PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 11-10

Computación de alto desempeño para la reducción de modelos

Pablo Ezzatti **Enrique S. Quintana-Ortí**

Alfredo Remón

2011

Computación de alto desempeño para la reducción de modelos
Ezzatti, Pablo; Quintana-Ortí, Enrique S.; Remón, Alfredo
ISSN 0797-6410
Reporte Técnico RT 11-10
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, junio de 2011

Computación de alto desempeño para la reducción de modelos

Pablo Ezzatti[1], Enrique S. Quintana-Ortí[2] and Alfredo Remón[2]

[1]Centro de Cálculo, Instituto de Computación, Facultad de Ingeniería,
Universidad de la República, Uruguay
pezzatti@fing.edu.uy

[2]Depto. de Ingeniería y Ciencia de Computadores,
Universidad Jaume I, 12.071–Castellón, España
{quintana,remon}@icc.uji.es

junio 2011

Palabras claves: reducción de modelos, computación de alto desempeño, álgebra lineal numérica.

Resumen

El interés creciente por contar con modelos matemáticos que permitan realizar simulaciones, evaluar posibles diseños, estudiar impactos, etc. en distintos campos de ingeniería, pero a su vez la necesidad de que estos modelos sean tratables en un tiempo aceptable, dan origen al campo de trabajo de la reducción de modelos. Estas técnicas buscan, dado un modelo matemático, encontrar otro cuya dimensión sea considerablemente menor pero que presente un comportamiento similar al del modelo original. De esta forma, es posible utilizar el modelo reducido en posteriores simulaciones o estudios, disminuyendo así las necesidades de cómputo y tiempo de ejecución.

Este trabajo presenta una pequeña introducción a la temática de reducción de modelos, con particular interés en los métodos basados en realizaciones balanceadas. Además, se estudia las técnicas de HPC y su aplicación para la aceleración de los métodos de reducción de modelos.

1. Introducción

Diversos fenómenos físicos se pueden modelar mediante ecuaciones diferenciales. Estos modelos matemáticos se utilizan por ejemplo en el diseño de dispositivos electrónicos, donde es necesario especificar la posición y conexiones entre los circuitos. El alto costo temporal y económico hace inviable afrontar la fabricación y evaluación empírica de cada posible diseño de estos dispositivos. Una opción más realista consiste en especificar un modelo matemático que describa esta realidad (cómo afecta al funcionamiento y costo de producción del dispositivo la posición de cada uno de los circuitos y sus conexiones) y evaluar, utilizando el modelo matemático, todos los diseños posibles.

Generalmente, la simulación de los modelos matemáticos derivados de aplicar técnicas de control requiere un número elevado de cálculos. Este número depende de la dimensión del modelo, o lo que es lo mismo, del número de variables involucradas en el modelado. Además, la cantidad de operaciones necesarias para simular un modelo crece rápidamente con la dimensión del modelo, de forma que, aún en casos en los que se opera con modelos de dimensiones modestas, se require de la ayuda de computadoras para su resolución. El desarrollo vertiginoso de la computación en los últimos años, con la propuesta de nuevos algoritmos y técnicas de programación, y sobre todo, la aparición de nuevas plataformas hardware de gran desempeño, han posibilitado resolver cada vez problemas más complejos, de mayor dimensión y con mayor precisión. A pesar de ello, la necesidad de resolver problemas de mayor dimensión supera con creces las posibilidades de las computadoras actuales. Utilizando el caso antes mencionado de los dispositivos electrónicos, se puede tomar como ejemplo los teléfonos móviles, donde en un espacio físico cada día más reducido se integra una cantidad mayor de funcionalidades (teléfono propiamente dicho, cámara de foto, GPS, etc.) y cuyas componentes deben respetar diversas restricciones y cualidades deseables (consumo y trasmisión de energía, campo magnético, interferencias, etc.).

Los métodos de reducción de modelos pueden ser vistas como una función, que va del conjunto de modelos originales al de los modelos reducidos. Ello implica que estos métodos deben operar sobre el sistema original, y dada su dimensión, en muchos casos su costo computacional es excesivo para ser tratados con computadores personales. Dicha situación ha motivado la aplicación de técnicas de computación de alto desempeño (HPC, del inglés *high performance computing*) para poder abordar problemas de grandes dimensiones y acelerar las técnicas. En particular, y debido al tipo de cómputos involucrados, el uso de técnicas de HPC aplicadas a la resolución de operaciones de álgrebra lineal numérica (ALN).

El resto del trabajo se estructura de la siguiente forma. En la Sección 2 se estudian y describen los métodos numéricos para el cómputo de la reducción de modelos, profundizando en las técnicas basadas en la función de signo. Luego, en la Sección 3 se expone una pequeña introducción al uso de las técnicas de computación de alto desempeño. Un relevamiento de la aplicación de técnicas de HPC a la reducción de modelos se encuentra en la Sección 4. Finalmente, el trabajo se resume en la Sección 5.

2. El problema de reducción de modelos

En lo que resta de la sección se describen algunos métodos numéricos para la reducción de modelos y las técnicas computacionales para su resolución.

2.1. Reducción de modelos

La formulación clásica de un sistema dinámico lineal (SDL) continuo e invariante en el tiempo, mediante el modelo de espacio de estados [90], queda definida por dos ecuaciones matriciales de la siguiente forma:

$$\begin{array}{rcll} \dot{x}(t) & = & Ax(t) + Bu(t), & t > 0, \quad x(0) = x_0, \\ y(t) & = & Cx(t) + Du(t), & t \geq 0, \end{array} \tag{1}$$

donde $x(t) \in \mathbb{R}^n$ denota el vector de variables de estado, $x(0) = x_0$ es el estado inicial del sistema, $u(t) \in \mathbb{R}^m$ es el vector de entradas o controles, $y(t) \in \mathbb{R}^p$ es el vector de salidas, $A \in \mathbb{R}^{n \times n}$ se conoce como la matriz de estados, $B \in \mathbb{R}^{n \times m}$ es la matriz de entradas, $C \in \mathbb{R}^{p \times n}$ es la matriz de salidas, y $D \in \mathbb{R}^{p \times m}$. Además, n es el orden (o la dimensión del espacio de estados) del sistema. La función de transferencia matricial asociada a (1), deducida de tomar su transformada de Laplace y asumiendo $x_0 = 0$, está definida como

$$G(s) = C(sI_n - A)^{-1}B + D, \tag{2}$$

donde I_n es la matriz identidad de orden n.

El modelo de representación en el espacio de estados ha permitido la aplicación de numerosos resultados provenientes del álgebra lineal numérica (ALN) al ámbito de la teoría de control [72, 90]. Así, los nuevos métodos desarrollados para este modelo permiten abordar, de manera numéricamente estable, problemas de grandes dimensiones que serían difícilmente tratables usando otro tipo de técnicas [79, 90].

Considerando el SDL de orden n en (1) y la matriz de función de transferencia asociada en (2), los métodos de reducción de modelos buscan obtener un segundo SDL,

$$\begin{array}{rcll} \dot{x}_r(t) & = & A_r x_r(t) + B_r u(t), & t > 0, \quad x_r(0) = \bar{x}_0, \\ y_r(t) & = & C_r x_r(t) + D_r u(t), & t \geq 0, \end{array} \tag{3}$$

de orden r mucho menor que n, y con función de transferencia:

$$G_r(s) = C_r(sI_r - A_r)^{-1}B_r + D_r, \tag{4}$$

que mantenga la información fundamental sobre la dinámica del sistema original; en particular, el objetivo es que las funciones $G_r(s)$ y $G(s)$ presenten un

comportamiento similar (en otras palabras, que $\|y - y_r\|$ sea "pequeño"). Bajo estas premisas, el nuevo SDL en (3), de orden $r \ll n$, es una aproximación del SDL original en (1), lo que permite reemplazarlo en diferentes análisis o simulaciones posteriores.

Esta sustitución puede ser de vital importancia, entre otras, en las siguientes circunstancias:

- Cuando los recursos disponibles no permiten operar con sistemas de gran dimensión (por ejemplo, en dispositivos empotrados con una memoria limitada).
- Cuando la aplicación impone un tiempo de respuesta máximo que es imposible de alcanzar si se opera con sistemas de gran dimensión (por ejemplo, en entornos que operan en tiempo real).

Los métodos de reducción de modelos más populares se pueden agrupar en dos grandes familias: los métodos de aproximación basados en descomposiciones SVD [81, 89] y los métodos de aproximación por subespacios de Krylov [48, 68, 67, 71, 93].

Entre los métodos de reducción de modelos basados en aproximación por SVD se pueden encontrar dos clases, los métodos de error absoluto y los de error relativo. Entre los primeros destacan los algoritmos de truncamiento balanceado (BT, del inglés *balanced truncation*) [83, 96, 99, 102], los algoritmos de aproximación de perturbaciones singulares (*singular perturbation approximation*) [82] y los algoritmos de aproximación de la norma de Hankel (*Hankel-norm approximation*) [51]. Todos estos métodos presentan, como problema computacional principal, la resolución de un par de ecuaciones matriciales (lineales) de Lyapunov que tienen por operandos las matrices de estados, entradas y salidas del SDL. Entre los métodos de error relativo destaca el método de truncamiento estocástico balanceado (BST, del inglés *balanced stochastic truncation*) [44]. En estos métodos, el problema computacional clave es la solución de una ecuación de Lyapunov y una ecuación matricial (cuadrática) de Riccati.

Los métodos de aproximación por subespacios de Krylov básicamente requieren el cálculo de algún tipo de factorización de la matriz de controlabilidad (o alcanzabilidad) del SDL, definida por:

$$R_k(A, B) = \left[B,\ AB,\ (A)^2B,\ \ldots, (A)^{k-1}B\right].$$

Esta factorización se obtiene, habitualmente, mediante el algoritmo iterativo de Arnoldi o alguna de sus variantes [53, 95].

Este trabajo se centra en los métodos SVD y, más concretamente, en los métodos de error absoluto BT, ya que es una de las técnicas de reducción de modelos más popular en teoría de control. Si bien los métodos basados en la SVD en general presentan un mayor costo computacional que la aproximación por subespacios de Krylov, los primeros tienen como ventaja principal que garantizan la preservación de importantes propiedades del sistema original, como la estabilidad y la pasividad [14]. Además, este tipo de métodos proporciona una cota del error introducido por el modelo reducido, lo cual permite desarrollar métodos adaptativos que reducen la dimensión del sistema en función de un margen de error requerido. En forma complementaria, el trabajo abarca también una aproximación primaria a las técnicas de error relativo, estudiando el método BST.

2.1.1. Métodos basados en la SVD

La controlabilidad de un sistema es un concepto similar a encontrar el gramiano de controlabilidad. En particular, los métodos de reducción de modelos BT se basan en encontrar un sistema de coordenadas apropiado para el espacio de estados, en el cual los gramianos de controlabilidad y observabilidad del sistema, $W_c \in \mathbb{R}^{n\times n}$ y $W_o \in \mathbb{R}^{n\times n}$ respectivamente, son diagonales e iguales. Es más, el método BT explota el hecho de que los gramianos del SDL en (1) pueden ser obtenidos como la solución de las siguientes ecuaciones acopladas de Lyapunov:

$$AW_c + W_cA^T + BB^T = 0, \quad A^TW_o + W_oA + C^TC = 0. \tag{5}$$

Esta propiedad se puede aplicar siempre y cuando el SDL sea estable, es decir, si todos sus polos (valores propios de A) están en el semiplano complejo izquierdo (tienen parte real negativa). En otras palabras, la matriz A debe ser estable (o *Hurwitz*), es decir, el espectro de A, denotado por $\Lambda(A)$, debe satisfacer $\Lambda(A) \subset \mathbb{C}_-$. De la teoría de estabilidad de Lyapunov (consultar, por ejemplo, [76, Capítulo 13]) se deriva que, para una matriz A estable, las ecuaciones duales de Lyapunov en (5) tienen soluciones semidefinidas positivas y únicas W_c y W_o.

Además, se establece que si la matriz W_c es definida positiva, el sistema es controlable, y si la matriz W_o es definida positiva, el sistema es observable.

Los conceptos de controlabilidad y observabilidad son equivalentes a minimizar el sistema, por lo cual, para sistemas mínimos todos los valores propios del producto W_cW_o son números reales estrictamente positivos. Las raíces cuadradas de estos valores propios, denotadas en orden decreciente como

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0,$$

se conocen como valores singulares de Hankel[1] (HSVs, del inglés *Hankel singular values*) del SDL y son invariantes a transformaciones de coordenadas del espacio de estados.

Una vez modificado el sistema de coordenadas, transformando los gramianos en matrices diagonales con las entradas positivas y en orden decreciente, el modelo de orden reducido se obtiene truncando los estados correspondientes a los $n - r$ HSVs menores.

El método directo para hallar la solución de la ecuación de Lyapunov $AW_c + W_cA^T + BB^T = 0$ implica resolver un sistema lineal $K_cx = q$, donde $K_c = I_n \otimes A^T + A^T \otimes I_n \in \mathbb{R}^{n^2\times n^2}$, $\otimes$ representa el producto de Kronecker, $q = \text{vec}(BB^T)$, $x = \text{vec}(W_c)$, y $\text{vec}(M)$ es el vector que se obtiene al disponer las columnas de la matriz M de forma consecutiva. De manera análoga, la solución de $A^TW_o + W_oA + C^TC = 0$ puede obtenerse del sistema lineal $K_oy = p$, con $K_o = I_n \otimes A + A \otimes I_n$, $p = \text{vec}(C^TC)$ e $y = \text{vec}(W_o)$. Esta estrategia es prohibitiva por su elevado costo, ya que la resolución de estos sistemas de ecuaciones lineales precisa $\mathcal{O}(n^6)$ operaciones aritméticas en coma flotante (flops) y almacenamiento para $\mathcal{O}(n^4)$ números. Por esta razón, se han desarrollado diversas alternativas para abordar el problema. A continuación se describen las técnicas computacionales más difundidas para la resolución de ecuaciones de Lyapunov.

[1] Notar la similitud de la técnica con el cálculo de los valores singulares de una matriz.

El método de Bartels-Stewart [21] se basa en transformar la matriz coeficiente de la ecuación a una forma triangular superior por bloques (en concreto, la forma real de Schur) mediante transformaciones de semejanza. Luego es necesario ir resolviendo ecuaciones de Lyapunov con matrices de dimensión 1 ó 2 (dependiendo del tamaño de bloque); en el primer caso se resuelve directamente el sistema; en cambio, si las matrices son 2×2, se puede utilizar el producto de Kronecker para hallar la solución. Este proceso se repite para cada una de las columnas de la solución. Otro método utilizado para resolver ecuaciones matriciales lineales es el de Hessenberg-Schur [52]; sin embargo, en el caso de las ecuaciones de Lyapunov este método es equivalente al de Bartels-Stewart. Para la resolución de las ecuaciones de Lyapunov destaca también el método de Hammarling [63], una variante ingeniosa del método de Bartels-Stewart que obtiene de forma directa el factor de Cholesky de la solución. En estos métodos existe una última etapa en la que se recupera la solución de la ecuación original, antes de la reducción de la matriz coeficientes. Estos métodos tienen como característica su elevada exigencia computacional, $\mathcal{O}(n^3)$ flops, y un alto requerimiento de almacenamiento, $\mathcal{O}(n^2)$ números. Además, las operaciones involucradas en estos métodos (en particular, la reducción a la forma real de Schur) no son propicias para la aplicación de técnicas de computación de alto desempeño (HPC) y menos sobre arquitecturas masivamente paralelas.

Otra técnica muy extendida para la resolución de ecuaciones de Lyapunov con matriz de coeficientes estable es el método de la función signo [92]. Este estudio se centra en la optimización de este método para el cómputo de los factores de rango bajo de los gramianos. Si bien este método presenta un costo aritmético ($\mathcal{O}(n^3)$ flops) y espacial ($\mathcal{O}(n^2)$ números) similar al de los métodos comentados anteriormente, sus características permiten la aplicación de técnicas de programación paralela y de computación de altas prestaciones. Por lo tanto, esta alternativa es más apropiada para su implementación sobre las nuevas arquitecturas de hardware que disponen decenas, o incluso centenas, de unidades computacionales.

El problema de la reducción de modelos para SDL discretos puede formularse de forma análoga y la mayoría de los métodos discutidos en este trabajo son aplicables a este tipo de sistemas. Para una descripción detallada sobre técnicas para sistemas discretos, consultar el trabajo de Obinata y Anderson [86]. Asimismo existen técnicas específicas para afrontar los casos en que las matrices que definen el SDL son dispersas; estas técnicas obtienen el modelo reducido con un costo proporcional a la cantidad de elementos no nulos de la matriz A. Entre estos métodos destacan los basados en Smith [61] y ADI (del inglés *Alternating Direction Implicit*) [88, 105]. Este trabajo únicamente aborda la reducción de modelos de SDL continuos, con matriz de coeficiente densa.

2.1.2. Realizaciones balanceadas

Una realización de un SDL está definida por el conjunto (A, B, C, D) de matrices, asociado a la Ec. (1). En general, un SDL tiene infinitas realizaciones y su función de transferencia es invariante ante transformaciones en el espacio de estados. Dada la transformación

$$\mathcal{T} : \left\{ \begin{array}{ccc} x & \rightarrow & Tx, \\ (A, B, C, D) & \rightarrow & (TAT^{-1}, TB, CT^{-1}, D), \end{array} \right. \tag{6}$$

con un simple cálculo se puede demostrar que

$$(CT^{-1})(sI_n - TAT^{-1})^{-1}(TB) + D = C(sI_n - A)^{-1}B + D = G(s)$$

En base a lo anterior, se puede deducir que la representación asociada a un SDL no es única. Es más, cualquier adición en los estados que no modifique la relación entre las entradas y salidas del SDL (es decir, que para una entrada u, la misma salida y es alcanzada por ambos sistemas), conduce a una realización válida del mismo SDL. De esta forma, el orden de un sistema puede ser arbitrariamente extendido sin modificar el mapeo entrada-salida. Por otro lado, para cada sistema existe una cantidad mínima de estados, $\hat{n}$, necesarios para describir por completo el comportamiento de la relación entrada-salida establecida en el sistema. Este número se denomina *grado de McMillan* del sistema y la realización $(\hat{A}, \hat{B}, \hat{C}, \hat{D})$ de orden $\hat{n}$ es una *realización mínima* del sistema. Es importante notar que el grado de McMillan es único y la transformación del espacio de estados del sistema en (6) conduce a otra realización mínima del mismo.

Encontrar una realización mínima para un sistema dado, donde los estados redundantes (no mínimos) son eliminados del sistema, puede considerarse como un primer paso en la resolución del problema de reducción de modelos.

Aunque las realizaciones no son únicas, un SDL estable tiene un conjunto de estados invariantes con respecto a la transformación del espacio de estados que provee una buena vía para buscar el modelo de orden reducido. Además, como se comentó anteriormente, si la matriz A es estable, las ecuaciones de Lyapunov en (5) tienen sendas soluciones únicas y semidefinidas positivas, W_c y W_o. Dado que controlabilidad y observabilidad son equivalentes a minimizar el sistema, y que para un sistema mínimo todos los valores propios del producto W_cW_o son estrictamente positivos, se pueden calcular los HSVs del SDL que son invariantes en el sistema; esto se puede corroborar fácilmente. Considérese

$$(\hat{A}, \hat{B}, \hat{C}, D) = (TAT^{-1}, TB, CT^{-1}, D);$$

la realización transformada con la ecuación de controlablidad asociada

$$0 = \hat{A}\hat{W}_c + \hat{W}_c\hat{A}^T + \hat{B}\hat{B}^T = TAT^{-1}\hat{W}_c + \hat{W}_cT^{-T}A^TT^T + TBB^TT^T,$$

es equivalente a

$$0 = A(T^{-1}\hat{W}_cT^{-T}) + (T^{-1}\hat{W}_cT^{-T})A^T + BB^T.$$

En consecuencia, la unicidad de la solución de la ecuación de Lyapunov (ver, por ejemplo [76]) implica que $\hat{W}_c = TW_cT^T$, $\hat{W}_o = T^{-T}W_oT^{-1}$ y, por lo tanto,

$$\hat{W}_c\hat{W}_o = TW_cW_oT^{-1},$$

mostrando que $\Lambda(\hat{W}_c\hat{W}_o) = \Lambda(W_cW_o) = \{\sigma_1^2, \ldots, \sigma_n^2\}$. La extensión del espacio de estados mediante estados no minimales implica agregar HSVs de magnitud cero, mientras que los restantes HSVs distintos de cero se mantienen invariantes.

Una clase de realizaciones importante (que induce el nombre de la técnica) son las *realizaciones balanceadas*. Una realización (A, B, C, D) se denomina

balanceada si y solo si

$$W_c = W_o = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix};$$

esto es, los gramianos de controlabilidad y observabilidad son diagonales e iguales entre sí, con los HSVs ordenados de forma decreciente como entradas de sus respectivas diagonales.

Para la realización mínima ($n = \hat{n}$) siempre existe una transformación balanceada en el espacio de estados de la forma (6), con una matriz no singular $T_b \in \mathbb{R}^{n\times n}$. Para un sistema no mínimo ($n > \hat{n}$), los gramianos pueden ser transformados en matrices diagonales, de forma que

$$\hat{W}_c\hat{W}_o = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_{\hat{n}}^2, 0, \ldots, 0).$$

Usando una realización balanceada obtenida mediante la matriz de transformación T_b, los HSVs permiten una interpretación de la energía de los estados; ver por ejemplo el trabajo de Van Dooren [101] para un estudio detallado. Específicamente, la energía mínima necesaria para alcanzar el estado x^0 es

$$\inf_{\substack{u\in\mathcal{L}_2(-\infty,0]\\ x(0)=x^0}} \int_{-\infty}^{0} u(t)^T u(t)\,dt = (x^0)^T W_c^{-1} x^0 = (\hat{x}^0)^T \hat{W}_c^{-1}\hat{x}^0 = \sum_{k=1}^{n} \frac{1}{\sigma_k}\hat{x}_k^2,$$

donde $\hat{x}^0 := \begin{bmatrix} \hat{x}_1 \\ \vdots \\ \hat{x}_n \end{bmatrix} = T_b x^0$, ya que los HSVs menores se corresponden con estados que son difíciles de alcanzar. La energía de salida resultante de un estado inicial x^0 y $u(t) \equiv 0$ para $t > 0$ viene dada por

$$\|y\|_2^2 = \int_0^\infty y(t)^T y(t)\,dt = x_0^T W_o x_0 = (\hat{x}^0)^T \hat{W}_o \hat{x}^0 = \sum_{k=1}^{n} \sigma_k \hat{x}_j^2;$$

aquí los HSVs mayores se corresponden con los estados que contienen la mayor parte de la energía del sistema. La transferencia de energía desde las entradas a las salidas puede ser computada mediante la siguiente expresión

$$E = \sup_{\substack{u\in\mathcal{L}_2(-\infty,0]\\ x(0)=x^0}} \frac{\|y\|_2^2}{\int_{-\infty}^{0} u(t)^T u(t)\,dt} = \frac{(x^0)^T W_o x^0}{(x^0)^T W_c^{-1} x^0} = \frac{(\bar{x}^0)^T W_c^{\frac{1}{2}} W_o W_c^{\frac{1}{2}} \bar{x}^0}{(\bar{x}^0)^T \bar{x}^0},$$

donde $\bar{x}^0 = W_c^{-\frac{1}{2}} x^0$. Entonces, los HSVs $\left(\Lambda\left(W_c W_o\right)\right)^{\frac{1}{2}} = \left(\Lambda\left(W_c^{\frac{1}{2}} W_o W_c^{\frac{1}{2}}\right)\right)^{\frac{1}{2}}$, siendo éste un buen estimador de cuán implicado está un estado en la transferencia de energía desde las entradas a las salidas.

En resumen, es posible obtener un modelo de orden reducido que aproxime al original mediante la eliminación de los estados menos controlables o menos observables, manteniendo los estados que contienen la mayor parte de la energía, ya que éstos son los que están más implicados en la transferencia de energía desde las entradas a las salidas. Esto es lo que se consigue manteniendo los estados correspondientes a los HSVs de mayor valor, y es exactamente la idea que persigue al método de BT que se describe a continuación.

2.1.3. Truncamiento balanceado

El objetivo en el método de truncamiento balanceado es computar una realización balanceada

$$(TAT^{-1}, TB, CT^{-1}, D) = \left(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \end{bmatrix}, D \right), \quad (7)$$

donde $A_{11} \in \mathbb{R}^{r\times r}$, $B_1 \in \mathbb{R}^{r\times m}$, $C_1 \in \mathbb{R}^{p\times r}$, con $r \leq \hat{n}$, y entonces definir el modelo de orden reducido mediante la realización truncada

$$(\hat{A}, \hat{B}, \hat{C}, \hat{D}) = (A_{11}, B_1, C_1, D). \quad (8)$$

Esta técnica se trata en forma exhaustiva, entre otros, en los trabajos de Moore [83], y Mullins y Roberts [84].

Tomando los resultados de [51, 83, 99], se pueden resumir las siguientes propiedades de aplicar la técnica de truncamiento balanceado.

Dada (A, B, C, D), una realización de un SDL estable con función de transferencia $G(s)$, y dado un modelo $(\hat{A}, \hat{B}, \hat{C}, \hat{D})$ de orden reducido r con función de transferencia asociada $\hat{G}(s)$ que lo aproxima, computado como en (7)–(8):

a) El sistema de orden reducido $\hat{G}$ es balanceado, mínimo y estable, si sus gramianos son

$$W_o = W_c = \tilde{\Sigma} = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}.$$

b) El error absoluto de la transferencia sobre el modelo reducido está acotado por

$$\|G - G_r\|_\infty \leq 2 \sum_{k=r+1}^{\hat{n}} \sigma_k. \quad (9)$$

c) Si $r = \hat{n}$, entonces (8) es una realización mínima de G y $G = G_r$.

Particularmente importante es la cota del error en (9), ya que permite una elección adaptativa de la dimensión del modelo de orden reducido (r) que satisfaga un umbral de tolerancia preestablecido.

Es posible demostrar que, para un sistema controlable y observable (mínimo), por ejemplo un sistema con gramianos no singulares, la matriz

$$T = \Sigma^{\frac{1}{2}} U^T R^{-T} \quad (10)$$

donde $W_c = R^T R$ y $RW_oR^T = U\Sigma^2U^T$ (descomposición en valores singulares), provee una transformación del espacio de estados balanceada. Además, a partir de un sistema no mínimo (8) puede computarse un SDL de orden reducido sin la necesidad de calcular la matriz T completamente. Dado $W_o = S^T S$ y $W_c = R^T R$, entonces

$$S^{-T}(W_cW_o)S^T = (SR^T)(SR^T)^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2U^T,$$

por lo cual U y Σ pueden ser computados mediante una SVD de la matriz SR^T,

$$SR^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}, \qquad \Sigma_1 = \mathrm{diag}(\sigma_1, \ldots, \sigma_r). \tag{11}$$

De este modo, los proyectores T_l y T_r pueden ser obtenidos según

$$T_l = \Sigma_1^{-1/2} V_1^T R, \quad T_r = S^T U_1 \Sigma_1^{-1/2}, \tag{12}$$

para finalmente obtener el nuevo modelo como

$$\hat{A} = T_l A T_r, \quad \hat{B} = T_l B, \quad \hat{C} = C T_r \quad y \quad \hat{D} = D. \tag{13}$$

Este método es equivalente a computar en primera instancia una realización mínima de (1), luego balancear el sistema (7) con T como se presenta en (10), y finalmente truncar la realización balanceada como en (8). En particular, las realizaciones obtenidas en (8) y (13) son la misma, puesto que T_l contiene las primeras r filas de T y T_r las primeras r columnas de T^{-1}; ambas partes de T son necesarias para computar A_{11}, B_1 y C_1 en (7). Cabe destacar que el producto $T_r T_l$ es un proyector en el subespacio r-dimensional del espacio de estados y el modelo de orden reducido obtenido mediante (13) puede ser visto como una proyección de la dinámica del sistema en el subespacio de estados.

El algoritmo descrito en las Ecs. (12) y (13) es comúnmente referido como el método SRBT (del inglés *square root balanced truncation*). En los textos de referencia [80, 99] y varios otros que tratan el truncamiento balanceado, se asume que las matrices S y R son los factores de Cholesky de los gramianos del sistema. En el artículo de Benner et al. [35] se muestra que los factores de Cholesky de rango bajo de los gramianos del sistema pueden ser igualmente utilizados. Esto permite implementaciones más eficientes del algoritmo de BT, especialmente cuando $\hat{n} \ll n$. Además, el rango numérico bajo de los gramianos implica un rápido decaimiento de los valores propios asociados y, por consiguiente, un rápido decaimiento de los HSVs. Esta propiedad se estudia con mayor profundidad más adelante en la sección.

Si el sistema original es pronunciadamente desbalanceado (en este caso, la matriz de transformación del espacio de estados T está mal condicionada), es interesante el algoritmo sin raíz cuadrada (BFSR, del inglés *balancing free square-root*) para BT propuesto por Varga [102], que permite, en presencia de errores de redondeo, obtener modelos de orden reducido de mejor precisión. Este método combina la implementación del método SR formulado en [80, 99] y la reducción de modelos sin balanceo (*balancing-free model reduction*) propuesta por Safanov y Chiang [96]. El algoritmo BFSR solamente difiere de la implementación de SR en el procedimiento de obtención de los proyectores T_l y T_r a partir de la factorización SVD (Ec. (11)) de SR^T, y en que en este método el modelo de orden reducido no es balanceado. La idea principal es que, para computar el modelo de orden reducido, es suficiente usar bases ortogonales para Im(T_l) e Im(T_r), donde Im(M) denota al conjunto imagen de la transformación M, que pueden obtenerse calculando las siguientes factorizaciones QR:

$$S^T U_1 = [P_1 \; P_2] \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}, \qquad R^T V_1 = [Q_1 \; Q_2] \begin{bmatrix} \bar{R} \\ 0 \end{bmatrix}, \tag{14}$$

donde las columnas de P_1, $Q_1 \in \mathbb{R}^{n\times r}$ son ortonormales, mientras que $\hat{R}$, $\bar{R} \in \mathbb{R}^{r\times r}$ son matrices triangulares superiores. El sistema de orden reducido queda definido según (13) con

$$T_l = (Q_1^T P_1)^{-1} Q_1^T, \qquad T_r = P_1, \tag{15}$$

donde el factor $(Q_1^T P_1)^{-1}$ es necesario para preservar la condición de proyector de la matriz $T_r T_l$.

El error absoluto de la realización de orden r computada mediante la implementación BFSR de truncamiento balanceado satisface la misma cota superior del error en la Eq. (9).

2.2. La función signo

En 1971, Roberts [92] propuso la función signo matricial y mostró como utilizarla para resolver las ecuaciones de Sylvester y Lyapunov. El método ha sido aplicado con éxito en posteriores trabajos [22, 43, 69].

Considérese la matriz $Z \in \mathbb{R}^{n\times n}$ sin valores propios en el eje imaginario (esto es, $\Lambda(Z) \cap \jmath\mathbb{R} = \emptyset$), donde su descomposición de Jordan [53] está definida por

$$Z = T^{-1} \begin{pmatrix} J_- & 0 \\ 0 & J_+ \end{pmatrix} T, \tag{16}$$

y los valores propios de $J_- \in \mathbb{R}^{j\times j}$ / $J_+ \in \mathbb{R}^{(n-j)\times(n-j)}$ tienen parte real negativa/positiva [53]. La función signo matricial de Z queda definida por

$$\mathrm{fsign}(Z) = T^{-1} \begin{pmatrix} -I_j & 0 \\ 0 & I_{n-j} \end{pmatrix} T. \tag{17}$$

Existen esquemas iterativos simples para computar la función signo. El más difundido es la iteración de Newton, definida de la siguiente forma:

$$Z_0 = Z, \tag{18}$$

$$Z_{k+1} = \frac{1}{2}(Z_k + Z_k^{-1}), \quad k = 0, 1, 2, \ldots. \tag{19}$$

Esta estrategia es particularmente atractiva por su simplicidad, eficiencia y desempeño computacional, así como por presentar una convergencia asintóticamente cuadrática [34, 38].

Si bien se pueden encontrar en la literatura diversos esquemas iterativos para computar la función signo con interesantes propiedades, como por ejemplo una acelerada convergencia y la posibilidad de emplear técnicas de programación paralela en su implementación (consultar el trabajo de Kenney y Laub [74] para un estudio cabal del tema), la iteración básica de Newton aparece como uno de los métodos más robustos y que ofrece mejores tiempos de cómputo, tanto en implementaciones secuenciales como paralelas. En particular, la iteración de Newton en (19) solo requiere calcular sumas e inversiones de matrices, siendo ambas operaciones ampliamente estudiadas por la comunidad científica que trabaja en computación de altas prestaciones, y contando con gran caudal de desarrollo.

Otra forma de considerar el cómputo basado en la función signo es como un *método de proyección espectral*, ya que

$$P_- = \frac{1}{2}(I_n - \mathrm{fsign}\,(Z)), \tag{20}$$

es un proyector espectral en el subespacio estable Z-invariante. Además, $P_+ = (I_n + \mathrm{fsign}\,(Z))/2$ es un proyector espectral en el subespacio Z-invariante correspondiente a los valores propios en el semiplano complejo derecho (parte real positiva). Sin embargo, cabe resaltar que P_- y P_+ no son proyectores ortogonales sino proyectores oblicuos al subespacio Z-invariante complementario.

A continuación, se resumen algunas propiedades importantes de la función signo. Dada una matriz $Z \in \mathbb{R}^{n\times n}$, con $\Lambda\,(Z) \cap \jmath\mathbb{R} = \emptyset$, entonces:

a) $(\mathrm{fsign}\,(Z))^2 = I_n$;

b) $\mathrm{fsign}\left(T^{-1}ZT\right) = T^{-1}\mathrm{fsign}\,(Z)\,T$ para todas las matrices $T \in \mathbb{R}^{n\times n}$ no singulares;

c) $\mathrm{fsign}\left(Z^T\right) = \mathrm{fsign}\,(Z)^T$;

d) si p_+ y p_- denotan el número de valores propios de Z con parte real positiva y negativa respectivamente, entonces:

$$p_+ = \frac{1}{2}(n + \mathrm{tr}\,(\mathrm{fsign}\,(Z))), \quad p_- = \frac{1}{2}(n - \mathrm{tr}\,(\mathrm{fsign}\,(Z))),$$

donde $\mathrm{tr}\,(M)$ define la traza de una matriz M;

e) si Z es estable, entonces

$$\mathrm{fsign}\,(Z) = -I_n, \quad \mathrm{fsign}\,(-Z) = I_n.$$

La función signo es criticada por varias razones, siendo la más prominente la necesidad de computar explícitamente la matriz inversa en cada paso, lo que conlleva un alto costo computacional. Además, este método no está definido en contextos con valores propios puramente imaginarios y puede tener problemas de condicionamiento cuando las matrices poseen valores propios cercanos al eje imaginario. Sin embargo, esto se da únicamente cuando existen valores propios con parte imaginaria de magnitud menor a la raíz cuadrada de la precisión de la máquina, es decir, si se trabaja con números en doble precisión la magnitud relativa de los valores propios debería ser menor a 10^{-8}. Habitualmente, en aplicaciones de control como las consideradas en este trabajo, los polos se encuentran apartados del eje imaginario, lo que permite emplear el método de la función signo. Por otro lado, el método de la función signo generalmente está mejor condicionado que las opciones basadas en el complemento de Schur, dado que la primera separa los espacios estables de los anti-estables mientras que el método de Schur esencialmente implica separar n subespacios. Para un análisis exhaustivo del cómputo de subespacios invariantes basados en la función signo, consultar los trabajos [19, 39]. El condicionamiento de la forma de Schur y de la forma triangular a bloques (como la computada por la función signo) es discutido en el trabajo de Konstantinov et al. [75]. Es más, en las aplicaciones consideradas en el área de control, en general $\mathrm{cond}\,(\mathrm{fsign}\,(Z)) = 1$ ya que Z es estable o anti-estable, y por lo tanto, el cómputo de $\mathrm{fsign}\,(Z)$ es un problema bien condicionado.

2.2.1. Aceleración de la convergencia de la función signo

Existen diversas alternativas discutidas en la literatura que permiten acelerar la convergencia de la iteración de Newton para el cálculo de la función signo. Una discusión profunda se puede encontrar tanto en [73] como en [18], donde se ofrece un relevamiento así como una comparativa de los distintos esquemas propuestos. Una estrategia extendida y efectiva es el uso de un factor de escalado, γ, que se aplica en cada iteración del método de la siguiente forma:

$$Z_0 = Z, \tag{21}$$

$$Z_{k+1} = \frac{1}{2\gamma_k}(Z_k + \gamma_k^2 Z_k^{-1}), \quad k = 0, 1, 2, \ldots, \tag{22}$$

La elección de los diferentes γ_k se discute brevemente a continuación.

***Escalado por determinante* [38]:** el objetivo es minimizar la media de la distancia geométrica de los valores propios de Z_k a 1. Para ello se utiliza el factor de escalado

$$\gamma_k = \det(Z_k)^{\frac{1}{n}}, \tag{23}$$

donde $\det(M)$ representa el determinante de la matriz M. Cabe notar que el determinante, $\det(Z_k)$, se puede obtener como un resultado intermedio de los cómputos necesarios para calcular la inversa Z_k^{-1}.

***Escalado por norma* [66]:** esta elección tiene ciertas propiedades de minimización en el contexto del cómputo de descomposiciones polares. Es también beneficiosa sobre los errores de redondeo, ya que iguala las normas de los dos sumandos en el cálculo de la norma finita $(\frac{1}{\gamma_k} Z_k) + (\gamma_k Z_k^{-1})$. En este caso, el escalado está definido por:

$$\gamma_k = \sqrt{\frac{\|Z_k\|_2}{\|Z_k^{-1}\|_2}}. \tag{24}$$

siendo $\|M\|_2$ la 2-norma (o espectral) matricial de la matriz M.

Escalado por norma aproximada: dado que la norma espectral es computacionalmente costosa de calcular, en algunos trabajos [66, 73] se sugiere utilizar cotas conocidas de la misma (ver Golub y Van Loan [53]) o aproximar dicha norma por otra más económica de calcular. Por ejemplo, puesto que $\|Z_k\|_2 \leq \sqrt{\|Z_k\|_1 \|Z_k\|_\infty}$, con $\|M\|_1$ y $\|M\|_\infty$ las normas matriciales 1 e infinito respectivamente, se puede utilizar la siguiente expresión:

$$\gamma_k = \sqrt{\frac{\|Z_k\|_1 \|Z_k\|_\infty}{\|Z_k^{-1}\|_1 \|Z_k^{-1}\|_\infty}}. \tag{25}$$

Experimentos numéricos y consideraciones analíticas parciales (ver [36]) sugieren que un escalado basado en la norma aproximada es habitualmente una buena alternativa. Además, el cómputo de la 1-norma o la norma infinito puede ser paralelizado fácilmente, por lo cual se presenta como una buena opción en matemática computacional. Sin embargo, la elección de la mejor opción

de escalado está fuertemente condicionada por las características de cada problema y, en la práctica, ha de ser realizada en forma empírica.

2.3. Resolución de las ecuaciones de Sylvester y Lyapunov mediante la función signo

En este apartado, en primer lugar se estudia el uso de la función signo para la resolución de la ecuación de Sylvester y luego se extiende la técnica para la resolución de la ecuación de Lyapunov.

La ecuación de Sylvester se define como

$$AX + XB + W = 0, \tag{26}$$

donde $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{m\times m}, W, X \in \mathbb{R}^{n\times m}$ y $\Lambda(A) \cap \Lambda(-B) = \emptyset$. Esta propiedad implica que la Ec. (26) tiene una solución única [76]. El cálculo

$$\begin{bmatrix} I_n & 0 \\ X & I_m \end{bmatrix} \begin{bmatrix} A & 0 \\ W & -B \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -X & I_m \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & -B \end{bmatrix} \tag{27}$$

revela que las columnas de $\begin{bmatrix} I_n \\ -X \end{bmatrix}$ generan el subespacio invariante de $Z = \begin{bmatrix} A & 0 \\ W & -B \end{bmatrix}$ correspondiente a $\Lambda(A)$. Este subespacio, tras aplicar un cambio de base apropiado y el cálculo de la matriz X, puede ser computado mediante la utilización de un proyector espectral sobre su subespacio Z-invariante. La función signo es una herramienta apropiada para resolver este problema siempre y cuando las matrices A y B sean estables, puesto que P_-, definido en la Ec. (20), es el proyector espectral requerido. Un análisis de la iteración (19) aplicada a Z permite observar que no es necesario generar P_-; en este caso la solución puede ser obtenida directamente de la matriz $\mathrm{fsign}(Z)$. En particular, basándose en la Ec. (27) y las propiedades de la función signo resumidas anteriormente, se puede demostrar que:

$$\mathrm{fsign}(Z) = \mathrm{fsign}\left(\begin{bmatrix} A & 0 \\ W & -B \end{bmatrix}\right) = \begin{bmatrix} -I_n & 0 \\ 2X & I_m \end{bmatrix},$$

y por lo tanto la solución de la Ec. (26) está dada por el bloque inferior izquierdo del límite dividido entre 2. Es más, la estructura triangular a bloques de Z permite desacoplar la iteración (19) como se presenta en el Algoritmo GECSNW. Cuando este algoritmo converge, $X_* = \frac{1}{2}\lim_{k\to\infty} W_k$, siendo X_* la solución de (26).

Algoritmo GECSNW**:**
Iteración básica de Newton para la ecuación de Sylvester
$A_0 \leftarrow A$, $B_0 \leftarrow B$, $W_0 \leftarrow W$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
$A_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(A_k + \gamma_k^2 A_k^{-1}\right)$
$B_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(B_k + \gamma_k^2 B_k^{-1}\right)$
$W_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(W_k + \gamma_k^2 A_k^{-1} W_k B_k^{-1}\right)$

Para considerar la formulación estándar de la ecuación de Lyapunov se deben particularizar los resultados anteriores para la ecuación

$$AX + XA^T + Q \;=\; 0, \tag{28}$$

donde $A, Q, X \in \mathbb{R}^{n\times n}$, $Q = Q^T$ y $X = X^T$ es la incógnita. Asumiendo que todos los valores propios de A tienen parte real negativa, se puede aplicar el método de la iteración de Newton (18)–(19) a la matriz Z,

$$Z \;=\; \begin{bmatrix} A^T & 0 \\ Q & -A \end{bmatrix}, \tag{29}$$

de forma que,

$$Z_\infty \;=\; \lim_{k\to\infty} Z_k \;=\; \begin{bmatrix} -I & 0 \\ 2X & I \end{bmatrix}. \tag{30}$$

En la práctica, cada iteración con matrices Z_k de dimensión $2n \times 2n$ puede ser reemplazada con dos iteraciones con matrices de dimensión $n \times n$, según se muestra en el Algoritmo `GECLNW`.

Algoritmo `GECLNW`**:**
Iteración básica de Newton para la ecuación de Lyapunov
$A_0 \leftarrow A$, $Q_0 \leftarrow Q$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
$A_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(A_k + \gamma_k^2 A_k^{-1}\right)$
$Q_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(Q_k + \gamma_k^2 A_k^{-1} Q_k A_k^{-T}\right)$

Una posible condición de parada para el Algoritmo `GECLNW` es:

$$\|A_k + I_n\|_F < \tau_{\text{iter}} \cdot \|A_k\|_F, \tag{31}$$

donde, dada la convergencia asintóticamente cuadrática de la iteración, comúnmente se especifica $\tau_{\text{iter}} = 10 \cdot n \cdot \sqrt{\varepsilon}$, siendo ε la precisión de la máquina. Ejecutando dos iteraciones más una vez que el criterio de condición de parada se satisface, en la práctica se asegura la obtención de una solución suficientemente precisa y se evitan problemas de convergencia lenta derivados de un mal condicionamiento del problema (nótese que esta misma discusión puede adaptarse fácilmente al Algoritmo `GECSNW`).

2.3.1. Disminución de requerimientos de memoria

En modelos de gran escala que surgen en problemas de teoría de control, como por ejemplo la reducción de modelos y el control óptimo, el número de estados en las ecuaciones en (1) es mucho mayor que el número de entradas ($n \gg m$) y salidas ($n \gg p$). Esto supone que las ecuaciones de Lyapunov frecuentemente exhiban una matriz de términos independientes Q en forma factorizada y que, además, la magnitud de los valores propios de la matriz solución (X) decaiga rápidamente, es decir, la matriz X presenta un rango numérico bajo (ver [16, 55, 89]). La Figura 1 (ejemplo extraído del trabajo de Benner et al.

[32]) muestra el comportamiento antes descrito para el gramiano de controlabilidad de un SDL estable aleatorio con $n = 500$, $m = 10$, y *margen de estabilidad* (distancia mínima de $\Lambda(A)$ a $\jmath\mathbb{R}$) $\approx 0{,}055$. En este contexto, si $n_\varepsilon = \mathrm{rank}\,(X)$ es el rango numérico de X, existe una matriz $S_\varepsilon \in \mathbb{R}^{n\times n_\varepsilon}$ tal que $X \approx S_\varepsilon S_\varepsilon^T$ al nivel de la precisión de la máquina.

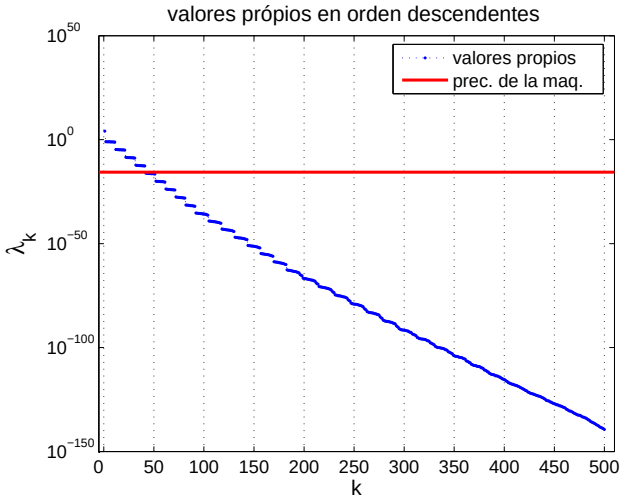

Figura 1: Ritmo de decaimiento de los valores propios para el gramiano de controlabiliad de un SDL aleatorio, con $n = 500$, $m = 10$, y margen de estabilidad $\approx 0{,}055$.

En la práctica, algunas aplicaciones no requieren explícitamente la solución de la ecuación, sino que es suficiente con conocer el factor de rango reducido S_ε. En estos casos el Algoritmo `GECLNC` para la solución de la ecuación $AX + XA^T + BB^T = 0$, presentado a continuación, es especialmente atractivo.

Algoritmo `GECLNC`**:**
Iteración factorizada de Newton para la ecuación de Lyapunov

$A_0 \leftarrow A$, $B_0 \leftarrow B$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
$A_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(A_k + \gamma_k^2 A_k^{-1}\right)$
$B_{k+1} \leftarrow \frac{1}{\sqrt{2}\gamma_k}[\, B_k,\ \gamma_k A_k^{-1} B_k \,]$

Esta observación sirve como idea básica en la mayoría de los algoritmos propuestos para resolver ecuaciones de Lyapunov de grandes dimensiones (ver [15, 89]), ya que almacenar S_ε (en el Algoritmo `GECLNC` se utiliza el espacio de memoria empleado por B_k para almacenarlo) es más económico que almacenar X; en concreto, en lugar de almacenar n^2 números sólo se almacenan $(n \times n_\varepsilon)$. En el ejemplo utilizado anteriormente para ilustrar el ritmo de decaimiento de los valores propios, se puede observar un ahorro de un 90 % (500 columnas frente a 50) para el almacenamiento del gramiano de controlabilidad, aunque

es habitual encontrar situaciones en las que se reduce la necesidad de almacenamiento hasta en un 99 % [27].

Por otro lado, en el nuevo esquema iterativo el espacio requerido para almacenar B_k se dobla en cada paso. Existen diversas maneras para limitar dicho espacio. La primera técnica, propuesta en [78], trabaja con una matriz de $n \times n$, asignando en B_0 el factor de Cholesky BB^T, computando la factorización QR de $\begin{bmatrix} B_k \\ \gamma_k A_k^{-1} B_k \end{bmatrix}$ en cada iteración, y usando dicho factor R en la siguiente iteración. Una versión un poco más económica de esta técnica se presenta en el trabajo de Benner et al. [34], donde la factorización se aplica siempre y cuando $k \leq \log_2 \frac{n}{p}$, y sólo se computa una factorización QR en los pasos posteriores. En ambos casos, se puede demostrar que B_k converge a un factor de Cholesky de la solución X de la Ec. (28). Sin embargo, esto puede ser mejorado computando la factorización QR con pivotamiento por columnas de B_{k+1}^T en cada iteración,

$$B_{k+1}^T \Pi_{k+1} = U_{k+1} \begin{bmatrix} R_{k+1} \\ 0 \end{bmatrix}, \tag{32}$$

donde Π_{k+1} es una matriz de permutación, U_{k+1} es ortogonal, y R_{k+1} es triangular superior. Dado que

$$B_{k+1} B_{k+1}^T = \Pi_{k+1} R_{k+1}^T U_{k+1}^T U_{k+1} R_{k+1} \Pi_{k+1}^T = \tag{33}$$

$$(\Pi_{k+1} R_{k+1}^T)(R_{k+1} \Pi_{k+1}) = \tilde{R}_{k+1}^T \tilde{R}_{k+1}, \tag{34}$$

es posible reemplazar B_{k+1} por $\tilde{R}_{k+1}^T$. La reducción del costo de almacenamiento y cómputo de cada iteración utilizando esta factorización es fuertemente dependiente de las características del problema y, por lo tanto, es necesario evaluarla experimentalmente en cada caso concreto.

2.3.2. Ecuaciones de Lyapunov acopladas

Como se mencionó en el Apartado 2.1.1, el tratamiento del problema de reducción de modelos mediante el método BT implica la resolución de dos ecuaciones estándares de Lyapunov. Una primera aproximación a este problema es ejecutar dos instancias del Algoritmo `GECLNC`, una para la resolución de cada ecuación. Sin embargo, en el artículo de Lang y Lezius [77] se demuestra que no es necesario realizar dos iteraciones de la función signo para resolver las ecuaciones acopladas de Lyapunov. Dicha resolución puede ser unificada, debido a que esencialmente son la misma iteración para las matrices A_k, donde la única diferencia reside en que las matrices de las iteraciones son traspuestas entre ellas; por lo tanto solamente es necesario efectuar una iteración (o mejor dicho, una inversión por paso).

Esta idea fue generalizada y combinada con la iteración de rango bajo en los trabajos [29, 35]. El algoritmo de la función signo `CECLNW` es el resultante de aplicar las técnicas de optimización mencionadas hasta este momento, y obtiene los factores de rango bajo de los gramianos de controlabilidad y observabilidad del SDL.

Algoritmo `CECLNW`**:**
Iteración factorizada de Newton para las ecuaciones de Lyapunov acopladas
$A_0 \leftarrow A$, $\tilde{S}_0 \leftarrow B^T$, $\tilde{R}_0 \leftarrow C$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
$A_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(A_k + \gamma_k^2 A_k^{-1}\right)$

Computa la descomposición RRQR (*rank-revealing* QR)
$\frac{1}{\sqrt{2\gamma_k}}\left[\tilde{S}_k,\ \ \gamma_k \tilde{S}_k (A_k^{-1})^T\right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$
$\tilde{S}_{k+1} \leftarrow U_s \Pi_s$

Computa la descomposición RRQR
$\frac{1}{\sqrt{2\gamma_k}}\left[\tilde{R}_k,\ \ \gamma_k (\tilde{R}_k A_k^{-1})\right] = Q_r \begin{bmatrix} U_r \\ 0 \end{bmatrix} \Pi_r$
$\tilde{R}_{k+1} \leftarrow U_r \Pi_r$

2.4. Reducción de modelos: caso generalizado

Consideremos ahora el SDL generalizado

$$\begin{array}{rcll} E\dot{x}(t) & = & Ax(t) + Bu(t), & t > 0, \qquad x(0) = x_0, \\ y(t) & = & Cx(t) + Du(t), & t \geq 0, \end{array} \tag{35}$$

donde $E \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times n}$, con función de transferencia asociada

$$G(s) = C(sE - A)^{-1}B + D. \tag{36}$$

El problema de reducción de modelos se puede plantear en este caso como la búsqueda de un segundo SDL,

$$\begin{array}{rcll} E_r\dot{x}_r(t) & = & A_r x_r(t) + B_r u(t), & t > 0, \qquad x_r(0) = \hat{x}^0, \\ y_r(t) & = & C_r x_r(t) + D_r u(t), & t \geq 0, \end{array} \tag{37}$$

donde $E \in \mathbb{R}^{r\times r}$, $A \in \mathbb{R}^{r\times r}$, $B \in \mathbb{R}^{r\times m}$, $C \in \mathbb{R}^{p\times r}$, $D \in \mathbb{R}^{p\times r}$, $r \ll n$ e $\|y - y_r\|$ "pequeño".

En la resolución del problema de reducción de modelos generalizado son aplicables la mayor parte de los conceptos desarrollados previamente para el caso estándar ($E = I_n$), pero es necesario cuidar ciertos aspectos.

En este contexto, para encontrar los gramianos de controlabilidad y observabilidad, W_c y W_o, es necesario resolver dos ecuaciones de Lyapunov generalizadas de la forma

$$AW_cE^T + EW_cA^T + BB^T = 0, \tag{38}$$
$$A^T\tilde{W}_oE + E^T\tilde{W}_oA + C^TC = 0, \tag{39}$$

donde $W_o = E^T\tilde{W}_oE$.

Tras aplicar el método de la función signo (con las consiguiente modificaciones, expuestas en el siguiente apartado), se obtienen los factores S y R de

rango bajo, donde $W_c = RR^T$ y $W_o = S^T S$. Considérese la descomposición en valores singulares del producto

$$SR^T = U\Sigma V^T = [\, U_1 \; U_2 \,] \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} [\, V_1 \; V_2 \,]^T , \tag{40}$$

donde U y V son matrices ortogonales, y $\Sigma = \mathrm{diag}\,(\sigma_1, \sigma_2, \ldots, \sigma_n)$ es una matriz diagonal que contiene los valores singulares de SR^T. Dada una partición de Σ en $\Sigma_1 \in \mathbb{R}^{r\times r}$ y $\Sigma_2 \in \mathbb{R}^{(n-r)\times(n-r)}$, y una partición conforme de U y V en la Ec. (40), el método SRBT para el caso generalizado determina un modelo (E_r,A_r,B_r,C_r,D_r) de orden r ($r \ll n$) al realizar las siguientes operaciones:

$$\begin{array}{lllllllll} E_r & = & T_l E T_r, & A_r & = & T_l A T_r, & & & \\ B_r & = & T_l B, & C_r & = & C T_r, & D_r & = & D, \end{array} \tag{41}$$

donde

$$T_l = \Sigma_1^{-1/2} V_1^T R E^{-1} \quad \text{y} \quad T_r = S^T U_1 \Sigma_1^{-1/2}. \tag{42}$$

2.4.1. Ecuaciones de Lyapunov generalizadas

Una primera aproximación para la resolución del problema de reducción de modelos asociado a un SDL generalizado es mediante la transformación que toma como entrada la tupla $(\tilde{E}, \tilde{A}, \tilde{B}, \tilde{C})$, donde el par $\tilde{E}, \tilde{A} \in \mathbb{R}^{n\times n}$ define el haz de matrices de estados, $\tilde{B} \in \mathbb{R}^{n\times m}$ es la matriz de entradas, y $\tilde{C} \in \mathbb{R}^{p\times n}$ es la matriz de salidas, y produce el SDL estándar $(I_n, \tilde{E}^{-1}\tilde{A}, \tilde{E}^{-1}\tilde{B}, \tilde{E}^{-1}\tilde{C})$.

Al aplicar estas operaciones se obtiene un SDL donde los gramianos se derivan de la resolución de las ecuaciones de Lyapunov estándar. Sin embargo, esta aproximación puede presentar problemas de estabilidad numérica [87] por lo cual, en general, no es una vía recomendable.

Otro enfoque, más estable desde el punto de vista numérico, es la resolución directamente de las ecuaciones generalizadas de Lyapunov (Ecs. (38)-(39)); esto puede ser abordado con el método de la función signo y la iteración de Newton, tal y como muestra el Algoritmo `CGCLNW`.

Algoritmo `CGCLNW`**:**
Iteración factorizada de Newton para las Ecs. de Lyapunov generalizadas acopladas

$A_0 \leftarrow A$, $\tilde{S}_0 \leftarrow B^T$, $\tilde{R}_0 \leftarrow C$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
  $A_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(A_k + \gamma_k^2 (E A_k^{-1}) E\right)$
  Computa la descomposición RRQR
  $\frac{1}{\sqrt{2\gamma_k}}\left[\, \tilde{S}_k, \;\; \gamma_k \tilde{S}_k (E A_k^{-1})^T \,\right] = Q_s \begin{bmatrix} U_s \\ 0 \end{bmatrix} \Pi_s$
  $\tilde{S}_{k+1} \leftarrow U_s \Pi_s$
  Computa la descomposición RRQR
  $\frac{1}{\sqrt{2\gamma_k}}\left[\, \tilde{R}_k, \;\; \gamma_k (\tilde{R}_k A_k^{-1}) E \,\right] = Q_r \begin{bmatrix} U_r \\ 0 \end{bmatrix} \Pi_r$
  $\tilde{R}_{k+1} \leftarrow U_r \Pi_r$

Si el algoritmo converge tras $\tilde{k}_0$ iteraciones, entonces las aproximaciones de rango completo de S y R vienen dadas, respectivamente, por

$$\tilde{S} = \frac{1}{\sqrt{2}}\tilde{S}_{\tilde{k}_0}E^{-T} \quad \text{y} \quad \tilde{R} = \frac{1}{\sqrt{2}}\tilde{R}_{\tilde{k}_0}E^{-1}, \tag{43}$$

donde $\tilde{S} \in R^{\tilde{k}_o \times n}$, $\tilde{R} \in R^{\tilde{k}_c \times n}$ y, habitualmente, $\tilde{k}_o, \tilde{k}_c \ll n$, de forma que $W_c = S^T S \approx \tilde{S}^T \tilde{S}$ y $W_o = R^T R \approx \tilde{R}^T \tilde{R}$.

El núcleo del Algoritmo `CGCLNW`, desde el punto de vista del costo computacional, es la etapa que computa $\left(A_k + \gamma_k^2 (E A_k^{-1})E\right)$. A diferencia de lo que sucede en el método para la resolución de las ecuaciones estándar de Lyapunov, donde es necesario calcular la inversa de A_k, en este caso es más conveniente evitar el cálculo explícito de la inversa. En su lugar, resulta más eficiente calcular la factorización LU de la matriz A_k (asumiendo que esta matriz no presenta ninguna estructura particular), y resolver los dos sistemas triangulares correspondientes para encontrar la solución de $XLU = E$. De esta manera se realizan $\frac{2}{3}n^3$ flops para la factorización, n^3 flops para la resolución de cada uno de los dos sistemas triangulares y $2n^3$ flops más para el producto de EA_k^{-1} con E. Frente a esto, la alternativa supone $2n^3$ flops para la inversión de la matriz A_k más $4n^3$ flops adicionales que implica las multiplicaciones de matrices EA_k^{-1} y $(EA_k^{-1})E$.

2.5. Métodos de error relativo

Los métodos de reducción de modelos de error absoluto descritos en los apartados anteriores se basan en minimizar $\|\Delta_a\| = \|G - G_r\|$ para alguna norma. En particular, el método de realizaciones balanceadas proporciona una cota del error Δ_a según lo expuesto en la Ec. (9). Esta cota del error solo aporta información sobre la peor desviación del modelo de orden reducido con respecto al sistema original. Sin embargo, no se dispone de ninguna pauta de para qué entradas ocurre ese máximo de desviación. Por ejemplo, el método BT [83, 99] tiende a aproximar con mucha precisión las frecuencias altas ($\lim_{\omega\to\infty} \Delta_a(\jmath\omega) = 0$), mientras que el método de aproximación por perturbación singular [82] no tiene errores en los estados estacionarios (es decir, $G(0) = G_r(0)$) y presenta buenas propiedades de aproximación para las frecuencias bajas.

En diversas aplicaciones es necesario que el sistema de orden reducido provea una aproximación uniforme sobre todo el rango de frecuencias, $0 \leq \omega \leq \infty$, mientras que otras precisan únicamente de una buena aproximación para un rango de frecuencias preestablecidas. Este es el caso, por ejemplo, si el SDL describe el comportamiento de un controlador de alto orden que tiene que trabajar correctamente en un determinado rango de frecuencias. Este requerimiento puede ser satisfecho por los métodos de error relativo.

Los métodos de error relativo buscan minimizar el error relativo $\|\Delta_r\|_\infty$, definido implícitamente por $G - G_r = G\Delta_r$. Entre éstos, el método de truncamiento estocástico balanceado (BST) [44, 56, 104] y los métodos derivados de él son particularmente relevantes. Sin embargo, teniendo en cuenta su costo computacional y los cálculos involucrados, estos métodos pueden ser aplicados solo en sistemas de tamaño modesto, es decir, modelos donde n es del orden de algunos miles.

En cuanto a los fundamentos matemáticos, BST es un método de reducción de modelos basado en truncar una realización estocásticamente balanceada. Sean $\Phi(s) = G(s)G^T(-s)$ y W un factor espectral de fase mínima de Φ, es decir $\Phi(s) = W^T(-s)W(s)$, con D una matriz de rango completo, $\tilde{D} = DD^T$ definida positiva; entonces, una realización en el espacio de estados mínima (A_W, B_W, C_W, D_W) de W viene dada por [12, 13]:

$$A_W = A, \quad B_W = BD^T + W_c C^T, \quad C_W = \tilde{D}^{-\frac{1}{2}}(C - B_W^T X_W), \quad D_W = \tilde{D}^{\frac{1}{2}}.$$

Aquí, W_c es el gramiano de controlabilidad de $G(s)$, la solución de la ecuación de Lyapunov estándar,

$$AW_c + W_c A^T + BB^T \ = \ 0 \tag{44}$$

mientras X_W es el gramiano de observabilidad de $W(s)$, obtenido como la solución estabilizante de la ecuación algebraica de Riccati (EAR)

$$(A - B_W\tilde{D}^{-1}C)^T X + X(A - B_W\tilde{D}^{-1}C) + XB_W\tilde{D}^{-1}B_W^T X + C^T\tilde{D}^{-1}C = 0. \tag{45}$$

Es decir, para esta solución particular,

$$\hat{A} = A - B_W\tilde{D}^{-1}C + B_W\tilde{D}^{-1}B_W^T X_W \tag{46}$$

es estable. Además, los gramianos W_c y X_W son matrices simétricas definidas (semi-)positivas y admiten descomposiciones $W_c = S^T S$ y $X_W = R^T R$ (factorizaciones de Cholesky). Como en el cómputo de una realización balanceada, puede obtenerse una transformación en el espacio de estados T con los subespacios invariantes dominantes por izquierda y derecha de $W_c X_W$, o como con los subespacios singulares por izquierda y derecha de SR^T, tal que la transformación del sistema dada por $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}) = (T^{-1}AT, T^{-1}B, CT, D)$ tiene el gramiano de controlabiliad $\tilde{W}_c$ que satisface:

$$\tilde{W}_c = T^{-1}W_c T^{-T} = \text{diag}\,(\sigma_1, \ldots, \sigma_n) = T^T X_W T = \tilde{X}_W, \tag{47}$$

donde $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$. Esto es, el gramiano de observabilidad $\tilde{X}_W$ de la realización transformada de factor espectral derecho de $\Phi(s)$ es igual al gramiano de controlablidad de la realización transformada de $G(s)$. La realización de $G(s)$ se conoce como realización estocástica balanceada (BSR, del inglés *balanced stochastic realization*). La reducción de modelos basada en BSR se puede obtener truncando la realización $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ a un orden r donde $\sigma_r \gg \sigma_{r+1}$. Las propiedades del método BST se resumen en el siguiente enunciado, que reúne resultados de diversos trabajos [44, 56, 57].

Considérese la función de transferencia matricial del SDL $G(s) = C(sI_n - A)^{-1}B + D$ con A estable y D no singular y suponiendo que

$$(T^{-1}AT, T^{-1}B, CT, D) = \left(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \end{bmatrix}, D \right)$$

es una BSR tal que satisface (47); entonces

$$(A_r, B_r, C_r, D_r) = (A_{11}, B_1, C_1, D)$$

es estable y mínima con las siguientes propiedades:

a) $G_r(s) = C_r(sI_r - A_r)^{-1}B_r + D_r$ satisface la cota de error relativo

$$\|\Delta_r\|_\infty = \|G^{-1}(G - G_r)\|_\infty \leq \prod_{j=r+1}^{n} \frac{1+\sigma_j}{1-\sigma_j} - 1. \tag{48}$$

b) $G(s)$ es de fase mínima, es decir, G no tiene ceros en $\mathbb{C}_+$. De igual modo, $G_r(s)$ es de fase mínima.

En este trabajo se aborda el método de la raíz cuadrada, especialmente adecuado cuando se desea trabajar con SR^T en lugar de $W_c X_w$; ver [35]. En el mismo trabajo se puede encontrar un profundo relevamiento de los distintos enfoques para abordar la resolución del método.

Como se mencionó anteriormente, en el método BST es necesario resolver primero la ecuación de Lyapunov (44), para lo cual se puede utilizar el método de la función signo, obteniendo la factorización de rango completo de $W_c = S^T S$, es decir, $S \in \mathbb{R}^{n_c \times n}$ donde $n_c = \operatorname{rank}(S) = \operatorname{rank}(W_c)$. Además, es necesario resolver una ecuación de Riccati, para la cual se puede emplear tanto el método de Newton [17, 65] como el método de la función signo [23, 92]. En los siguientes apartados se describen ambas técnicas y su aplicación en el método BST.

2.5.1. Método de Newton para la ecuación de Riccati

El método de Newton permite resolver una ecuación de Riccati de la forma

$$F^T X + XF - XGX + Q = 0, \tag{49}$$

donde $F, G \in \mathbb{R}^{n \times n}$ son matrices simétricas definidas positivas.

El método se basa en una iteración en la cual se corrige la solución obtenida en cada paso, obteniéndose la matriz de corrección mediante la resolución de una ecuación de Lyapunov sobre la matriz F y con término independiente el error de la solución obtenida en el paso previo.

Para acelerar la convergencia de la iteración se puede utilizar la técnica *exact line search* [26], que permite disminuir la cantidad de pasos de la iteración. Hay que destacar el elevado costo computacional que implica cada paso del método, pues es necesario resolver una ecuación de Lyapunov, por lo que los beneficios que puedan derivarse de una aceleración de la convergencia son claramente relevantes. La técnica se basa en modificar el tamaño del paso, utilizando el óptimo, es decir aquel que minimiza la norma Frobenius del próximo residuo. Esto supone un costo muy inferior al de una sola iteración, según [25] entre un 5 – 10 %. El algoritmo resultante, `GECRNW`, se detalla a continuación. Para especificar el método de Newton estándar basta con fijar la variable $t_k = 1$ para todas las iteraciones k.

Algoritmo `GECRNW`**:**
Método de Newton con *exact line search* para la ecuación de Riccati
$F_0 \leftarrow F$, $X_0 \leftarrow X$
`for` $k = 0, 1, 2, \ldots$ *until convergence*
  $F_k \leftarrow F + GX_k$
  $R(X_k) \leftarrow F_k^T X_k + X_k F_k - X_k G X_k + Q$
  Resolver $F_k^T N_k + N_k F_k + R(X_k) = 0$
  Calcular el parámetro t_k
  $X_{k+1} \leftarrow X_k + t_k N_k$

Se puede observar que la EAR que se resuelve en el método Newton, Algoritmo `GECRNW`, difiere de la que aparece en el método BST (en 45); sin embargo, la Ec. (45) se puede transformar sencillamente en la Ec. (49) mediante los siguientes cambios de variables:

$$F = A - B_W \tilde{D}^{-1} C, \quad G = B_W \tilde{D}^{-1} B_W^T, \quad Q = C^T \tilde{D}^{-1} C. \tag{50}$$

El núcleo de mayor peso en cuanto al costo computacional del método es la resolución de la ecuación de Lyapunov, para lo cual se pueden utilizar las técnicas desarrolladas en este mismo capítulo basadas en la función signo.

2.5.2. Función signo para la ecuación de Riccati

La solución de la ecuación de Riccati (49) es una matriz $X \in \mathbb{R}^{n \times n}$ que cumple que los valores propios de la matriz $F - GX$ tienen parte real negativa. La solución de la EAR puede definirse también en términos de los subespacios invariantes del haz $H - \lambda I_{2n}$, donde H es la matriz Hamiltoniana

$$H = \begin{bmatrix} F & G \\ -Q & -F^T \end{bmatrix}. \tag{51}$$

Además, se puede demostrar que la matriz formada por una base de los espacios invariantes de la matriz H es la solución de la EAR asociada [28]. Entonces, una forma de calcular esta solución es separando el espectro de H en dos subespacios complementarios (estable y anti-estable). Se puede emplear esta propiedad, combinándola con la función signo, para resolver la primera etapa del procedimiento requerido. A partir de la función signo de H,

$$\operatorname{fsign}(H) = Y = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}, \tag{52}$$

para encontrar la solución a la EAR únicamente es necesario resolver (por ejemplo mediante aproximaciones de mínimos cuadrados) el siguiente sistema,

$$\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}. \tag{53}$$

El procedimiento resultante de aplicar los pasos antes descritos se resume en el Algoritmo `GECRSG`.

Algoritmo `GECRSG`**:**
Método de la función signo para la ecuación de Riccati

$H_0 \leftarrow \begin{bmatrix} F & G \\ -Q & -F^T \end{bmatrix}$

`for` $k = 0, 1, 2, \ldots$ *until convergence*

$H_{k+1} \leftarrow \frac{1}{2\gamma_k}\left(H_k + \gamma_k^2 H_k^{-1}\right)$

Resolver $\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}$

3. Técnicas de programación paralela/distribuida

En esta sección se ofrece una breve introducción al uso de las técnicas de programación paralela y distribuidas. Las técnicas de programación paralela están estrechamente relacionadas con las arquitecturas de hardware, por esta razón, en este apartado se describen las diferentes familias de arquitecturas utilizadas para HPC.

3.1. Arquitecturas de computadoras paralelas

En la década de 1960, con la computadora CDC 6600, con 9 MFLOPS (millones de operaciones aritméticas en coma flotante por segundo) de poder de cómputo, se comenzó a utilizar el término supercomputadora. Desde entonces, el avance y las mejoras en el hardware de HPC han sido constantes y notables. Las opciones de arquitecturas paralelas disponibles son diferentes, tanto desde el punto de vista del poder de cómputo de los equipos, como de las conexiones entre los elementos de procesamiento, el uso de memoria, etc., teniendo cada una de las plataformas de HPC características, limitaciones y bondades específicas. La gran variedad de plataformas de hardware de HPC ha propiciado la existencia de múltiples clasificaciones basadas en diferentes criterios. A continuación se presentan algunas de las taxonomías más difundidas.

Flynn [47] utiliza como criterio para clasificar las computadoras el manejo que realizan de las instrucciones y de los datos. Se distingue así entre instrucciones unitarias (SI, del inglés *single instruction*) y múltiples (MI, del inglés *multiple instruction*) y datos unitarios (SD, del inglés *single data*) o múltiples (MD, del inglés *multiple data*), especificando cuatro clases, como se puede apreciar en la Tabla 1.

		Datos	
		SD	MD
Instrucciones	SI	SISD	SIMD
	MI	MISD	MIMD

Tabla 1: Clasificación de arquitecturas según la taxonomía de Flynn.

Dentro de la categoría SISD se encuentran las computadoras secuenciales. En las categorías SIMD y MIMD se encuentran generalmente las computado-

ras paralelas utilizadas en la actualidad. En los equipos SIMD, todos los procesadores reciben la misma instrucción y operan concurrentemente sobre distintos datos, mientras que en los MIMD las unidades de proceso pueden ejecutar simultáneamente diferentes instrucciones sobre distintos conjuntos de datos, trabajando asíncronamente.

Otra clasificación muy utilizada para arquitecturas con múltiples unidades de cómputo [10] se basa en la organización del sistema de memoria. Esta taxonomía reconoce dos clases de arquitecturas:

- Multiprocesadores de memoria compartida. Todos los procesadores comparten y tienen acceso a la memoria del sistema.
- Multiprocesadores de memoria distribuida. Cada unidad de proceso dispone de una fracción de la memoria total del sistema que solo puede ser accedida por la unidad propietaria. La compartición de datos entre los procesadores se realiza a través del intercambio de mensajes. La categoría también se denomina computadores paralelos débilmente acoplados. Un ejemplo típico de equipos pertenecientes a esta categoría son los clusters beowulf [24], sistemas de desempeño escalable en base a hardware de bajo costo, software libre y que utilizan redes de tecnología estándar para interconectar los nodos.

En la actualidad, otra taxonomía ampliamente extendida propone agrupar los computadores paralelos en base a grandes clases de arquitecturas [10]. A continuación se describen las cinco categorías.

- Multiprocesadores simétricos (SMP, del inglés *Symmetric MultiProcessor*). Son equipos con varias unidades de procesamiento muy acopladas (comparten gran cantidad de los recursos incluida la memoria). También suelen denominarse sistemas de grano grueso.
- Equipos con acceso a memoria no uniforme (NUMA). Estos computadores están compuestos por varios procesadores, cada uno de los cuales posee su propia memoria. La característica que distingue a los computadores de esta clase es que a pesar de que cada procesador posee su propia memoria, se dispone de un direccionamiento de memoria global. Existen equipos que emplean coherencia de caché (cc-NUMA) para facilitar su programación, en contraposición con los que no (nc-NUMA).
- Computadores masivamente paralelos (MPP, del inglés *Massively Parallel Processing*). Son equipos que poseen varias unidades de procesamiento poco acopladas, en los que cada procesador accede a su propia memoria. También son denominados sistemas de grano fino.
- Clusters de PCs. Se constituyen por un conjunto de computadores personales (PCs) interconectados mediante una red.
- Grids. Se componen de un conjunto de equipos autónomos conectados mediante una red. En este caso los equipos pueden ser heterogéneos en cuanto a arquitectura (incluso pueden pertenecer a distintas clases de las antes mencionadas), velocidad, sistema operativo, etc.

Si bien las clasificaciones descritas anteriormente son suficientes a los efectos de este trabajo, se destaca que existen otras clasificaciones en base a distintos criterios: basadas en la forma de trabajo de los procesadores de los equipos (síncrona o asíncrona), el poder de cómputo de las unidades de proceso, la cantidad de procesadores, la topología de la red de interconexiones, etc. [10].

3.1.1. Programación paralela

Como se comentó en el apartado anterior, existen diversas arquitecturas de computadoras para HPC. En consecuencia, existen distintas técnicas de programación paralela, cada una de las cuales se adapta mejor a las características de un tipo específico de arquitectura paralela. La programación paralela consiste, esencialmente, en dividir el problema para su resolución simultánea en diferentes unidades de proceso de forma cooperativa. Para efectuar la división del problema existen dos estrategias principales:

- Descomposición funcional. En este caso se dividen las distintas tareas a realizar por el algoritmo entre las distintas unidades de procesamiento disponibles.
- Descomposición de datos. En esta estrategia todas las unidades de procesamiento disponibles realizan las mismas tareas, pero cada una trabaja sobre un subconjunto de los datos del problema.

Además de las dos estrategias descritas anteriormente, en la práctica son ampliamente utilizadas las estrategias híbridas, que dividen tanto las tareas como los datos entre las diferentes unidades de procesamiento.

Al igual que ocurre con las estrategias de división de problemas, las distintas técnicas de programación paralela se encuentran muy ligadas al hardware subyacente, y en particular, a la forma en que las unidades computacionales pueden comunicarse y compartir información. A continuación se presentan las principales técnicas de programación paralela y su relación con las diferentes arquitecturas.

- Pasaje de mensajes: las unidades de procesamiento se comunican y coordinan entre sí mediante el pasaje explícito de mensajes. Esta técnica es muy utilizada cuando se trabaja sobre hardware poco acoplado, típicamente clusters o sistemas distribuidos.
- Memoria compartida: las unidades de procesamiento trabajan de forma coordinada gracias al uso compartido de la memoria central. Esta técnica es típicamente utilizada sobre multiprocesadores de memoria compartida.

Al programar en paralelo aparecen varios factores específicos que influyen en el rendimiento. Entre ellos destacan:

- Balance de carga: es necesario dividir el trabajo a realizar de forma equitativa entre las distintas unidades de proceso. La distribución será igualitaria si todos los elementos de procesamiento tienen la misma capacidad (es decir, arquitectura homogénea) o se deberá distribuir el trabajo de forma proporcional a las capacidades de cómputo de los componentes si se trata de computadores heterogéneos.

- Buen nivel de concurrencia: es prioritario lograr que todas las unidades de proceso trabajen de forma concurrente, evitando tiempos superfluos de espera y sincronización.
- Bajo sobrecosto: se deben minimizar las tareas de coordinación necesarias para realizar el trabajo en paralelo, por ejemplo, minimizando comunicaciones, trabajo redundante, etc.

Otro aspecto a tener en cuenta al desarrollar programas paralelos es el tratamiento de las tareas no paralelizables, es decir, aquellas tareas que sólo pueden ser realizadas por una única unidad de procesamiento. En este contexto, el tiempo de ejecución de las tareas secuenciales es una cota inferior del tiempo total de ejecución del algoritmo paralelo. El enunciado de este concepto es conocido como ley de Amdahl [11].

3.2. Herramientas para el diseño e implementación de aplicaciones paralelas

Así como existen diversas arquitecturas de computadores paralelos y se dispone de diferentes técnicas de programación paralela, también se han implementado diversas herramientas para desarrollar programas bajo el paradigma de la programación paralela y distribuida. Durante muchos años la tendencia por parte de los vendedores de hardware fue desarrollar herramientas específicas para sus equipos, existiendo incluso herramientas diseñadas para un determinado modelo de computador. Sin embargo, en las últimos décadas la tendencia gradualmente fue cambiando hacia el desarrollo de herramientas genéricas, buscándose cada día un mayor equilibrio entre el desempeño computacional y la portabilidad/productividad.

Por lo general, las características de las distintas herramientas están asociadas con alguna técnica particular de programación paralela. Por este motivo, en muchas ocasiones la elección de una herramienta u otra se ve condicionada por la técnica de programación a utilizar, que a su vez generalmente se encuentra supeditada por el hardware subyacente. Esto implica que es imposible afirmar de forma general que una herramienta es mejor que otra, siendo necesario evaluar en cada ocasión, y dependiendo del hardware y estrategia de programación paralela a utilizar, qué herramienta se adapta mejor a la resolución de un problema determinado en una arquitectura concreta.

En este trabajo se reseñan las características más importantes de dos herramientas de uso general, el estándar MPI (*Message Passing Interface*) [60] para plataformas de memoria distribuida y la interfaz OpenMP [6] para memoria compartida.

El estándar MPI

Existen diversas bibliotecas para la programación paralela que implementan el paradigma de pasaje de mensajes. Muchas de las grandes compañías que venden computadoras paralelas han creado sus propias bibliotecas. De forma frecuente estas bibliotecas se encuentran diseñadas y optimizadas para un tipo de arquitectura de hardware en particular, siendo en general de difícil comprensión, poca legibilidad y no portables a otras arquitecturas. En las últimas

décadas, con el propósito de mejorar la portabilidad de aplicaciones, se han introducido las especificaciones de tipo estándar. Pese a la potencial disminución de la eficiencia en los programas debido a una adaptación incompleta a las propiedades del hardware, el uso de estas especificaciones se ve compensada en una visión de largo plazo, dado que estas estrategias permiten portar códigos a nuevas arquitecturas; facilitar el intercambio de experiencias, el mantenimiento y legibilidad del software; etc.

El estándar MPI [60] fue definido en el año 1992 por un conjunto de desarrolladores de software y aplicaciones científicas en coordinación con importantes empresas (IBM, Intel), con el objetivo de proporcionar una especificación de biblioteca implementable sobre una variada gama de arquitecturas, permitiendo así diseñar aplicaciones distribuidas portables y eficientes. En el estándar se definen las funcionalidades y diversas características deseables que debe incorporar una biblioteca orientada al pasaje de mensajes. El estándar MPI se basa en la experiencia obtenida con herramientas anteriores, como por ejemplo la biblioteca PVM (*Parallel Virtual Machine*) [50], ampliamente utilizada en los finales de la década de 1980 en el ámbito científico.

Entre las principales características del estándar MPI destacan:

- El paralelismo es explícito, es decir, es definido y controlado en su totalidad por el programador.
- Utiliza como único mecanismo de comunicación entre los procesos el pasaje de mensajes explícito.
- Tiene como objetivo principal el trabajo sobre plataformas de memoria distribuida, aunque incluye primitivas para el trabajo sobre plataformas de memoria compartida.
- Considera como mecanismo de diseño de programas el modelo SPMD (del inglés *single program multiple data*), aunque posee la capacidad de adaptarse de manera sencilla al desarrollo de programas bajo un modelo MIMD.

El estándar completo de MPI posee más de 130 funciones que permiten la sincronización, comunicación y trabajo cooperativo. Entre las tareas que implementan las funciones de MPI, destacan el envío y recepción punto a punto de mensajes (bloqueantes o no), las operaciones de difusión (*broadcasting*, envío de mensajes de un proceso a todos los restantes), las operaciones de reducción (recolectar y reducir datos de varios procesos en uno), y la obtención de información del proceso y entorno de ejecución.

Existen diversas implementaciones de la especificación MPI para trabajar con los lenguajes C y Fortran. Dos de las más difundidas son MPICH [5, 58, 59], del Argonne National Laboratory (ANL) [7] de Estados Unidos, y OpenMPI [49].

La interfaz OpenMP

El éxito del estándar MPI por un lado y el renovado interés por el paradigma de paralelismo de memoria compartida debido al crecimiento de capacidad de los computadores multinúcleo, impulsaron a un grupo de instituciones, fabricantes de computadoras y empresas desarrolladoras de compiladores a propon-

er estándar para programación sobre memoria compartida. En el año 1997 se presentó la interfaz OpenMP.

El diseño de la interfaz posee como principales características ser portable, escalable y relativamente simple de utilizar para el desarrollo de programas paralelos tanto para pequeñas computadoras como para equipos de gran porte.

Actualmente se dispone de versiones de OpenMP para los lenguajes C/C++ y las distintas versiones de Fortran. Entre las funcionalidades de la interfaz se encuentran el definir secciones paralelas, secciones críticas y bucles paralelos. Además, emplea variables de entorno para controlar la ejecución de un programa; por ejemplo, la cantidad máxima de hilos a utilizar durante la ejecución se especifica mediante la variable $OMP\_NUM\_THREADS$.

Sobre las diversas opciones para trabajar con la interfaz OpenMP se puede profundizar en el trabajo de Narus et al. [85].

3.2.1. Medidas del desempeño computacional

Una primera aproximación intuitiva para medir el desempeño de los computadoras puede obtenerse mediante la observación de la velocidad del procesador o del número de instrucciones que es capaz de ejecutar, evaluando los millones de instrucciones por segundo (MIPS) que puede realizar el equipo. Sin embargo, los MIPS no aportan información para comparar dos computadores que poseen distintas arquitecturas, y sobre todo, distintos juegos de instrucciones.

Una forma, más completa y ampliamente utilizada en computación científica, es medir el desempeño de una computadora o algoritmo evaluando cuántos millones de operaciones de coma flotante puede realizar en un segundo (MFLOPS), o en computadoras más potentes cuántos miles de millones de flops (GFLOPS) pueden realizar.

Otra estrategia orientada a la evaluación de los equipos es emplear alguno de los *benchmark* de uso público. Un *benchmark* es un conjunto de programas que se utilizan para comparar el rendimiento de distintos computadores o algoritmos. Entre los benchmarks de uso más extendido en la comunidad se encuentran el HINT [62], propuesto por Gustafson y Snell, y el Linpack [4], propuesto por Dongarra. Este último benchmark emplea subrutinas de álgebra lineal, como la factorización LU, para evaluar el desempeño de los computadores y es utilizado para confeccionar la lista Top500.

En muchas situaciones o estudios, el interés radica en la evaluación de distintos algoritmos o estrategias para la resolución de un mismo problema. Si durante el estudio se emplea siempre la misma computadora para realizar las ejecuciones, utilizar el tiempo de cómputo como medida de desempeño es una opción válida. El tiempo efectivo de cómputo permite realizar una evaluación de forma global para una arquitectura; esta evaluación será válida independientemente de si el algoritmo utiliza más instrucciones, más operaciones de coma flotante, menos movimientos en la memoria, etc.

El análisis del desempeño computacional de un programa es una tarea sumamente compleja. Cuando se trata de medir el desempeño computacional de un algoritmo que utiliza técnicas de programación paralela, la dificultad es mayor, ya que sumado a la complejidad de evaluar un programa secuencial aparecen nuevos condicionantes como son los tiempos de comunicación entre procesos,

los tiempos de sincronización, etc., que tienen que ser evaluados de forma separada. Otro aspecto a tener en cuenta al medir el desempeño de algoritmos que se ejecutan en paralelo es conocer la ganancia al ejecutarlos en paralelo en relación con la versión secuencial. Esta ganancia es habitualmente conocida como el *speedup* (también se estila utilizar el *speedup* normalizado, conocido como eficiencia computacional). Estos resultados están fuertemente ligados al equipo en el cual se ejecutan. En resumen, para evaluar programas paralelos se puede utilizar:

- El tiempo total de ejecución, comparando los tiempos de ejecución de las distintas versiones de un algoritmo en un mismo equipo.
- El *speedup* y la eficiencia computacional, que se definen en el siguiente apartado.

Existe otra gran cantidad de métricas para evaluar el desempeño de programas de computación paralela. Alguna de las más utilizadas son la escalabilidad, la isoeficiencia y la isovelocidad. Para profundizar sobre el tema se puede consultar en el trabajo publicado por Sahni y Thanvantri [97]. Además, en los últimos años existe un creciente interés por el uso de medidas que incluyan aspectos económicos, como la capacidad de cómputo por dólar invertido, o la capacidad de cómputo por potencia consumida.

Speedup y eficiencia de un algoritmo paralelo

El *speedup* absoluto relaciona el tiempo de ejecución del mejor algoritmo secuencial conocido con el tiempo de ejecución del algoritmo ejecutado en paralelo. La métrica se define por la expresión $S(n) = \frac{Tsec}{T(n)}$ donde *Tsec* es el tiempo de ejecución del mejor algoritmo secuencial conocido y $T(n)$ representa el tiempo de ejecución del algoritmo paralelo utilizando n unidades de cómputo. La definición del *speedup* absoluto es poco práctica, porque no siempre se conoce el mejor algoritmo secuencial para resolver un problema, y en caso de que éste se conozca, en la mayoría de las ocasiones no se dispone de una implementación para evaluar su ejecución en un equipo determinado. Por este motivo, en general suele utilizarse como base de la comparación el tiempo de ejecución del algoritmo paralelo trabajando en una única unidad de cómputo. La medida resultante, denominada *speedup* algorítmico, se define mediante la expresión $S(n) = \frac{T(1)}{T(n)}$ donde $T(1)$ es el tiempo de ejecución del algoritmo paralelo ejecutando en una única unidad de cómputo.

En ocasiones, no sólo es importante el valor de *speedup* para una cantidad dada de unidades de cómputo, sino que también es relevante observar la variación de la ganancia a medida que crece la cantidad de unidades computacionales utilizadas. En este caso se puede utilizar la métrica eficiencia, que corresponde al valor del *speedup* normalizado por el número de unidades computacionales utilizadas. La métrica se define como $E(n) = \frac{S(n)}{n}$, y dependiendo del modo de calcular el *speedup*, se tendrá un valor de eficiencia absoluta o de eficiencia algorítmica (también denominada eficiencia relativa).

3.3. HPC y ALN

Diversas aplicaciones científicas tienen la particularidad de que sus etapas más costosas, desde el punto de vista de tiempo de ejecución, requieren el cómputo de operaciones de ALN. A modo ilustrativo, algunos ejemplos donde se puede corroborar esta afirmación son los problemas de optimización lineal, resolución de ecuaciones diferenciales, problemas de radiosidad en computación gráfica y la reducción de modelos. Este hecho ha motivado que el área de ALN concentre, desde varias décadas atrás, gran caudal de trabajo, buscando desarrollar métodos eficientes que exploten las características de las diferentes arquitecturas de computadoras empleadas.

Una política impuesta en el área es el desarrollo de especificaciones de bibliotecas (o interfaces) cuyas implementaciones permiten resolver problemas básicos de ALN, que pueden ser utilizadas para construir métodos que permitan tratar problemas más complejos. También, de esta manera se puede fácilmente optimizar las aplicaciones sin necesidad de grandes modificaciones y tiempos de desarrollo, únicamente empleando implementaciones eficientes de las especificaciones de las bibliotecas que resuelven los núcleos básicos.

Un caso paradigmático de lo antes expuesto es la especificación BLAS (del inglés *Basic Linear Algebra Subprograms*), que incluye en su funcionalidad operaciones básicas de ALN. Buscando replicar este caso se han desarrollado diversos esfuerzos similares, como LAPACK y SCALAPACK. A continuación se presenta una breve reseña de estas bibliotecas.

Existen múltiples bibliotecas de características similares a las presentadas a continuación, así como esfuerzos exitosos para desarrollar soluciones a problemas más específicos. En el repositorio NetLib [1] se encuentra un buen compendio del trabajo en el área, teniendo disponible una amplia gama de bibliotecas, algoritmos y artículos científicos en este ámbito.

3.4. BLAS

En 1973, Hanson et al. [64] dieron cuenta de la necesidad de adoptar un conjunto de rutinas básicas para problemas de ALN. Pretendían definir un estándar para las operaciones elementales entre vectores que permitiera mejorar la claridad, portabilidad y modularidad de las rutinas numéricas, simplificando el mantenimiento y desarrollo de futuro software. La idea tuvo un éxito considerable; la especificación BLAS fue desarrollada por varias empresas de fabricación de computadoras, que implementaron el estándar optimizando las operaciones para la arquitectura de sus equipos. Al correr de los años, con la evolución de las arquitecturas y en especial con la difusión de las computadoras vectoriales, la biblioteca BLAS pasó a ser una limitante al momento de mejorar el desempeño de los códigos numéricos. Como consecuencia, en el año 1984 comenzó a discutirse la extensión de los subprogramas incluidos en la primera especificación de BLAS. Tras varias reuniones y publicaciones se presentó la extensión, agregando funciones para las operaciones matriz-vector elementales. Las nuevas funciones se denominaron BLAS-2, reservándose el término de BLAS-1 para las funciones originales. Nuevamente, la evolución de la tecnología, en especial el creciente uso de memoria jerárquica y la difusión de los procesadores de memoria compartida, motivaron la extensión de la especificación, debido a que las funciones de BLAS-1 y BLAS-2 exhibían una relación entre las opera-

ciones de coma flotante realizadas y los movimientos de datos entre los distintos niveles de memoria poco eficiente. La extensión incluyó operaciones de tipo matriz-matriz, que fueron denominadas funciones BLAS-3.

Los tres niveles de BLAS resuelven múltiples operaciones con vectores y varios tipos de matrices (generales y estructuradas) para varios tipos de datos (simple y doble precisión, reales y complejos). En el trabajo de Dongarra et al. [45] se puede obtener una descripción completa de los distintos niveles de funciones de la biblioteca BLAS.

Una extensión sobre la implementación de BLAS consistió en dotarlas de la capacidad de ser ejecutadas en paralelo, sobre un modelo de memoria compartida. En este sentido existen implementaciones que utilizan estrategias multihilo como la presentada por T. Guignon, denominada BLASTH [98], la desarrollada por K. Goto disponible en su sitio web [54] y la incluida en la distribución de Intel, la biblioteca MKL [70].

Otra extensión de la biblioteca BLAS disponible es la que se define para vectores y matrices dispersas. Algunos ejemplos de implementaciones de esta extensión son la NIST Sparse BLAS [9] y la disponible en el repositorio Netlib de Dodson et al. [3].

3.5. LAPACK

En el año 1987 se desarrolló en la Universidad de Tennessee, Estados Unidos, un proyecto para implementar una biblioteca que permitiera la resolución de los problemas más comunes de ALN. El esfuerzo se cristalizó en la especificación LAPACK [42] (*Linear Algebra PACKage*), y una implementación de la misma, conocida como implementación de referencia. La especificación incluye funciones para la resolución de sistemas de ecuaciones lineales, aproximación de mínimos cuadrados, búsqueda de valores propios y de valores singulares.

La implementación de referencia de LAPACK utiliza rutinas de BLAS para efectuar las operaciones elementales sobre vectores y matrices. Este hecho permite la portabilidad y adaptación de LAPACK a diferentes arquitecturas sin afectar negativamente su desempeño.

La implementación de referencia de LAPACK está originalmente desarrollada en el lenguaje Fortran 77, es de dominio público y se puede obtener en el repositorio Netlib [2]. Además, existen diversas implementaciones optimizadas, entre las que destaca la incluida en la biblioteca MKL de Intel.

3.6. SCALAPACK

El aumento en la utilización de computadoras paralelas de bajo costo, y en especial, de aquellas con memoria distribuida, motivó la evolución de LAPACK a una nueva versión basada en técnicas de programación paralela distribuida; esta nueva biblioteca se denominó SCALAPACK [41] (*SCAlable* LAPACK) y la primera versión fue publicada en el año 1995.

Una de las principales características de la biblioteca SCALAPACK es que su diseño especifica distintos niveles de abstracción, buscando encapsular diferentes tareas en distintas componentes. En la Figura 2 se puede observar el diseño de componentes de la biblioteca SCALAPACK.

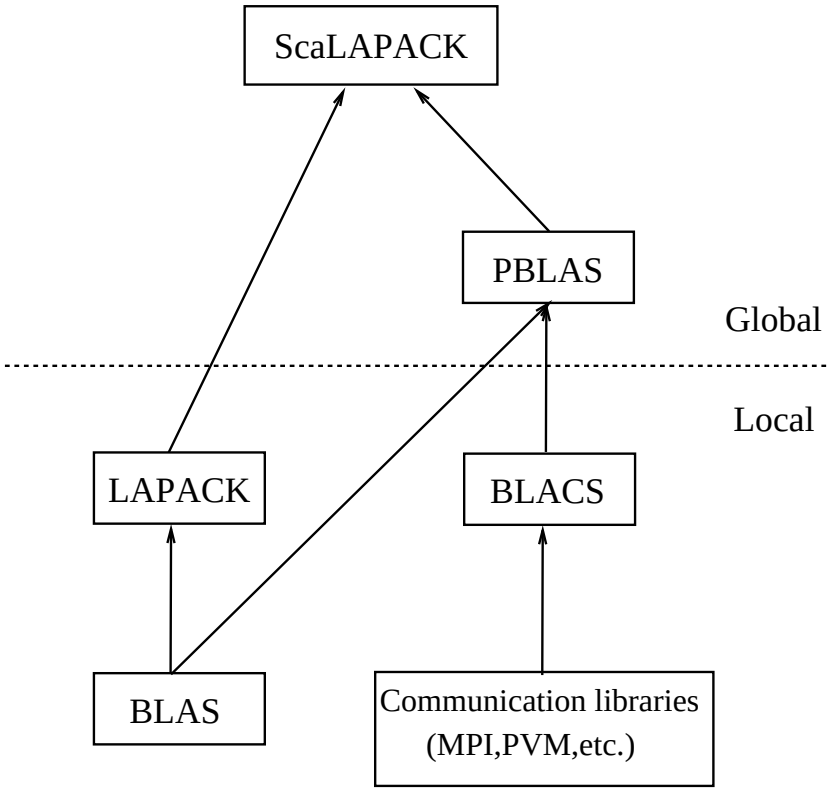

Figura 2: Diseño simplificado de la biblioteca SCALAPACK. Extraído de [46].

Al igual que LAPACK, SCALAPACK posee funciones para las operaciones más comunes en el campo del ALN, tales como la factorización LU y la factorización QR. Sin embargo, SCALAPACK está diseñada para su ejecución en plataformas de memoria distribuida, en las que las matrices y vectores se distribuyen entre las unidades de procesamiento de forma cíclica sobre una estructura de comunicación 2D.

Al diseñar la biblioteca SCALAPACK se decidió encapsular las operaciones sobre vectores que se debían realizar de forma distribuida. En este sentido, y debido a la notable influencia que tuvo en la comunidad científica la biblioteca BLAS, se creó la especificación de la biblioteca PBLAS (*Parallel* BLAS) [40]. La especificación de PBLAS posee las mismas operaciones que dispone BLAS, pero en este caso las operaciones están diseñadas para ser ejecutadas en paralelo sobre memoria distribuida mediante el paradigma de pasaje de mensajes (MPI, PVM, etc.).

Las operaciones de comunicaciones que utilizan pasaje de mensajes son resueltas mediante la invocación a funciones de la biblioteca BLACS. BLACS consta de un conjunto de rutinas para la transferencia de datos mediante el pasaje de mensajes específicamente diseñadas para dar soporte a la resolución de operaciones de ALN en plataformas con memoria distribuida. La biblioteca incluye la especificación de rutinas de envío y recepción de mensajes de forma síncrona

y asíncrona, tanto para realizar la comunicación de matrices y submatrices entre procesadores, como para realizar envíos en forma de difusión y reducciones globales (sumas, máximo, mínimo, etc.). En el repositorio NetLib se pueden obtener versiones de BLACS implementadas para diversas bibliotecas de pasaje de mensaje (MPI, PVM, NX), y con interfaz para los lenguajes Fortran y C, al igual que una implementación de la biblioteca PBLAS.

4. Aplicación de técnicas de HPC a la reducción de modelos

Dada la importancia del área de reducción de modelos, los grandes costos computacionales que implica su tratamiento y el creciente uso de las técnicas de HPC en los últimos años se han presentado diversos trabajos que abordan la aplicación de técnicas de HPC a la reducción de modelos.

Uno de los esfuerzos pioneros en el desarrollo de herramientas que incluyan estrategias de HPC para la resolución de problemas de control y en particular de reducción de modelos es la biblioteca SLICOT (*Subroutine Library in Control Theory*)[2] [103], que fue desarrollada a finales de la década de los 90, en el marco del proyecto NICONET (*Numerics in Control Network*) [8]. La biblioteca está implementada en Fortran 77 y también es posible su utilización desde Matlab. Su diseño se basa en el uso de las especificaciones LAPACK y BLAS, por lo cual se puede obtener fácilmente una versión paralela de la biblioteca sobre memoria compartida utilizando una versión multihilo de BLAS. SLICOT es fuertemente utilizada por investigadores del área aún en la actualidad. Los esfuerzos previos a la introducción de SLICOT por desarrollar bibliotecas para la resolución de problemas de control están relevados en el trabajo de Benner et al. [31].

Posteriormente, se desarrollaron diferentes esfuerzos para incorporar estrategias de paralelismo de memoria distribuida a la biblioteca SLICOT, desembocando en la realización de la biblioteca PLiC (*Parallel Library in Control*); los principales trabajos relacionados a este proceso se resumen a continuación.

El trabajo de Blanquer et al. [37] presenta algunas propuestas tendentes a paralelizar la resolución de la ecuación de Lyapunov proponiendo PSLICOT (*Parallel* SLICOT). Posteriormente, uno de los principales esfuerzos se vio plasmado en el desarrollo de la primera versión de la biblioteca PLiC [33] propiamente dicho. La biblioteca PLiC posee dos componentes fundamentales, PLiCOC y PLiCMR (del inglés *Parallel Library in Control: Model Reduction*). La primera da soporte a la resolución de problemas de control mientras que la segunda componente está especialmente diseñada para la resolución de problemas de reducción de modelos. PLiC incluye rutinas para la resolución de problemas de análisis y diseño de SDLs invariantes en el tiempo y de gran escala sobre arquitecturas de computadoras distribuidas y paralelas. Contiene más de 30 subrutinas que permiten resolver ecuaciones matriciales lineales y cuadráticas (Lyapunov, Sylvester, Stein y la ecuación algebraica de Riccati), reducción de modelos, estabilización parcial, y diversas rutinas de soporte. El modelo de paralelismo emula el utilizado por la biblioteca SCALAPACK. En particular, la resolución de las operaciones de álgebra se hacen invocando a rutinas de SCALAPACK,

[2] Disponible en `http://www.win.tue.nl/niconet/NIC2/slicot.html`.

siempre que es posible, y las comunicaciones se efectúan mediante invocación a rutinas de la biblioteca BLACS.

Otra extensión a la biblioteca PLiC fue el desarrollo de una versión accesible mediante servicios web. Esta propuesta permite resolver problemas de reducción de modelos en sistemas remotos a través de la red, eliminando la necesidad de contar con plataformas de HPC locales para la resolución de este tipo de problemas. En el trabajo de Benner et al. [30] se puede profundizar en la temática. Con el mismo objetivo que el trabajo anterior, facilitar el uso de la biblioteca PLiC, en el trabajo de Galiano et al. [100] se presenta el paquete PyPLiC. En él, se ofrece una interfaz de alto nivel en el lenguaje de *scripting* Python para la biblioteca PLiCMR.

En problemas con modelos dispersos o en presencia de matrices estructuradas banda en los SDL, los métodos LR-ADI [61, 88, 105] son particularmente eficientes. En el trabajo de Remón et al. [91], se aplican diferentes técnicas de paralelismo al método. Los autores presentan dos implementaciones paralelas sobre memoria compartida (arquitecturas SMP) para la factorización LU de matrices de banda. En el trabajo se muestra también cómo aplicar las variantes de factorización desarrolladas para acelerar los métodos LR-ADI en la resolución de problemas de reducción de modelos, consiguiendo porcentajes de mejora superiores al $50\,\%$.

En el caso de los métodos de reducción de modelos basados en sub-espacios de Krylov, existe gran caudal de trabajos referentes a la aplicación de técnicas de HPC [20, 94]. Generalmente, dada la estrategia de resolución de estos métodos, las técnicas de HPC se aplican a la resolución del sistema, es decir el método de Lanczos o alguna de sus variantes.

5. Resumen

En el trabajo se presentan algunos conceptos básicos del problema de reducción de modelos, y en particular, el método BT. Este método precisa, como etapa fundamental desde el punto de vista computacional, la resolución de un par de ecuaciones acopladas de Lyapunov. También se extiende el estudio para abordar de forma primaria el método BST, donde la principal etapa de cómputo es la resolución de una ecuación algebraica de Riccati (EAR), que se puede tratar mediante un algoritmo iterativo que resuelve una ecuación de Lyapunov por paso, o bien mediante el método de la función signo.

La herramienta principal para la resolución de estas ecuaciones matriciales es la iteración de Newton para la función signo, que se basa en el cálculo de operaciones básicas de ALN (fundamentalmente inversión de matrices) de gran exigencia computacional. Por lo tanto, cuando las ecuaciones a resolver son de gran dimensión, la ejecución de estas operaciones precisa del uso de estrategias de computación de alto desempeño.

En la Sección 3 se ofrece una pequeña introducción a las técnicas de programación paralela y las arquitecturas de HPC. Mientras que en la Sección 4 se relevan los esfuerzos tendentes a aplicar las técnicas de HPC para la aceleración de la resolución de los problemas de reducción de modelos, constatando que los principales esfuerzos se nuclean entorno al proyecto NICONET.

Referencias

[1] Repositorio Netlib. `www.netlib.org/`. Consultado en abril 2011.

[2] Repositorio Netlib, biblioteca LAPack. `www.netlib.org/lapack/`. Consultado en abril 2011.

[3] Repositorio Netlib, biblioteca sparse BLAS. `www.netlib.org/sparse-blas/`. Consultado en abril 2011.

[4] Sitio web de benchmarks. `www.netlib.org/benchmark/`. Consultado en abril 2011.

[5] Sitio web de MPICH. `www-unix.mcs.anl.gov/mpi/mpich/`. Consultado en abril 2011.

[6] Sitio web de OpenMP. `www.openmp.org/drupal/`. Consultado en abril 2011.

[7] Sitio web del ANL. `www.anl.gov/`. Consultado en abril 2011.

[8] Sitio web del proyecto the Numerics in Control Network NICONET. `www.icm.tu-bs.de/NICONET/`. Consultado en abril 2011.

[9] Sito web de NIST Sparse BLAS. `http://math.nist.gov/spblas/`. Consultado en abril 2011.

[10] J. Aguilar and E. Leiss. *Introducción a la computción paralela*. Editorial: Universidad de los Andes. Mérida – Venezuela, 2004.

[11] G. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS*, pages 483–485, 1967.

[12] B.D.O. Anderson. An algebraic solution to the spectral factorization problem. *IEEE Trans. Automat. Control*, AC-12:410–414, 1967.

[13] B.D.O. Anderson. A system theory criterion for positive real matrices. *SIAM J. Cont.*, 5:171–182, 1967.

[14] A.C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM Publications, Philadelphia, PA, 2005.

[15] A.C. Antoulas and D.C. Sorensen. Approximation of large-scale dynamical systems: An overview. *Int. J. Appl. Math. Comp. Sci.*, 11(5):1093–1121, 2001.

[16] A.C. Antoulas, D.C. Sorensen, and Y. Zhou. On the decay rate of Hankel singular values and related issues. *Sys. Control Lett.*, 46(5):323–342, 2002.

[17] W.F. Arnold, III and A.J. Laub. Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proc. IEEE*, 72:1746–1754, 1984.

[18] Z. Bai and J. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In R.F. Sincovec et al., editor, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 391–398. SIAM, Philadelphia, PA, 1993. *See also:* Tech. Report CSD-92-718, Computer Science Division, University of California, Berkeley, CA 94720.

[19] Z. Bai and J. Demmel. Using the matrix sign function to compute invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 19(1):205–225, 1998.

[20] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[21] R.H. Bartels and G.W. Stewart. Solution of the matrix equation $AX + XB = C$: Algorithm 432. *Comm. ACM*, 15:820–826, 1972.

[22] A. N. Beavers and E. D. Denman. A new solution method for the Lyapunov matrix equations. *SIAM J. Appl. Math.*, 29:416–421, 1975.

[23] A.N. Beavers and E.D. Denman. A new solution method for quadratic matrix equations. *Mathematical Biosciences*, 20:135–143, 1974.

[24] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, pages 11–14, 1995.

[25] P. Benner. *Contributions to the Numerical Solution of Algebraic Riccati Equations and Related Eigenvalue Problems*. Logos–Verlag, Berlin, Germany, 1997. *Also:* Dissertation, Fakultät für Mathematik, TU Chemnitz–Zwickau, 1997.

[26] P. Benner. Accelerating Newton's method for discrete-time algebraic Riccati equations. In A. Beghi, L. Finesso, and G. Picci, editors, *Mathematical Theory of Networks and Systems*, pages 569–572, Il Poligrafo, Padova, Italy, 1998.

[27] P. Benner. Symplectic balancing of Hamiltonian matrices. *SIAM J. Sci. Comput.*, 22(5):1885–1904, 2001.

[28] P. Benner, R. Byers, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search. *Parallel Comput.*, 26(10):1345–1368, 2000.

[29] P. Benner, J.M. Claver, and E.S. Quintana-Ortí. Efficient solution of coupled Lyapunov equations via matrix sign function iteration. In A. Dourado et al., editor, *Proc. 3^{rd} Portuguese Conf. on Automatic Control CONTROLO'98,* Coimbra, pages 205–210, 1998.

[30] P. Benner, R. Mayo, E. S. Quintana-Ortí, and G. Quintana-Ortí. Enhanced services for remote model reduction of large-scale dense linear

systems. In Juha Fagerholm, Juha Haataja, Jari Järvinen, Mikko Lyly, Peter Råback, and Ville Savolainen, editors, *PARA*, volume 2367 of *Lecture Notes in Computer Science*, pages 329–338. Springer, 2002.

[31] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT - a subroutine library in systems and control theory. In *Applied and Computational Control, Signals, and Circuits*, pages 499–539. Birkhäuser, 1997.

[32] P. Benner and E. S. Quintana-ortí. Model reduction based on spectral projection methods. In *Dimension Reduction of Large-Scale Systems*, pages 5–45. Springer-Verlag, 2005.

[33] P. Benner, E. S. Quintana-Ortí, and G. Quintana-Ortí. A portable subroutine library for solving linear control problems on distributed memory computers. In *Workshop on Wide Area Networks and High Performance Computing*, pages 61–87, London, UK, 1999. Springer-Verlag.

[34] P. Benner and E.S. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. *Numer. Algorithms*, 20(1):75–100, 1999.

[35] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Balanced truncation model reduction of large-scale dense systems on parallel computers. *Math. Comput. Model. Dyn. Syst.*, 6(4):383–405, 2000.

[36] P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Solving linear matrix equations via rational iterative schemes. Technical Report SFB393/04-08, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, FRG, 2004. Available from `http://www.tu-chemnitz.de/sfb393/preprints.html`.

[37] I. Blanquer, D. Guerrero, V. Hernández, E. S. Quintana-Ortí, and P. A. Ruiz. Parallel-SLICOT implementation and documentation standards. Technical report, SLICOT Working Note, 1998.

[38] R. Byers. Solving the algebraic Riccati equation with the matrix sign function. *Linear Algebra Appl.*, 85:267–279, 1987.

[39] R. Byers, C. He, and V. Mehrmann. The matrix sign function method and the computation of invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 18(3):615–632, 1997.

[40] J. Choi, J. Dongarra, and D. Walker. PB-BLAS: A set of parallel block basic linear algebra subprograms. In *Proc. of the 1994 Scalable High Performance Computing Conference, IEEE Computer Society Press*, 1994.

[41] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Comput. Soc. Press, 1992.

[42] J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. Prospectus for the development of a linear algebra library for high-performance computers. Technical Report ANL/MCS-TM-97, 9700 South Cass Avenue, Argonne, IL 60439-4801, USA, 1987.

[43] E.D. Denman and A.N. Beavers. The matrix sign function and computations in systems. *Appl. Math. Comput.*, 2:63–94, 1976.

[44] U.B. Desai and D. Pal. A transformation approach to stochastic model reduction. *IEEE Trans. Automat. Control*, AC–29:1097–1100, 1984.

[45] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990.

[46] J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5):403–407, July 1987.

[47] M. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers*, 21:948–960, 1972.

[48] R. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 12:267–319, 2003.

[49] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, Ralph H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[50] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, USA, 1994.

[51] K. Glover. All optimal Hankel-norm approximations of linear multivariable systems and their L^∞ norms. *Internat. J. Control*, 39:1115–1193, 1984.

[52] G. H. Golub, S. Nash, and C. F. Van Loan. A Hessenberg–Schur method for the problem $AX + XB = C$. *IEEE Trans. Automat. Control*, AC-24:909–913, 1979.

[53] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition, 1996.

[54] K. Goto and R. A. van de Geijn. On reducing TLB misses in matrix multiplication. Technical Report CS-TR-02-55, Department of Computer Sciences, The University of Texas at Austin, 2002.

[55] L. Grasedyck. Existence of a low rank or H-matrix approximant to the solution of a Sylvester equation. *Numer. Lin. Alg. Appl.*, 11:371–389, 2004.

[56] M. Green. Balanced stochastic realization. *Linear Algebra Appl.*, 98:211–247, 1988.

[57] M. Green. A relative error bound for balanced stochastic truncation. *IEEE Trans. Automat. Control*, AC-33(10):961–965, 1988.

[58] W. Gropp and E. Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[59] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, sep 1996.

[60] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 1994.

[61] S. Gugercin, D.C. Sorensen, and A.C. Antoulas. A modified low-rank Smith method for large-scale Lyapunov equations. *Numer. Algorithms*, 32(1):27–55, 2003.

[62] J. Gustafson and Q. Snell. Hint: A new way to measure computer performance. In *HICSS (2)*, pages 392–401, 1995.

[63] S.J. Hammarling. Numerical solution of the stable, non-negative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2:303–323, 1982.

[64] R. Hanson, F. Krogh, and L. Lawson. A proposal for standard linear algebra subprograms. *Jet Propulsion Lab., Pasadena, Calif. TM 33-660*, 5(3), 1973.

[65] G.A. Hewer. An iterative technique for the computation of steady state gains for the discrete optimal regulator. *IEEE Trans. Automat. Control*, AC-16:382–384, 1971.

[66] N.J. Higham. Computing the polar decomposition—with applications. *SIAM J. Sci. Statist. Comput.*, 7:1160–1174, 1986.

[67] M. Hochbruck and G. Starke. Preconditioned Krylov subspace methods for Lyapunov matrix equations. *SIAM J. Matrix Anal. Appl.*, 16(1):156–171, 1995. (See also: IPS Research Report 92–17, ETH Zürich, Switzerland (1992)).

[68] A.S. Hodel, B. Tenison, and K.R. Poolla. Numerical solution of the Lyapunov equation by approximate power iteration. *Linear Algebra Appl.*, 236:205–230, 1996.

[69] W.D. Hoskins, D.S. Meek, and D.J. Walton. The numerical solution of $A'Q + QA = -C$. *IEEE Trans. Automat. Control*, AC-22:882–883, 1977.

[70] Intel Corporation., `http://www.intel.com/`. Consultado en abril 2011.

[71] I.M. Jaimoukha and E.M. Kasenally. Krylov subspace methods for solving large Lyapunov equations. *SIAM J. Numer. Anal.*, 31:227–251, 1994.

[72] T. Kailath. *Systems Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[73] C. Kenney and A.J. Laub. On scaling Newton's method for polar decomposition and the matrix sign function. *SIAM J. Matrix Anal. Appl.*, 13:688–706, 1992.

[74] C. Kenney and A.J. Laub. The matrix sign function. *IEEE Trans. Automat. Control*, 40(8):1330–1348, 1995.

[75] M.M. Konstantinov, V. Mehrmann, and P.Hr. Petkov. Perturbation analysis for the Hamiltonian Schur form. *SIAM J. Matrix Anal. Appl.*, 23(2):387–424, 2001.

[76] P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, Orlando, 2nd edition, 1985.

[77] W. Lang and U. Lezius. Numerical realization of the balanced reduction of a control problem. In H. Neunzert, editor, *Progress in Industrial Mathematics at ECMI94*, pages 504–512. John Wiley & Sons Ltd and B.G. Teubner, New York and Leipzig, 1996.

[78] V.B. Larin and F.A. Aliev. Construction of square root factor for solution of the Lyapunov matrix equation. *Sys. Control Lett.*, 20:109–112, 1993.

[79] A.J. Laub. Numerical linear algebra aspects of control design computations. *IEEE Trans. Automat. Control*, AC-30:97–108, 1985.

[80] A.J. Laub, M.T. Heath, C.C. Paige, and R.C. Ward. Computation of system balancing transformations and other application of simultaneous diagonalization algorithms. *IEEE Trans. Automat. Control*, 34:115–122, 1987.

[81] J.-R. Li and J. White. Reduction of large circuit models via low rank approximate gramians. *Int. J. Appl. Math. Comp. Sci.*, 11(5):1151–1171, 2001.

[82] Y. Liu and B.D.O. Anderson. Controller reduction via stable factorization and balancing. *Internat. J. Control*, 44:507–531, 1986.

[83] B.C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Trans. Automat. Control*, AC-26:17–32, 1981.

[84] C. Mullis and R.A. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. *IEEE Trans. Circuits and Systems*, CAS-23(9):551–562, 1976.

[85] C. Narus, M. Narus, L. Dagum, D. Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2000.

[86] G. Obinata and B.D.O. Anderson. *Model Reduction for Control System Design*. Communications and Control Engineering Series. Springer-Verlag, London, UK, 2001.

[87] G. Quintana P. Benner, E. S. Quintana. Solving linear-quadratic optimal control problems on parallel computers. *Optimization Methods & Software*, 23(6):879–909, 2008.

[88] T. Penzl. A cyclic low rank Smith method for large, sparse Lyapunov equations with applications in model reduction and optimal control. Technical Report SFB393/98-6, Fakultät für Mathematik, TU Chemnitz, 09107 Chemnitz, FRG, 1998. Available from `http://www.tu-chemnitz.de/sfb393/sfb98pr.html`.

[89] T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 2000.

[90] P.H. Petkov, N.D. Christov, and M.M. Konstantinov. *Computational Methods for Linear Control Systems*. Prentice-Hall, Hertfordshire, UK, 1991.

[91] A. Remón, E. Quintana-Ortí, and G. Quintana-Ortí. Parallel solution of band linear systems in model reduction. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 4967 of *Lecture Notes in Computer Science*, pages 678–687. Springer Berlin / Heidelberg, 2008.

[92] J.D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32:677–687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).

[93] Y. Saad. Numerical solution of large Lyapunov equation. In M. A. Kaashoek, J. H. van Schuppen, and A. C. M. Ran, editors, *Signal Processing, Scattering, Operator Theory and Numerical Methods*, pages 503–511. Birkhäuser, 1990.

[94] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[95] J. Saak. Effiziente numerische Lösung eines Optimalsteuerungsproblems für die Abkühlung von stahlprofilen. Diplomarbeit, Fachbereich 3/Mathematik und Informatik, Universität Bremen, D-28334 Bremen, September 2003.

[96] M.G. Safonov and R.Y. Chiang. A Schur method for balanced-truncation model reduction. *IEEE Trans. Automat. Control*, AC–34:729–733, 1989.

[97] S. Sahni and V. Thanvantri. Parallel computing: Performance metrics and models, 1995. Research Report, Computer Science Department, University of Florida.

[98] G. Thomas. BLASTH, a BLAS library for dual SMP computer. In *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, pages 24–24, Berkeley, CA, USA, 2000. USENIX Association.

[99] M.S. Tombs and I. Postlethwaite. Truncated balanced realization of a stable non-minimal state-space system. *Internat. J. Control*, 46(4):1319–1330, 1987.

[100] H. Migallón V. Migallón J. Penadés V. Galiano, A. Martín and E.S. Quintana-Ortí. PyPLiC: A high-level interface to the parallel model reduction library PLiCMR. In B. H. V. Topping, editor, *Proceedings of the*

Eleventh International Conference on Civil, Structural and Environmental Engineering Computing, Stirlingshire, United Kingdom, 2007. Civil-Comp Press. paper 62.

[101] P. Van Dooren. Gramian based model reduction of large-scale dynamical systems. In D.F. Griffiths and G.A. Watson, editors, *Numerical Analysis 1999. Proc. 18th Dundee Biennial Conference on Numerical Analysis*, pages 231–247, London, UK, 2000. Chapman & Hall/CRC.

[102] A. Varga. Efficient minimal realization procedure based on balancing. In *Prepr. of the IMACS Symp. on Modelling and Control of Technological Systems*, volume 2, pages 42–47, 1991.

[103] A. Varga. Task II.B.1 – selection of software for controller reduction. SLICOT Working Note 1999–18, The Working Group on Software (WGS), December 1999. Available from `http://www.win.tue.nl/niconet/NIC2/reports.html`.

[104] A. Varga and K. H. Fasol. A new square–root balancing–free stochastic truncation model reduction algorithm. In *Prepr. 12th IFAC World Congress*, volume 7, pages 153–156, Sydney, Australia, 1993.

[105] E.L. Wachspress. ADI iteration parameters for the Sylvester equation, 2000. available from the author.