UNIVERSIDAD EAFIT

Abierta a la investigación

UNA INTRODUCCIÓN AL USO DE LAPACK

CARLOS E. MEJÍA
TOMÁS RESTREPO
CHRISTIAN TREFFTZ

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

MEDELLÍN, Agosto de 2002

Favor enviar sus comentarios a la dirección electrónica cemejia@perseus.unalmed.edu.co

T A B L A D E C O N T E N I D O

RESUMEN

El software LAPACK (Linear Algebra Package) es una colección de subrutinas escritas en FORTRAN 77 que sirve para resolver los problemas matemáticos más comunes que surgen a partir del modelamiento matemático y que se enmarcan en el campo de álgebra lineal numérica. Las subrutinas de LAPACK se basan en unas subrutinas más sencillas que se conocen por la sigla BLAS (Basic Linear Algebra Subprograms). En estas notas, presentamos a LAPACK y al álgebra lineal numérica, destacamos las rutinas más comunes de LAPACK, ilustramos el uso de las rutinas con ejemplos seleccionados y todo lo hacemos en un tono motivador hacia el uso de software de calidad disponible en el dominio público.

ABSTRACT

LAPACK (Linear Algebra Package) is a collection of FORTRAN 77 subroutines that are useful in the solution of the mathematical problems related to mathematical modeling and belonging to the field of numerical linear algebra. The LAPACK subroutines are based on simpler subroutines called BLAS (Basic Linear Algebra Subprograms). In these notes, we introduce LAPACK and numerical linear algebra, explain in detail the more common LAPACK subroutines, illustrate their use by selected examples and all the time we encourage the readers to use quality software available in the public domain.

SOBRE LOS AUTORES

CARLOS ENRIQUE MEJÍA
Ph.D. Matemáticas, University of Cincinnati, Cincinnati, Ohio, E.U. Profesor asociado, Universidad Nacional de Colombia, Medellín.

TOMÁS RESTREPO
Ingeniero de Sistemas, Universidad EAFIT.

CHRISTIAN TREFFTZ
Ph.D. Ciencias de la Computación, Michigan State University, East Lansing, Michigan, E.U. Profesor de la Universidad EAFIT por varios años. Actualmente profesor de Grand Valley State University, Allendale, Michigan, E.U.

Los autores expresamos nuestro reconocimiento a la Universidad EAFIT, por ofrecer todos los recursos que necesitamos durante la realización de esta investigación. Este documento surgió del trabajo de seminario que sostuvimos los autores por un período de un año mientras el tercer autor todavía era profesor en esta Universidad.

El primer autor agradece el apoyo recibido de DINAIN (Proyecto DI00C1236) y COLCIENCIAS (Grupo de Investigación en Matemáticas y Doctorado en Matemáticas, Universidad Nacional).

1

Introducción

El modelamiento matemático ha sido siempre fundamental para las ciencias y las ingenierías pero ha cobrado mucha más importancia desde que se empezaron a usar computadores en gran escala. LAPACK (Linear Algebra Package) es una colección de subrutinas escritas en FORTRAN 77 para resolver los problemas matemáticos más comunes que surgen a partir del modelamiento y que se enmarcan en el campo del álgebra lineal numérica. El desarrollo de LAPACK es una tarea que se empezó hacia 1987 y aun no termina. Lo hace, desde universidades y laboratorios de investigación de Estados Unidos y Europa, un equipo de investigadores de primera línea apoyados financieramente por varias instituciones entre las que se destaca la Fundación Nacional de Ciencias de Estados Unidos (National Science Foundation, NSF).

Las subrutinas de LAPACK [1] se basan en llamadas a unas subrutinas más sencillas que se conocen por la sigla BLAS (Basic Linear Algebra Subprograms). Hasta el momento se dispone de subprogramas BLAS de tres niveles: Nivel 1, publicado en 1979, que se encarga de operaciones de vector con vector. Ejemplo: Subrutina SAXPY, que sirve para sumar un múltiplo de un vector con otro vector. Nivel 2, publicado en 1988, que se ocupa de operaciones de matriz con vector. Ejemplo: Subrutina SGEMV, que sirve para sumar el producto de un múltiplo de una matriz por un vector, con un múltiplo de otro vector. Nivel 3, publicado en 1990, que se dedica a operaciones entre matrices. Ejemplo: Subrutina SGEMM, que sirve para sumar un múltiplo de una matriz con un múltiplo del producto de otras dos matrices.

La construcción de LAPACK con base en los subprogramas BLAS genera un alto nivel de estandarización, proporciona gran rapidez de cálculo y hace más fácil hacer un seguimiento cuando se presentan dificultades. La utilización de LAPACK es universal, no solo directamente, sino también como motor de cálculo en software con interfase gráfica como OCTAVE, <http://www.octave.org>, desarrollado en la Universidad de Wisconsin.

LAPACK incluye rutinas para resolver sistemas de ecuaciones lineales, sistemas de ecuaciones lineales por mínimos cuadrados, problemas de valores propios y problemas de valores singulares. Se trata pues, de un software de primera calidad, que no requiere de pago alguno para ser utilizado y que sirve para el tratamiento numérico de los problemas que se encuentran por igual investigadores,

profesores y estudiantes en una gran diversidad de disciplinas científicas y técnicas. Las preguntas que surgen son:

¿Por qué motivo LAPACK es tan desconocido en Colombia?

¿Por qué razón es frecuente que en Colombia se recurra a software de dudosa calidad o a software de calidad comparable a la de LAPACK pero de alto precio?

Para evitar que nos tengamos que seguir haciendo estas preguntas, nos propusimos divulgar, para una audiencia bastante general, la existencia de LAPACK, la forma de obtenerlo y de instalarlo y una revisión somera de su contenido y su alcance. Este documento y el artículo [2] son el resultado. Lo dirigimos a personas acostumbradas a usar un computador, que conocen rudimentos de FORTRAN y que están familiarizadas con las nociones básicas del álgebra lineal.

Aunque no es indispensable, es recomendable que la máquina en la que se trabaje LAPACK tenga a LINUX como sistema operativo. El sistema operativo LINUX, que es de dominio público, puede instalarse fácilmente en microcomputadores con procesador Intel o compatible, que son la mayoría entre nosotros. Lo más común, es que toda instalación de LINUX incluya compiladores de dominio público de varios lenguajes de programación, incluyendo FORTRAN. Asi que, en principio, estamos ante una plataforma de cálculo numérico de primera calidad basada en software no comercial. Una de las distribuciones más bien mantenidas de LINUX es REDHAT LINUX, con dirección Internet

$$\mathrm{http://www.redhat.com.}$$

Estas notas están organizadas de la siguiente forma: En lo que resta de esta introducción, nos dedicamos a explicar las formas de obtener e instalar LAPACK y a la convención que se sigue para los nombres de las subrutinas. En el capítulo 2 sobre álgebra lineal numérica, explicamos procesos de modelamiento que conducen a problemas de álgebra lineal susceptibles de ser resueltos con LAPACK y ofrecemos algunas definiciones importantes. Los capítulos 3, 4 y 5, los dedicamos a considerar en detalle las tres clases de problemas principales que se pueden resolver con LAPACK, es decir, sistemas de ecuaciones lineales, problemas de valores y vectores propios y descomposición en valores singulares. En el capítulo 6, de naturaleza más técnica, consideramos aspectos relacionados con almacenamiento de matrices y uso de la memoria.

1.1 Instalación de LAPACK

1. **Descarga**: Si el LINUX de su máquina trae LAPACK como uno de los paquetes para instalación opcional, siga las instrucciones de instalación de dichos paquetes y tendrá LAPACK y

las BLAS funcionando bien en muy corto tiempo. Si no dispone de LAPACK como paquete opcional o si desea obtenerlo desde la fuente por si ha tenido algunas mejoras recientes, el principal sitio del que se puede obtener LAPACK es NetLib, mantenido por la Universidad de Tennessee y ORNL (Oak Ridge National Laboratory). La página oficial es

$$http://www.netlib.org/lapack/index.html$$

Al momento de escribir este texto, la última versión de la colección es la 3.0.

Hay tres archivos principales con el código fuente:

- lapack.tgz : contiene la distribución completa de LAPACK 3.0 en un archivo comprimido usando tar y gzip.
- lapack-pc.zip : archivo en formato zip con la versióon 3.0 de LAPACK para Microsoft Windows. Requiere de la rutina NMAKE de Microsoft y el compilador FORTRAN de Digital Equipment Corporation, distribuído ahora por la propia Microsoft.
- lapack-pc-wfc.zip : archivo en formato zip con la version 3.0 de LAPACK para Microsoft Windows. Requiere de la rutina NMAKE de Microsoft y el compilador FORTRAN 77/32 de Watcom version 11.0.

Adicionalmente es posible descargar las colecciones precompiladas, así como algunas de las rutinas individuales. Para estas notas suponemos que se desea instalar LAPACK para LINUX, por lo que el archivo correcto que se debe descargar es lapack.tgz.

2. **Compilación** Lo primero que se necesita hacer es descomprimir el archivo de la distribución. Esto se realiza usando los comandos:

   gunzip lapack.tgz

   tar -xvf lapack.tar

   Para compilar, cambiamos al directorio ./LAPACK/INSTALL. Desde aquí podemos configurar nuestra instalación de LAPACK para obtener las opciones correctas para la compilación. Lo primero es usar el archivo de configuracion apropiado para LINUX

   cp make.inc.LINUX ../make.inc

   Antes de poder compilar LAPACK propiamente dicho, debemos compilar las BLAS:

   cd ../BLAS/SRC

make

Si éstas compilan sin ningún problema, podemos compilar la colección y los programas de prueba:

cd ../../

make

Lo primero que la compilación realiza es compilar unas cortas pruebas de desempeño que se encuentran en el directorio INSTALL, y las ejecuta, mostrando su salida en pantalla. Luego, procede a compilar la colección en si misma, cuyo código fuente se encuentra en el directorio SRC. Luego, procederá a compilar los programas de prueba. Tenga en cuenta que la compilación puede tardar bastante tiempo, así que prepárese a esperar un buen rato.

Si no está interesado en los programas de prueba, y solo desea compilar la colección, ejecute make directamente desde el directorio SRC.

1.2 Convenciones para los nombres

LAPACK está escrita en Fortran. Posiblemente debido a que los antiguos compiladores de FORTRAN exigían que los nombres de los programas tuvieran máximo 6 letras, los diseñadores de LAPACK adoptaron los siguientes formatos para los nombres de las rutinas:

XYYZZ

XYYZZZ

donde la primera letra, X, indica el tipo de datos sobre el que opera la rutina. Las convenciones utilizadas para esta primera letra son:

- S : denota precisión sencilla y números reales
- C : denota precisión sencilla y números complejos
- D : denota precisión doble y números reales
- Z : denota precisión doble y números complejos

Las siguientes dos letras, YY, describen el tipo de matriz sobre el que opera la rutina. En la siguiente tabla se describen los códigos y los tipos de matrices correspondientes:

BD	Bidiagonal
DI	Diagonal
GB	General definida por bandas
GE	General (puede ser rectangular)
GG	Matrices generales para un problema generalizado
GT	General Tridiagonal
HB	Hermitiana (compleja) definida por bandas
HE	Hermitiana (compleja)
HG	Matriz de Hessenberg superior (problema generalizado)
HP	Hermitiana (compleja) con almacenamiento compacto
HS	Matriz de Hessenberg superior
OP	Ortogonal (real) con almacenamiento compacto
OR	Ortogonal (real)
PB	Simétrica o Hermitiana definida positiva (por bandas)
PO	Simétrica o Hermitiana definida positiva
PP	Simétrica o Hermitiana definida positiva (compacto)
PT	Simétrica o Hermitiana definida positiva tridiagonal
SB	(Real) Simétrica definida por bandas
SP	(Real) Simétrica con almacenamiento compacto
ST	(Real) Simétrica tridiagonal
SY	Simétrica
TB	Triangular definida por bandas
TG	Matrices Triangulares de un problema generalizado
TP	Triangular con almacenamiento compacto
TR	Triangular
TZ	Trapezoidal
UN	(Compleja) Unitaria
UP	(Compleja) Unitaria, con almacenamiento compacto

Las últimas dos o tres letras indican el cálculo que se realiza. Por ejemplo, SGEBRD es una rutina de precisión sencilla que realiza una reducción bidiagonal (BRD) sobre una matriz real general y SGESV es una rutina de precisión sencilla que resuelve un sistema de ecuaciones con matriz real general por factorización LU. La lista completa de rutinas puede verse en [1].

2

Algebra Lineal Numérica

2.1 Introducción

La teoría matemática necesaria para emprender con éxito la solución numérica de problemas de álgebra lineal tiene un poco de álgebra, análisis, topología y posiblemente de otras ramas de la matemática. Toda esa teoría se denomina actualmente Algebra Lineal Numérica (ALN). Se trata de un campo muy activo de la matemática y uno de los que más aplicaciones tiene. Su desarrollo se aceleró enormemente a partir de 1950, por la aparición y desarrollo de los computadores. En este capítulo presentamos una introducción a este fascinante mundo con los siguientes objetivos en mente:

1. Enunciar definiciones y resultados centrales.
2. Resaltar dificultades típicas en el cálculo numérico, generalmente relacionadas con problemas mal condicionados.
3. Justificar la necesidad de herramientas sólidas de cálculo como LAPACK.
4. Esbozar la idea central de discretización, que sirve para pasar de un problema analítico en un espacio infinito dimensional, a un problema finito dimensional de álgebra lineal.

Hay dos grandes familias de problemas en ALN que se considerarán brevemente en estas notas

1. Sistemas de ecuaciones algebraicas de la forma $Ax = b$, con A matriz y x, b vectores. Aqui se incluyen las soluciones por mínimos cuadrados.
2. Problemas de valores y vectores propios de la forma $Ax = \lambda x$, con A matriz, x vector y λ escalar (real o complejo).

En la siguiente sección nos ocupamos de motivar el estudio del ALN y en las siguientes, nos concentramos en las definiciones y resultados que queremos resaltar. Ultimamente han aparecido buenos libros sobre el tema como [3] y [4] que se unen a libros ya clásicos como [5] y [6]. Un libro de nivel intermedio y muy completo que recomendamos es [7].

2.2 Motivación y ejemplos

Para dar una idea concreta acerca de este tema de estudio, consideramos diversos ejemplos que responden a los objetivos propuestos arriba y que sirven de motivación, no solo para el estudio somero de los conceptos y métodos que ofrecemos enseguida sino también para profundizar en algunos de ellos.

1. El primer ejemplo sirve para precisar la idea de discretización. Consideremos la ecuación diferencial ordinaria de segundo orden

$$y''(t) = f(t, y, y'), \qquad a \leq t \leq b$$

junto con condiciones de borde

$$y(a) = A, \qquad y(b) = B.$$

La función f puede depender de forma lineal o no lineal de sus variables segunda y tercera. Son muchos los problemas de la física y la ingeniería que admiten un modelo matemático de este estilo para su solución. Basta pensar, por ejemplo, en la segunda ley de Newton de la que surgen un gran número. Una posible discretización de diferencias finitas para este problema es como sigue: Se genera una subdivisión en $n+1$ subintervalos de igual longitud $h = \frac{b-a}{n+1}$ del intervalo $[a, b]$. Los puntos extremos de los subintervalos los denotamos $a = t_0 < t_1 < ... < t_{n+1} = b$. Enseguida para $j = 1, 2, ..., n$, aproximamos la segunda derivada $y''(t_j)$ por el cociente

$$\frac{1}{h^2}\left(y(t_{j-1}) - 2y(t_j) + y(t_{j+1})\right)$$

y entonces resulta natural querer resolver el sistema

$$\frac{1}{h^2}\left(v_{j-1} - 2v_j + v_{j+1}\right) = f\left(t_j, v_j, \frac{v_{j+1} - v_{j-1}}{2h}\right), \qquad j = 1, 2, ..., n$$

con $v_0 = A$, $v_{n+1} = B$ y v_j para $j = 1, ..., n$ aproximaciones de los valores $y(t_j)$ utilizados para los cálculos. Tenemos entonces un sistema, en general no lineal, de n ecuaciones con n incógnitas $v_1, v_2, ..., v_n$ que generalmente se resuelve por un método iterativo de tipo Newton. Expandido, el sistema es el siguiente:

$$\begin{array}{rcl}
2v_1 - v_2 - A + h^2 f(t_1, v_1, \frac{v_2 - A}{2h}) & = & 0 \\
-v_1 + 2v_2 - v_3 + h^2 f(t_2, v_2, \frac{v_3 - v_1}{2h}) & = & 0 \\
\vdots & & \vdots \\
-v_{n-2} + 2v_{n-1} - v_n + h^2 f\left(t_{n-1}, v_{n-1}, \frac{v_n - v_{n-2}}{2h}\right) & = & 0 \\
-v_{n-1} + 2v_n - B + h^2 f\left(t_n, v_n, \frac{B - v_{n-1}}{2h}\right) & = & 0.
\end{array} \tag{2.1}$$

Suponemos que este sistema tiene solución para preservar sencilla esta exposición. Advertimos sin embargo que estudiar condiciones suficientes o necesarias para que haya solución a sistemas no lineales es un tema de investigación actual en análisis numérico que está lleno de sutilezas y dificultades.

El método de Newton para nuestro sistema se enuncia más fácilmente con notación vectorial. Denotamos por V al vector columna cuyas componentes son $v_1, v_2, \ldots, v_n$ y definimos una función de variable y valor vectorial G de la siguiente manera:

$$\begin{array}{rccc} G: & \mathbb{R}^n & \to & \mathbb{R}^n \\ & V & \longmapsto & GV \end{array}$$

con la componente j–ésima de GV igual al lado izquierdo de la ecuación j–ésima en (2.1). De esta forma, el sistema que nos ocupa se puede denotar simplemente como $GV = 0$. Su solución por el método iterativo de Newton exige en cada iteración, la solución de un sistema lineal de ecuaciones. Más precisamente, se procede asi:

$$\begin{array}{ll} V^{(0)} & \text{aproximación inicial,} \\ V^{(k+1)} & = V^{(k)} + W, \end{array}$$

donde el superíndice indica orden de iteración y W es la solución del sistema de ecuaciones

$$JW = -GV^{(k)},$$

donde la matriz J es el jacobiano de G en el punto $V^{(k)} = (v_1, v_2, \ldots, v_n)^T$ en el cual omitimos los superíndices para no recargar la exposición. Para cada iteración, J es la matriz tridiagonal dada por

$$\begin{aligned} J_{i-1,i} &= -1 - \frac{h}{2} f_{y'}\left(t_i, v_i, \frac{v_{i+1} - v_{i-1}}{2h}\right) \\ J_{ii} &= 2 + h^2 f_y\left(t_i, v_i, \frac{v_{i+1} - v_{i-1}}{2h}\right) \\ J_{i,i+1} &= -1 + \frac{h}{2} f_{y'}\left(t_i, v_i, \frac{v_{i+1} - v_{i-1}}{2h}\right), \end{aligned}$$

para $i = 1, 2, \ldots, n$ y haciendo $v_0 = A$ y $v_{n+1} = B$.

El algoritmo numérico debe incluir un número máximo de iteraciones como criterio de parada y algún otro criterio de parada basado en lo cercanos que son iterados consecutivos. Esta noción de cercanía es una noción topológica y la precisaremos más adelante.

En resumen: este ejemplo presenta un problema con valores en la frontera en dos puntos, posiblemente no lineal, que generalmente debe tratarse por un método iterativo para su solución. Cuando se enuncia un método de Newton, que es uno de los más comunes y poderosos, se ve la necesidad de resolver, en cada iteración un sistema tridiagonal de ecuaciones lineales. Para resolver sistemas de ese tipo, LAPACK proporciona la rutina SGTSVX, que ofrece la máxima calidad y especialización.

2. El segundo ejemplo se ocupa de un sencillo sistema dinámico en el que se puede apreciar la decisiva participación del álgebra lineal para comprenderlo cabalmente. También aprovechamos la ocasión para observar que un pequeño cambio en los datos puede originar cambios grandes en la solución. Esta es una situación que se presenta intempestivamente que *no depende de la clase de máquina o de algoritmo que se utilice*.

Consideremos el siguiente problema de valor inicial

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_1 \\ y_1(0) &= 1 \\ y_2(0) &= -1 \end{aligned} \tag{2.2}$$

El método más apropiado para obtener la solución exacta de un sistema como éste, está basado enteramente en álgebra lineal. Notamos

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, Y_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \text{ y } A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Reenunciamos el problema en forma vectorial asi:

$$\begin{aligned} Y' &= AY \\ Y(0) &= Y_0 \end{aligned}$$

La idea clave para resolver un sistema lineal como éste es la de *Exponencial de una Matriz*. Una buena introducción a este tema aparece en [7]. El procedimiento en este caso es el siguiente:

a. Encontrar los valores propios de A. Dependiendo de la clase de matriz, LAPACK ofrece varias rutinas para esta tarea. Por supuesto con nuestra matriz no hay necesidad de usar LAPACK, es inmediato reconocer que sus valores propios son 1 y -1.

b. Encontrar vectores propios asociados con los valores propios de A. En casos menos sencillos, también es posible recurrir a LAPACK. Tanto la búsqueda de valores como la de vectores propios

son procedimientos en los que la propagación de errores de redondeo puede causar desastres. Es muy importante utilizar la mejor herramienta de software disponible para sortear con éxito esta fase del procedimiento. LAPACK cumple los mejores estándares de calidad a este respecto.

En nuestro caso, es fácil ver que $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ y $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ son vectores propios asociados con 1 y -1 respectivamente. De manera que la matriz $S = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ es tal que $S^{-1}DS = A$, donde $D = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Esta es una situación ideal. A una matriz con una base de vectores propios para el espacio de definición se le llama *diagonalizable*. No todas las matrices son diagonalizables, por ejemplo, si $C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ fuera diagonalizable, sería semejante a la matriz nula (por tanto igual a ella), lo cual es imposible pues C es no nula.

c. El cambio de variable $U = SY$ nos lleva al sistema lineal *desacoplado*

$$\begin{aligned} U' &= DY \\ U(0) &= \begin{bmatrix} 0 \\ 2 \end{bmatrix}. \end{aligned}$$

Su solución es $U = \begin{bmatrix} 0 \\ 2e^{-t} \end{bmatrix}$. De aqui obtenemos la solución para el sistema (2.2), dada por

$$Y = S^{-1}U = \begin{bmatrix} e^{-t} \\ -e^{-t} \end{bmatrix}.$$

d. Una característica fundamental de la solución Y es que $\lim_{t\to\infty} Y(t) = 0$. Más precisamente, si el sistema (2.2) se refiere a un par de poblaciones que interactúan de acuerdo con la ley lineal dada por la matriz A, vemos que ambas poblaciones eventualmente desaparecerán.

e. Sea $\epsilon > 0$ un número real. Si en lugar de $Y_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ tuviéramos una condición inicial $Y_0^\epsilon = \begin{bmatrix} 1+\epsilon \\ -1 \end{bmatrix}$, entonces

$$U(0) = \begin{bmatrix} \epsilon \\ 2+\epsilon \end{bmatrix}, U(t) = \begin{bmatrix} \epsilon e^t \\ (2+\epsilon)\, e^{-t} \end{bmatrix}$$

y

$$Y(t) = S^{-1}U(t) = \begin{bmatrix} \frac{1}{2}\left(\epsilon e^{t} + (2+\epsilon)\, e^{-t}\right) \\ \frac{1}{2}\left(\epsilon e^{t} - (2+\epsilon)\, e^{-t}\right) \end{bmatrix}.$$

El comportamiento de Y cuando $t \to \infty$, es completamente diferente. Ahora ambas componentes de Y tienden a ∞. Nótese que ϵ puede ser lo pequeño que se desee y que todos los cálculos se hicieron exactamente. Es decir, el comportamiento de la nueva solución, que es totalmente distinto al de la solución original, no obedece a aproximaciones durante los cálculos. Aqui es pertinente hacerse la siguiente reflexión: son pocos los problemas con solución exacta en forma cerrada con base en funciones elementales como polinomios o funciones exponenciales o trigonométricas. En los demás casos, solo podemos buscar una solución aproximada. En el proceso, se trabaja con aproximaciones a los verdaderos valores y si el problema tiene la sensibilidad a pequeños errores en los datos que tiene el que estudiamos aqui, podríamos encontrar sorpresas desagradables. Asi es que lo mejor que uno puede hacer, es apoyarse en herramientas confiables de software. Que las sorpresas que se presenten sean solamente las inevitables.

2.3 Normas y otros conceptos preliminares

Existen nociones topológicas y geométricas que son indispensables en ALN. Posiblemente la más importante sea la idea de norma que logra generalizar a espacios abstractos los conceptos escalares de valor absoluto y módulo.

Sea V un espacio vectorial no vacío, real o complejo. Una norma en V es una función $N : V \to \mathbb{R}$ que cumple las siguientes condiciones:

1. $N(x) \geq 0$ para todo $x \in V$ y $N(x) = 0$ si y solo si $x = 0$.
2. $N(\alpha x) = |\alpha|\, N(x)$ para todo $x \in V$ y todo escalar α.
3. $N(x+y) \leq N(x) + N(y)$ para todo x, $y \in V$.

A la norma $N(x)$ generalmente se le denota $\|x\|$. Se le agrega un subíndice si hay lugar a confusión.

Para el espacio vectorial $\mathbb{R}^n$ de vectores de n componentes reales, las siguientes son las normas más utilizadas:

Sea $x \in \mathbb{R}^n$ con componentes x_j, $j = 1, 2, ..., n$.

1. Norma infinito o norma uniforme: $\|x\|_\infty = \max\limits_{j=1,2,...,n} |x_j|$.
2. Norma 1: $\|x\|_1 = \sum\limits_{j=1}^{n} |x_j|$.

3. Norma 2 o euclideana: $\|x\|_2 = \left[\sum_{j=1}^{n} |x_j|^2\right]^{\frac{1}{2}}$.

Las mismas definiciones son válidas en el espacio vectorial $\mathbb{C}^n$ de vectores de n componentes complejas. En este caso, $|x_j|$ significa módulo del número complejo x_j.

Para el espacio vectorial $\mathbb{R}^{n\times n}$ de matrices cuadradas $n \times n$ con elementos reales, también se requiere el concepto de norma. La notación que se usa es la misma, pero, además de las propiedades 1 a 3 listadas arriba, se pide que se cumplan además las siguientes dos condiciones:

4. $\|AB\| \leq \|A\| \|B\|$, para toda $A, B \in \mathbb{R}^{n\times n}$,

5. Toda norma matricial debe ser *compatible* con alguna norma vectorial, es decir, $\|Ax\| \leq \|A\| \|x\|$, para toda $A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n$.

Las condiciones para una norma en el espacio matricial $\mathbb{C}^{n\times n}$ son análogas a las de arriba y no las precisaremos aqui.

Un concepto necesario para la definición de una de las normas matriciales más utilizadas es el de radio espectral. Sea $B \in \mathbb{R}^{n\times n}$ con $L = \{\lambda_1, \lambda_2, ..., \lambda_r\}$ su espectro, es decir, el conjunto de sus valores propios distintos. El radio espectral de B es $\rho(B) = \max_{\lambda\in L} |\lambda|$.

Las normas matriciales más utilizadas son las siguientes:

Sea $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ (Definiciones análogas para $A \in \mathbb{C}^{n\times n}$).

1. Norma infinito: $\|A\|_\infty = \max_{i=1,2,...,n} \sum_{j=1}^{n} |a_{ij}|$. (Compatible con norma infinito vectorial).

2. Norma 1: $\|A\|_1 = \max_{j=1,2,...,n} \sum_{i=1}^{n} |a_{ij}|$. (Compatible con norma 1 vectorial).

3. Norma 2: $\|A\|_2 = \left[\rho\left(A^T A\right)\right]^{\frac{1}{2}}$. (Compatible con norma 2 vectorial).

4. Norma de Frobenius: $\|A\|_F = \left[\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2\right]^{\frac{1}{2}}$. (Compatible con norma 2 vectorial).

2.4 Tipos de Datos

Al utilizar LAPACK es importante recordar los tipos de datos que pueden ser manipulados por los algoritmos y las limitaciones, costos y errores que resultan de utilizarlos. Recordemos que las convenciones utilizadas son:

1. S : denota precisión sencilla y números reales
2. C : denota precisión sencilla y números complejos
3. D : denota precisión doble y números reales

4. Z : denota precisión doble y números complejos

Actualmente la mayoría de los computadores utilizan el estándar 754 de la IEEE [8, 9] para representar números reales. Este estándar contempla representaciones de números reales que utilizan 32 bits (precisión sencilla) o 64 bits (precisión doble).

En este estándar los números reales se representan en 32 bits de la siguiente manera:

- El primer bit es el bit de signo (0 si el número es positivo, 1 si el número es negativo).
- Los siguientes 8 bits se utilizan para almacenar el exponente E. El exponente se almacena en formato de exceso 127, es decir hay que restar 127 del valor almacenado en estos bits para obtener el valor del exponente.
- Los 23 bits restantes se emplean para almacenar la mantisa M. Esta mantisa es una fracción menor que 0 a la que se le suma 1 para obtener el valor efectivo de la mantisa.

El número N es entonces:

$$N = (-1)^S 2^{E-127}(1.M)$$

donde E varía entre 1 y 254.

Los números de punto flotante diferentes a cero en este formato tienen magnitudes que varían entre $2^{-126}(1.0)$ y $2^{+127}(2-2^{-23})$ lo cual en base 10 se traduce aproximadamente a un rango entre 1.18×10^{-38} y 3.4×10^{38}.

La versión de 64 bits es una extensión del caso de 32 bits en la que el exponente utiliza 11 bits y la mantisa se almacena en 52 bits.

El número N, en este caso es:

$$N = (-1)^S 2^{E-1023}(1.M)$$

donde E varía entre 1 y 2046.

El tipo de datos complejo es llamado COMPLEX en Fortran y se implementa con dos variables, una que contiene la parte real y otra que contiene la parte imaginaria del número complejo. En Fortran el tipo de datos complejo de doble precisión se llama COMPLEX*16. No todos los compiladores de Fortran permiten utilizar este tipo de datos.

2.5 Limitaciones y Errores

Es importante recordar que se intenta representar en estos tipos de datos entidades de naturaleza continua y que al operar aritméticamente sobre las representaciones discretas y finitas de entidades continuas se introducirán errores de diferentes tipos.

LAPACK tiene rutinas de soporte dedicadas exclusivamente al descubrimiento en tiempo de ejecución de los parámetros de la máquina. Estas rutinas tienen como propósito determinar los siguientes parámetros:

- Epsilon de la máquina (eps): precisión relativa de la máquina, es decir, la mínima diferencia que puede ser detectada entre dos números.
- Mínimo seguro ($sfmin$): Valor mínimo tal que $1 \div sfmin$ no incurra en un desbordamiento ($overflow$).
- Base de la máquina ($base$ o $beta$): es el valor (entero) de la base de exponenciación de la máquina. Usualmente es 2.
- Precisión ($prec$): definida como $eps \times base$.
- Número de digitos de la mantisa (t)
- Ocurrencia de redondeo en la adición (rnd).
- Mínimo exponente antes de underflow gradual ($emin$).
- Umbral de underflow ($rmin$): calculado como $base^{(emin-1)}$.
- Máximo exponente antes de desbordamiento ($emax$).
- Umbral de desbordamiento ($rmax$): calculado como $(base^{emax}) \times (1 - eps)$.

Todos estos parámetros son usados por diferentes rutinas de LAPACK para tareas como escalado de matrices. Los valores de los parámetros no son precalculados, sino que siempre se calculan de nuevo cada vez que se requieren.

Para calcular sus valores, LAPACK cuenta con dos rutinas básicas: SLAMCH para precisión simple y DLAMCH para precisión doble. Cada una de estas rutinas cuenta con 5 rutinas adicionales de soporte que hacen el trabajo real, llamadas SLAMC1 hasta SLAMC5 y DLAMC1 hasta DLAMC5. El trabajo se divide de la siguiente manera:

- XLAMC1: Calcula $beta, t, rnd$ y adicionalmente determina si el redondeo que se está utilizando es simétrico, al estilo IEEE.
- XLAMC2: Calcula $eps, rmin$ y $emax$.
- XLAMC3: Es una rutina de soporte que solo calcula la suma de dos valores de punto flotante. Es usada para evitar que ciertos valores sean mantenidos en registros, y evitar problemas con las optimizaciones realizadas por algunos compiladores.
- XLAMC4: Calcula $emin$.
- XLAMC5: Calcula $rmax$.

2.5.1 Error Absoluto y Error Relativo

Sean a y b escalares. Entonces el **error absoluto**[1] en b considerado como una aproximación a a es el número

$$\epsilon = |b - a| \tag{2.3}$$

El error absoluto es difícil de interpretar sin información adicional acerca del tamaño de los números.

Sean a y b escalares con $a \neq 0$. Entonces el **error relativo** en b como una aproximación a a es el número

$$\rho = \frac{|b - a|}{|a|} \tag{2.4}$$

El error relativo está relacionado con el número de dígitos significativos en el que concuerdan los dos números a y b. Considere el siguiente ejemplo: Examinemos los siguientes números como aproximaciones a $e = 2.71828...$ y sus errores relativos:

Aproximación	Error Relativo
2	0.3
2.7	0.007
2.71	0.003
2.718	0.0001
2.7182	0.00005
2.71828	0.0000006

[1] Se recomienda al lector consultar este material en [9] páginas 121 a 141

Puede observarse que si el error relativo en b es ρ, entonces a y b concuerdan aproximadamente en $-\log(\rho)$ cifras decimales significativas.

Para vectores y matrices se definen error absoluto y error relativo con fórmulas análogas a (2.3) y (2.4) respectivamente.

2.5.2 Número de Condición

Dada una función f y dos argumentos x y y, es deseable conocer una cota para la distancia entre $f(x)$ y $f(y)$ en términos de la distancia entre x y y.

En otras palabras, queremos saber qué efecto tiene un error relativo en el dominio en el error relativo que se produce en el rango. Ese factor de amplificación es conocido como número de condición [10].

Cuando el número de condición de un problema determinado es grande, el problema se dice que está mal condicionado, porque un pequeño error relativo en el dominio muy probablemente generará un error relativo grande en el rango. Si el número de condición es pequeño, el problema se dice que está bien condicionado, no se amplifica considerablemente el error relativo al aplicar la función f.

El número de condición lo definiremos únicamente para matrices no singulares, es decir, aquellas matrices que tienen inversa. Estas son por cierto las matrices que más nos interesan pues en la solución de sistemas de ecuaciones siempre nos concentramos en sistemas con solución única.

Sea $A \in \mathbb{R}^{n\times n}$ no singular. Su número de condición es $\kappa(A) = \|A\| \left\|A^{-1}\right\|$, donde la norma que se usa es la misma en ambos casos. Una definición análoga es válida para matrices en $\mathbb{C}^{n\times n}$.

La importancia del número de condición de una matriz se ilustra claramente con un análisis de la propagación de una perturbación en la solución de un sistema lineal. Más precisamente:

Consideremos una matriz no singular $A \in \mathbb{R}^{n\times n}$ y un vector $b \in \mathbb{R}^n$. Existe una única solución x para el sistema $Ax = b$. Supongamos que en lugar de b se conoce a $\bar{b} = b + r$, una aproximación de b. Si denotamos por $\bar{x}$ a la solución del sistema $A\bar{x} = \bar{b}$, entonces se cumple que

$$\frac{1}{\kappa(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|\bar{x} - x\|}{\|x\|} \leq \kappa(A)\frac{\|r\|}{\|b\|}. \tag{2.5}$$

Es decir, una perturbación en el lado derecho b de un sistema lineal, lleva a un nuevo sistema $A\bar{x} = \bar{b}$ y el error relativo $\frac{\|\bar{x}-x\|}{\|x\|}$ en la solución del nuevo sistema, está acotado por debajo y por encima por múltiplos del error relativo en $\bar{b}$. Si $\kappa(A)$ es un número grande, el intervalo en el que se puede situar $\frac{\|\bar{x}-x\|}{\|x\|}$ es muy amplio y se corre el peligro de haber llegado a una solución muy lejana de la original.

Ejemplos:

1. Las matrices de Hilbert son ejemplos típicos de matrices mal condicionadas. La matriz de Hilbert de orden n, denotada H_n, se define asi: H_n es una matriz real cuadrada $n \times n$ y su elemento i, j está dado por $\frac{1}{i+j-1}$. Asi, por ejemplo,

$$H_5 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$$

Los números de condición para las primeras matrices de Hilbert son:

n	$\kappa(H_n)$
3	5.24×10^2
4	1.55×10^4
5	4.77×10^5
6	1.50×10^7
7	4.75×10^8
8	1.53×10^{10}
9	4.93×10^{11}
10	1.60×10^{13}

2. El segundo ejemplo consiste en la apreciación directa del cambio en la solución de un sistema lineal cuando se perturba levemente el vector del lado derecho.

Sean $A = \begin{bmatrix} 0.89 & 0.53 \\ 0.47 & 0.28 \end{bmatrix}$ y $b = \begin{bmatrix} 0.36 \\ 0.19 \end{bmatrix}$. La solución del sistema $Ax = b$ es $x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. En cambio, la solución exacta del sistema con la misma matriz A pero con $\bar{b} = \begin{bmatrix} 0.37 \\ 0.18 \end{bmatrix}$ en lugar de b, es $\bar{x} = \begin{bmatrix} 82 \\ -137 \end{bmatrix}$. Es ilustrativo comparar el error relativo en $\bar{x}$ con el error relativo en $\bar{b}$ y calcular $k(A)$ para confirmar que la doble desigualdad (2.5) se cumple.

3

Sistemas de Ecuaciones Lineales

Consideramos el sistema de ecuaciones lineales

$$Ax = b$$

donde A es la matriz de coeficientes, b es el lado derecho y x es la solución. El objetivo es hallar x dados A y b.

Si hay varios lados derechos, se escribe:

$$AX = B$$

donde las columnas de B son los lados derechos individuales y las columnas de X son las soluciones correspondientes. La tarea es calcular X, dadas A y B.

LAPACK cuenta con once variedades de rutinas para la solución de sistemas de ecuaciones lineales, uno para cada tipo y almacenamiento de matriz. Además, para cada tipo se cuenta con rutinas simples y expertas para cuatro clases de datos diferentes.

Las once variedades de almacenamiento de matrices son:

1. **General**: Matrices generales almacenadas en la forma usual, es decir, arreglos bidimensionales.

2. **Banda General**: Matrices banda generales, con cualquier número de sub- y super-diagonales. Se almacenan en un arreglo bidimensional, donde cada fila representa una diagonal, comenzando con la superior.

3. **Tridiagonal General**: Las matrices tridiagonales se almacenan en tres vectores independientes, uno por cada diagonal.

4. **Simétrica Definida Positiva**: Se almacenan como matrices bidimensionales estándares, pero solo se requieren los valores en el triángulo superior o inferior del arreglo. Las demás posiciones son ignoradas por las subrutinas de LAPACK.

5. **Simétrica Definida Positiva con almacenamiento compacto**: Las columnas se almacenan en un solo vector, una detrás de otra.

6. **Simétrica Definida Positiva Banda**: Similar al tipo 2, pero solo se almacena el triángulo relevante.
7. **Simétrica Positiva Definida Tridiagonal**: Similar al tipo 3, pero solo se requieren 2 vectores debido a la simetría.
8. **Simétrica Indefinida**: Similar al tipo 4.
9. **Simétrica Compleja**: El empacamiento es similar al del tipo 4, pero aquí la matriz puede contener elementos complejos no reales.
10. **Simétrica Indefinida con almacenamiento compacto:** El empacamiento es similar al del tipo 5.
11. **Simétrica Compleja con almacenamiento compacto**: Similar a la anterior pero con elementos de tipo complejo.

Para cada uno de los diferentes almacenamientos existen rutinas en los cuatro tipos de datos básicos:

- Real simple
- Complejo simple
- Real con doble precisión
- Complejo con doble precisión

La excepción son las rutinas para los almacenamientos tipo 9 y 11, que solo están disponibles en las 2 variedades de datos complejos.

A continuación describimos tres subrutinas de las llamadas *driver subroutines* que se encargan de resolver problemas completos con matriz general, tridiagonal y simétrica definida positiva dada por bandas, respectivamente.

3.1 SGESV

SGESV es un manejador simple para matrices generales. Soluciona un sistema real de ecuaciones lineales de la forma:

$$AX = B$$

donde A es una matriz N × N, y X y B son matrices de N filas y NRHS columnas.

La subrutina SGESV utiliza un algoritmo de factorización LU con pivoteo parcial e intercambio de filas para resolver el sistema de ecuaciones, haciendo uso de las rutinas computacionales SGETRF y SGETRS. La primera de éstas realiza la factorización LU de la matriz A, mientras que la segunda utiliza los factorez L y U para resolver el sistema de ecuaciones propiamente dicho. Con este algoritmo, una matriz de coeficients A se descompone en 3 matrices P, L y U, tal que:

$$A = PLU$$

donde P es la matriz de permutaciones causadas por el pivoteo, L es la matriz triangular inferior y U es la matriz triangular superior. Vale la pena recordar que el realizar pivoteo parcial no afecta el orden de las respuestas que se obtienen del procedimiento, a diferencia de lo que ocurre cuando se utiliza pivoteo total.

El prototipo de la subrutina SGESV es:

```
SUBROUTINE SGESV ( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
      INTEGER  INFO, LDA, LDB, N, NRHS
      INTEGER  IPIV( * )
      REAL     A( LDA, * ), B( LDB, * )
```

donde

N : Orden de la matriz A y el número de ecuaciones en el sistema.

NRHS : Número de columnas de la matriz B.

A : Matriz de coeficientes. Cuando la subrutina retorna, A contiene los factores L y U resultado de la factorización. Nótese que los elementos unitarios en la diagonal de L no se almacenan.

LDA : Número máximo de filas que se pueden almacenar en la matriz A. Debe ser mayor o igual a N.

IPIV : Arreglo de dimensión N en que se retornan las permutaciones realizadas durante el pivoteo. La fila i de la matriz A fue intercambiada con la fila IPIV(i). Pueden aparecer valores repetidos en IPIV. Por ejemplo, para $n = 4$:

$$IPIV = \begin{bmatrix} 3 \\ 4 \\ 3 \\ 4 \end{bmatrix} \Longrightarrow P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

B: Una matriz de LDB filas y NRHS columnas, que a la entrada contiene los conjuntos de valores del lado derecho de las ecuaciones, y a la salida la matriz X de soluciones del sistema.

LDB : Número máximo de filas que se pueden almacenar en la matriz B. Debe ser mayor o igual a N.

INFO : Un valor entero que indica si la subrutina tuvo éxito o no, de la siguiente forma:

- INFO $= 0$: El sistema pudo ser resuelto sin ningún problema.
- INFO $= -i$: El argumento i-ésimo pasado a la subrutina era inválido. Por ejemplo, si se pasa un N menor que 0, se retorna INFO $= -1$, o si se pasa LDA menor que N, se retorna INFO $= -4$.
- INFO $= i$: La posición (i, i) del factor U calculado es 0, por lo que U es singular. Debido a ésto, tratar de resolver el sistema daría como resultado una división por cero.

3.2 SGTSV

La subrutina SGTSV es un manejador simple para matrices tridiagonales. Resuelve un sistema real de ecuaciones lineales de la forma

$$AX = B$$

donde A es una matriz $N \times N$ de tres diagonales centrales, y X y B son matrices de N filas y NRHS columnas. Al igual que SGESV, la subrutina SGTSV puede tomar múltiples conjuntos de vectores al lado derecho y resolver el sistema para todos ellos en una sola llamada. Para ésto, cada vector se pasa como una columna en la matriz B.

La subrutina SGTSV utiliza factorización LU con pivoteo parcial e intercambio de filas para resolver el sistema de ecuaciones, pero a diferencia de SGESV, la rutina hace todo el trabajo sin ayuda de otras rutinas computacionales o de las BLAS.

La subrutina SGTSV no devuelve los multiplicadores (los valores de la subdiagonal de L) directamente, sino que deben ser calculados manualmente a partir de los valores de U retornados.

El prototipo de la subrutina SGTSV es:

```
SUBROUTINE SGTSV( N, NRHS, DL, D, DU, B, LDB, INFO )
        INTEGER  INFO, LDB, N, NRHS
        REAL     B( LDB, * ), D( * ), DL( * ), DU( * )
```

donde

N : Orden de la matriz A y el número de ecuaciones en el sistema.

NRHS : Número de columnas de la matriz B.

DU, D, DL : Vectores de dimensiones $N-1$, N y $N-1$ respectivamente que almacenan la superdiagonal de A, la diagonal principal de A y la subdiagonal de A respectivamente. Cuando la subrutina retorna, el vector D contiene los elementos de la diagonal principal de U, el vector DU contiene los elementos de la primera superdiagonal de U y DL contiene los elementos de la segunda superdiagonal de la matriz U. Se espera que los $n-2$ primeros elementos de DL sean todos cero, si no lo son, se debe revisar el sistema para ver si existe alguna particularidad que induzca a errores de redondeo significativos.

B : Una matriz de LDB filas y NRHS columnas, que a la entrada contiene los conjuntos de vectores del lado derecho y a la salida la matriz X de soluciones del sistema.

LDB : Número máximo de filas que se pueden almacenar en la matriz B. Debe ser mayor o igual a N.

INFO : Un valor entero que indica si la subrutina tuvo éxito o no, de la siguiente forma:

- INFO $= 0$: El sistema fue resuelto sin ningún problema.
- INFO $= -i$: El argumento i-ésimo era inválido.
- INFO $= i$: La posición (i, i) del factor U calculado es 0, por lo que si i es diferente de N la factorización no se completó.

3.3 SPBSV

SPBSV soluciona un sistema real de ecuaciones lineales de la forma

$$AX = B$$

donde A es una matriz $N \times N$ simétrica definida positiva dada por bandas y X y B son matrices de N filas y NRHS columnas. Al igual que las dos subrutinas anteriores, SPBSV puede tomar múltiples conjuntos de vectores de lado derecho y resolver el sistema para todos ellos en una sola llamada. Para ésto, cada conjunto de valores se pasa como una columna en la matriz B.

La subrutina SPBSV utiliza las subrutinas computacionales SPBTRF, que realiza la descomposición Cholesky y SPBTRS, que resuelve el sistema. Debido a que la matriz A es definida positiva, no se requiere de pivoteo.

La rutina SPBSV solo devuelve los valores de uno de los factores, según la matriz de entrada y el parámetro UPLO, como se explica abajo.

El prototipo de la subrutina SPBSV es:

```
SUBROUTINE SPBSV( UPLO, N, KD, NRHS, AB, LDAB, B, LDB, INFO )
        CHARACTER      UPLO
        INTEGER        INFO, KD, LDAB, LDB, N, NRHS
        REAL           AB( LDAB, * ), B( LDB, * )
```

donde

UPLO : Indica si la rutina debe tomar en cuenta los valores en la triangular inferior o superior de la matriz A y si se debe retornar el factor U o L:

Si UPLO $= 'U'$, es porque lo almacenado corresponde a la parte triangular superior de la matriz A. Análogamente si UPLO $= 'L'$.

N : Orden de la matriz A y el número de ecuaciones en el sistema.

KD : Número de super o sub diagonales en la matriz A dada por bandas.

NRHS : Número de columnas de la matriz B.

AB : Matriz de coeficientes que contiene la parte triangular superior o inferior de A, dada por bandas, en sus primeras KD $+ 1$ filas.

LDAB : Número de filas en la matriz AB, debe ser mayor o igual a KD $+ 1$.

B : Una matriz de LDB filas y NRHS columnas, que a la entrada contiene el conjunto de vectores de lado derecho de las ecuaciones, y a la salida la matriz X de soluciones del sistema.

LDB : Número máximo de filas que se pueden almacenar en la matriz B. Debe ser mayor o igual a N.

INFO : Un valor entero que indica si la subrutina tuvo éxito o no, de la siguiente forma:

- INFO $= 0$: El sistema fue resuelto sin ningún problema.
- INFO $= -i$: El argumento i-ésimo era inválido.
- INFO $= i$: La matriz i-ésima menor principal no es definida positiva, por lo que el sistema no se puede resolver.

A continuación incluimos un programa que soluciona un sistema lineal de ecuaciones cuya matriz de entrada está dada por bandas.

```
* PROGRAMA DE PRUEBA PARA SPBSV()
* EL ARCHIVO DE DATOS TIENE EL SIGUIENTE FORMATO:
*       N             ORDEN DEL SISTEMA
*       KD            NUMERO DE SUPER/SUB-DIAGONALES
*       NRHS          NUMERO DE LADOS DERECHOS
*       SUPER/SUB-DIAGONAL KD DE A
*       SUPER/SUB-DIAGONAL KD-1 DE A
*       ...
*       DIAGONAL PRINCIPAL DE A
*       FILA 1 DE B
*       ...
*       FILA N DE B
*
EXTERNAL SPBSV
* N: NUMERO DE ECUACIONES
* NMAX: MAXIMO NUMERO DE ECUACIONES PERMITIDO(200)
* NRHS: NUMERO DE VECTORES DEL LADO DERECHO
* NRHMAX: VALOR MAX. PERMITIDO PARA NRHS(10)
* KD: NUMERO DE SUPERDIAGONALES
* INFO: RESULTADO OPERACION
INTEGER N, NMAX, NRHS, NRHMAX, KD, INFO
PARAMETER (NMAX = 200, NRHMAX=10)

* MATRIZ A DE ENTRADA Y B DE LADOS DERECHOS
REAL A(NMAX,NMAX), B(NMAX,NRHMAX)


* VERIFICACION DE PARAMETROS
* EN ENTRADA
READ *, N, KD, NRHS
IF (N .GT. NMAX) THEN
   PRINT *, 'N DEMASIADO GRANDE'
```

```
   RETURN
ENDIF
IF (NRHS .GT. NRHMAX) THEN
   PRINT *, 'NRHS DEMASIADO GRANDE'
   RETURN
ENDIF
IF (KD .LT. 0 ) THEN
   PRINT *, 'KD INVALIDO'
   RETURN
ENDIF


DO 1000 I = 1, N
   DO 1010 J = 1, N
              A(I,J) = 0
 1010      CONTINUE
 1000   CONTINUE
*     LEER A
*     VA POR DIAGONALES ALINEADAS A LA DERECHA
*     USAMOS SUPERDIAGONALES (U)
DO 10 I = 1, KD + 1
   READ (5, FMT = *)(A(I,J), J = (KD-I+2), N)
 10 CONTINUE
DO 20 I = 1, N
   READ (5, FMT = *)(B(I,J), J = 1, NRHS)
 20 CONTINUE

*     LLAMAMOS A SPBSV CON UPLO = 'U'
CALL SPBSV ( 'U', N, KD, NRHS, A, NMAX, B, NMAX, INFO )

IF (INFO .EQ. 0) THEN
PRINT *, 'SOLUCIONADO!'
PRINT *, 'X:'
DO 70 I = 1, N
```

```
      WRITE (6, FMT = *)(B(I,J), J = 1, NRHS)
70          CONTINUE
PRINT *, 'MATRIZ U:'
DO 80 I = 1, N
      WRITE (6, FMT = *)(A(I,J), J = 1, N)
80          CONTINUE
ELSE IF (INFO .LT. 0) THEN
PRINT *, 'PARAMETRO ', -INFO, ' INCORRECTO'
ELSE
PRINT *, 'U(', INFO, ',', INFO, ') = 0'
ENDIF

END
```

Este programa imprimirá la matriz B de resultados y la matriz factor de Cholesky.

Bajo LINUX, el lenguaje FORTRAN que más se utiliza es g77, <http://gcc.gnu.org>, que es el ofrecido por el proyecto GNU, especializado en software de dominio público, cuya página WEB es

$$< \text{http} : //www.\text{gnu.org} > .$$

La ejecución de un programa FORTRAN en una instalación estándar de g77 bajo LINUX, se realiza con el siguiente par de comandos:

1. *g77 miarchivo.f -lblas -llapack*

Aqui el programa fuente *miarchivo.f* se compila y se agrupa con todas las librerías que pueda requerir, incluyendo las que se agregaron expresamente, BLAS y LAPACK. En inglés, esta agrupación del programa con las librerías que requiere se denomina *link edition*. El producto final de este doble proceso se denomina código ejecutable correspondiente a *miarchivo.f*. Generalmente el nombre por defecto que se le asigna es *a.out*.

2. *a.out*

Este paso es la ejecución del programa.

Aqui se está presentando únicamente información genérica y no se consideran posibles errores, otras formas de invocar los comandos o formas de actuar en caso de haber problemas. Recomendamos tener a la mano manuales para LINUX, FORTRAN *g77* y LAPACK. Todos pueden obtenerse en Internet afortunadamente. Además, tener manuales a la mano significa simplemente poder consultarlos en línea o en papel. En particular, no recomendamos imprimir páginas de ayuda residentes en el computador local que se pueden consultar en línea.

4

Valores y Vectores Propios

El problema estandar de valores propios o simplemente problema de valores propios para una matriz A consiste en encontrar escalares (valores propios) λ y vectores propios correspondientes $z \neq 0$ tales que

$$Az = \lambda z \tag{4.1}$$

A los vectores z que cumplen (4.1) también se les llama *vectores propios derechos asociados con el valor propio* λ y los *vectores propios izquierdos asociados con el valor propio* λ se definen como los vectores $u \neq 0$ tales que $u^H A = \lambda u^H$.

La matriz A puede ser real o compleja, lo mismo que los escalares λ y los vectores z.

Distinguimos tres problemas generalizados de valores propios para dos matrices simétricas A y B. Consisten en encontrar escalares (valores propios generalizados) λ y vectores propios generalizados correspondientes $z \neq 0$ tales que:

Problema 1: $Az = \lambda Bz$

Problema 2: $ABz = \lambda z$

Problema 3: $BAz = \lambda z$.

Para matrices cuadradas no simétricas A y B, distinguimos dos problemas generalizados de valores propios, a saber:

1. Encontrar escalares (valores propios generalizados) λ y vectores propios generalizados correspondientes $x \neq 0$ tales que $Ax = \lambda Bx$.

2. Encontrar escalares (valores propios generalizados) ρ y vectores propios generalizados correspondientes $y \neq 0$ tales que $\rho Ay = By$.

Con fines ilustrativos, presentamos algunos detalles de la definición de tres subrutinas para resolver problemas de valores propios. Se trata de las subrutinas SSYEV, SGEES y SGEEV. La primera se encarga de calcular todos o solamente algunos de los valores propios y vectores propios correspondientes para una matriz real simétrica. Se basa en factorización QR. La segunda subrutina encuentra todos los valores propios y la forma real de Schur para una matriz real no necesariamente simétrica. La última se encarga, no solo de calcular los valores propios sino que además calcula vectores propios derechos y/o izquierdos de una matriz general real no necesariamente simétrica.

Estas dos rutinas son apenas una pequeña parte de la amplia colección de rutinas que tiene

LAPACK para el complicado problema de aproximar valores y vectores propios. Algunas de las rutinas disponibles se basan en algoritmos recientemente desarrollados, como *RRR (Relatively Robust Representation)* o en algoritmos que solo recientemente están llamando la atención de los investigadores como el algoritmo *Divide y vencerás*. Sugerimos mantener contacto con la página WEB de LAPACK

$$\text{http} : //www.\text{netlib.org}/\text{lapack}/\text{index.html}$$

para más explicaciones y referencias acerca de los métodos numéricos utilizados en las subrutinas.

4.1 SSYEV

Esta subrutina es el manejador simple para encontrar valores y/o vectores propios de una matriz simétrica A. Se basa en los cuatro pasos siguientes:

1. Escalar la matriz, si es necesario, de acuerdo a la precisión y el epsilon de la máquina. Para los numerales 2 y 3 siguientes, suponemos que, después del escalamiento, la matriz se sigue llamando A.

2. Reducir la matriz a su forma tridiagonal , mediante una llamada a SSYTRD. Es decir, se trata de encontrar una matriz ortogonal Q tal que $Q^T A Q = T$, con T tridiagonal simétrica.

3. Calcular los valores propios de A, mediante una llamada a SSTERF, que utiliza una variante del algoritmo QR para encontrar valores propios de matrices tridiagonales simétricas. Si también se requiere de los vectores propios, entonces primero se genera la matriz ortogonal Q del paso anterior mediante una llamada a la subrutina SORGTR, y luego se computan valores y vectores propios de T, con una sola llamada a la subrutina SSTEQR. Los valores propios de A son los mismos de T. Los vectores propios correspondientes del problema con matriz A, se consiguen a partir de los que se acaban de calcular recurriendo a la transformación ortogonal del paso anterior.

4. Se re-escala la matriz nuevamente si se realizó algún escalamiento en el paso 1.

El prototipo de la rutina SSYEV es:

```
SUBROUTINE SSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )
```

```
CHARACTER       JOBZ, UPLO

INTEGER         INFO, LDA, LWORK, N

REAL            A( LDA, * ), W( * ), WORK( * )
```

donde

JOBZ: Un caracter de control que especifica qué tareas se desea realizar. Los valores posibles son:

- 'N': Calcular solo los valores propios
- 'V': Calcular tanto los valores como los vectores propios

UPLO: Un caracter de control que indica si se almacenó la parte triangular superior ('U') o la parte triangular inferior ('L')de A.

N: Orden de la matriz A.

A: Matriz que se desea usar.

LDA: Número de filas en la matriz A. Debe ser mayor o igual a N.

W: Un vector de dimensión N en el que se retornan los valores propios de A, en orden ascendente.

WORK: Vector real que la rutina usa como área de trabajo.

LWORK: Longitud del vector WORK.

INFO: Valor entero que indica si hubo éxito o no de la siguiente forma:

- INFO $= 0$: Salida exitosa.
- INFO $= -i$: El argumento i -ésimo de entrada es inválido.
- INFO $= i$: El algoritmo falló. Un elemento de la diagonal i-ésima en una matriz tridiagonal intermedia no converge a cero.

4.2 SGEES

La rutina SGEES es un manejador simple que calcula los valores propios y la factorización Schur de una matriz cuadrada y posiblemente no simétrica A.

La rutina calcula la factorización real de Schur dada por

$$A = ZTZ^T$$

donde Z es una matriz ortogonal y T es una matriz casi triangular superior. Es decir, en su diagonal, pueden aparecer bloques tanto 1×1 como 2×2. Estos últimos corresponden a pares conjugados de valores propios complejos de la matriz A. El cálculo de la matriz Z es opcional. Los bloques 2×2 en la matriz T son de la forma

$$\begin{bmatrix} a & b \\ c & a \end{bmatrix}$$

donde $bc < 0$. Los valores propios serán entonces de la forma $a \pm \sqrt{bc}$.

La primera tarea de esta subrutina es reducir A a una forma Hessenberg H, es decir, encontrar una matriz ortogonal Q tal que $A = QHQ^T$. Las matrices en forma de Hessenberg son las que tienen ceros debajo de su primera subdiagonal.

La segunda tarea es reducir H a su forma de Schur T, es decir, encontrar una matriz ortogonal S tal que $H = STS^T$. La matriz Z resulta ser $Z = QS$. Todo ésto se hace recurriendo a subrutinas computacionales especializadas.

El prototipo de la rutina SGEES es:

```
SUBROUTINE SGEES( JOBVS, SORT, SELECT, N, A, LDA, SDIM, WR, WI,

                  VS, LDVS, WORK, LWORK, BWORK, INFO )

      CHARACTER   JOBVS, SORT

      INTEGER     INFO, LDA, LDVS, LWORK, N, SDIM

      LOGICAL     BWORK(*)

      REAL        A(LDA,*), VS(LDVS,*), WI(*), WORK(*), WR(*)
```

donde

JOBVS: Si es 'V' se deben calcular expresamente las columnas de Z, llamadas vectores de Schur. Si es 'N', no se calculan.

SORT: Si es 'S', se ordenan los valores propios y no se ordenan si es 'N'.

SELECT: Select es la función usada para ordenar los valores propios, si así se especifica, llevando aquellos valores seleccionados hacia la parte superior izquierda de la matriz T (retornada en los arreglos *WR* y *WI*).

Select debe ser declarada con el siguiente prototipo:

EXTERNAL LOGICAL FUNCTION SELECT(WR, WI)

REAL WR, WI

El vector *WR* contiene la parte real de los valores propios y el vector *WI* contiene la parte imaginaria.

Es importante notar, sin embargo, que el proceso de ordenamiento puede cambiar algunos de los valores propios seleccionados, especialmente si se presentan problemas de mal condicionamiento.

N: Orden de la matriz de entrada A.

A: Matriz de entrada, en forma de arreglo de dimensiones$(\mathrm{LDA}, \mathrm{N})$. Al retornar, A contiene la matriz T resultado de la factorización.

LDA: Número de filas en la matriz A.

SDIM: Al retornar, SDIM contiene el número de valores propios seleccionados tras el ordenamiento. Nótese que, debido a los problemas antes mencionados, es posible que SDIM sea diferente al número de veces que la función SELECT retornó un valor .TRUE.

WR, *WI*: Arreglos reales de dimensión N que contienen a la salida partes real e imaginaria respectivamente de los valores propios de A.

VS: Arreglo real de dimensión $(\mathrm{LDVS}, \mathrm{N})$, en el que se retorna la matriz ortogonal Z, si ésta fue solicitada.

LDVS**:** Número de filas de la matriz VS. Si no se solicitó el cálculo de la matriz Z, entonces LDVS debe ser por lo menos 1. Si se calcula Z, entonces LDVS debe ser mayor o igual al orden de la matriz de entrada A.

WORK, LWORK**:** Área de trabajo y su tamaño. Funciona de la misma manera que en SSYEV. La dimensión LWORK debe ser mayor o igual a 3N.

BWORK**:** Arreglo lógico de dimensión N que se usa como área de trabajo en caso de que se solicite el ordenamiento de los valores propios. No es referenciado en caso contrario.

INFO**:** Retorna un código de error:

- INFO $= 0$: No hubo error.
- INFO < 0: El i-ésimo parámetro de entrada tiene un valor no válido.
- $0 <$ INFO $\leq$ N: El algoritmo falló al calcular los valores propios.
- INFO $=$ N $+ 1$: Los valores propios no pudieron ser ordenados debido a que algunos tienen valores muy cercanos. Usualmente indica problemas de condicionamiento.
- INFO $=$ N $+ 2$: Luego del ordenamiento, algunos valores propios cambiaron debido a redondeos o problemas de escalamiento. Por lo tanto, es probable que algunos de estos valores ya no satisfagan que SELECT retorne verdadero para ellos.

4.3 SGEEV

La rutina SGEEV es el manejador simple para calcular los valores propios y los vectores propios izquierdos (u) y/o derechos (v) de una matriz real general A. Es importante notar que SGEEV normaliza los vectores propios calculados de forma tal que tengan norma euclidiana igual a 1 y parte real lo más grande posible.

SGEEV inicia escalando la matriz A, balanceándola posteriormente mediante una llamada a la rutina SGEBAL con el objetivo de reducir la norma 1 de la matriz y mejorar así la precisión de los valores y vectores propios calculados. El balanceo se realiza en 2 pasos básicos: en el primer paso se permuta A con una transformación de semejanza que aísle los valores propios en la diagonal. El segundo paso consiste en aplicar una transformación de semejanza diagonal a ciertas filas y columnas de manera tal que sus normas respectivas se acerquen en valor.

Una vez balanceada, se reduce A a una forma superior de Hessenberg y se calculan los vectores propios izquierdos y derechos (si se solicitan). Para ésto, se calculan primero vectores de Schur

usando una técnica similar a la usada por SGEES, y luego se realiza una llamada a STREVC para calcular los vectores propios a partir de una matriz quasi-triangular superior en forma canónica de Schur generada a partir de los vectores de Schur calculados.

Finalmente, se normalizan los vectores propios y se reescalan los resultados para compensar el escalado inicial.

El prototipo de SGEEV es:

```
SUBROUTINE SGEEV( JOBVL, JOBVR, N, A, LDA, WR, WI, VL, LDVL,
    $             VR, LDVR, WORK, LWORK, INFO )
    CHARACTER     JOBVL, JOBVR
    INTEGER       INFO, LDA, LDVL, LDVR, LWORK, N
    REAL          A( LDA, * ), VL( LDVL, * ), VR( LDVR, * ),
    $             WI( * ), WORK( * ), WR( * )
```

JOBVL: Si es $'N'$, no se calculan los vectores propios izquierdos; si es $'V'$, si se calculan.

JOBVR: Igual a JOBVL, pero se refiere al cálculo de los vectores propios derechos.

N: Orden de la matriz de entrada A.

A: Matriz de entrada, de dimensión $(\mathrm{LDA}, \mathrm{N})$. Téngase en cuenta que durante el proceso los elementos de la matriz son destruidos, aún a pesar de que no se retorne ningún valor útil en ella.

LDA: Número de filas de la matriz A.

WR y WL: Vectores reales de dimensión N, que al retornar contienen las partes reales e imaginarias, respectivamente, de los valores propios calculados.

VL: Matriz real de dimensión $(\mathrm{LDVL}, \mathrm{N})$ en la que se retornan los vectores propios izquierdos calculados. No es necesaria si se especificó JOBVL $=' \mathrm{N}'$. Los vectores se almacenan en columnas consecutivas de VL, en el mismo orden en que sus valores propios respectivos aparecen en WR y WI.

LDVL: Número de filas de la matriz VL. LDVL debe ser mayor que 1, incluso si no se solicita el cálculo de VL, y mayor que N en caso contrario.

VR: Matriz real de dimensión $(\mathrm{LDVR}, \mathrm{N})$, que al retornar contiene los vectores propios derechos calculados. Funciona de la misma manera que VL.

LDVR: Número de filas de la matriz VR. Sujeto a las mismas restricciones que LDVL.

WORK, *LWORK*: Área de trabajo y su tamaño. Funciona de la misma manera que en SSYEV. LWORK debe ser, como mínimo, mayor o igual a 3N si no se solicita el cálculo de vectores propios, o mayor o igual a 4N en caso contrario.

INFO: Indica el resultado de la operación:

- INFO $= 0$: Operación completada con éxito.
- INFO < 0: El i-ésimo parámetro de entrada tiene un valor no válido.
- INFO > 0: Falló el algoritmo QR en calcular todos los valores propios. Dado que INFO retorne el valor i, entonces los elementos $i + 1$ hasta N de los vectores *W*R y *W*L contendrán aquellos valores propios ya calculados.

En lugar de ofrecer un ejemplo de programa que llame la rutina SGEEV, incluimos un pequeño ejemplo de un programa que halla los valores propios de una matriz tridiagonal simétrica por medio de la rutina SSTEBZ de LAPACK.

```
      PROGRAM VALORESPROPIOS
C     Este programa ilustra el uso de una rutina
C     que halla los valores propios de una matriz
C     tridiagonal simetrica

C     Variable con la que se especifica el rango de
C     de valores propios que se va a calcular
      CHARACTER*1 RANGE
C     Orden en que se retornan los valores propios
      CHARACTER*1 ORDER
C     Tamanyo de la matriz
      INTEGER N
C     Intervalo en el que se buscaran valores propios
      REAL VL
      REAL VU
C     En caso de que solo se calcule un rango
C     en las siguientes variables se especifica el rango
C     de valores propios que se va a calcular
```

```
      INTEGER IL1
      INTEGER IU1
C     Tamanyo maximo del intervalo en
C     el que se buscan intervalos
      REAL ABSTOL
C     La diagonal principal y las sub y superdiagonal
      REAL D(10000)
      REAL E(10000)
C     Numero de valores propios encontrados
      INTEGER M
C     Numero de bloques diagonales en matriz
      INTEGER NSPLIT
C     Los resultados retornan en el arreglo W
      REAL W(10000)
C     Indicador de los lugares donde la matriz
C     se divide en sublbloques
      INTEGER IBLOCK(10000)
      INTEGER ISPLIT(10000)
C     Areas de trabajo
      REAL WORK(40000)
      INTEGER IWORK(30000)
C     Codigo de retorno
      INTEGER INFO
      N = 10000
      ABSTOL = 0.0
C     Se inicializan los arreglos D y E
      DO 10 I = 1,10000
         D(I) = 2.0
         E(I) = 1.0
 10   CONTINUE
C     Se solicita que se calculen algunos valores
C     propios
      RANGE = 'I'
C     Se solicita que se ordenen todos los valores propios
```

```
      ORDER = 'E'
C     El rango de valores propios que se va a calcular
      IL = 1
      IU = 2500

      T1 = secnds(0.0)

      CALL SSTEBZ(RANGE,ORDER,N,VL,VU,IL,IU,ABSTOL,D,E,
     *            M,NSPLIT,W,IBLOCK,ISPLIT,WORK,IWORK,INFO)

      T2 = secnds(T1)
      WRITE(*,*) 'Codigo de retorno: ',INFO
      WRITE(*,*) 'Demoro: ',T2
      END
```

En este programa en particular solo se están imprimiendo el código de retorno y el número de segundos que demoró el programa. Si se desea, se puede imprimir el contenido del vector W que contiene los valores propios.

5

Descomposición en Valores Singulares

Sea A una matriz general $m \times n$. La descomposición en valores singulares (Singular Value Decomposition SVD) de A es la factorización $A = U\Sigma V^T$, donde U y V son ortogonales y $\Sigma = diag(\sigma_1, \cdots, \sigma_r), r = min(m, n)$ con $\sigma_1 \geq \cdots \sigma_r \geq 0$. Si A es una matriz compleja, entonces su descomposición en valores singulares es $A = U\Sigma V^H$ donde U y V son unitarias y Σ tiene elementos reales en la diagonal. Los σ_i son llamados los valores singulares. Las primeras r columnas de V son los vectores singulares derechos y las primeras r columnas de U son los vectores singulares izquierdos.

Es importante recordar que [9, 5]

- El rango de una matriz es el número de sus valores singulares diferentes de cero.
- La norma 2 de una matriz es su mayor valor singular.
- El cuadrado de la norma de Frobenius de una matriz es la suma de los cuadrados de sus valores singulares.
- El número de condición κ_2 de una matriz A (utilizando la norma 2) cumple la siguiente igualdad:

$$k_2(A) = \frac{\sigma_1}{\sigma_s}$$

donde σ_1 es el mayor valor singular de A y σ_s es el menor valor singular no nulo de A.

La descomposición, en LAPACK, se calcula en dos etapas:

1. La matriz A se reduce a formato bidiagonal: Si A es real se calcula $A = U_1 B V_1^T$. Si A es compleja se calcula

   $A = U_1 B V_1^H$. En estas descomposiciones, U_1 y V_1 son ortogonales (unitarias si A es compleja), B es real y contiene elementos:

   - En la diagonal principal y en la primera superdiagonal (bidiagonal superior) si $m \geq n$.
   - En la diagonal principal y en la primera subdiagonal (bidiagonal inferior) si $m < n$.

2. Se calcula la descomposición singular de la matriz bidiagonal B:

$$B = U_2 \Sigma V_2^T$$

donde U_2 y V_2 son ortogonales, Σ es diagonal y tiene el formato descrito previamente. Los vectores singulares de A están en $U = U_1 U_2$ y $V = V_1 V_2$.

En LAPACK existen dos rutinas manejadoras para encontrar Valores Singulares: SGESVD para matrices reales y CGESVD para matrices complejas. A continuación describimos SGESVD:

La subrutina SGESVD calcula la descomposición en valores singulares (SVD) de una matriz real $M \times N$. Opcionalmente puede calcular los vectores singulares izquierdos y/o derechos.

El prototipo de la subrutina SGESVD es:

```
 SUBROUTINE SGESVD ( JOBU, JOBVT, M, N, A, LDA, S, U, LDU,
$                    VT, LDVT, WORK, LWORK, INFO )
      CHARACTER JOBU, JOBVT
      INTEGER  INFO, LDA, LDU, LDVT, LWORK, M, N
      REAL     A( LDA, * ), S( * ), U( LDU, * ),
$              VT( LDVT, * ), WORK ( * )
```

JOBU (entrada) : Especifica qué partes de la matriz U se van a calcular. Los valores posibles son:

- $'A'$: Se calculan todas las M columnas de U y se retornan en la matriz U.
- $'S'$: Se calculan las primeras $\min(M, N)$ columnas de U y se retornan en la matriz U.
- $'O'$: Se calculan las primeras $\min(M, N)$ columnas de U y se retornan en la matriz A. Se borra el contenido anterior de la matriz A.
- $'N'$: No se calcula ninguna columna de la matriz U (o sea, ningún vector singular izquierdo).

JOBVT (entrada) : Especifica qué partes de la matriz V^T se van a calcular. Los valores posibles son:

- $'A'$: Se calculan todas las filas de V^T y se retornan en la matriz VT.
- $'S'$: Se calculan las primeras $\min(M, N)$ columnas de V^T y se retornan en la matriz VT.

- 'O' : Se calculan las primeras $\min(M, N)$ columnas de V^T y se retornan en la matriz A.
- 'N' : No se calcula ninguna columna de la matriz V^T.

M (entrada) : El número de filas de la matriz A. $M > 0$.

N (entrada): El número de columnas de la matriz A. $N > 0$.

A (entrada/salida): Matriz cuyos valores singulares se van a calcular. Tiene dimensión $M \times N$. Almacena información de salida si JOBU o JOBVT se especifican como O.

LDA (entrada) : Número de filas de la matriz A. $LDA \geq \max(1, M)$.

S (salida) : Arreglo REAL de dimensión $\min(M, N)$. Contendrá al final los valores singulares de A, almacenados de mayor a menor $S(i) \geq S(i+1)$.

U (salida) : Arreglo REAL de dimensión:

- (LDU, M) si JOBU vale A
- $(LDU, \min(M, N))$ si JOBU vale S

LDU (entrada) : El número de filas del arreglo U.

VT (salida) : Arreglo REAL de dimensión $(LDVT, N)$.

LDVT (entrada) : El número de filas del arreglo VT.

WORK (área de trabajo) : Arreglo de dimensión LWORK. A la salida, si INFO vale 0, la posición WORK(1) retorna el valor óptimo para LWORK.

LWORK (entrada) : El número de filas del arreglo WORK. El valor de LWORK debe ser como mínimo:

$\max(3 * \min(M, N) + \max(M, N), 5 * \min(M, N) - 4)$

Si el valor es mayor, se obtendrá un rendimiento mejor.

INFO (salida) : Indicador del resultado de la llamada. Los posibles valores retornados son:

- 0 si la rutina funcionó bien
- < 0 Si INFO vale $-i$, el i-ésimo argumento contenía un valor inválido.

- > 0 Si INFO vale i, el algoritmo no convergió; i elementos que estaban por fuera de la diagonal principal en la matriz de formato bidiagonal no convergieron a cero.

Para finalizar incluimos un ejemplo de un programa que halla la descomposición en valores singulares de una matriz aleatoria real.

```
C
C     Programa que ilustra como se llama la rutina SGESVD
C     que calcula la descomposicion en valores singulares
C     de una matriz general
C
      PROGRAM ENSAYOSVD

C     Parametro que indica que partes de la matriz U
C     se van a calcular (entrada)
      CHARACTER*1 JOBU
C     Parametro que indica que partes de la matriz VT
C     se van a calcular (entrada)
      CHARACTER*1 JOBVT
C     Dimensiones de la matriz A.
C     M es el numero de filas
C     N es el numero de columnas (entrada)
      INTEGER M
      INTEGER N
C     A La matriz cuya descomposicion se va a hallar
C     (entrada)
      REAL A(1000,1000)
C     El numero de filas de la matriz A
C     (entrada)
      INTEGER LDA
C     S : Los valores singulares de A (salida)
      REAL S(1000)
C     U : La matriz U (salida)
      REAL U(1000,1000)
C     El numero de filas de la matriz U
```

```
C     (entrada)
      INTEGER LDU
C     U : La matriz U (salida)
      REAL VT(1000,1000)
C     El numero de filas de la matriz U
C     (entrada)
      INTEGER LDVT
C     Area de trabajo
      REAL WORK(5000)
C     Tamanyo del area de trabajo
      INTEGER LWORK
C     Indicador de los resultados (salida)
      INTEGER INFO

C     Se solicita que se calcule la matriz U completa
      JOBU = 'A'
C     Se solicita que se calcule la matriz VT completa
      JOBVT = 'A'
      M = 1000
      N = 1000
      LDA = 1000
      LDU = 1000
      LDVT = 1000
      LWORK = 5000

C     Se inicializa la matriz A
      DO 10 I = 1,1000
         DO 20 J = 1,1000
            A(I,J) = RANDOM(0)
 20   CONTINUE
 10   CONTINUE

      T1 = secnds(0.0)
```

```
      CALL SGESVD(JOBU,JOBVT,M,N,A,LDA,S,U,LDU,VT,LDVT,
     *            WORK,LWORK,INFO)

      T2 = secnds(T1)
      WRITE(*,*) 'Codigo de retorno: ',INFO
      WRITE(*,*) 'Demoro: ',T2
      END
```

De nuevo en este programa sólo se están imprimiendo el código de retorno y el número de segundos que demoró el programa. Si se desea, se puede imprimir el contenido del vector S que contiene los valores singulares y las matrices U y VT que contienen los vectores singulares derechos e izquierdos.

6

Manejo de Bloques en Lapack

Para mejorar el desempeño de LAPACK y comprender cómo funciona la colección, es importante entender cómo manipula las matrices sobre las que opera.

6.1 ¿Por qué bloques?

Existen muchas razones por las cuales puede ser útil dividir las matrices en sub-bloques. La más importante es la ganancia en velocidad que se obtiene al utilizar más eficientemente la jerarquía de memoria que se encuentra en la gran mayoría de los equipos actuales.

La mayor parte de las máquinas usadas actualmente tienen una jerarquía de memoria de por lo menos cinco niveles:

- Disco (Memoria Virtual)
- Memoria Principal (física)
- Memoria Cache L2
- Memoria Cache L1
- Registros

Por el momento, nos concentraremos principalmente en los 3 niveles intermedios. Cada nivel es aproximadamente un orden de magnitud más rápido que el nivel inmediatamente superior, lo cual nos indica que hay gran potencial para incrementar el desempeño de nuestros algoritmos si podemos mantener los datos sobre los que operamos en el nivel más bajo posible. Esto es conocido como el principio de localidad espacial.

El dividir las matrices en sub-bloques y operar sobre éstos de manera independiente nos permite aprovechar el principio de localidad al operar sobre conjuntos de datos más pequeños cuyos elementos están relativamente muy cercanos.

Consideremos, por ejemplo, una operación aparentemente simple como es la multiplicación de matrices. Supongamos una máquina con una configuración común hoy en día: un cache L2 de 512KB y un par de matrices relativamente grandes de 1000×1000.

Supongamos también que cada elemento en cada una de las matrices se almacena como un valor de punto flotante de doble precisión, necesitando 8 bytes (64-bits) de almacenamiento cada uno. Según ésto, en el mejor de los casos, el cache L2 no podría retener en un momento dado ni el 3% de cada una de las matrices. Si consideramos el algoritmo usual para multiplicar matrices, notamos que para la matriz que aparece en el lado derecho de la operación, las referencias a memoria son a datos no contiguos, por lo que el cache se vería obligado a remplazar bloques con mucha frecuencia.

El desempeño se reduciría notablemente debido al incremento en la rata de *cache misses*. Es de notar además, que estos cálculos son, en realidad, optimistas, ya que usualmente, los caches L2 son unificados para datos e instrucciones, lo cual reduce aún más el espacio efectivo que se puede utilizar.

Casos como éstos requieren de algoritmos capaces de operar sobre porciones relativamente pequeñas de una matriz a la vez, y así tomar plena ventaja del principio de localidad espacial. El éxito de esta estrategia depende de dos factores críticos:

Tamaño de Bloque (NB): para aprovechar el cache de la máquina al máximo, necesitamos escoger un NB tal que el cache sea capaz de acomodar fácilmente todos los bloques sobre los que estamos operando. Por ejemplo, con un tamaño de bloque de 64 y un cache de 512KB, se podrían acomodar tres bloques completos de 8×8 y habría espacio de sobra. Es de notar que la selección de NB también depende del proceso numérico que se desee realizar, así como de las restricciones o posibilidades que nos ofrezca el algoritmo desarrollado.

Estabilidad: Es claro que, en muchos casos, de nada sirve desarrollar algoritmos super-veloces si se pierden la estabilidad y precisión numérica de las que depende el éxito de la operación que se realiza.

Una pregunta que se presenta aquí es: *¿Por qué no aprovechar la estructura que presentan algunas matrices para seleccionar el tamaño de bloque?*

Algunas matrices, notablemente las diagonales por bloques, se prestan a ser procesadas de esta manera. Sin embargo, actualmente LAPACK no posee rutinas especializadas para operar sobre este tipo de matrices, ni, como veremos en detalle más adelante, tiene en cuenta esta información en la escogencia de NB.

6.2 Implementación de LAPACK

LAPACK implementa los algorimos por bloques en dos niveles separados: las BLAS y las rutinas computacionales.

6.2.1 Las BLAS

Las *Basic Linear Algebra Subprograms* o BLAS implementan las operaciones básicas de álgebra lineal de manera independiente del resto de LAPACK. Esto implica que es fácil remplazar las BLAS originales escritas en FORTRAN por versiones altamente optimizadas para la plataforma en la que se está trabajando. Las operaciones que las BLAS implementan se dividen, como vimos al principio, en operaciones de niveles 1 (vector-vector), 2 (matriz-vector) y 3 (matriz-matriz).

De manera análoga, las rutinas computacionales estan divididas en versiones diferentes, dependiendo de cuál categoría de funciones BLAS invoquen. Las rutinas que hacen uso del nivel 3 de las BLAS son aquellas que más relevantes resultan en esta discusión, puesto que son las que realmente implementan los algoritmos por bloques.

Las rutinas computacionales y las BLAS de nivel 3 tienen una característica particular: Implementan algoritmos $O(n^3)$ cuando bastaría con $O(n^2)$. Inicialmente, podría parecer que ésto va a afectar negativamente el desempeño, pero dadas las condiciones antes mencionadas, es posible lograr todo lo contrario. ¿Cómo? Las BLAS de nivel 3 ejecutan más operaciones, pero de forma tal que se realice todo el trabajo posible de una vez sobre los datos traídos de memoria.

Con ésto, se puede aumentar el factor q, que es la razón de operaciones realizadas sobre los datos con respecto a las referencias de memoria requeridas. En otras palabras, q es un estimador de que tan bien se puede aprovechar el principio de localidad espacial.

En algunos casos, las rutinas computacionales con bloques no pueden invocar a las funciones de BLAS 3, como cuando el usuario fuerza el tamaño de bloque 1. En estos casos las rutinas invocan a sus contrapartes sin bloques para hacer el trabajo, que solo invocan funciones de nivel 1 o 2 de las BLAS.

6.2.2 Selección de NB

En LAPACK, el trabajo de seleccionar el tamaño de bloque apropiado recae sobre la rutina ILAENV. Esta rutina devuelve un valor para NB de acuerdo a dos factores:

Subrutina que solicita el valor de NB.

Que tan óptimo debe ser el valor retornado para NB por esta rutina.

El valor retornado por ILAENV es una constante predefinada por los autores de LAPACK, hallada de forma experimental como un valor óptimo del tamaño de bloque para las máquinas en que se escribió LAPACK originalmente.

Es concebible pensar que, dados los avances en la velocidad de procesamiento y la complejidad de los procesadores en los últimos diez años, sea factible usar valores más grandes.

Es una lástima, sin embargo, que ILAENV no haga parte de las BLAS sino de LAPACK propiamente dicha, ya que así al reemplazar las BLAS por una versión optimizada para nuestra máquina, obtendríamos un tamaño de bloque también óptimo.

Referencias

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*. SIAM, 1992.

[2] C. E. Mejía, T. Restrepo, and C. Trefftz, "Lapack una colección de rutinas para resolver problemas de algebra lineal numérica," *Revista Universidad EAFIT*, vol. 123, pp. 73–80, 2001.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[4] L.Ñ. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[5] G. H. Golub and C. F. VanLoan, *Matrix Computations*. Johns Hopkins University Press, second ed., 1989.

[6] G. W. Stewart, *Introduction to Matrix Computations*. Academic Press, 1973.

[7] B.Ñoble and J. W. Daniel, *Algebra Lineal Aplicada*. Prentice Hall, tercera ed., 1989.

[8] J. P. Hayes, *Computer Architecture and Organization*. McGraw Hill, 1988.

[9] G. W. Stewart, *Matrix Algorithms - Volume I: Basic Decompositions*. SIAM, 1998.

[10] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*. Springer Verlag, segunda ed., 1992.

Indice