a que la fase de restauración en cada método no está completamente desarrollada (se están usando versiones del método Nelder-Mead simplex en ambas implementaciones).
- El método `fnlsdp` obtuvo el 76 % de los casos donde el valor de la función $\theta(x)$ (como se describe en (2.88)) es menor a $0{,}01$.

En base a esta información, se puede deducir que el método `fnlsdp` tiene resultados mejores en términos del valor de la función objetivo y de mérito, sin embargo no se puede asegurar nada con respecto a la terminación del algoritmo, pues sólo en 1 caso se llegó a una solución donde $\|dx\| \approx 0$ (valor de la norma de la solución del problema tangencial $QP(x_k, \rho)$). Se espera que lo anterior pueda mejorar al introducir nuevas fases de restauración, lo cual quedará como trabajo a futuro para la implementación `fnlsdp`.

Prob.	Sistema	Tiempo	Iter.	fr	$\|dx\|$	$f(x)$	$\|h_1(x)\|$	$\|h_2(x)\|$	$\lambda_1(V)$	Tipo
AC1	matlab	11.15	4	0	0.01	1.06	0	0	0	e
	fnlsdp	7.13	9	2	7.13	-42.45	0	0	0	h
AC2	matlab	13.73	3	3	0.01	1.35	0	0	0	h
	fnlsdp	10.23	6	9	8.62	-48.46	0	0	0	h
AC3	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	8.7	8	2	3.2	-17.89	0	0	0	h
AC5	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
AC6	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	17.62	2	2	17.18	-100136.49	311762.8	784.64	0.03	w
AC11	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	12.43	7	9	6.93	-5.05	0	0.01	0.29	h
AC12	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
AC15	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	32.45	14	2	390.54	-9633684.91	13.51	38.02	0	h
AC16	matlab	4.22	1	0	112.43	3853.01	1.98	0.01	0	e
	fnlsdp	8.8	16	2	12.19	-786.27	0	0	0	h
AC17	matlab	3.86	2	0	193.12	57777.29	1.26	0	0	e
	fnlsdp	1.19	1	2	19.31	0.17	0	0	0	w
AC18	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
HE1	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	9.15	6	2	161.3	-167.69	0.13	1.63	0	h
HE2	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	2.61	3	2	19.64	-1.27	0	0	0	h
HE3	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
HE4	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	102.62	9	2	39.97	18.22	0	0.03	0.42	h
HE5	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	138.33	4	5	3475.25	-3343.22	12.5	35.63	0.33	h
REA1	matlab	5.11	3	0	0.77	25.11	0.02	0.04	0	e
	fnlsdp	1.71	1	2	5.14	-0.09	0	0	0.09	w
REA2	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	2.48	5	2	5.34	30.71	0	0	0	h
DIS1	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	50.91	10	2	4.71	-31.39	0	0.01	0	h
DIS2	matlab	2.98	1	0	7.04	41.37	1.5	0	0	e
	fnlsdp	1.69	5	2	4.05	-3.35	0	0	0	h
DIS3	matlab	14.54	2	0	4.3	326.8	1.23	0.08	0	e
	fnlsdp	16.99	7	2	4.4	12.34	0	0	0	h
DIS4	matlab	27.75	4	0	0.72	8.36	0.41	0.98	0	e
	fnlsdp	31.49	7	5	1.95	2.59	0	0.01	0	h
DIS5	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*

Cuadro 4.3: Comparación de ambos sistemas para batería de problemas COMPleib (parte 1)

Prob.	Sistema	Tiempo	Iter.	fr	$\|dx\|$	$f(x)$	$\|h_1(x)\|$	$\|h_2(x)\|$	$\lambda_1(V)$	Tipo
BDT1	matlab	29.87	2	0	0	-0.2	0	0	0	h
	fnlsdp	234.7	4	5	386.01	0.02	0	0.68	0.01	h
MFP	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	6.6	3	2	31.89	-73.78	1.15	2.75	0	w
EB1	matlab	702.45	6	0	0.87	477.29	0	0.01	0	e
	fnlsdp	*	*	*	*	*	*	*	*	*
EB2	matlab	123.9	4	0	0.06	35.81	0	0	0	e
	fnlsdp	*	*	*	*	*	*	*	*	*
EB3	matlab	39.79	2	3	0.02	54.38	0.19	0.03	0	f
	fnlsdp	*	*	*	*	*	*	*	*	*
TF1	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	18.7	5	2	65.92	-1.1	0	0	0	h
PSM	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	8.59	4	2	9.05	0.04	0	0	0.04	h
NN1	matlab	5.27	1	0	363.21	166.38	1.73	0.02	0	e
	fnlsdp	2.13	7	2	17.06	-7739.5	0	0	0	h
NN2	matlab	3.55	3	0	0.53	3.25	0.11	0.09	0	e
	fnlsdp	*	*	*	*	*	*	*	*	*
NN3	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
NN4	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	2.04	2	2	2.79	2.63	0	0	0	h
NN5	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
NN8	matlab	4.09	3	0	3.26	170.87	0.16	0.11	0	e
	fnlsdp	1.47	4	2	4.11	-15.66	0	0	0	h
NN9	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	13.3	9	5	96.19	-210.79	0	0	0	h
NN10	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
NN12	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
NN13	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	21.9	11	2	54.64	38.19	0.01	0	0.03	h
NN14	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	21.92	11	2	54.64	390.91	0.01	0	0.03	h
NN15	matlab	8.52	1	0	333.19	0.02	4.89	2.18	0.24	e
	fnlsdp	0.8	7	0	0	0.02	0	0	0	e
NN16	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	24.31	6	2	18.88	0.58	0	0	0	h
NN17	matlab	5.44	6	0	1.1	366.05	0.01	0	0	e
	fnlsdp	*	*	*	*	*	*	*	*	*
TMD	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	7.29	4	2	29.02	-0.01	0	0	0	h
FS	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*

Cuadro 4.4: Comparación de ambos sistemas para batería de problemas COMPleib (parte 2)

Prob.	Sistema	Tiempo	Iter.	fr	$\|dx\|$	$f(x)$	$\|h_1(x)\|$	$\|h_2(x)\|$	$\lambda_1(V)$	Tipo
DLR1	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
ROC1	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
ROC3	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
ROC4	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
ROC6	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	*	*	*	*	*	*	*	*	*
ROC7	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	3.66	3	2	51.2	0	0	0	0	h
ROC8	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	795.1	9	9	164.73	-39235.63	0.01	0.01	0	h
ROC9	matlab	*	*	*	*	*	*	*	*	*
	fnlsdp	13.13	8	2	30.52	-184.91	0	0	0	h

Cuadro 4.5: Comparación de ambos sistemas para batería de problemas COMPleib (parte 3)

4.3. Cálculo paralelo

Para revisar el speedup obtenido por la aplicación `fnlsdp`, se deben realizar pruebas midiendo el tiempo de ejecución. Las pruebas se debían realizar con problemas de tamaño grande provenientes de las tablas 4.1 y 4.2. Debían ser de tamaño grande, pues de lo contrario no se podrían observar las ventajas en el speedup. La variable de decisión utilizada en este caso fue el tiempo de ejecución secuencial. Se consideraron aquellos tests de la sección anterior cuyos tiempos de ejecución fueran mayor a 100 segundos.

También se decidió desarrollar pruebas de speedup para las componentes paralelizadas, cada una por separado. Estas componentes son: la resolución del problema $QP(x_k, \rho)$ y el cálculo de $\theta(x)$.

El ambiente de trabajo utilizado fue un cluster de 4 servidores Linux, cada uno con 8 procesadores y una memoria compartida de 16GB para cada servidor. La red utilizada fue de 100 mbps, por lo que se decidió utilizar sólo 8 procesos, para no perjudicar a las pruebas realizadas con mayor cantidad, pues la velocidad de la red (mucho menor que la velocidad del bus de datos interno de cada servidor) perturba sus resultados.

4.3.1. Speedup: `fnlsdp`

Los problemas utilizados para medir el speedup de la aplicación `fnlsdp` fueron los siguientes (sus tiempos de ejecución secuencial supera los 100 segundos): HE4, HE5,

Problema	1 proceso	2 procesos	4 procesos	8 procesos
HE4	102.62	100.97	119.42	1153.21
HE5	138.33	222.38	248.03	858.92
BDT1	234.70	211.07	187.33	436.45
ROC8	795.10	814.02	653.07	2095.69

Cuadro 4.6: Tiempos de ejecución de `fnlsdp`

Problema	1 proceso	2 procesos	4 procesos	8 procesos
HE1	1	0.98	1.16	11.24
HE5	1	1.61	1.79	6.21
BDT1	1	0.89	0.79	1.86
ROC8	1	1.02	0.82	2.63

Cuadro 4.7: Speedup de `fnlsdp`

BDT1 y ROC8. Los tiempos de ejecución se pueden ver en la tabla 4.6, y el speedup de las pruebas en la tabla 4.7.

Se observa que el speedup con 8 procesos no sigue la continuidad esperada. Para tener más información sobre las partes de la aplicación que pueden producir las discontinuidades, se decidió estudiar el speedup de las partes por separado. El speedup con 4 procesos presenta los mejores resultados para los problemas de mayor tamaño (requieren mayor tiempo de ejecución secuencial).

4.3.2. Speedup: resolución de $QP(x_k, \rho)$

Para testear el speedup de la resolución del subproblema, se decidió verificar que el solver `PCSDP` funcionara correctamente y entregara un incremento en el speedup creciente con el número de procesos involucrados. Se escogió testear problemas provenientes de la batería SDPLIB [Bor98]. La librería se puede descargar del sitio:

`http://euler.nmt.edu/~brian/sdplib/sdplib.zip`

Los tiempos de ejecución obtenidos para algunos problemas de dimensiones representativas se pueden ver en la tabla 4.8. En este caso, m representa el número de matríces simétricas involucradas en el problema (matrices A_i según la formulación (3.1)) y n es la dimensión de esas matrices.

El speedup obtenido en cada prueba se puede ver en la tabla 4.9.

Se puede observar que para los problemas de tamaño grande, control10, equalG11 y qap10, salvo qpG51, se obtuvo un speedup menor que 1, lo que significa una mejora

Problema	m	n	1 proceso	2 procesos	4 procesos	8 procesos
truss4	12	19	0.153	1.172	1.24	1.485
truss3	27	31	0.18	1.184	1.242	2.489
qap5	136	26	0.22	1.22	1.244	1.38
gpp124-1	125	124	0.68	1.711	1.744	2.945
arch0	174	335	4.847	4.493	3.909	6.731
gpp250-1	250	250	3.802	16.233	5.433	5.075
gpp500-1	501	500	36.909	92.411	27.557	50.493
equalG11	801	801	168.734	132.221	143.258	137.585
qap10	1021	101	3.558	3.632	2.976	3.002
control10	1326	150	171.367	120.690	91.213	109.806
qpG51	1000	2000	299.713	311.123	301.104	446.001

Cuadro 4.8: Tiempos de ejecución de PCSDP

Problema	m	n	1 proceso	2 procesos	4 procesos	8 procesos
truss4	12	19	1	7.66	8.1	9.71
truss3	27	31	1	6.58	6.9	13.83
qap5	136	26	1	5.55	5.65	6.27
gpp124-1	125	124	1	2.52	2.56	4.33
arch0	174	335	1	0.93	0.81	1.39
gpp250-1	250	250	1	4.27	1.43	1.33
gpp500-1	501	500	1	2.5	0.75	1.37
equalG11	801	801	1	0.78	0.85	0.82
qap10	1021	101	1	1.02	0.84	0.84
control10	1326	150	1	0.7	0.53	0.64
qpG51	1000	2000	1	1.04	1	1.49

Cuadro 4.9: Speedup de PCSDP

Problema	Dimensión	1 proceso	2 procesos	4 procesos	8 procesos
sherman1	1000	0.714214	1.545932	1.490187	1.610511
lshp1009	1009	0.651241	1.550980	1.499325	2.543561
rajat02	1960	5.032212	3.890884	2.629093	3.685942
ex14	3251	26.976499	16.496059	10.672434	11.024562
c-26	4307	58.171369	35.130799	25.281388	26.590259
c-30	5321	100.305306	64.575935	47.436703	47.751464
bcsstk17	10974	850.734038	534.985589	384.939793	456.559349

Cuadro 4.10: Tiempos de ejecución de `PDSYEVX`

en el tiempo de resolución. Para los casos medianos, arch0 y gpp500-1, no se obtuvo un speedup menor que 1 en ninguna prueba, salvo para 4 procesos. Para gpp124-1 y gpp250-1 no hubo resultados con speedup menor que 1. Para los casos pequeños, el speedup es mayor que 1 en todos los casos, lo que indica que no mejora el tiempo de ejecución.

El speedup con mejor rendimiento se genera al utilizar 4 procesos, con 5 casos de éxito. Esto se puede deber a que como las matrices son cuadradas, el balance de carga óptimo se produce al asignar igual cantidad de bloques de la matríz a cada proceso. También influye en el speedup la densidad de las matrices A_i involucradas en el problema.

4.3.3. Speedup: cálculo de $\theta(x_k)$

Para testear el speedup del cálculo de la función, se decidió verificar que la rutina `PDSYEVX` funcionara correctamente y entregara un incremento en el speedup creciente con el número de procesos involucrados. Se escogió testear el cálculo del valor propio máximo de una matríz simétrica. Las matrices se obtuvieron del sitio:

`http://www.cise.ufl.edu/research/sparse/matrices/list_by_dimension.html`

Los tiempos de ejecución obtenidos para algunas matrices de dimensiones representativas se pueden ver en la tabla 4.10.

El speedup obtenido en cada prueba se puede ver en la tabla 4.11.

Se puede observar que el speedup con mejor rendimiento se genera al utilizar 4 procesos, al igual que en las pruebas para PCSDP. Igualmente, esto se puede deber a que como las matrices son cuadradas, el balance de carga óptimo se produce al asignar igual cantidad de bloques de la matríz a cada proceso. Tambien se observa que para los problemas pequeños no hay ganancia en el speedup, es decir, no se obtuvo un speedup menor que 1. Para los problemas grandes si se obtuvo un speedup menor que 1, lo que indica que si se obtiene una ganancia en tiempo de ejecución al utilizar múltiples procesos.

Problema	Dimensión	1 proceso	2 procesos	4 procesos	8 procesos
sherman1	1000	1	2.16	2.09	2.25
lshp1009	1009	1	2.38	2.3	3.91
rajat02	1960	1	0.77	0.52	0.73
ex14	3251	1	0.61	0.4	0.41
c-26	4307	1	0.6	0.43	0.46
c-30	5321	1	0.64	0.47	0.48
bcsstk17	10974	1	0.63	0.45	0.54

Cuadro 4.11: Speedup de `PDSYEVX`

4.4. Fases de restauración

Cada problema se testeó usando 4 métodos, como se describen en el capítulo 3.3:

- Método 1: `fminsearch` (original) + SOF
- Método 2: `fminsearch` (original) + posicionamiento de polos
- Método 3: `lsdp` (restauración inexacta) + SOF
- Método 4: `lsdp` (restauración inexacta) + posicionamiento de polos

Los resultados se pueden ver en la tabla 4.17. Resulta interesante observar que la mayoría de las pruebas terminaron en pocas iteraciones. Este hecho es de consideración, sin embargo, varios problemas llegaron a la solución en ese número de iteraciones, lo cual supone que el punto inicial, generado automáticamente (y aleatoria según el método de posicionamiento de polos) , es sumamente bueno o cercano al óptimo.

Se pueden calcular algunas estadísticas sobre estos resultados. Contando los casos ganadores en cada columna, y sumando en ambos métodos en caso de empate, se obtienen los siguientes resultados:

- 10 de 54 problemas se resolvieron con éxito, llegando a la condición de parada de $\|dx\| < \epsilon$, con $\epsilon = 10^{-3}$.
- 40 de 54 problemas terminaron su ejecución entregando un valor en al menos un método. El resto no se detuvo o falló en su fase de restauración.
- En un 35 % de los problemas que terminaron, el método 3 obtuvo un menor valor de $\|dx\|$, la solución del problema $QP(x_k, \rho)$, utilizada como condición de parada si $\|dx\| < \epsilon$. Le sigue el método 1 con un 28 %, el método 4 con un 21 % y el método 2 con un 16 %.

- En un 50 % de los problemas que terminaron, el método 3 obtuvo un menor valor de $f(x)$, la función objetivo del problema, seg'un la fórmula (2.87). Le sigue el método 1 con un 43 %, el método 2 con un 7 % y el método 4 con un 0 %.

- En un 33 % de los problemas que terminaron, el método 3 obtuvo un menor valor de $\|h_1(x)\|$, la norma de la primera restricción del problema (1.2). Le sigue el método 2 con un 27 %, el método 1 con un 24 % y el método 4 con un 16 %.

- En un 38 % de los problemas que terminaron, el método 3 obtuvo un menor valor de $\|h_2(x)\|$, la norma de la segunda restricción del problema (1.2). Le sigue el método 1 con un 24 %, el método 4 con un 23 % y el método 2 con un 14 %.

- En un 29 % de los problemas que terminaron, el método 1 obtuvo un menor valor de $\lambda_1(V)$, el mayor valor propio de la matríz V, según el tercer sumando de la fórmula (2.88). Le sigue el método 4 con un 27 %, el método 2 con un 23 % y el método 3 con un 21 %.

De estos datos, se infiere que el método 3 posee mejor rendimiento en comparación con las otras fases, sin embargo, conviene tambien analizar la calidad de los puntos de parada, para ver si el valor de $\|dx\|$ es cercano a cero, analizar los valores de la función objetivo y los valores de las componentes de la función de mérito. Las estadísticas obtenidas en este caso se pueden ver en las tablas 4.12, 4.13, 4.14, 4.15 y 4.16. Estas tablas entregan la siguiente información:

- Los métodos 1 y 4 entregan mejores resultados (promedio más cercano a cero y con menor desviación estándar) en el valor de $\|dx\|$, sin embargo, todos los métodos están alejados en promedio del valor de tolerancia, es decir, la condición de parada al encontrar un punto óptimo no se alcanzó en promedio.

- El método 2 entrega los mejores resultados para la minimización de la función objetivo, sin embargo, posee una alta desviación estándar. Le sigue el método 3, bastante alejado, pero con una desviación estándar pequeña.

- El método 4 entrega los mejores resultados para el valor de $\|h_1(x)\|$ (cercano a cero en promedio y con baja desviación estándar). El método 1 le sigue con resultados similares.

- El método 1 entrega los mejores resultados para el valor de $\|h_2(x)\|$ (cercano a cero en promedio y con baja desviación estándar). El método 4 le sigue con resultados similares.

- El método 1 entrega los mejores resultados para el valor de $\lambda_1(V)$ (cercano a cero y con baja desviación estándar). Le sigue el método 4 con resultados levemente peores.

	Método 1	Método 2	Método 3	Método 4
# casos	24	40	38	22
Promedio	18.6960	121.5108	72.3756	18.6809
Desv. Estándar	33.5783	328.9834	142.9171	30.9918

Cuadro 4.12: Estadísticas para $\|dx\|$, por método

	Método 1	Método 2	Método 3	Método 4
# casos	24	40	38	22
Promedio	87.4881	-3326.1257	12.2544	82.5667
Desv. Estándar	118.6212	14674.9903	39.8084	290.6753

Cuadro 4.13: Estadísticas para $f(x)$, por método

Utilizando estos datos, se puede llegar a una conclusión sobre que método conviene implementar en C y MPI para mejorar la implementación paralela. Ciertamente falta desarrollo del algoritmo en sí, pero es un avance con respecto a una nueva fase de restauración, que sirve como alternativa en caso de que la fase original no funcione. Esta meta quedará como trabajo a futuro, sin embargo, los resultados que se pueden obtener pueden ser de gran alcance, pues se podrán abordar problemas de mayor tamaño y provenientes de otros ámbitos, no sólo de la batería COMPleib.

	Método 1	Método 2	Método 3	Método 4
# casos	24	40	38	22
Promedio	1.0795	623.7996	14.5255	0.7825
Desv. Estándar	1.4921	3197.5112	52.1860	1.0009

Cuadro 4.14: Estadísticas para $\|h_1(x)\|$, por método

	Método 1	Método 2	Método 3	Método 4
# casos	24	40	38	22
Promedio	0.0048	52.1308	0.1845	0.0132
Desv. Estándar	0.0123	231.3172	0.7846	0.0283

Cuadro 4.15: Estadísticas para $\|h_2(x)\|$, por método

	Método 1	Método 2	Método 3	Método 4
# casos	24	40	38	22
Promedio	0.1842	11.0617	19.3023	0.3867
Desv. Estándar	0.8656	29.8066	62.4200	1.1233

Cuadro 4.16: Estadísticas para $\lambda_1(V)$, por método

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
AC1	1	1	4.3866	0.1164	0.1852	0.0000	2.8727
AC1	2	1	0.8340	6.3492	0.0496	0.0000	0.0000
AC1	3	1	4.3866	0.1164	0.1852	0.0000	2.8727
AC1	4	1	0.8340	6.3492	0.0496	0.0000	0.0000
AC2	1	1	1.6901	0.1119	0.0491	0.0031	0.7892
AC2	2	4	0.0103	0.3979	0.0000	0.0000	0.0000
AC2	3	1	1.6942	0.1129	0.0828	0.0000	0.7911
AC2	4	1	1.0602	13.7624	0.1305	0.0000	0.0000
AC3	1	2	2.6576	78.1688	0.2928	0.0094	0.0000
AC3	2	1	98.2309	41.9975	2.2135	0.0000	0.0000
AC3	3	1	43.4506	2.7724	2.1833	0.0000	6.3104
AC3	4	1	33.8421	242.8459	2.1717	0.0000	0.0000
AC5	1	*	*	*	*	*	*
AC5	2	*	*	*	*	*	*
AC5	3	*	*	*	*	*	*
AC5	4	*	*	*	*	*	*
AC6	1	2	1.7083	244.4431	0.6714	0.0018	0.0000
AC6	2	*	*	*	*	*	*
AC6	3	2	1.2899	76.7079	1.6399	0.0315	0.0000
AC6	4	*	*	*	*	*	*
AC11	1	1	15.1268	2.0034	2.0823	0.0000	5.9381
AC11	2	*	*	*	*	*	*
AC11	3	1	19.6708	1.7942	2.0518	0.0000	6.8002
AC11	4	1	127.7429	5.3683	2.2187	0.0000	0.0000
AC12	1	2	20.8187	29.3325	1.6609	31.5109	0.0000
AC12	2	*	*	*	*	*	*
AC12	3	1	375.2711	0.1439	0.1155	0.0000	374.2121
AC12	4	*	*	*	*	*	*
AC15	1	3	162.6017	-54143.1469	3972.3096	596.4333	0.0000
AC15	2	*	*	*	*	*	*
AC15	3	1	519.1385	1.3216	1.9961	0.0000	49.6278
AC15	4	1	5.7748	285.5207	1.6791	0.0000	0.0000
AC16	1	1	122.5108	-0.2739	1.9808	0.0000	65.8394
AC16	2	1	10.8867	1390.7171	0.9247	0.0150	0.0000
AC16	3	1	79.6824	1.8682	1.9750	0.0000	3.8055
AC16	4	1	9.3886	403.7466	1.6424	0.0000	0.0000

Cuadro 4.17: Resultados comparación de métodos para COMPleib (parte 1)

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
AC17	1	4	0.3259	28.8093	0.0046	0.0045	0.0000
AC17	2	4	0.1378	17.2703	0.0078	0.0078	0.0000
AC17	3	4	0.3184	39.3730	0.0012	0.0012	0.0000
AC17	4	3	0.2214	22.8899	0.0099	0.0042	0.0000
AC18	1	*	*	*	*	*	*
AC18	2	*	*	*	*	*	*
AC18	3	*	*	*	*	*	*
AC18	4	*	*	*	*	*	*
HE1	1	1	12.3812	1.4779	0.1471	0.0871	7.3372
HE1	2	3	0.0010	0.0710	0.0000	0.0000	0.0000
HE1	3	3	0.0004	0.0267	0.0000	0.0000	0.0000
HE1	4	4	0.0019	0.0309	0.0000	0.0000	0.0000
HE2	1	1	24.0194	2.2934	1.4070	0.0186	0.0000
HE2	2	*	*	*	*	*	*
HE2	3	3	0.5167	149.9490	0.0295	0.0339	0.0000
HE2	4	1	118.5320	2.3308	1.9833	0.0000	0.0000
HE3	1	1	157.9239	0.1297	1.2161	0.4461	100.5195
HE3	2	*	*	*	*	*	*
HE3	3	*	*	*	*	*	*
HE3	4	*	*	*	*	*	*
HE4	1	*	*	*	*	*	*
HE4	2	*	*	*	*	*	*
HE4	3	*	*	*	*	*	*
HE4	4	*	*	*	*	*	*
HE5	1	*	*	*	*	*	*
HE5	2	*	*	*	*	*	*
HE5	3	*	*	*	*	*	*
HE5	4	*	*	*	*	*	*
REA1	1	1	5.4040	0.8369	1.8793	2.4567	0.0000
REA1	2	2	1.1118	74.6705	0.2813	0.0051	0.0000
REA1	3	3	1.6303	1.2185	1.0537	0.0000	0.7669
REA1	4	2	1.8836	35.4145	0.5399	0.0413	0.0000
REA2	1	1	9.1555	0.7186	1.0038	0.0000	0.7124
REA2	2	1	8.6059	3.1552	1.2317	0.0000	2.3928
REA2	3	1	4.0832	-1.0982	1.2531	0.0000	2.5744
REA2	4	3	18.2908	11.8250	1.8964	0.0000	0.0000
DIS1	1	3	11.7021	2.1781	3.5936	0.0640	6.2139
DIS1	2	*	*	*	*	*	*
DIS1	3	1	6.1180	1.7997	3.4764	0.0000	0.0000
DIS1	4	*					

Cuadro 4.18: Resultados comparación de métodos para COMPleib (parte 2)

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
DIS2	1	1	3.1093	-2.2073	0.1528	0.0000	0.8466
DIS2	2	1	6.4714	21.8346	1.5006	0.0000	0.0000
DIS2	3	1	3.8367	-1.2159	1.1554	0.0000	2.9398
DIS2	4	1	1.4680	23.1539	0.3506	0.0107	0.0000
DIS3	1	2	4.2485	85.3857	1.0167	0.0358	0.0000
DIS3	2	1	43.0401	63.0160	2.3938	0.0000	0.0000
DIS3	3	1	13.0010	0.2299	2.2669	0.0000	3.8068
DIS3	4	1	2.8132	171.5497	0.5748	0.0000	0.0000
DIS4	1	2	19.2898	-15.2208	3.6008	4.1959	0.0000
DIS4	2	1	113.9877	9.7626	2.4282	0.0000	0.0000
DIS4	3	1	16.5108	-26.2749	4.8688	4.3600	0.0000
DIS4	4	1	7.4186	3.9286	2.1599	0.0000	0.0000
DIS5	1	*	*	*	*	*	*
DIS5	2	*	*	*	*	*	*
DIS5	3	*	*	*	*	*	*
DIS5	4	*	*	*	*	*	*
BDT1	1	*	*	*	*	*	*
BDT1	2	2	0.0000	-0.0123	0.0000	0.0000	0.0000
BDT1	3	*	*	*	*	*	*
BDT1	4	*	*	*	*	*	*
MFP	1	2	3.0211	355.6885	0.0793	0.0002	0.0000
MFP	2	2	31.2628	257.5035	0.0317	0.0308	0.0000
MFP	3	2	5.0481	159.5406	0.1969	0.0010	0.0000
MFP	4	1	23.8421	255.3249	0.0515	0.0500	0.0000
EB1	1	1	2008.7962	0.0267	2.4192	0.0000	7.8641
EB1	2	*	*	*	*	*	*
EB1	3	3	0.0059	3.6418	0.0000	0.0000	0.0000
EB1	4	*	*	*	*	*	*
EB2	1	1	710.3949	-0.3467	2.3848	0.0000	9.8252
EB2	2	4	0.0638	35.6053	0.0000	0.0000	0.0000
EB2	3	3	0.0040	1.2860	0.0000	0.0000	0.0000
EB2	4	4	0.0642	35.9487	0.0000	0.0000	0.0000
EB3	1	2	0.0002	0.8964	0.0000	0.0000	0.0000
EB3	2	*	*	*	*	*	*
EB3	3	2	0.0002	0.7890	0.0000	0.0000	0.0000
EB3	4	*	*	*	*	*	*
TF1	1	2	152.2033	0.0537	0.1159	30.1364	0.0000
TF1	2	*	*	*	*	*	*
TF1	3	2	17.7844	0.0279	0.0566	2.5073	0.0000
TF1	4	*	*	*	*	*	*

Cuadro 4.19: Resultados comparación de métodos para COMPleib (parte 3)

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
PSM	1	2	33.1951	-1.2402	9.7687	3.8840	0.0000
PSM	2	3	0.0045	2.4821	0.0000	0.0001	0.0000
PSM	3	4	0.4200	6.2828	0.0597	0.0685	0.0000
PSM	4	4	0.0754	3.1696	0.0023	0.0056	0.0000
NN1	1	1	195.2775	-1.1856	1.3770	0.7739	9.3955
NN1	2	1	18.4685	103.7321	1.3412	0.0002	0.0000
NN1	3	1	618.5051	1.8081	1.6388	0.0000	9.1050
NN1	4	1	38.9023	110.0768	1.6886	0.0000	0.0000
NN2	1	3	0.1101	2.5360	0.0016	0.0008	0.0000
NN2	2	3	0.6063	7.3500	0.0230	0.0155	0.0000
NN2	3	4	0.2390	5.6421	0.0010	0.0009	0.0000
NN2	4	4	0.2481	13.0989	0.0002	0.0002	0.0000
NN3	1	*	*	*	*	*	*
NN3	2	*	*	*	*	*	*
NN3	3	*	*	*	*	*	*
NN3	4	*	*	*	*	*	*
NN4	1	1	5.7558	1.0458	1.6808	0.0000	1.6554
NN4	2	1	6.3570	9.4184	1.2566	0.0106	0.0000
NN4	3	1	13.8776	0.8754	1.8483	0.0000	6.0786
NN4	4	1	5.6259	14.5298	1.6975	0.0000	0.0000
NN5	1	*	*	*	*	*	*
NN5	2	*	*	*	*	*	*
NN5	3	*	*	*	*	*	*
NN5	4	*	*	*	*	*	
NN8	1	35	0.0009	5.1971	0.0000	0.0000	0.0000
NN8	2	3	1.3400	30.5314	0.0823	0.0571	0.0000
NN8	3	1	30.1128	-0.5495	1.0561	0.0000	2.4466
NN8	4	3	0.3150	8.1234	0.0040	0.0029	0.0000
NN9	1	2	159.0649	-79626.6575	20466.2561	1394.4496	0.0000
NN9	2	*	*	*	*	*	*
NN9	3	1	130.2842	1.6699	44.4547	0.0000	4.4442
NN9	4	*	*	*	*	*	*
NN10	1	*	*	*	*	*	*
NN10	2	*	*	*	*	*	*
NN10	3	*	*	*	*	*	*
NN10	4	*	*	*	*	*	*
NN12	1	*	*	*	*	*	*
NN12	2	*	*	*	*	*	*
NN12	3	*	*	*	*	*	*
NN12	4	*	*	*	*	*	*

Cuadro 4.20: Resultados comparación de métodos para COMPleib (parte 4)

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
NN13	1	1	148.9382	-13.0810	235.0481	0.2278	0.6892
NN13	2	*	*	*	*	*	*
NN13	3	1	86.7877	-63.7041	232.5441	0.0000	0.2206
NN13	4	*	*	*	*	*	*
NN14	1	1	50.8780	-82.2349	223.3093	1.3485	0.1814
NN14	2	*	*	*	*	*	*
NN14	3	1	83.4662	83.0369	241.0373	0.0000	0.2509
NN14	4	*	*	*	*	*	*
NN15	1	4	0.0002	0.0479	0.0000	0.0000	0.0000
NN15	2	2	0.0006	0.3081	0.0000	0.0001	0.0000
NN15	3	2	0.0001	0.0086	0.0000	0.0000	0.0000
NN15	4	3	0.0001	0.3075	0.0000	0.0000	0.0000
NN16	1	1	148.9903	0.0714	2.8060	0.1322	60.4053
NN16	2	*	*	*	*	*	*
NN16	3	2	1.5911	19.0031	0.5648	0.0086	0.0000
NN16	4	*	*	*	*	*	*
NN17	1	1	4.0421	-3.3646	0.0714	0.1632	1.1563
NN17	2	3	9.7076	0.2253	0.0003	0.1312	1.0723
NN17	3	1	321.3220	-3.4302	1.9707	0.0000	41.0302
NN17	4	4	1.1487	333.7756	0.0026	0.0008	0.0000
TMD	1	1	13.4962	0.5847	0.3050	0.0000	3.6368
TMD	2	*	*	*	*	*	*
TMD	3	1	187.8849	-0.5205	0.3452	0.0000	105.7353
TMD	4	*	*	*	*	*	*
FS	1	*	*	*	*	*	*
FS	2	*	*	*	*	*	*
FS	3	*	*	*	*	*	*
FS	4	*	*	*	*	*	*
DLR1	1	3	0.0229	0.3522	0.0000	0.0000	0.0000
DLR1	2	*	*	*	*	*	*
DLR1	3	3	0.0080	0.2266	0.0000	0.0000	0.0000
DLR1	4	*	*	*	*	*	*
ROC1	1	*	*	*	*	*	*
ROC1	2	*	*	*	*	*	*
ROC1	3	*	*	*	*	*	*
ROC1	4	*	*	*	*	*	*
ROC3	1	*	*	*	*	*	*
ROC3	2	*	*	*	*	*	*
ROC3	3	*	*	*	*	*	*
ROC3	4	*	*	*	*	*	*

Cuadro 4.21: Resultados comparación de métodos para COMPleib (parte 5)

Código	Método	iter	$\|d_*\|$	$f(x^*)$	$\|h_1(x^*)\|$	$\|h_2(x^*)\|$	$\lambda_1(V^*)$
ROC4	1	*	*	*	*	*	*
ROC4	2	*	*	*	*	*	*
ROC4	3	*	*	*	*	*	*
ROC4	4	*	*	*	*	*	*
ROC6	1	1	26.3493	-0.8178	6.4408	0.0000	1.1556
ROC6	2	1	59.8519	-259.9164	3.4495	0.0180	5.0435
ROC6	3	*	*	*	*	*	*
ROC6	4	1	49.2123	96.6433	7.0554	0.0000	4.4215
ROC7	1	1	526.7379	-0.0074	1.0299	0.0000	151.0282
ROC7	2	*	*	*	*	*	*
ROC7	3	4	0.0129	1.2261	0.0000	0.0000	0.0000
ROC7	4	*	*	*	*	*	*
ROC8	1	2	79.8849	2.9700	5.4419	18.8520	0.0000
ROC8	2	*	*	*	*	*	*
ROC8	3	1	148.0863	-0.9942	0.9821	0.0000	102.0155
ROC8	4	*	*	*	*	*	*
ROC9	1	1	14.2141	-0.7213	0.1958	0.0000	4.4063
ROC9	2	*	*	*	*	*	*
ROC9	3	1	14.2343	0.9556	0.8810	0.0000	7.6559
ROC9	4	*	*	*	*	*	*

Cuadro 4.22: Resultados comparación de métodos para COMPleib (parte 6)

Capítulo 5

Conclusiones y Trabajo a futuro

5.1. Conclusiones

Las conclusiones obtenidas a partir de los resultados son las siguientes:

- La utilización de cálculo paralelo en la implementación realizada tiene utilidad sólo si el problema a abordar es de grandes dimensiones. Las pruebas de speedup realizadas muestran, en particular para el solver PCSDP, que el tiempo de ejecución no necesariamente disminuye al aumentar el número de procesos en el cálculo. Con respecto al cálculo del valor propio máximo, el escalamiento en los valores del speedup sigue un patrón similar, pero con mejores resultados.
- Se debe definir con exactitud la fase de restauración que se implementará en C y MPI. Para ello se realizaron las pruebas de comparación entre los 4 métodos propuestos.
- Los métodos 1 y 4, que proponen realizar fases de restauración con la función `fminsearch` (Nelder-Mead simplex) más soluciones SOF subóptimales, y con la función `lsdp` (restauración inexacta) más posicionamiento de polos, entregan los mejores rendimientos estadísticos con respecto a la batería de prueba COMPleib. Sin embargo, el método 1 entrega mejores resultados para la función objetivo que el método 4, por lo que se conluye que tiene un comportamiento superior a los otros.
- Conviene tener varias fases de restauración activas simultáneamente, pues en varios casos, un método obtuvo una solución y los otros no.
- Para cada tipo de problema a resolver con el algoritmo `Filter-SDP`, se debe realizar un estudio de las fases de restauración posibles. En este caso, se trabajó con problemas provenientes del control de sistemas (*Static Output Feedback*), y las fases de restauración propuestas utilizaban métodos y teoremas provenientes de esa área.

Las conclusiones obtenidas a partir del proceso de trabajo realizado son las siguientes:

- En este trabajo se intentó desarrollar una aplicación que implementaba un algoritmo proveniente del área de la Optimización, en particular la Programación semidefinida y los métodos de filtro. Esa aplicación debía utilizar cálculo de alto desempeño en áreas específicas del código y utilizar librerías existentes para su funcionamiento. Esos objetivos se lograron, sin embargo, el correcto funcionamiento del algoritmo, en particular su fase de restauración, hicieron muy difícil verificar la ventaja comparativa en tiempo que podía entregar la paralelización.

- Se aprendió a utilizar librerías clásicas en el área del cálculo de alto desempeño. Estas librerías son BLAS, LAPACK y ScaLAPACK. El valor de este aprendizaje es muy alto, puesto que continuaré estudios de postgrado en esta misma área, y tener un aprendizaje en estas librerías es muy valioso para los proyectos en los que participe en el futuro.

- El área de la programación semidefinida tiene un potencial enorme en la resolución de problemas provenientes de minería e investigación de operaciones. Dado que he trabajado en proyectos relacionados con esas áreas, he visto diferentes formulaciones y modelos matemáticos que abordan problemas industriales. Al conocer la programación semidefinida y su potencial en cuanto a velocidad de cálculo, creo que se le debe dar un mayor enfasis al momento de modelar y resolver un problema, al menos dentro del Centro de Modelamiento Matemático (CMM). Las herramientas estudiadas, instaladas y desarrolladas en este trabajo pueden facilitar la inclusión de esta área de la programación matemática en los proyectos futuros que se aborden.

Como comentario final, conviene mencionar como es el proceso de desarrollo realizado en un centro de estudios avanzados de cálculo de alto desempeño, como lo es el Centro de Supercomputación de España (*Barcelona Supercomputing Center*, BSC) y compararlo con el desarrollo realizado en este trabajo. En el BSC realizan 2 caminos para implementar una aplicación en sus sistemas de alto desempño. En el primero, una persona tiene un código ya desarrollado en algún lenguaje no orientado al cálculo paralelo, por ejemplo MATLAB ©. El equipo de desarrollo revisa ese código y genera una nueva aplicación similar a la original, pero desarrollada en un lenguaje orientado al cálculo paralelo, por ejemplo C o Fortran. Junto con el desarrollo, se genera una cooperación a nivel de trabajo, pues es valioso que la persona que originó el código vea la estructura y los resultados obtenidos, junto con las modificaciones implementadas para que pueda funcionar en alto desempeño. En el segundo camino, hay una colaboracion más profunda, se escribe código tipo maqueta, que sea correcto del punto de vista funcional y para el objetivo original del código (por ejemplo, una aplicación que resuelve problemas provenientes de la mecánica), después se optimiza algoritmicamente, primero en version serial y despues paralela.

La manera más eficiente de desarrollar una aplicación ya implementada en un lenguaje no orientado al cálculo paralelo, es contar previamente con un framework que permita el

desarrollo de aplicaciones en forma de módulos, donde los módulos centrales (o que estén en la capa más profunda del framework) realizan las operaciones de cálculo paralelo, de tal forma que el desarrollador no tiene que estar en contacto con esos módulos y sólo se concentra en implementar su aplicación en este nuevo framework.

Este trabajo sirvió como una primera aproximación al desarrollo de una aplicación en lenguaje no orientado al cálculo paralelo, utilizando un framework de trabajo (y modificandolo para una utilización futura), basado en la librería CSDP. La manera de programar las funciones y el encapsulamiento de las rutinas que realizan operaciones de alto desempeño, intentó seguir el esquema descrito en el párrafo anterior.

5.2. Trabajo a futuro

El trabajo a futuro que generó este desarrollo fue el siguiente:

- Implementación de una fase de restauración en C y MPI que garantice la convergencia y buen funcionamiento de la aplicación `fnlsdp`. Este aspecto quedó pendiente debido a que no se alcanzó a implementar una fase de restauración correcta en este sistema. Se implementó el mismo algoritmo que utilizaba la implementación original, sin embargo no se obtuvieron buenos resultados. Se requiere mayor investigación y desarrollo en esta área.

- Diseño de otras fases de restauración, o modificación de las ya existentes. Por ejemplo, la función `lsdp` , perteneciente al método de restauración inexacta, posee diversas formulaciones posibles, y en este trabajo sólo se revisó una.

- Depuración de la aplicación `fnlsdp` para generar resultados utilizando problemas más grandes y verificar el speedup obtenido. Una vez que la aplicación esté completa, se pueden generar diversos tipos de pruebas, por ejemplo, utilizando distintos solvers de programación semidefinida (SDPARA, [YFF+05]) o distintas formas de cálculo de valores propios (P_ARPACK, [SM96]).

- Estructuración de la continuación de este proyecto, pero con objetivos acotados (realización de cierto tipo de pruebas, implementación de determinadas fases de restauración, etc.), para aprovechar las instalaciones del BSC y del nuevo centro de cálculo del CMM, y poder realizar pruebas con mayor número de procesos.

Apéndice A

Utilización de librerías para álgebra lineal

A.1. Introducción

En este capítulo se revisarán ejemplos sobre aplicaciones que utilizan las librerías BLAS, LAPACK, BLACS y ScaLAPACK en un ambiente con sistema operativo Linux. Se detalla la manera de programar, compilar y ejecutar los ejemplos en lenguaje C, utilizando la librería asociada a arquitecturas Intel, *Intel MKL Library*.

A.2. BLAS

En este ejemplo se compilará una aplicación que realiza la multiplicación de 2 matrices utilizando las librerías provistas por *Intel MKL*. La estructura del ejemplo es la siguiente:

```
minimal
|-- Makefile
|-- examples
|   |-- Makefile
|   `-- test.c
|-- include
|   `-- minimal.h
`-- lib
    |-- Makefile
    `-- mat_mult.c
```

Primero revisemos la carpeta `include`. El contenido del archivo `minimal.h` se puede ver en el código A.1.

Código A.1: Archivo `minimal.h` para ejemplo BLAS

```
1  void mat_mult_raw(int n, double scale1, double scale2, double *A, double *B, double *C);
2  void print_matrix(int n,double *A);
3
4  #ifdef CAPSBLAS
5  #ifdef NOUNDERBLAS
6  void DGEMM();
7  #else
8  void DGEMM_();
9  #endif
10 #else
11 #ifdef NOUNDERBLAS
12 void dgemm();
13 #else
14 void dgemm_();
15 #endif
16 #endif
```

El las primeras líneas se observa la declaración de las funciones `mat_mult_raw` y `print_matrix`, las cuales realizan la multiplicación de matrices según la función `DGEMM` de BLAS, la cual se declara en las líneas siguientes de distintas formas según sea la implementación de BLAS utilizada (en este caso la implementación provista por *Intel MKL Library*), y la impresión por la salida estandar de la matríz resultado.

En la carpeta `lib` se encuentran las fuentes que conformarán la librería a compilar, la cual sólo contiene las 2 funciones declaradas en `minimal.h`. La fuente contenida en la carpeta `lib` es el archivo `mat_mult.c` y su contenido se puede ver en el código A.2. El archivo `Makefile` se puede ver en el código A.3.

En las líneas 15, 17, 21 y 23 de A.2 se realiza el llamado a la función `DGEMM`, la cual realiza operaciones de multiplicación entre matrices de la forma:

$$C \leftarrow \alpha AB + \beta C$$

donde $\alpha, \beta \in \mathbb{R}$, $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$ y $C \in \mathbb{R}^{m \times n}$. La especificación completa de la función se puede revisar aca:

`http://www.netlib.org/blas/dgemm.f`

La carpeta `examples` contiene al archivo `test.c` cuyo código se puede ver en A.4. En él se realiza el link (al momento de compilar) con la librería generada que contiene a las funciones declaradas y se implementa un ejemplo de ejecución.

El la línea 3 del archivo `Makefile` de la carpeta `examples` se puede observar la ruta a la librería Intel MKL en su versión 10.0.3.020 para arquitecturas de 64 bits. En la línea 4 se observa la ruta y el link con la librería de ejemplo que contiene a la función para multiplicar matrices. Y en la línea 5 se realizan los links con la librería BLAS (`-lmkl`).

La ejecución del ejemplo `test.c` entrega como resultado lo siguiente:

Código A.2: Archivo `mat_mult.c` para ejemplo BLAS

```
#include <stdlib.h>
#include <stdio.h>
#include "minimal.h"

void mat_mult_raw(n,scale1,scale2,ap,bp,cp)
     int n;
     double scale1;
     double scale2;
     double *ap;
     double *bp;
     double *cp;
{
#ifdef NOUNDERBLAS
#ifdef CAPSBLAS
          DGEMM("N","N",&n,&n,&n,&scale1,ap,&n,bp,&n,&scale2,cp,&n);
#else
          dgemm("N","N",&n,&n,&n,&scale1,ap,&n,bp,&n,&scale2,cp,&n);
#endif
#else
#ifdef CAPSBLAS
          DGEMM_("N","N",&n,&n,&n,&scale1,ap,&n,bp,&n,&scale2,cp,&n);
#else
          dgemm_("N","N",&n,&n,&n,&scale1,ap,&n,bp,&n,&scale2,cp,&n);
#endif
#endif

}

void print_matrix(int n,double *A){
        int i=0,j=0;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)
                        printf("%f\t",A[i+j*n]);
                printf("\n");
        }
}
```

Código A.3: Archivo `Makefile` de carpeta `lib` para ejemplo BLAS

```
CC=gcc

CFLAGS= -g -O0 -I../include -DBIT64

libexample.a: mat_mult.o
        ar cr libexample.a  mat_mult.o
clean:
        rm -f *.o
        rm -f libexample.a
```

Código A.4: Archivo `test.c` para ejemplo BLAS

```
#include "minimal.h"

double A[]={
3,1,3,
1,5,9,
2,6,5
};

double B[]={
1,0,0,
0,2,0,
0,0,3
};

double C[]={
0,0,0,
0,0,0,
0,0,0
};

int main(){
        int n=3;
        double scale1=1.0;
        double scale2=1.0;

        mat_mult_raw(n,scale1,scale2,A,B,C);

        print_matrix(n,C);

        return 0;
}
```

Código A.5: Archivo `Makefile` de carpeta `examples` para ejemplo BLAS

```
CC=gcc
CFLAGS= -g -O0 -I../include  -DBIT64
MKLLIB=-L /opt/intel/mkl/10.0.3.020/lib/64
LIBS=-L../lib -lexample
BLAS= -lmkl -lguide -lpthread
LIBLM= -lm

test: test.o
        $(CC) $(CFLAGS) test.o $(LIBS) $(MKLLIB) $(BLAS) $(LIBLM)  -o test
        rm -f *.o
clean:
        rm -f *.o*
        rm -f *.e*
```

```
[operedo@syntagma examples]$ ./test
3.000000        2.000000        6.000000
1.000000        10.000000       18.000000
3.000000        18.000000       15.000000
```

Código A.6: Archivo `minimal.h` para ejemplo LAPACK

```
1  int chol_raw(int n,int lda,double *A);
2  void print_matrix(int n,double *A);
3
4  #ifdef CAPSLAPACK
5  #ifdef NOUNDERLAPACK
6  void DPOTRF();
7  #else
8  void DPOTRF_();
9  #endif
10 #else
11 #ifdef NOUNDERLAPACK
12 void dpotrf();
13 #else
14 void dpotrf_();
15 #endif
16 #endif
```

A.3. LAPACK

En este ejemplo se compilará una aplicación que realiza la descomposición de Cholesky de una matríz utilizando las librerías provistas por *Intel MKL*. La estructura del ejemplo es la siguiente:

```
minimal
|-- Makefile
|-- examples
|   |-- Makefile
|   `-- test.c
|-- include
|   `-- minimal.h
`-- lib
    |-- Makefile
    `-- chol.c
```

Primero revisemos la carpeta `include`. El contenido del archivo `minimal.h` se puede ver en el código A.6.

El las primeras líneas se observa la declaración de las funciones `chol_raw` y `print_matrix`, las cuales realizan la descomposición de Cholesky según la función `DPOTRF` de LAPACK, la cual se declara en las líneas siguientes de distintas formas según sea la implementación de BLAS utilizada (en este caso la implementación provista por *Intel MKL Library*), y la impresión por la salida estandar de la matríz resultado.

En la carpeta `lib` se encuentran las fuentes que conformarán la librería a compilar, la cual sólo contiene las 2 funciones declaradas en `minimal.h`. La fuente contenida en la carpeta `lib` es el archivo `chol.c` y su contenido se puede ver en el código A.7. El archivo `Makefile` se puede ver en el código A.8.

Código A.7: Archivo `chol.c` para ejemplo LAPACK

```c
#include <stdlib.h>
#include <stdio.h>
#include "minimal.h"

int chol_raw(n,lda,A)
      int n;
      int lda;
      double *A;
{
  int info;
  int i;
  int j;

  info=0;

#ifdef NOUNDERLAPACK
  #ifdef CAPSLAPACK
    DPOTRF("U",&n,A,&lda,&info);
  #else
    dpotrf("U",&n,A,&lda,&info);
  #endif
#else
  #ifdef CAPSLAPACK
    DPOTRF_("U",&n,A,&lda,&info);
  #else
    dpotrf_("U",&n,A,&lda,&info);
  #endif
#endif

   if (info != 0)
    {
      return(1);
    };
   return(0);
}

void print_matrix(int n,double *A){
        int i=0,j=0;
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)
                        printf("%f\t",A[i+j*n]);
                printf("\n");
        }
}
```

Código A.8: Archivo `Makefile` de carpeta `lib` para ejemplo LAPACK

```
CC=gcc

CFLAGS= -g -O0 -I../include -DBIT64

libexample.a: chol.o
        ar cr libexample.a  chol.o
clean:
        rm -f *.o
        rm -f libexample.a
```

Código A.9: Archivo `test.c` para ejemplo LAPACK

```
1  #include "minimal.h"
2
3  double A[]={
4  25,15,-5,
5  15,18,0,
6  -5,0,11
7  };
8
9  int main(){
10         int n=3;
11         int lda=3;
12
13         chol_raw(n,lda,A);
14
15         print_matrix(n,A);
16
17         return 0;
18 }
```

Código A.10: Archivo `Makefile` de carpeta `examples` para ejemplo LAPACK

```
1  CC=gcc
2  CFLAGS= -g -O0 -I../include  -DBIT64
3  MKLLIB=-L /opt/intel/mkl/10.0.3.020/lib/64
4  LIBS=-L../lib -lexample
5  BLAS= -lmkl_lapack -lmkl -lguide -lpthread
6  LIBLM= -lm
7
8  test: test.o
9          $(CC) $(CFLAGS) test.o $(LIBS) $(MKLLIB) $(BLAS) $(LIBLM)  -o test
10         rm -f *.o
11 clean:
12         rm -f *.o*
13         rm -f *.e*
```

En las líneas 15, 17, 21 y 23 de A.7 se realiza el llamado a la función `DPOTRF`, la cual realiza una descomposición de Cholesky de la matríz entregada como input. La opción Ü"guarda la parte triangular superior de la descomposición y la sobreescribe en la memoria de la matríz input. La especificación completa de la función se puede revisar aca:

`http://www.netlib.org/lapack/double/dpotrf.f`

La carpeta `examples` contiene al archivo `test.c` cuyo código se puede ver en A.9. En él se realiza el link (al momento de compilar) con la librería generada que contiene a las funciones declaradas y se implementa un ejemplo de ejecución.

El la línea 3 del archivo `Makefile` de la carpeta `examples` se puede observar la ruta a la librería Intel MKL en su versión 10.0.3.020 para arquitecturas de 64 bits. En la línea 4 se observa la ruta y el link con la librería de ejemplo que contiene a la función para multiplicar matrices. Y en la línea 5 se realizan los links con las librerías LAPACK

y BLAS (`-lmkl_lapack -lmkl`). La ejecución del ejemplo `test.c` entrega como resultado lo siguiente:

```
[operedo@syntagma examples]$ ./test
5.000000        3.000000        -1.000000
15.000000       3.000000        1.000000
-5.000000       0.000000        3.000000
```

Código A.11: Archivo `minimal.h` para ejemplo ScaLAPACK

```
1  #include "mpi.h"
2  #include "PBblacs.h"
3
4  extern int descinit_(int *desc, int *m, int *n, int *mb, int *nb, int *irsrc,
5                       int *icsrc, int *ictxt, int *lld, int *info);
6  extern int numroc_(  int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
7  extern int pdelset_( double *a, int *ia, int *ja, int *desca, double *alpha);
8  extern int pdsyev_(  char *jobz, char *uplo, int *n, double *a, int *ia, int *ja,
9                       int *desca, double *w, double *z__, int *iz, int *jz,
10                      int *descz, double *work, int *lwork, int *info);
11 extern int pdsyevx_( char *jobz, char *range, char *uplo, int *n, double *a,
12                      int *ia, int *ja, int *desca, double *vl, double *vu,
13                      int *il, int *iu, double *abstol, int *m, int *nz, double *w,
14                      double *orfac, double *z__, int *iz, int *jz, int *descz,
15                      double *work, int *lwork, int *iwork, int *liwork, int *ifail,
16                      int *iclustr, double *gap, int *info);
```

A.4. BLACS y ScaLAPACK

En este ejemplo se compilarán dos aplicaciones que realizan el cálculo de valores propios de una matríz utilizando las librerías provistas por *Intel MKL*. La primera rutina calcula la totalidad de los valores propios y la segunda calcula el mayor. La estructura del ejemplo es la siguiente:

```
minimal
|-- Makefile
|-- examples
|   |-- Makefile
|   |-- pdsyev.c
|   |-- pdsyevx.c
|   `-- script.sh
`-- include
    |-- minimal.h
    `-- PBblacs.h
```

Primero revisemos la carpeta `include`. El contenido del archivo `minimal.h` se puede ver en el código A.6.

El la primera línea del código A.11 se observa la utilización del header de MPI, la librería que permite la comunicación entre procesos a bajo nivel. Luego se observa la inclusión de `PBblacs.h` que contiene los prototipos de las funciones de BLACS, las cuales realizan la comunicación de procesos a alto nivel, trabajando con el mapeo de datos entre las matrices y los procesadores involucrados. El archivo `PBblacs.h` se puede encontrar en la instalación de ScaLAPACK bajo la ruta

`$(PATH_TO_SCALAPACK)/PBLAS/SRC/PBblacs.h`.

Las 5 funciones declaradas (`descinit, numroc, pdelset, pdsyev, pdsyevx`) provienen de la librería ScaLAPACK. Las funciones `descinit, numroc` y `pdelset` se utilizan para gestionar la comunicación con BLACS. Las funciones `pdsyev` y `pdsyevx` calculan la totalidad de los valores propios y el valor propio máximo de una matríz. Ambas utilizan la distribución de datos sobre los procesos realizada por BLACS para disminuir el tiempo de cálculo de su ejecución. Las versiones secuenciales (mono-procesador) de ambas funciones, `dsyev` y `dsyevx`, se encuentran en la librería LAPACK. El detalle de las funciones en FORTRAN se puede ver en los siguientes enlaces:

- `http://www.netlib.org/scalapack/tools/descinit.f`
- `http://www.netlib.org/scalapack/tools/numroc.f`
- `http://www.netlib.org/scalapack/tools/pdelset.f`
- `http://www.netlib.org/scalapack/double/pdsyev.f`
- `http://www.netlib.org/scalapack/double/pdsyevx.f`

En la carpeta `example`, el contenido del archivo `pdsyev.c` se puede ver en el código A.12. En este ejemplo se utilizan 4 procesadores para calcular los valores propios de una matríz de 4×4. La distribución realizada es la siguiente (ver explicación de distribución cíclica por bloques de la subsección 2.4.4):

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \end{bmatrix}$$

En la línea 20 se puede observar la definición de los parámetros r, c, P_r y P_c. En las líneas 16 y 22 a 25, se inicia la paralelización con MPI y se observan las instrucciones de inicialización de la grilla de procesos utilizando BLACS. En las líneas 27 y 28 se calcula el número de filas y columnas que tendrá la submatríz utilizada por el proceso, según la fórmula (2.110) y en la línea 30 se pide memoria para almacenar localmente la submatríz. En las líneas 36, 37 y 38 se copia la submatríz desde la matríz global a la memoria local. En las líneas 42 y 47 se ejecuta la rutina para calcular los valores propios. Se ejecuta 2 veces pues en la primera corrida se calcula el tamaño de ciertos parámetros de ejecución y en la segunda se ejecuta el cálculo verdadero. en las líneas 53, 54 y 55 se cierra la grilla y se finaliza el ambiente de paralelización. El archivo `pdsyevx.c` se puede ver en el código A.13, y su funcionamiento es análogo al explicado para el ejemplo `pdsyev.c`. El archivo `Makefile` se puede ver en el código A.14

El la línea 6 del archivo `Makefile` de la carpeta `examples` se puede observar la ruta a la librería Intel MKL en su versión 10.0.3.020 para arquitecturas de 64 bits. En la línea 8 se realizan los links con las librerías LAPACK y BLAS (`-lmkl_lapack`

Código A.12: Archivo `pdsyev.c` para ejemplo ScaLAPACK

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include "minimal.h"
#define N 4
double a[N][N]={{2,1,1,1},{1,4,1,1},{1,1,2,1},{1,1,1,2}};

int main(int argc, char ** argv){
        int iam, nprocs, myrank_mpi, nprocs_mpi;
        int ictxt, prow, pcol, myrow, mycol, brow, bcol, info, lwork, i, j, n, LDA, LDB;
        int desca[9],descz[9];
        double *a0, *z0, *w0,*work;
        int izero=0, ione=1;
        char jobz, uplo;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&myrank_mpi);
        MPI_Comm_size(MPI_COMM_WORLD, &nprocs_mpi);
        n=N;
        brow=2;bcol=2;prow=2;pcol=2;
        jobz='N'; uplo='L';
        Cblacs_pinfo(&iam, &nprocs);
        Cblacs_get(-1,0,&ictxt);
        Cblacs_gridinit(&ictxt,"Row",prow, pcol);
        Cblacs_gridinfo(ictxt, &prow,&pcol, &myrow, &mycol);
        //Compute the size of the local matrices
        LDA=numroc_(&n, &brow, &myrow, &izero, &prow);
        LDB=numroc_(&n, &bcol, &myrow, &izero, &pcol);
        //allocat space for A, Z and W
        a0=(double *) malloc(LDA*LDB*sizeof(double));
        w0=(double *) malloc(n*sizeof(double));
        z0=(double *) malloc(LDA*LDB*sizeof(double));
        //initialize the array descriptor
        descinit_(desca, &n, &n, &brow, &bcol, &izero, &izero, &ictxt, &LDA, &info);
        descinit_(descz, &n, &n, &brow, &bcol, &izero, &izero, &ictxt, &LDA, &info);
        //distribute matrix to grid
        for(j=1;j<n+1;j++)
                for(i=1;i<n+1;i++)
                        pdelset_(a0,&i,&j,desca,&a[i-1][j-1]);
        work=(double *)malloc(1*sizeof(double));
        lwork=-1;
        pdsyev_(&jobz, &uplo, &n, a0 , &ione, &ione, desca, w0, z0, &ione, &ione,
                descz, work, &lwork, &info);
        lwork=(int)work[0];
        free(work);
        work=(double *)malloc(lwork*sizeof(double));
        pdsyev_(&jobz, &uplo, &n, a0, &ione, &ione, desca, w0, z0, &ione, &ione,
                descz, work, &lwork, &info);
        if(myrow==0 && mycol==0){
                printf("e1=%f\n e2=%f\n e3=%f\n e4=%f\n", w0[0], w0[1], w0[2], w0[3]);
        }
        free(work); free(w0); free(a0); free(z0);
        Cblacs_gridexit(0);
        Cblacs_exit(1);
        MPI_Finalize();
        exit(0);
}
```

Código A.13: Archivo `pdsyevx.c` para ejemplo ScaLAPACK

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "minimal.h"
#define N 4
double a[N][N]={{2,1,1,1},{1,2,1,1},{1,1,2,1},{1,1,1,2}};

int main(int argc, char ** argv){
        int iam, nprocs, myrank_mpi, nprocs_mpi;
        int ictxt, prow, pcol, myrow, mycol, brow, bcol, info, lwork, liwork;
        int i, j, n=N, LDA, LDB, izero=0, ione=1;
        int desca[9], descz[9], *queryiwork, *iwork, *ifail, *iclustr;
        double *a0, *z0, *w0, *querywork, *work, *gap, mone=-1.0, dzero=0.0;
        char jobz, uplo, range;
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &nprocs_mpi);
        brow=2;bcol=2;prow=2;pcol=2;
        jobz='N';uplo='U';range='I';
        Cblacs_pinfo(&iam, &nprocs);
        Cblacs_get(0,0,&ictxt);
        Cblacs_gridinit(&ictxt,"Row",prow, pcol);
        Cblacs_gridinfo(ictxt, &prow,&pcol, &myrow, &mycol);
        //Compute the size of the local matrices
        LDA=numroc_(&n, &brow, &myrow, &izero, &prow);
        LDB=numroc_(&n, &bcol, &mycol, &izero, &pcol);
        //allocat space for A, Z and W
        a0=(double *) malloc(LDA*LDB*sizeof(double));
        w0=(double *) malloc(n*sizeof(double));
        z0=(double *) malloc(LDA*LDB*sizeof(double));
        //initialize the array descriptor
        descinit_(desca, &n, &n, &brow, &bcol, &izero, &izero, &ictxt, &LDA, &info);
        descinit_(descz, &n, &n, &brow, &bcol, &izero, &izero, &ictxt, &LDA, &info);
        //distribute matrix to grid
        for(j=1;j<n+1;j++)
                for(i=1;i<n+1;i++)
                        pdelset_(a0,&i,&j,desca,&a[i-1][j-1]);
        char cmach = 'U'; double abstol=-1.0;
        querywork=(double *)malloc(1*sizeof(double));lwork=-1;
        queryiwork=(int *)malloc(1*sizeof(int));liwork=-1;
        ifail=(int *)malloc(n*sizeof(int));
        iclustr=(int *)malloc(2*prow*pcol*sizeof(int));
        gap=(double *)malloc(prow*pcol*sizeof(double));
        int found=0;int nz=0;
        pdsyevx_(&jobz, &range, &uplo, &n, a0, &ione, &ione, desca, &dzero, &dzero,
                 &n, &n, &mone, &found, &nz, w0, &mone, z0, &ione, &ione, descz,
                 querywork, &lwork, queryiwork, &liwork, ifail, iclustr, gap, &info);
        lwork=((int)querywork[0]);liwork=((int)queryiwork[0]);
        iwork=(int *)malloc( liwork * sizeof(int));
        work=(double *)malloc( lwork*sizeof(double));
        pdsyevx_(&jobz, &range, &uplo, &n, a0, &ione, &ione, desca, &dzero, &dzero,
                 &n, &n, &mone, &found, &nz, w0, &mone, z0, &ione, &ione, descz,
                 work, &lwork, iwork, &liwork, ifail, iclustr, gap, &info);
        if(myrow==0 && mycol==0){printf("e1=%f\n", w0[0]);}
        MPI_Barrier(MPI_COMM_WORLD);
        free(querywork);free(queryiwork);free(gap);free(iclustr);free(ifail);
        free(iwork);free(work);free(w0);free(a0);free(z0);free(desca);free(descz);
        Cblacs_gridexit(ictxt);
        MPI_Finalize();
        return(0);
}
```

Código A.14: Archivo `Makefile` de carpeta `examples` para ejemplo ScaLAPACK

```
1  CC=mpicc
2  F77=mpif77
3  CFLAGS= -Wall -g -O0 -I../include  -DBIT64
4  FFLAGS=
5
6  MKLLIB=/opt/intel/mkl/10.0.3.020/lib/64
7  LIBS=-L/home/operedo/pcsdp/pcsdp1.0gplR1/lib -lsdp -L$(MKLLIB)
8  SCALAPACKLIB= -lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64 -lmkl_lapack -lmkl -lguide
9                -lpthread
10 BLAS=
11 LIBLM= -lm -lg2c
12
13 pdsyev: pdsyev.o
14         $(CC) $(FFLAGS) pdsyev.o $(LIBS) $(SCALAPACKLIB) $(BLAS) $(LIBLM)  -o pdsyev
15         rm -f *.o
16
17 pdsyevx: pdsyevx.o
18         $(CC) $(FFLAGS) pdsyevx.o $(LIBS) $(SCALAPACKLIB) $(BLAS) $(LIBLM)  -o pdsyevx
19         rm -f *.o
20
21 clean:
22         rm pdsyev pdsyevx
23         rm -f *.o*
24         rm -f *.e*
25         rm -f *.po*
26         rm -f *.pe*
```

`-lmkl`). Y en la misma línea se realizan los links con las librerías ScaLAPACK y BLACS (`-lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64`).

La ejecución del ejemplo se realiza ejecutando el archivo `script.sh` ubicado en la carpeta `examples`. En este caso, se ejecuta bajo el sistema SGE que permite encolar trabajos a una lista de ejecución. El script es el siguiente:

La ejecución y salida es la siguiente:

```
[operedo@syntagma examples]$ qsub script.sh
Your job 151846 ("out_test_scalapack") has been submitted
[operedo@syntagma examples]$ qstat
job-ID  prior   name       user    state submit/start at     queue
-----------------------------------------------------------------
 151846 0.00000 out_test_s operedo qw    02/03/2010 20:22:15
[operedo@syntagma examples]$ qstat
job-ID  prior   name       user    state submit/start at     queue
-----------------------------------------------------------------
 151846 0.55500 out_test_s operedo r     02/03/2010 20:22:27 all.q
[operedo@syntagma examples]$ qstat
[operedo@syntagma examples]$ cat out_test_scalapack.o151846
running pdsyev...
e1=1.000000
e2=1.000000
e3=2.267949
e4=5.732051
running pdsyevx...
```

Código A.15: Archivo `script.sh` de carpeta `examples` para ejemplo ScaLAPACK

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -N out_test_scalapack
#$ -pe openmpi 4
#$ -S /bin/bash
## Nodos (queues) donde funciona ok la libreria Intel MKL
#$ -q all.q@compute-1-1
#$ -q all.q@compute-1-4
#$ -q all.q@compute-1-5
#$ -q all.q@compute-1-6
#$ -q all.q@compute-1-7
#$ -q all.q@compute-1-8
#$ -q all.q@compute-1-9
#$ -q all.q@compute-1-10
#$ -q all.q@compute-1-11
#$ -q all.q@compute-1-13

MPI=/opt/ofed/mpi/intel/openmpi-1.2.6/bin/mpirun

echo "running pdsyev..."
$MPI -np $NSLOTS -machinefile $TMPDIR/machines ./pdsyev

echo "running pdsyevx..."
$MPI -np $NSLOTS -machinefile $TMPDIR/machines ./pdsyevx
```

```
e1=5.000000
```

Apéndice B

Utilización de PCSDP

En este capítulo se explicará la instalación y ejecución del software CSDP, en su versión para memoria distribuída, llamada PCSDP. Tambien se explicará el modo de uso del sistema con un ejemplo.

El ambiente de trabajo utilizado fue:

- 32 (4 servidores con 8 cada uno) Procesadores Intel(R) Xeon(R) CPU E5405 @ 2.00GHz.
- 64 bits.
- 16 GB de memoria RAM por procesador (cada servidor tiene su propia memoria, es decir, cada grupo de 8 tiene memoria compartida).
- Red intraservidor de 1 Gbps.
- Red interservidor de 100 Mbps.

B.1. Descarga e instalación

Se debe descargar la última versión del software desde el link

`http://www.st.ewi.tudelft.nl/~ivanov/pcsdp1.0gplR1.tar.gz`

Para la instalación, se deben seguir los pasos detallados en el archivo `INSTALL` y `README`.

Código B.1: Archivo `Makefile` para ejemplo PCSDP

```
#
# Use this line to specify the C and Fortran compilers.
#
CC=mpicc
F77=mpif77
#
# Use this line to specify options for the C compiler.  You'll probably
# want to turn on optimizations. You may also have to use some of the
# following flags:
#
#  -DCAPSBLAS          if BLAS routine names are capitalized.
#  -DCAPSLAPACK        if LAPACK routine names are capitalized.
#  -DNOUNDERBLAS       if BLAS routine names have no underscore.
#  -DNOUNDERLAPACK     if LAPACK routine names have no underscore.
#  -DBIT64             For I32LP64 systems.
#
CFLAGS= -g -O0 -I../include  -DBIT64
#
# Use this line to specify options for the Fortran compiler. In fact
# Fortran is used only to link the code with the external libraries.
# In case  you expirience problems as second_underscore in the filenames
# of you BLAS, LAPACK or ScaLAPACk libraties it is a good idea to
# use the option -fno-second-underscore.
#
FFLAGS=  #-fno-second-underscore
#
# Use this line to specify where the SDP and linear algebra libraries are
# to be found.
#
# -L../lib            look in the ../lib directory
# -lsdp               get libsdp.a
# -llapack            get liblapack.a
# -lblas              get libblas.a
# -lm                 Get C math library.
#
# It's extremely likely that you'll have to change the LIBS= line for
# your particular system.
# BLACS library might have slightly different names than the specified here,
# so use the proper names available on your cluster. Please, pay attention
# on the order of ScaLAPACK and BLACS libs.
#
MKLLIB=/opt/intel/mkl/10.1.0.015/lib/em64t
LIBS=-L../lib -lsdp -L$(MKLLIB)
SCALAPACKLIB= -lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64 -lmkl_lapack
              -lmkl -lguide -lpthread
BLAS=
LIBLM= -lm

#
#  This builds the pcsdp program.
#
pcsdp: pcsdp.o
  $(CC) $(CFLAGS) pcsdp.o $(LIBS) $(SCALAPACKLIB) $(BLAS) $(LIBLM)  -o pcsdp
  rm -f *.o
#
# To clean out the directory:
#
clean:
  rm -f *.o*
  rm -f *.e*
  rm -f *.po*
  rm -f *.pe*
```

B.2. Compilación

Se realizó la compilación con las librerías *Intel MKL*, como se ve en el archivo `Makefile` del directorio `solver`, cuyo código es B.1.

Para la compilación sólo se necesita ejecutar `make` en la línea de comandos.

B.3. Ejecución

Para ejecutar la aplicación en un ambiente sin SGE, se realiza lo siguiente:

```
[operedo@leloo solver]$ mpirun -np 4 pcsdp testprob/control10.dat-s
 Iter:  0 Ap: 0.00e+00 Pobj:  1.3636364e+04 Ad: 0.00e+00 Dobj:  0.0000000e+00
 Iter:  1 Ap: 9.57e-01 Pobj:  1.4178644e+04 Ad: 9.60e-01 Dobj:  1.8138155e+05
 Iter:  2 Ap: 8.91e-01 Pobj:  1.5238147e+04 Ad: 9.62e-01 Dobj:  2.0350437e+05
 Iter:  3 Ap: 7.45e-01 Pobj:  1.1670090e+04 Ad: 1.00e+00 Dobj:  2.6289699e+05
 Iter:  4 Ap: 9.07e-01 Pobj:  1.1277979e+03 Ad: 1.00e+00 Dobj:  2.6949224e+05
 Iter:  5 Ap: 7.72e-01 Pobj:  4.4576374e+02 Ad: 1.00e+00 Dobj:  1.9014982e+05
 Iter:  6 Ap: 7.81e-01 Pobj:  2.1412945e+02 Ad: 1.00e+00 Dobj:  5.3126968e+04
 Iter:  7 Ap: 7.28e-01 Pobj:  1.0939944e+02 Ad: 1.00e+00 Dobj:  2.6194609e+04
 Iter:  8 Ap: 7.23e-01 Pobj:  6.3980919e+01 Ad: 1.00e+00 Dobj:  9.1985255e+03
 Iter:  9 Ap: 7.02e-01 Pobj:  3.6437813e+01 Ad: 1.00e+00 Dobj:  3.7330116e+03
 Iter: 10 Ap: 8.32e-01 Pobj:  1.8581541e+01 Ad: 1.00e+00 Dobj:  1.3107333e+03
 Iter: 11 Ap: 7.15e-01 Pobj:  1.1254154e+01 Ad: 1.00e+00 Dobj:  3.3532671e+02
 Iter: 12 Ap: 8.54e-01 Pobj:  6.0812516e+00 Ad: 1.00e+00 Dobj:  1.2609641e+02
 Iter: 13 Ap: 1.00e+00 Pobj:  5.5519139e+00 Ad: 9.84e-01 Dobj:  5.2875895e+01
 Iter: 14 Ap: 3.75e-01 Pobj:  2.0172361e+01 Ad: 5.45e-01 Dobj:  6.4867227e+01
 Iter: 15 Ap: 5.46e-01 Pobj:  2.3478729e+01 Ad: 1.00e+00 Dobj:  4.9079672e+01
 Iter: 16 Ap: 1.00e+00 Pobj:  3.1424294e+01 Ad: 1.00e+00 Dobj:  4.2846017e+01
 Iter: 17 Ap: 1.00e+00 Pobj:  3.5248006e+01 Ad: 1.00e+00 Dobj:  3.9365557e+01
 Iter: 18 Ap: 1.00e+00 Pobj:  3.7434806e+01 Ad: 1.00e+00 Dobj:  3.8663207e+01
 Iter: 19 Ap: 1.00e+00 Pobj:  3.8243104e+01 Ad: 1.00e+00 Dobj:  3.8558215e+01
 Iter: 20 Ap: 1.00e+00 Pobj:  3.8477625e+01 Ad: 1.00e+00 Dobj:  3.8537653e+01
 Iter: 21 Ap: 1.00e+00 Pobj:  3.8524166e+01 Ad: 1.00e+00 Dobj:  3.8533864e+01
 Iter: 22 Ap: 1.00e+00 Pobj:  3.8531906e+01 Ad: 1.00e+00 Dobj:  3.8533178e+01
 Iter: 23 Ap: 1.00e+00 Pobj:  3.8532664e+01 Ad: 1.00e+00 Dobj:  3.8533100e+01
 Iter: 24 Ap: 1.00e+00 Pobj:  3.8532919e+01 Ad: 1.00e+00 Dobj:  3.8533060e+01
 Iter: 25 Ap: 1.00e+00 Pobj:  3.8533007e+01 Ad: 1.00e+00 Dobj:  3.8533042e+01
 Iter: 26 Ap: 1.31e-01 Pobj:  3.8533002e+01 Ad: 6.88e-01 Dobj:  3.8533038e+01
 Iter: 27 Ap: 9.16e-01 Pobj:  3.8533050e+01 Ad: 1.00e+00 Dobj:  3.8533036e+01
Stuck at edge of primal feasibility, giving up.
Partial Success: SDP solved with reduced accuracy
 Primal objective value: 3.8533050e+01
 Dual objective value: 3.8533036e+01
 Relative primal infeasibility: 1.86e-07
 Relative dual infeasibility: 1.45e-09
 Real Relative Gap: -1.74e-07
 XZ Relative Gap: 9.44e-08
 DIMACS error measures: 1.86e-07 0.00e+00 6.39e-09 0.00e+00 -1.74e-07 9.44e-08
```

Para ejecutar la aplicación en un ambiente con SGE, se debe proceder de la misma manera como se describe en la subsección 3.2.7.

Apéndice C

Implementación de fases de restauración

Todos los códigos de este capítulo están escritos en lenguaje M perteneciente al software MATLAB.

C.1. Enfoque original

El código correspondiente al algoritmo 8 se puede observar en C.1.

C.2. Restauración Inexacta

Los códigos correspondientes al algoritmo 9 se puede observar en C.2 y con su función `lsdp` en C.3, C.4, C.5.

C.3. SOF

Los códigos correspondientes al algoritmo 13 se puede observar en C.6.

C.4. Posicionamiento de polos

Los códigos correspondientes al algoritmo 16 se puede observar en C.7 y con su función `generatePoleplace` en C.8, C.9, C.10, C.11, C.12.

Código C.1: Enfoque para fase de restauración: Original

```
if(step==1)
    ac=0; fr=100; step_busca=0;
    %la solucion debe ser factible y aceptable
    while((fr>0 | ac==0) & step_busca<=step_1_fail)
        if (very_first_step==1)
            very_first_step=0;
            if (no_start_point)
                step_x=genera_punto(num_iter);
            else
                step_x=x0;
            end;
        else
            if step_busca==0
                nada=outputer(1,output_info);
                x_inicial=x_current;
            else
                step_x=genera_punto(ceil(step_busca));
                x_inicial=step_x;
            end;
            step_x=busca_FL(itera,x_inicial);
        end
        filter_cand=tetha_ff(step_x);
        t=filter_cand.t;
        f=filter_cand.f;
        ac=aceptable(Fil,t,f,betta,gamma);
        display(['ac=',num2str(ac)]);
        ro=1;
        if(ac==1)  %si es aceptable, entonces es necesario ver si es factible
            display('encontro un punto aceptable');
            while(fr>0 & ro<romax)
                ro=2*ro;
                [step_dx,fr]=QP_nsdp(step_x,ro,nf_ob);
                display(['ro=',num2str(ro),', fr=',num2str(fr)]);
            end
        end
        display(['step_busca=',num2str(step_busca)]);
        step_busca=step_busca+1;
    end
    if (fr>0 | ac==0)
        %fprintf('FAILS OF STEP 1 (RESTORATION)\n');
        nada=outputer(2,output_info);
        stop=1;
    else
        x_current=step_x
        step_x=[];
        dx=step_dx
        step_dx=[];
    end;
    if stop~=1
        cont=cont+1;
        paso=1;
        output_info.alg_data_numerical=[num_iter;fr;ro];
        output_info.filter_cand=filter_cand;
        output_info.alg_data_char=tipo_iter;
        output_info.iterate=x_current;
        output_info.iterate_deriv=dx;
        nada=outputer(3,output_info);
        step=3;
    end;
end
```

Código C.2: Enfoque para fase de restauración: Inexacta

```
if(step==1)
    ac=0; fr=100; step_busca=0;
    %la solucion debe ser factible y aceptable
    while((fr>0 | ac==0) & step_busca<=step_1_fail)
        if (very_first_step==1)
            very_first_step=0;
            if (no_start_point)
                step_x=genera_punto(num_iter);
            else
                step_x=x0;
            end;
        else
            if step_busca==0
                nada=outputer(1,output_info);
                x_inicial=x_current;
            else
                step_x=genera_punto(ceil(step_busca));
                x_inicial=step_x;
            end;
            step_x=lsdp(itera,x_inicial,Fil,betta,gamma);
        end
        filter_cand=tetha_ff(step_x);
        t=filter_cand.t;
        f=filter_cand.f;
        ac=aceptable(Fil,t,f,betta,gamma);
        display(['ac=',num2str(ac)]);
        ro=1;
        if(ac==1)  %si es aceptable, entonces es necesario ver si es factible
            display('encontro un punto aceptable');
            while(fr>0 & ro<romax)
                ro=2*ro;
                [step_dx,fr]=QP_nsdp(step_x,ro,nf_ob);
                display(['ro=',num2str(ro),', fr=',num2str(fr)]);
            end
        end
        display(['step_busca=',num2str(step_busca)]);
        step_busca=step_busca+1;
    end
    if (fr>0 | ac==0)
        %fprintf('FAILS OF STEP 1 (RESTORATION)\n');
        nada=outputer(2,output_info);
        stop=1;
    else
        x_current=step_x
        step_x=[];
        dx=step_dx
        step_dx=[];
    end;
    if stop~=1
        cont=cont+1;
        paso=1;
        output_info.alg_data_numerical=[num_iter;fr;ro];
        output_info.filter_cand=filter_cand;
        output_info.alg_data_char=tipo_iter;
        output_info.iterate=x_current;
        output_info.iterate_deriv=dx;
        nada=outputer(3,output_info);
        step=3;
    end;
end
```

Código C.3: Enfoque para fase de restauración: Inexacta (parte 1)

```
function x0=lsdp(iter,x_inicial,Fil,betta,gamma)
        global p r n A B1 B C NLSDP_problem
        alpha=-1;
        nn=size(Fil,1);
        mintheta=100000;
        j=1;
        while (j<=nn)
             if((mintheta>=betta*Fil(j,1)))
                mintheta=betta*Fil(j,1);
             end
                j=j+1;
        end
        var_mat=conv_m(x_inicial);
        FF0=var_mat.var1; QQ0=var_mat.var2; VV0=var_mat.var3;
        QQ0=(QQ0+QQ0')/2; VV0=(VV0+VV0')/2;
        AF=A+B*FF0*C; MF1=AF*QQ0+QQ0*AF'+B1*B1'; MF2=AF*VV0+VV0*AF'+eye(n,n); G=-VV0;
        dF=sdpvar(p,r,'full','real');
        dQ=sdpvar(n,n,'symmetric','real');
        dV=sdpvar(n,n,'symmetric','real');
        t1=sdpvar(1,1,'full','real');
        t2=sdpvar(1,1,'full','real');
        t3=sdpvar(1,1,'full','real');
        s=sdpvar(1,1,'full','real');
        Z=sdpvar(n,n,'symmetric','real');
        constFil1=set(t3<=0.3*mintheta ); constFil2=set(t3*eye(n,n)+dV>=0);
        sigma=10; tol=1e-4; const=[]; numConst=0;
        t1Ok=0; t2Ok=0;
        if norm(MF1,'fro')<=tol && norm(MF2,'fro')>tol
                const1=set(AF*QQ0+QQ0*AF' + B1*B1' + B*dF*C*QQ0
                        + QQ0*C'*dF'*B'+AF*dQ+dQ*AF'==0);
                const2=set(t2>=norm(AF*VV0+VV0*AF' + eye(n,n) +
                        B*dF*C*VV0 + VV0*C'*dF'*B'+AF*dV+dV*AF'));
                Const=const1+const2+constFil1+constFil2;
                t2Ok=1;
        else if norm(MF2,'fro')<=tol && norm(MF1,'fro')>tol
                const1=set(AF*VV0+VV0*AF' + eye(n,n) + B*dF*C*VV0
                        + VV0*C'*dF'*B'+AF*dV+dV*AF'==0);
                const2=set(t2>=norm(AF*QQ0+QQ0*AF' + B1*B1' +
                        B*dF*C*QQ0 + QQ0*C'*dF'*B'+AF*dQ+dQ*AF'));
                Const=const1+const2+constFil1+constFil2;
                t2Ok=1;
        else if norm(MF2,'fro')<=tol && norm(MF1,'fro')<=tol
                const1=set(AF*QQ0+QQ0*AF' + B1*B1' + B*dF*C*QQ0
                        + QQ0*C'*dF'*B'+AF*dQ+dQ*AF'==0);
                const2=set(AF*VV0+VV0*AF' + eye(n,n) +
                        B*dF*C*VV0 + VV0*C'*dF'*B'+AF*dV+dV*AF'==0);
                Const=const1+const2+constFil1+constFil2;
                t2Ok=0;
        else if norm(MF2,'fro')>tol && norm(MF1,'fro')>tol
                const1=set(t2>=norm(AF*VV0+VV0*AF' + eye(n,n) +
                        B*dF*C*VV0 + VV0*C'*dF'*B'+AF*dV+dV*AF'));
                const2=set(t2>=norm(AF*QQ0+QQ0*AF' + B1*B1' +
                        B*dF*C*QQ0 + QQ0*C'*dF'*B'+AF*dQ+dQ*AF'));
                Const=const1+const2+constFil1+constFil2;
                t2Ok=1;
        end; end; end; end;
```

Código C.4: Enfoque para fase de restauración: Inexacta (parte 2)

```
[P,D]=eig(G);
[pp,idx]=esort(diag(D));
D=diag(pp);
P=P(:,idx);
GG=P*D*inv(P);
E=[];
eigsPositive=0; negative=0; allNegative=0; allPositive=0;
for i=1:size(D,1)
        if D(i,i)<=0
                eigsPositive=i-1; negative=1;
                if i==1
                        allNegative=1;
                end
                break;
        end
end
E=P(:,eigsPositive+1:size(P,2));
if negative
        const3=set(E'*(G-dV)*E<=0);
        numConst=3;
        if allNegative==0
                const4=set(t1-eigsPositive*s-trace(Z)>=0);
                const5=set(Z-(G-dV)+s*eye(n,n)>=0);
                const6=set(Z>=0);
                Const=Const+const3+const4+const5+const6;
                t1Ok=1;
        else
                Const=Const+const3;
                t1Ok=0;
        end
else
        const4=set(t1-n*s-trace(Z)>=0);
        const5=set(Z-(G-dV)+s*eye(n,n)>=0);
        const6=set(Z>=0);
        Const=Const+const4+const5+const6;
        t1Ok=1;
end
options=sdpsettings('solver','csdp','savesolveroutput',1,'verbose',1,'debug',1);
if t1Ok && t2Ok
        Obj_function=t1+sigma*t2;
else if t1Ok==0 && t2Ok
        Obj_function=sigma*t2;
else if t1Ok && t2Ok==0
        Obj_function=t1;
else if t1Ok==0 && t2Ok==0
        Obj_function=0;
end; end; end; end;

sol=solvesdp(Const,Obj_function,options);
tt1=double(t1);
tt2=double(t2);
```

Código C.5: Enfoque para fase de restauración: Inexacta (parte 3)

```
        ac=0;
        while(ac==0 && abs(alpha)>=10e-12)
            dFF=FF0+alpha*double(dF); dQQ=QQ0+alpha*double(dQ); dVV=VV0+alpha*double(dV);
            mat.var1=dFF; mat.var2=dQQ; mat.var3=dVV;
            step_x=conv_vec(mat);
            filter_cand=tetha_ff(step_x);
            t=filter_cand.t; f=filter_cand.f;
            ac=aceptable(Fil,t,f,betta,gamma);
            x0=step_x;

            H1x=(A+B*FF0*C)*QQ0+QQ0*(A+B*FF0*C)'+B1*B1';
            H2x=(A+B*FF0*C)*VV0+VV0*(A+B*FF0*C)'+eye(n,n);
            vpmix=max(0,max(eig(-VV0)));
            hx=norm(H1x,'fro')+norm(H2x,'fro')+vpmix;

            H1z=(A+B*dFF*C)*dQQ+dQQ*(A+B*dFF*C)'+B1*B1';
            H2z=(A+B*dFF*C)*dVV+dVV*(A+B*dFF*C)'+eye(n,n);
            vpmiz=max(0,max(eig(-dVV)));
            hz=norm(H1z,'fro')+norm(H2z,'fro')+vpmiz;

            H1=norm(H1x,'fro')-norm(H1z,'fro');
            H2=norm(H2x,'fro')-norm(H2z,'fro');
            vpmi=vpmix-vpmiz;

            if ac==1 && H1>0 && H2>0 && vpmi>=0
                ac=1; break;
            else
                ac=0; alpha=alpha/2;
            end
        end
end
```

Código C.6: Enfoque para fase de restauración: SOF

```
function [F,LL,K,mu] = pen1
    global p r n A B1 B C1 C D11 D12 D21 NLSDP_problem
    tau=0.4; gamma=0.8; estab=10e-5;
    a=19; b=0.1; e0=0.9; sigma0=1.5;
    F=zeros(p,r);
    mu=1;
    Amu=A-mu*eye(n,n);
    while max(real(eig(Amu)))>=0
        mu=1.5*mu; Amu=A-mu*eye(n,n);
    end
    Abarra=A+B*F*C;
    Q=eye(n,n); P=eye(n,n); R=eye(p,p);
    L=lyap(Abarra',C'*F'*R*F*C+Q); K=lyap(Abarra,P);
    j=0; sigma=sigma0;
    Fj=F; muj=mu;
    for j=1:100
        cont=1; stop=0;
        k=1;
        while cont>0
            mu=(-1/sigma)*trace(L'*K);
            Fk=F;
            Fkplus1=-inv(R)*B'*L*K*C'*inv(C*K*C');
            Abarrak=A-mu*eye(n,n)+B*Fkplus1*C;
            i=0;
            while max(real(eig(Abarrak)))>=0
              gammai=power(gamma,i);
              Fkplus1=Fk+gammai*tau*(Fkplus1-Fk);
              if gammai*tau<=estab
                stop=1;break;
              else
                Abarrak=A-mu*eye(n,n)+B*Fkplus1*C;
                i=i+1;
              end
            end
            if stop
              break;
            end
            % calcular L0 y K0
            Abarra=A-mu*eye(n,n)+B*Fkplus1*C;
            Q=eye(n,n); P=eye(n,n); R=eye(p,p);
            Lkplus1=lyap(Abarra',C'*Fkplus1'*R*Fkplus1*C+Q);
            Kkplus1=lyap(Abarra,P);
            N1=2*(B'*Lkplus1+R*Fkplus1*C)*Kkplus1*C';          norma1=norm(N1,'fro');
            N2=2*(trace(Lkplus1'*Kkplus1))+2*(sigma*mu) ;      norma2=abs(N2);
            if max(norma1,norma2)<=e0
              cont=0;break;
            end
            F=Fkplus1; L=Lkplus1; K=Kkplus1;
            k=k+1;
        end
        sigma=a*sigma0; e0=b*e0;
        Fj=F; muj=mu;
        if stop || cont==0
          break;
        end
    end
    LL=lyap(A+B*F*C,B1*B1'); LL=LL/norm(LL,'fro');
end
```

Código C.7: Enfoque para fase de restauración: Polos

```
function [F,LL,K,mu] = pen1(pert)
    global p r n A B1 B C1 C D11 D12 D21 NLSDP_problem
    tau=0.4; gamma=0.8; estab=10e-5;
    a=19; b=0.1; e0=0.9; sigma0=1.5;
    addpath(genpath('../poles/pole'));
    delta=0.75*pert+0.1;
    [dd,F]=generateFpoleplace(A,B,C,n,delta);
    mu=1;
    Amu=A-mu*eye(n,n);
    while max(real(eig(Amu)))>=0
        mu=1.5*mu; Amu=A-mu*eye(n,n);
    end
    Abarra=A+B*F*C;
    Q=eye(n,n); P=eye(n,n); R=eye(p,p);
    L=lyap(Abarra',C'*F'*R*F*C+Q); K=lyap(Abarra,P);
    j=0; sigma=sigma0;
    Fj=F; muj=mu;
    for j=1:100
        cont=1; stop=0;
        k=1;
        while cont>0
            mu=(-1/sigma)*trace(L'*K);
            Fk=F;
            Fkplus1=-inv(R)*B'*L*K*C'*inv(C*K*C');
            Abarrak=A-mu*eye(n,n)+B*Fkplus1*C;
            i=0;
            while max(real(eig(Abarrak)))>=0
              gammai=power(gamma,i);
              Fkplus1=Fk+gammai*tau*(Fkplus1-Fk);
              if gammai*tau<=estab
                stop=1;break;
              else
                Abarrak=A-mu*eye(n,n)+B*Fkplus1*C;
                i=i+1;
              end
            end
            if stop
              break;
            end
            %calcular L0 y K0
            Abarra=A-mu*eye(n,n)+B*Fkplus1*C;
            Q=eye(n,n); P=eye(n,n); R=eye(p,p);
            Lkplus1=lyap(Abarra',C'*Fkplus1'*R*Fkplus1*C+Q);
            Kkplus1=lyap(Abarra,P);
            N1=2*(B'*Lkplus1+R*Fkplus1*C)*Kkplus1*C';          norma1=norm(N1,'fro');
            N2=2*(trace(Lkplus1'*Kkplus1))+2*(sigma*mu) ;      norma2=abs(N2);
            if max(norma1,norma2)<=e0
              cont=0;break;
            end
            F=Fkplus1; L=Lkplus1; K=Kkplus1;
            k=k+1;
        end
        sigma=a*sigma0; e0=b*e0;
        Fj=F; muj=mu;
        if stop || cont==0
          break;
        end
    end
    LL=lyap(A+B*F*C,B1*B1'); LL=LL/norm(LL,'fro');
end
```

Código C.8: Algoritmo de Posicionamiento de polos (interfaz)

```
function [dd,K]=generateFpoleplace(A,B,C,n,delta)
    tam=sqrt(norm(A,'fro'));
    dd=eig(-tam*ones(n,n)+(2*tam).*randn(n));
    dd=dd-max(real(dd))-delta;
    K0=randn(size(B,2),size(C,1));
    iters=2000;
    epsilon=1e-3;
    Newton=0;
    time=cputime;
    [K,normhist]=poleplace(A,B,C,dd,K0,iters,epsilon,Newton);
    time=cputime-time;
end
```

Código C.9: Algoritmo de Posicionamiento de polos (código central)

```
function [K,normhist]=poleplace(A,B,C,dd,K0,iters,epsilon,Newton)
%[K,normhist]=poleplace(A,B,C,dd,K0,iters,epsilon,Newton);
%
%This is a MATLAB based implementation of the algorithms described in
%
%  K. Yang and R. Orsi.
%  Static output feedback pole placement via a trust region approach.
%  To appear in IEEE Transactions on Automatic Control.
%
%Inputs:
%  A,B,C    : system matrices
%  dd       : vector of desired eigenvalues (order of entries does
%             not matter)
%  K0       : initial condition for K
%  iters    : maximum number of iterations
%  epsilon     : termination parameter (i.e., desired accuracy)
%  Newton      : 0 indicates use the Levenberg-Marquardt based algorithm,
%            : 1 indicates use the trust region Newton based algorithm.
Deltahat=4; Delta=Deltahat/2; eta=0.2;
x=K0(:);
[f,g,H]=calc_f_g_H(A,B,C,dd,x,Newton);
normhist=sqrt(f);
step_taken=0;
for k=1:iters,
    if step_taken
       [f,g,H]=calc_f_g_H(A,B,C,dd,x,Newton); %%Find the current cost, grad and Hess
    end
    if norm(g)<1e-8
        disp('gradient is approx. zero');
        disp(['iters = ', num2str(k)]);
        break
    end
    [p,lambda]=exacttrust(g,H,Delta,Newton);
    fnew=calc_f(A,B,C,dd,x+p); %%Find cost at x+p
    pred_change=-g'*p-0.5*p'*H*p;
    if pred_change<=0
        break
    end
    rho=(f-fnew)/pred_change;
    if rho<0.25
        Delta=0.25*Delta;
        if Delta<1e-30
            break
        end
    else
        if (rho>0.75)&(abs(norm(p)-Delta)<0.01*Delta)
            Delta=min(2*Delta,Deltahat);
        end
    end
    if rho>eta
        x=x+p;
        normhist=[normhist; sqrt(fnew)];
        step_taken=1;
        if normhist(end)<epsilon
            break
        end
    else
        normhist=[normhist; sqrt(f)];
        step_taken=0;
    end
end
K=reshape(x,size(B,2),size(C,1));
```

Código C.10: Algoritmo de Posicionamiento de polos (cálculo de primera y segunda derivadas)

```
function [f,g,H]=calc_f_g_H(A,B,C,dd,x,Newton)
n=size(A,1); m=size(B,2); p=size(C,1);
K=reshape(x,m,p);
[V,d]=eig(A+B*K*C);
d=diag(d);
dtemp=pp_hungarian(dd,d);
W=matchlsq(dtemp,[d, V']);
d=W(:,1);
V=W(:,2:n+1)';
eigdiff=d-dd;
f=abs(eigdiff'*eigdiff);
g=zeros(m,p);
G=zeros(n,m*p);   %%q'th column of G is the eig derivatives wrt
                  %%the q'th entry of vec(K)
invVB=inv(V)*B;
CV=C*V;
if ~Newton
    for j=1:p,
        for ii=1:m,
            G(:,(j-1)*m+ii)=invVB(:,ii).*(CV(j,:).');
        end
    end
else
    Z=invVB(:)*reshape(CV.',1,p*n);
    for j=1:p,
        for ii=1:m,
            G(:,(j-1)*m+ii)=diag(Z(n*(ii-1)+1:n*ii,n*(j-1)+1:n*j));
            %%Same as above but calculated using Z
        end
    end
end
g=2*real(G'*eigdiff);
H=G'*G;
if Newton
    H2=zeros(size(H));
    for j=1:p,
        for ii=1:m,
            P=Z(n*(ii-1)+1:n*ii,n*(j-1)+1:n*j);  %%P=invVB(:,ii)*CV(j,:);
            for J=1:p,
                for II=1:m,
                    if ((J-1)*m+II)<=((j-1)*m+ii)
                        v=zeros(n,1);
                        Q=Z(n*(II-1)+1:n*II,n*(J-1)+1:n*J);  %%Q=invVB(:,II)*CV(J,:);
                        for k=1:n,
                            for t=[1:k-1,k+1:n],
                                v(k)=v(k)+(Q(k,t)*P(t,k)+Q(t,k)*P(k,t))/(d(k)-d(t));
                            end
                        end
                        H2((J-1)*m+II,(j-1)*m+ii)=eigdiff'*v;
                    end
                end
            end
        end
    end
    H2=H2+H2'-diag(diag(H2));
    H=H+H2;
end
H=2*real(H);
```

Código C.11: Algoritmo de Posicionamiento de polos (cálculo de solución región de confianza)

```
function [p,lambda]=exacttrust(g,B,Delta,Newton)
%% Implementation of nearly exact solutions to trust region problem
%% Nocedal & Wright, p78-.
%% Currently does not deal with the 'hard case'.
  I=eye(size(B,1));
  [V,D]=eig(B);
  if Newton,
      d=min(diag(D));
      if d>0
          lambda=0;
      elseif d>-1e-12
          lambda=0.000001;
      else
          lambda=-d*1.0000001;
      end
  else
      d=max(min(diag(D)),0);
      lambda=0;
  end
  for j=1:20,
      if (lambda+d)<=0
          if d~=0
              lambda=-d*1.0000001;
          else
              lambda=0.000001;
          end
      end
      p=-V*((V'*g)./(diag(D)+lambda));    %% p=-E\g;
      q=((diag(D)+lambda).^(-0.5)).*(V'*p);    %% q=R'\p;
      z1=norm(p)-Delta;
      z2=q'*q*Delta;
      if (abs(z1)<1e-60)|(abs(z2)<1e-60)
          break
      end
      lambda=lambda+(p'*p)*z1/z2;
      lambda=max(0,lambda);
  end
end
```

Código C.12: Algoritmo de Posicionamiento de polos (cálculo de función objetivo)

```
function f=calc_f(A,B,C,dd,x)
    m=size(B,2); p=size(C,1);
    K=reshape(x,m,p);
    d=eig(A+B*K*C);
    d=pp_hungarian(dd,d);
    f=abs((d-dd)'*(d-dd));
end;

function v2s=pp_hungarian(v1,v2)
%% This code matches v2 to v1.
%% It is based on hungarian.m
    n=length(v1);
    vones=ones(n,1);
    v11=v1.';
    M=abs(v11(vones,:) - v2(:,vones));
    M=M.*M;
    sigma=hungarian(M);
end;
```

Apéndice D

Conversión CSDP/COMPleib

D.1. Introducción

Se desarrolló un conversor de formato desde Compleib hacia CSDP/PCSDP para resolver linealizaciones de los problemas provenientes de Compleib usando el solver CSDP/PCSDP directamente en ANSI C. El formato que acepta CSDP/PCSDP es el siguiente:

$$\begin{array}{rrcl} \text{máx} & \text{Tr}\,(CX) & & \\ & A(X) & = & a \\ & X & \succeq & 0 \end{array} \tag{D.1}$$

donde

$$A(X) = \begin{bmatrix} \text{tr}\,(A_1X) \\ \text{tr}\,(A_2X) \\ \cdots \\ \text{tr}\,(A_mX) \end{bmatrix}. \tag{D.2}$$

$X \succeq 0$ significa que X es semidefinida positiva. Todas las matrices A_i, X y C se asumen reales y simétricas. El dual del problema anterior es

$$\begin{array}{rrcl} \text{mín} & a^Ty & & \\ & A^T(y) - C & = & Z \\ & Z & \succeq & 0 \end{array} \tag{D.3}$$

donde

$$A^T(y) = \sum_{i=1}^{m} y_iA_i. \tag{D.4}$$

El problema semidefinido nolineal a resolver en COMPleib es el siguiente (su deducción se realiza en el capítulo Resultados):

$$(COMP)\begin{cases} \min_{F,Q,V} & Tr(\underbrace{(C_1+D_{12}FC)}_{C_F}Q(C_1+D_{12}FC)^T) & \\ \text{s.a} & \underbrace{(A+BFC)}_{A_F}Q+Q(A+BFC)^T+B_1B_1^T & =0 \\ & (A+BFC)V+V(A+BFC)^T+I & =0 \\ & V & \succ 0 \end{cases} \tag{D.5}$$

con $F \in \mathbb{R}^{n_u \times n_y}$ y $Q, V \in \mathbb{R}^{n_x \times n_x}$. Este problema se describe por primera vez en la ecuación (1.2) perteneciente a la Introducción. En nuestro algoritmo debemos resolver una linealización del problema anterior, de la forma:

$$(Q(x_k,\rho))\begin{cases} \min_{d\in\mathbb{R}^n} & \nabla f(x_k)^T d+\frac{1}{2}d^TBd \\ \text{s.a} & h(x_k)+Dh(x_k)d=0 \\ & G(x_k)+DG(x_k)d \preceq 0 \\ & \|d\|_\infty \le \rho \end{cases} \tag{D.6}$$

La linealización de $(COMP)$ se obtiene derivando la función objetivo, cada fila de las restricciones de igualdad y la restricción semidefinida positiva(negativa), con respecto a las variables en las matrices F, Q, V. Para ello, se considerará un vector de variables de la siguiente forma:

$$d = [\text{vec}(F_d); \text{vec}(Q_d); \text{vec}(V_d)] \tag{D.7}$$

donde F_d, Q_d, V_d representan las matrices variables en $(COMP)$ y $\text{vec}(A)$ representa al vector de las columnas de la matríz $A \in \mathbb{R}^{m\times n}$, es decir, si $A = [a_1|\dots|a_n]$, entonces

$$\text{vec}(A) = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

Con esto, $d \in \mathbb{R}^N$, con $N = n_u n_y + 2n_x n_x$ y el problema queda:

$$(Q_{comp}(x_k,\rho))\begin{cases} \min_{d\in\mathbb{R}^N} & Tr((D_{12}F_dC)Q_kC_{F_k}^T) & \\ & +Tr(C_{F_k}Q_dC_{F_k}^T) & \\ & +Tr(C_{F_k}Q_k(D_{12}F_dC)^T) & \\ & +Tr(F_d^TF_d)+Tr(Q_d^TQ_d)+Tr(V_d^TV_d) & \\ \text{s.a} & A_{F_k}Q_k+Q_kA_{F_k}^T+B_1B_1^T & \\ & +(BF_dC)Q_k+Q_k(BF_dC)^T & \\ & +A_{F_k}Q_d+Q_dA_{F_k}^T & =0 \\ & A_{F_k}V_k+V_kA_{F_k}^T+I & \\ & +(BF_dC)V_k+V_k(BF_dC)^T & \\ & +A_{F_k}V_d+V_dA_{F_k}^T & =0 \\ & -V_k-V_d & \preceq 0 \\ & \|d\| & \le \rho \end{cases} \tag{D.8}$$

A continuación, debemos desglosar cada elemento de este problema, para dejarlo en el formato de CSDP/PCSDP.

D.2. Función objetivo

La función objetivo a desglosar es la siguiente:

$$\begin{array}{c} Tr((D_{12}F_dC)Q_kC_{F_k}^T) \\ +Tr(C_{F_k}Q_dC_{F_k}^T) \\ +Tr(C_{F_k}Q_k(D_{12}F_dC)^T) \\ +Tr(F_d^TF_d)+Tr(Q_d^TQ_d)+Tr(V_d^TV_d) \end{array} \tag{D.9}$$

Primero conviene recordar algunas propiedades del operador traza:

Lema D.2.1. *Sean A, B y C matrices de dimensiones compatibles. Entonces se tiene:*

$$Tr(ABC) = Tr(BCA) = Tr(CAB)$$

$$Tr(A) = Tr(A^T), A \in \mathbb{R}^{n\times n}$$

Con esto la función objetivo se puede escribir:

$$\begin{array}{c} 2Tr(CQ_kC_{F_k}^TD_{12}F_d) \\ +Tr(C_{F_k}^TC_{F_k}Q_d) \\ +Tr(F_d^TF_d)+Tr(Q_d^TQ_d)+Tr(V_d^TV_d) \end{array} \tag{D.10}$$

Utilizando otra propiedad del operador traza, se puede desglosar aún más:

Lema D.2.2. *Sean A, B y C matrices de dimensiones compatibles. Se tiene:*

$$Tr(ABC) = vec(A^T)^T(I \otimes B)vec(C)$$

con $\otimes$ el producto de Kronecker. Como consecuencia,

$$Tr(AC) = vec(A^T)^Tvec(C)$$

La función objetivo queda:

$$\begin{array}{c} 2\text{vec}(D_{12}^TC_FQC^T)^T\text{vec}(F_d) \\ +\text{vec}(C_FC_F^T)^T\text{vec}(Q_d) \\ +d^Td \end{array} \tag{D.11}$$

En lo anterior se utilizó

$$Tr(F_d^TF_d)+Tr(Q_d^TQ_d)+Tr(V_d^TV_d) = d^Td$$

pues

$$\begin{aligned} d^T d &= [\text{vec}(F_d); \text{vec}(Q_d); \text{vec}(V_d)]^T [\text{vec}(F_d); \text{vec}(Q_d); \text{vec}(V_d)] \\ &= \text{vec}(F_d)^T \text{vec}(F_d) + \text{vec}(Q_d)^T \text{vec}(Q_d) + \text{vec}(V_d)^T \text{vec}(V_d) \\ &= Tr(F_d^T F_d) + Tr(Q_d^T Q_d) + Tr(V_d^T V_d) \end{aligned}$$

Con esto, la función objetivo se puede representar como una función cuadrática para el vector d:

$$\left[2\text{vec}(D_{12}^T C_{F_k} Q_k C^T); \text{vec}(C_{F_k} C_{F_k}^T); 0\right]^T d + d^T I d$$

Ahora, para transformar el término cuadrático, se utilizará la siguiente propiedad:

Lema D.2.3. *Si se tiene el problema*

$$\begin{array}{ll} \text{mín} & (A_0 x + b_0)^T (A_0 x + b_0) - c_0^T x - d_0 \\ & (A_i x + b_i)^T (A_i x + b_i) - c_i^T x - d_i \leq 0, \qquad i = 1, \ldots, L \end{array} \tag{D.12}$$

entonces una formulación equivalente es la siguiente

$$\begin{array}{ll} \text{mín} & t \\ & \left[\begin{array}{cc} I & A_0 x + b_0 \\ (A_0 x + b_0)^T & c_0^T x + d_0 + t \end{array} \right] \succeq 0 \\ & \left[\begin{array}{cc} I & A_i x + b_i \\ (A_i x + b_i)^T & c_i^T x + d_i \end{array} \right] \succeq 0, \qquad i = 1, \ldots, L \end{array} \tag{D.13}$$

Usando la propiedad anterior, nuestra función objetivo se puede escribir:

$$\begin{array}{ll} \text{mín} & t \\ & \left[\begin{array}{ccc} & & \text{vec}(F_d) \\ & I & \text{vec}(Q_d) \\ & & \text{vec}(V_d) \\ \text{vec}(F_d)^T & \text{vec}(Q_d)^T \text{vec}(V_d)^T & \left(2\text{vec}(D_{12}^T C_{F_k} Q_k C^T); \text{vec}(C_{F_k} C_{F_k}^T); 0\right)^T d + t \end{array} \right] \succeq 0 \end{array} \tag{D.14}$$

D.3. Primera restricción matricial

La primera restricción matricial es la siguiente:

$$\underbrace{A_{F_k} Q_k + Q_k A_{F_k}^T + B_1 B_1^T}_{M(F_k, Q_k)} + \underbrace{(BF_d C) Q_k + Q_k (BF_d C)^T}_{U(F_d)} + \underbrace{A_{F_k} Q_d + Q_d A_{F_k}^T}_{V(Q_d)} = 0 \tag{D.15}$$

El término $M(F_k, Q_k) \in \mathbb{R}^{n_x \times n_x}$ es constante, por lo tanto no entra en el análisis. Analicemos la ecuación por columnas, para la columna $j \in \{1, \ldots, n_x\}$:

$$U(F_d)_{\cdot,j} + V(Q_d)_{\cdot,j} = -M(F_k, Q_k)_{\cdot,j} \tag{D.16}$$

La primera componente de la ecuación D.16 se puede escribir de la forma:

$$\begin{aligned}((BF_dC)Q_k)_{\cdot,j} + (Q_k(BF_dC)^T)_{\cdot,j} &= (BF_d\underbrace{CQ_k}_{\tilde{C}})_{\cdot,j} + (\underbrace{Q_kC^T}_{\tilde{C}^T}F_d^TB^T)_{\cdot,j} \\ &= BF_d\tilde{C}_{\cdot,j} + \tilde{C}^TF_d^TB^T_{\cdot,j} \\ &= B\tilde{C}_{1,j}\begin{pmatrix}(F_d)_{1,1}\\ \vdots \\ (F_d)_{n_u,1}\end{pmatrix} + \cdots + B\tilde{C}_{n_y,j}\begin{pmatrix}(F_d)_{1,n_y}\\ \vdots \\ (F_d)_{n_u,n_y}\end{pmatrix} \\ &\quad +\tilde{C}^TB^T_{1,j}\begin{pmatrix}(F_d)_{1,1}\\ \vdots \\ (F_d)_{1,n_y}\end{pmatrix} + \cdots + \tilde{C}^TB^T_{n_u,j}\begin{pmatrix}(F_d)_{n_u,1}\\ \vdots \\ (F_d)_{n_u,n_y}\end{pmatrix}\end{aligned}$$

De esta manera, los términos se pueden expresar de la siguiente manera:

$$BF_d\tilde{C}_{\cdot,j} = \underbrace{\left[\begin{array}{ccc|c|ccc} B_{\cdot,1}\tilde{C}_{1,j} & \cdots & B_{\cdot,n_u}\tilde{C}_{1,j} & \cdots & B_{\cdot,1}\tilde{C}_{n_y,j} & \cdots & B_{\cdot,n_u}\tilde{C}_{n_y,j}\end{array}\right]}_{T_1^j}\left(\begin{array}{c}(F_d)_{1,1}\\ \vdots \\ \hline (F_d)_{n_u,1}\\ \vdots \\ \hline (F_d)_{1,n_y}\\ \vdots \\ (F_d)_{n_u,n_y}\end{array}\right) \tag{D.17}$$

$$\tilde{C}^TF_d^TB^T_{\cdot,j} = \underbrace{\left[\begin{array}{ccc|c|ccc} \tilde{C}^T_{\cdot,1}B^T_{1,j} & \cdots & \tilde{C}^T_{\cdot,1}B^T_{n_u,j} & \cdots & \tilde{C}^T_{\cdot,n_y}B^T_{1,j} & \cdots & \tilde{C}^T_{\cdot,n_y}B^T_{n_u,j}\end{array}\right]}_{T_2^j}\left(\begin{array}{c}(F_d)_{1,1}\\ \vdots \\ \hline (F_d)_{n_u,1}\\ \vdots \\ \hline (F_d)_{1,n_y}\\ \vdots \\ (F_d)_{n_u,n_y}\end{array}\right) \tag{D.18}$$

La segunda componente de la ecuación D.16 se puede escribir de la forma:

$$\begin{aligned}(A_{F_k}Q_d)_{\cdot,j} + (Q_dA^T_{F_k})_{\cdot,j} &= A_{F_k}(Q_d)_{\cdot,j} + Q_d(A^T_{F_k})_{\cdot,j} \\ &= (A_{F_k})_{\cdot,1}(Q_d)_{1,j} + \cdots + (A_{F_k})_{\cdot,n_x}(Q_d)_{n_x,j} \\ &\quad + \begin{pmatrix}(Q_d)_{1,1}\\ \vdots \\ (Q_d)_{n_x,1}\end{pmatrix}(A_{F_k})_{j,1} + \cdots + \begin{pmatrix}(Q_d)_{1,n_x}\\ \vdots \\ (Q_d)_{n_x,n_x}\end{pmatrix}(A_{F_k})_{j,n_x}\end{aligned}$$

De esta manera, los términos se pueden expresar de la siguiente manera:

$$A_{F_k}(Q_d)_{\cdot,j} = \underbrace{\left[\begin{array}{c|c|c|c} 0_{n_x\times n_x} & \dots & A_{F_k} & \dots & 0_{n_x\times n_x} \end{array}\right]}_{T_3^j} \left(\begin{array}{c} (Q_d)_{1,1} \\ \vdots \\ \hline (Q_d)_{n_x,1} \\ \vdots \\ \hline (Q_d)_{1,j} \\ \vdots \\ \hline (Q_d)_{n_x,j} \\ \vdots \\ \hline (Q_d)_{1,n_x} \\ \vdots \\ (Q_d)_{n_x,n_x} \end{array}\right) \tag{D.19}$$

$$Q_d(A_{F_k}^T)_{\cdot,j} = \underbrace{\left[\begin{array}{c|c|c} (A_{F_k})_{1,j}I_{n_x\times n_x} & \dots & (A_{F_k})_{n_x,j}I_{n_x\times n_x} \end{array}\right]}_{T_4^j} \left(\begin{array}{c} (Q_d)_{1,1} \\ \vdots \\ \hline (Q_d)_{n_x,1} \\ \vdots \\ \hline (Q_d)_{1,n_x} \\ \vdots \\ (Q_d)_{n_x,n_x} \end{array}\right) \tag{D.20}$$

Con esto, el problema se escribe:

$$\left[\begin{array}{c|c|c} T_1^1+T_2^1 & T_3^1+T_4^1 & 0_{n_x\times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x}+T_2^{n_x} & T_3^{n_x}+T_4^{n_x} & 0_{n_x\times n_x^2} \end{array}\right] \left(\begin{array}{c} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{array}\right) = \text{vec}(-M(F_k,Q_k)) \tag{D.21}$$

o equivalentemente:

$$\left[\begin{array}{c} \left[\begin{array}{c|c|c} T_1^1+T_2^1 & T_3^1+T_4^1 & 0_{n_x\times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x}+T_2^{n_x} & T_3^{n_x}+T_4^{n_x} & 0_{n_x\times n_x^2} \end{array}\right] \\ -\left[\begin{array}{c|c|c} T_1^1+T_2^1 & T_3^1+T_4^1 & 0_{n_x\times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x}+T_2^{n_x} & T_3^{n_x}+T_4^{n_x} & 0_{n_x\times n_x^2} \end{array}\right] \end{array}\right] \left(\begin{array}{c} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{array}\right) + \left(\begin{array}{c} \text{vec}(M(F_k,Q_k)) \\ \text{vec}(-M(F_k,Q_k)) \end{array}\right) \geq 0_{2n_x^2\times 1} \tag{D.22}$$

D.4. Segunda restricción matricial

La segunda restricción matricial es la siguiente:

$$\underbrace{A_{F_k}V_k + V_kA_{F_k}^T + I^T}_{M'(F_k,V_k)} + \underbrace{(BF_dC)V_k + V_k(BF_dC)^T}_{U'(F_d)} + \underbrace{A_{F_k}V_d + V_dA_{F_k}^T}_{V'(Q_d)} = 0 \qquad \text{(D.23)}$$

El término $M'(F_k, V_k) \in \mathbb{R}^{n_x \times n_x}$ es constante, por lo tanto no entra en el análisis.

Esta restricción es análoga a la primera y las matrices quedan de la forma (con $\tilde{C}' = CV_k$):

$$BF_d\tilde{C}'_{\cdot,j} = \underbrace{\left[\begin{array}{ccc|c|ccc} B_{\cdot,1}\tilde{C}'_{1,j} & \dots & B_{\cdot,n_u}\tilde{C}'_{1,j} & \dots & B_{\cdot,1}\tilde{C}'_{n_y,j} & \dots & B_{\cdot,n_u}\tilde{C}'_{n_y,j} \end{array}\right]}_{(T')_1^j} \left(\begin{array}{c} (F_d)_{1,1} \\ \vdots \\ \hline (F_d)_{n_u,1} \\ \vdots \\ \hline (F_d)_{1,n_y} \\ \vdots \\ (F_d)_{n_u,n_y} \end{array}\right) \qquad \text{(D.24)}$$

$$\tilde{C}'^T F_d^T B_{\cdot,j}^T = \underbrace{\left[\begin{array}{ccc|c|ccc} \tilde{C}'^T_{\cdot,1}B^T_{1,j} & \dots & \tilde{C}'^T_{\cdot,1}B^T_{n_u,j} & \dots & \tilde{C}'^T_{\cdot,n_y}B^T_{1,j} & \dots & \tilde{C}'^T_{\cdot,n_y}B^T_{n_u,j} \end{array}\right]}_{(T')_2^j} \left(\begin{array}{c} (F_d)_{1,1} \\ \vdots \\ \hline (F_d)_{n_u,1} \\ \vdots \\ \hline (F_d)_{1,n_y} \\ \vdots \\ (F_d)_{n_u,n_y} \end{array}\right) \qquad \text{(D.25)}$$

$$A_{F_k}(V_d)_{\cdot,j} = \left[\begin{array}{c|c|c|c} 0_{n_x\times n_x} & \dots & A_{F_k} & \dots & 0_{n_x\times n_x} \end{array}\right] \left(\begin{array}{c} (V_d)_{1,1} \\ \vdots \\ \hline (V_d)_{n_x,1} \\ \vdots \\ \hline (V_d)_{1,j} \\ \vdots \\ \hline (V_d)_{n_x,j} \\ \vdots \\ \hline (V_d)_{1,n_x} \\ \vdots \\ (V_d)_{n_x,n_x} \end{array}\right) \tag{D.26}$$

$$V_d(A^T_{F_k})_{\cdot,j} = \left[\begin{array}{c|c|c} (A_{F_k})_{1,j}I_{n_x\times n_x} & \dots & (A_{F_k})_{n_x,j}I_{n_x\times n_x} \end{array}\right] \left(\begin{array}{c} (V_d)_{1,1} \\ \vdots \\ \hline (V_d)_{n_x,1} \\ \vdots \\ \hline (V_d)_{1,n_x} \\ \vdots \\ (V_d)_{n_x,n_x} \end{array}\right) \tag{D.27}$$

Con esto, el problema se escribe:

$$\left[\begin{array}{c|c|c} (T')^1_1 + (T')^1_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \\ \vdots & \vdots & \vdots \\ (T')^{n_x}_1 + (T')^{n_x}_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \end{array}\right] \left(\begin{array}{c} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{array}\right) = \text{vec}(-M'(F_k, V_k)) \tag{D.28}$$

o equivalentemente:

$$\left[\begin{array}{c} \left[\begin{array}{c|c|c} (T')^1_1 + (T')^1_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \\ \vdots & \vdots & \vdots \\ (T')^{n_x}_1 + (T')^{n_x}_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \end{array}\right] \\ -\left[\begin{array}{c|c|c} (T')^1_1 + (T')^1_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \\ \vdots & \vdots & \vdots \\ (T')^{n_x}_1 + (T')^{n_x}_2 & 0_{n_x\times n_x^2} & T^1_3 + T^1_4 \end{array}\right] \end{array}\right] \left(\begin{array}{c} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{array}\right) + \left(\begin{array}{c} \text{vec}(M'(F_k, V_k)) \\ \text{vec}(-M'(F_k, V_k)) \end{array}\right) \geq 0 \tag{D.29}$$

Ahora, juntando las restricciones D.22 y D.30, se tiene:

$$\left[\begin{array}{c} \left[\begin{array}{c|c|c} T_1^1+T_2^1 & T_3^1+T_4^1 & 0_{n_x\times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x}+T_2^{n_x} & T_3^{n_x}+T_4^{n_x} & 0_{n_x\times n_x^2} \end{array}\right] \\ -\left[\begin{array}{c|c|c} T_1^1+T_2^1 & T_3^1+T_4^1 & 0_{n_x\times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x}+T_2^{n_x} & T_3^{n_x}+T_4^{n_x} & 0_{n_x\times n_x^2} \end{array}\right] \\ \left[\begin{array}{c|c|c} (T')_1^1+(T')_2^1 & 0_{n_x\times n_x^2} & T_3^1+T_4^1 \\ \vdots & \vdots & \vdots \\ (T')_1^{n_x}+(T')_2^{n_x} & 0_{n_x\times n_x^2} & T_3^1+T_4^1 \end{array}\right] \\ -\left[\begin{array}{c|c|c} (T')_1^1+(T')_2^1 & 0_{n_x\times n_x^2} & T_3^1+T_4^1 \\ \vdots & \vdots & \vdots \\ (T')_1^{n_x}+(T')_2^{n_x} & 0_{n_x\times n_x^2} & T_3^1+T_4^1 \end{array}\right] \end{array}\right] \left(\begin{array}{c} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{array}\right) + \left(\begin{array}{c} \text{vec}(M(F_k,Q_k)) \\ \text{vec}(-M(F_k,Q_k)) \\ \text{vec}(M'(F_k,V_k)) \\ \text{vec}(-M'(F_k,V_k)) \end{array}\right) \geq \left(\begin{array}{c} 0_{2n_x^2\times 1} \\ 0_{2n_x^2\times 1} \end{array}\right) \tag{D.30}$$

La matríz del lado izquierdo tiene dimensiones $4n_x^2 \times (n_u n_y + 2n_x^2)$ y el vector constante tiene dimensiones $4n_x^2 \times 1$.

Recordemos que una restricción lineal afín de la forma $Ax + b \geq 0$ se puede escribir de la siguiente manera:

$$Ax \geq b \Leftrightarrow \textbf{diag}(Ax+b) = \textbf{diag}(b) + \sum_{j=1}^{m} x_j \textbf{diag}(A_{\cdot,j}) \succeq 0 \tag{D.31}$$

De esta manera, usando D.31 se obtiene la restricción en el formato CSDP/PCSDP.

D.5. Restricción semidefinida positiva

La restricción semidefinida positiva(negativa) a desglosar es la siguiente:

$$-V_k - V_d \preceq 0 \tag{D.32}$$

o equivalentemente:

$$V_k + V_d \succeq 0 \tag{D.33}$$

Utilizando el vector d descrito en D.7:

$$V_k + V_d \succeq 0 \Leftrightarrow V_k + \sum_{i=1}^{n_u n_y} d_i 0_{n_x\times n_x} + \sum_{i=n_u n_y+1}^{n_u n_y+n_x^2} d_i 0_{n_x\times n_x} + \sum_{i=n_u n_y+n_x^2+1}^{n_u n_y+n_x^2+n_x^2} d_i E_i \succeq 0 \tag{D.34}$$

con $E_k \in \mathbb{R}^{n_x\times n_x}$ la matríz que satisface (para $i = (k-1\,\%n_x)+1$ y $j = \left\lfloor \frac{k-1}{n_x} \right\rfloor + 1$):

$$E_k = \begin{cases} \textbf{diag}(e_i) & i = j \\ \frac{1}{2}P^{i,j} & i \neq j \end{cases} \tag{D.35}$$

para $e_i \in \mathbb{R}^{n_x^2}$ y $(P^{p,q})_{i,j} = 1$ si $p = i \wedge q = j$ ó $p = j \wedge q = i$, y $(\Pi^{p,q})_{i,j} = 0$ en caso contrario.

D.6. Restricción región de confianza

La restricción asociada a la región de confianza es la siguiente:

$$\|d\| \leq \rho \tag{D.36}$$

con d descrito en D.7. Elevando al cuadrado ambos lados, se obtiene:

$$\begin{aligned} \left\| \begin{pmatrix} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{pmatrix} \right\|^2 &\leq \rho^2 \\ (\text{vec}(F_d)^T, \text{vec}(Q_d)^T, \text{vec}(V_d)^T) \begin{pmatrix} \text{vec}(F_d) \\ \text{vec}(Q_d) \\ \text{vec}(V_d) \end{pmatrix} &\leq \rho^2 \end{aligned}$$

Utilizando el lema D.2.3, se obtiene:

$$\begin{bmatrix} & & \text{vec}(F_d) \\ & I & \text{vec}(Q_d) \\ & & \text{vec}(V_d) \\ \text{vec}(F_d)^T & \text{vec}(Q_d)^T \text{vec}(V_d)^T & \rho^2 \end{bmatrix} \succeq 0 \tag{D.37}$$

D.7. Formulación final

Finalmente, hay que juntar las restricciones D.38, D.30, D.40 y D.37 y dejarlas en el formato CSDP/PCSDP como aparece en la formulación D.3. Primero escribamos la restricción D.38 en formato CSDP/PCSDP:

$$\begin{aligned} \text{mín} \quad & t \\ & \begin{bmatrix} I_{(n_u n_y + 2n_x^2)\times(n_u n_y + 2n_x^2)} & 0_{(n_u n_y + 2n_x^2)\times 1} \\ 0_{1\times(n_u n_y + 2n_x^2)} & 0 \end{bmatrix} + \textstyle\sum_{j=1}^{n_u n_y + 2n_x^2} d_j \begin{bmatrix} 0_{(n_u n_y + 2n_x^2)\times(n_u n_y + 2n_x^2)} & e_j \\ e_j^T & \gamma_j \end{bmatrix} \\ & + t \begin{bmatrix} 0_{(n_u n_y + 2n_x^2)\times(n_u n_y + 2n_x^2)} & 0_{(n_u n_y + 2n_x^2)\times 1} \\ 0_{1\times(n_u n_y + 2n_x^2)} & 1 \end{bmatrix} \succeq 0 \end{aligned} \tag{D.38}$$

donde $\gamma = \big(2\text{vec}(D_{12}^T C_{F_k} Q_k C^T); \text{vec}(C_{F_k} C_{F_k}^T); 0\big)$. La restricción D.30 se puede escribir de la forma:

$$\underbrace{\textbf{diag}(b)}_{4n_x^2 \times 4n_x^2} + \sum_{j=1}^{n_u n_y + 2n_x^2} d_j \underbrace{\textbf{diag}(A_{\cdot,j})}_{4n_x^2 \times 4n_x^2} \succeq 0 \tag{D.39}$$

donde

$$b = \begin{pmatrix} \text{vec}(M(F_k, Q_k)) \\ \text{vec}(-M(F_k, Q_k)) \\ \text{vec}(M'(F_k, V_k)) \\ \text{vec}(-M'(F_k, V_k)) \end{pmatrix}$$

$$A = \begin{bmatrix} \left[\begin{array}{c|c|c} T_1^1 + T_2^1 & T_3^1 + T_4^1 & 0_{n_x \times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x} + T_2^{n_x} & T_3^{n_x} + T_4^{n_x} & 0_{n_x \times n_x^2} \end{array}\right] \\ -\left[\begin{array}{c|c|c} T_1^1 + T_2^1 & T_3^1 + T_4^1 & 0_{n_x \times n_x^2} \\ \vdots & \vdots & \vdots \\ T_1^{n_x} + T_2^{n_x} & T_3^{n_x} + T_4^{n_x} & 0_{n_x \times n_x^2} \end{array}\right] \\ \left[\begin{array}{c|c|c} (T')_1^1 + (T')_2^1 & 0_{n_x \times n_x^2} & T_3^1 + T_4^1 \\ \vdots & \vdots & \vdots \\ (T')_1^{n_x} + (T')_2^{n_x} & 0_{n_x \times n_x^2} & T_3^1 + T_4^1 \end{array}\right] \\ -\left[\begin{array}{c|c|c} (T')_1^1 + (T')_2^1 & 0_{n_x \times n_x^2} & T_3^1 + T_4^1 \\ \vdots & \vdots & \vdots \\ (T')_1^{n_x} + (T')_2^{n_x} & 0_{n_x \times n_x^2} & T_3^1 + T_4^1 \end{array}\right] \end{bmatrix}$$

La restricción D.40 se escribe de la forma:

$$(V_k)_{n_x \times n_x} + \sum_{i=1}^{n_u n_y} d_i 0_{n_x \times n_x} + \sum_{i=n_u n_y+1}^{n_u n_y+n_x^2} d_i 0_{n_x \times n_x} + \sum_{i=n_u n_y+n_x^2+1}^{n_u n_y+n_x^2+n_x^2} d_i (E_i)_{n_x \times n_x} \succeq 0 \quad \text{(D.40)}$$

con $E_i \in \mathbb{R}^{n_x \times n_x}$ la matríz que satisface $\text{vec}(E_i) = e_i$, con e_i vector canónico en $\mathbb{R}^{n_x^2}$. La restricción D.37 se escribe de la forma:

$$\begin{bmatrix} I_{(n_u n_y+2n_x^2)\times(n_u n_y+2n_x^2)} & 0_{(n_u n_y+2n_x^2)\times 1} \\ 0_{1\times(n_u n_y+2n_x^2)} & \rho^2 \end{bmatrix} + \sum_{j=1}^{n_u n_y+2n_x^2} d_j \begin{bmatrix} 0_{(n_u n_y+2n_x^2)\times(n_u n_y+2n_x^2)} & e_j \\ e_j^T & 0 \end{bmatrix} \quad \text{(D.41)}$$

Apéndice E

Documentación de `fnlsdp`

Para revisar la documentación completa de la aplicación, visitar:

`http://www.dim.uchile.cl/~operedo/ma69f/fnlsdp/doc`

Esta documentación esta en continua actualización, por lo cual se debe enviar un correo a `operedo` en `dim.uchile.cl` para saber el estado actual de la aplicación.

Bibliografía

[ABD+90] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen, *Lapack: a portable linear algebra library for high-performance computers*, Supercomputing '90: Proceedings of the 1990 ACM/IEEE conference on Supercomputing (Washington, DC, USA), IEEE Computer Society, 1990, pp. 2–11.

[AHO96] F. Alizadeh, J. A. Haeberly, and M. L. Overton, *Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results*, SIAM Journal on Optimization **5** (1996), 13–51.

[Ali95] F. Alizadeh, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J. Optim. **5** (1995), no. 1, pp. 13–51.

[BCC+96] L. S. Blackford, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker, *Scalapack: a portable linear algebra library for distributed memory computers - design issues and performance*, Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM) (Washington, DC, USA), IEEE Computer Society, 1996, p. 5.

[BEFB93] S. Boyd, L. El-Ghaoui, E. Feron, and V. Balakrishnan, *Linear matrix inequalities in system and control theory*, Proc. Annual Allerton Conf. on Communication, Control and Computing (Allerton House, Monticello, Illinois), oct 1993.

[Bor98] Brian Borchers, *Sdplib 1.1, a library of semidefinite programming test problems*, in Optimization Methods and Software, 1998.

[Bor99] B. Borchers, *Csdp, a c library for semidefinite programming*, Optimization Methods and Software **11/12** (1999), 613–623.

[BSV02] H.Y. Benson, D.F. Shanno, and R.J. Vanderbei, *Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions*, Comput. Optim. Appl. **23** (2002), no. 2, pp. 257–272.

[BTe93] A. Ben-Tal and M. P. Bendsøe, *A new method for optimal truss topology design*, SIAM Journal on Optimization **3** (1993), no. 2, 322–358.

[BW80] P. Bentler and J. Woodward, *Inequalities among lower bounds to reliability: With applications to test construction and factor analysis*, Psychometrika **45** (1980), no. 2, 249–267.

[BY07] B. Borchers and J. G. Young, *Implementation of a primal—dual method for sdp on a shared memory parallel architecture*, Comput. Optim. Appl. **37** (2007), no. 3, 355–369.

[Che70] Chi-Tsong Chen, *Introduction to linear system theory*, Holt, Rinehart and Winston, New York, 1970.

[DMBS79] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *Linpack users' guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[Dun90] R. Duncan, *A survey of parallel computer architectures*, Computer **23** (1990), no. 2, 5–16.

[DW95] J. J. Dongarra and R. C. Whaley, *A user's guide to the blacs v1.1*, Tech. report, 1995.

[FGL+02] R. Fletcher, N.I.M. Gould, S. Leyffer, Ph.L. Toint, and A. Wächter, *Global convergence of a trust-region SQP-filter algorithms for nonlinear programming*, SIAM J. Optim. **13** (2002), no. 3, pp. 635–659.

[FL02] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Mathematical Programming **91** (2002), no. 2, Ser. A, pp. 239–269.

[Fle81] R. Fletcher, *A nonlinear programming problem in statistics (educational testing)*, SIAM Journal on Scientific and Statistical Computing **2** (1981), no. 3, 257–267.

[Fle85] ———, *Semidefinite matrix constraints in optimization*, SIAM Journal on Control and Optimization **23** (1985), 493–513.

[FLT02] R. Fletcher, S. Leyffer, and Ph.L. Toint, *On the global convergence of an sqp-filter algorithm*, SIAM J. Optimization **13** (2002), no. 1, pp. 44–59.

[For94] Message Passing Interface Forum, *Mpi: A message-passing interface standard*, International Journal on Supercomputer Applications and High Performance Computing **8** (1994), no. 3/4.

[GBDM77] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix eigensystem routines – eispack guide extension*, Lecture Notes in Computer Science, vol. 51, Springer-Verlag, Berlin and New York, 1977.

[GL96] G. H. Golub and C. F. Van Loan (eds.), *Matrix computations, third edition*, The Johns Hopkins University Press, Baltimore, 1996.

[GR06] W. Gómez and H. Ramírez, *A filter algorithm for nonlinear semidefinite programming*, Tech. report, Centro de Modelamiento Matemático, 2006.

[GW95] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM **42** (1995), no. 6, 1115–1145.

[HR00] C. Helmberg and F. Rendl, *A spectral bundle method for semidefinite programming*, SIAM J. Optimization **10** (2000), no. 3, pp. 673–696.

[HRVW96] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz, *An interior-point method for semidefinite programming*, SIAM Journal on Optimization **6** (1996), 342–361.

[HUL96] J.-B. Hiriart-Urruty and C. Lemarechal, *Convex analysis and minimization algorithms ii*, Springer, Berlin, 1996.

[HUY95] J.-B. Hiriart-Urruty and D. Ye, *Sensitivity analysis of all eigenvalues of a symmetric matrix*, Numer. Math. **70** (1995), no. 1, 45–72.

[IK07] I.D. Ivanov and E. de Klerk, *Parallel implementation of a semidefinite programming solver based on csdp in a distributed memory cluster*, Discussion Paper 2007-20, Tilburg University, Center for Economic Research, 2007.

[Inc97] The MathWorks Inc., *Using matlab*, Tech. report, The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, USA, 1997.

[inf] *Infiniband*, `http://www.infinibandta.org/`.

[Kat82] T. Kato, *A short introduction to perturbation theory for linear operators*, New York: Springer-Verlag, 1982.

[KR78] B. W. Kernighan and D. Ritchie, *The c programming language*, Prentice-Hall, 1978.

[KSH97] M. Kojima, S. Shindoh, and S. Hara, *Interior-point methods for the monotone semidefinite linear complementarity problem in symmetric matrices*, SIAM Journal on Optimization **7** (1997), no. 1, 86–125.

[Kur76] S. Kurcyusz, *On the existence and nonexistence of lagrange multipliers in banach spaces*, Journal of Optimization Theory and Applications **20** (1976), no. 1, pp. 81–110.

[LÖ4] J. Löfberg, *Yalmip : A toolbox for modeling and optimization in matlab*.

[Lei04] F. Leibfritz, *Compleib: Constrained matrix-optimization problem library- a collection of test examples for nonlinear semidefinite programming, control system design and related problems.*, Tech. report, Universität Trier, Trier, Germany, 2004.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for fortran usage*, ACM Trans. Math. Softw. **5** (1979), no. 3, 308–323.

[LL03] F. Leibfritz and W. Lipinski, *Description of the benchmark examples in compleib.*, Tech. report, Universität Trier, Trier, Germany, 2003.

[LL04] _____, *Compleib 1.0 - user manual and quick reference.*, Tech. report, Universität Trier, Trier, Germany, 2004.

[Lov79] L. Lovasz, *On the shannon capacity of a graph*, IEEE Transactions on Information Theory **25** (1979), no. 1, pp. 1–7.

[Mar63] D. W. Marquardt, *An algorithm for least-squares estimation of nonlinear parameters*, SIAM Journal on Applied Mathematics **11** (1963), no. 2, 431–441.

[Meh92] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization **2** (1992), no. 4, 575–601.

[mkl] *Intel math kernel library*, `http://www.intel.com/software/products/mkl/docs/WebHelp/mkl.htm`.

[Mon97] R. D. C. Monteiro, *Primal-dual path following algorithms for semidefinite programming*, SIAM Journal on Optimization **7** (1997), 663–678.

[Mos08] El-S. M.E. Mostafa, *First-order penalty methods for computing suboptimal output feedback controllers*, Appl. and Comput. Math. **7** (2008), no. 1, pp. 66–83.

[myr] *Myricom*, `http://www.myri.com/Myri-10G/overview/`.

[MZ99] R. D. C. Monteiro and P. Zanjácomo, *Implementation of primal-dual methods for semidefinite programming based on monteiro and tsuchiya newton directions and their variants*, Optimization Methods and Software **11** (1999), pp. 91–140.

[NM65] J. A. Nelder and R. Mead, *A simplex method for function minimization*, The Computer Journal **7** (1965), no. 4, pp. 308–313.

[NT97] Y. E. Nesterov and M. J. Todd, *Self-scaled barriers and interior-point methods for convex programming*, Math. Oper. Res. **22** (1997), no. 1, 1–42.

[NW99] J.Ñocedal and S. J. Wright, *Numerical optimization*, Springer, 1999.

[ope] *Openmp*, `http://openmp.org/wp/`.

[Ove92] M. L. Overton, *Large-scale optimization of eigenvalues*, SIAM J. Optimization **2** (1992), 88–120.

[Pac96] P. S. Pacheco, *Parallel programming with mpi*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Par87] B. Parlett, *The symmetric eigenvalue problem (classics in applied mathematics)*, Society for Industrial Mathematics, January 1987.

[pvm] *Pvm*, `http://www.csm.ornl.gov/pvm/pvm_home.html`.

[Rob76] S. M. Robinson, *Stability theorems for systems of inequalities, part ii: Differentiable nonlinear systems*, SIAM J. Numer. Anal. **13** (1976), pp. 497–513.

[SBD+76] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix eigensystem routines – eispack guide*, 2nd ed., Lecture Notes in Computer Science, vol. 6, Springer-Verlag, Berlin and New York, 1976.

[Sha82] A. Shapiro, *Weighted minimum trace factor analysis*, Psychometrika **47** (1982), pp. 243–264.

[SM96] D. C. Sorensen and K. J. Maschhoff, *P_arpack: An efficient portable large scale eigenvalue package for distributed memory parallel architectures*, In: Proceedings PARA96 conference, Lingby, Springer, 1996, pp. 478–86.

[SM08] C. Silva and M. Monteiro, *A filter inexact-restoration method for nonlinear programming*, TOP: An Official Journal of the Spanish Society of Statistics and Operations Research **16** (2008), no. 1, 126–146.

[Stu99] J. F. Sturm, *Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones*, Optimization Methods and Software **11/12** (1999), no. 1-4, 625–653.

[Ulb04] S. Ulbrich, *On the superlinear local convergence of a filter-SQP method*, Math. Program. **100** (2004), no. 1, Ser. B, pp. 217–245. MR MR2072932 (2005d:90120)

[UUV04] M. Ulbrich, S. Ulbrich, and L.N. Vicente, *A globally convergent primal-dual interior-point filter method for nonconvex nonlinear programming*, Mathematical Programming **100** (2004), no. 2, Ser. A, pp. 379–410.

[Wan92] X. Wang, *Pole placement by static output feedback*, J. Math. Systems Estim. Control **2** (1992), pp. 205–218.

[WB05a] A. Wächter and L.T. Biegler, *Line search filter methods for nonlinear programming: Local convergence*, SIAM J. Optimization **16** (2005), no. 1, pp. 32–48.

[WB05b] _____, *Line search filter methods for nonlinear programming: Motivation and global convergence*, SIAM J. Optimization **16** (2005), no. 1, pp. 1–31.

[YFF+05] M. Yamashita, K. Fujisawa, M. Fukuda, M. Kojima, and K.Ñakata, *Series b: Operations research b-415 parallel primal-dual interior-point methods for semidefinite programs*, 2005.

[YO07] K. Yang and R. Orsi, *Static output feedback pole placement via a trust region approach*, IEEE Transactions on Automatic Control **52** (2007), no. 11, pp. 2146–2150.

[Zha98] Y. Zhang, *On extending some primal-dual interior-point algorithms from linear programming to semidefinite programming*, SIAM Journal on Optimization **8** (1998), 365–386.