LAPACK

Una colección de rutinas para resolver problemas de Álgebra Lineal Numérica

Carlos Enrique Mejía
Tomás Restrepo
Christian Trefftz

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 hacen 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 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 sólo directamente, sino también como motor de cálculo en software con interfase gráfica como OCTAVE, 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

CARLOS ENRIQUE MEJÍA SALAZAR. Matemático, Universidad Nacional, Medellín, 1982. M.S. Matemáticas, Universidad Nacional, Medellín, 1985. Ph.D. Matemáticas, University of Cincinnati, 1993. Profesor asociado, Universidad Nacional en Medellín desde 1993.
E-mail: cemejias@epm.net.co

TOMÁS RESTREPO. Estudiante de Ingeniería de Sistemas, Universidad EAFIT.
E-mail: tomasr@mvps.org

CHRISTIAN TREFFTZ G. Ph.D. en Ciencias de la Computación de Michigan State University. Profesor Asistente Grand Valley State University en Allendale Michigan.
E-mail: trefftzc@gvsu.edu

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 sea necesario seguir haciendo estas preguntas, se propuso 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 artículo es un resumen de un documento que pretende lograr la divulgación descrita. Se dirige a personas acostumbradas a usar un computador, que conocen rudimentos de FORTRAN y que están familiarizadas con las nociones básicas del álgebra lineal.

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 sólo directamente, sino también como motor de cálculo en software con interfase gráfica como OCTAVE, desarrollado en la Universidad de Wisconsin.

El contenido de este artículo está organizado de la siguiente forma:

La sección 2 se dedica a explicar las formas de obtener e instalar LAPACK. En la sección 3 sobre álgebra lineal numérica, se explican procesos de modelamiento que conducen a problemas de álgebra lineal susceptibles de ser resueltos con LAPACK.

La sección 4 se dedica a considerar dos clases de problemas que se pueden resolver con LAPACK, sistemas de ecuaciones lineales y problemas de valores y vectores propios. En la sección 5, de naturaleza más técnica, se consideran aspectos relacionados con almacenamiento de matrices y uso de la memoria.

El artículo termina con algunas conclusiones presentadas en la sección 6.

2. INSTALACIÓN DE LAPACK

2.1 Descarga

Lo primero que se necesita para instalar LAPACK, es obtener una distribución. El principal sitio del que se puede obtener LAPACK es el sitio 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:

1. **lapack.tgz:** Contiene la distribución completa de LAPACK 3.0 en un archivo comprimido usando tar y gzip.
2. **lapack-pc.zip:** archivo en formato zip con la versión 3.0 de LAPACK para Microsoft Windows. Requiere de la rutina NMAKE de Microsoft y el compilador Fortran de Digital Equipment Corporation, distribuido ahora por la propia Microsoft.
3. **lapack-pc-wfc.zip:** archivo en formato zip con la versión 3.0 de LAPACK para Microsoft Windows. Requiere de la rutina NMAKE de Microsoft y el compilador Fortran 77/32 de Watcom versión 11.0.

Adicionalmente es posible descargar las colecciones precompiladas, así como algunas de las rutinas individuales.

Para estas notas se supone que se desea instalar LAPACK para LINUX, por lo que el archivo correcto que se debe descargar es lapack.tgz. El sistema operativo LINUX, que es de dominio público, puede instalarse fácilmente en microcomputadores con procesadores Intel o compatible, que son la mayoría. Lo más común, es que toda instalación de LINUX incluya compiladores de dominio público de varios lenguajes de programación, incluyendo Fortran. Así que, en principio, estamos ante una plataforma de cálculo numérico de primera calidad basada en software no comercial. Una de las distribuciones mejor mantenidas de LINUX es REDHAT LINUX, con dirección Internet **http://www.redhat.com**

2.2 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 LAPACK, se cambia al directorio ./LAPACK/INSTALL. Desde aquí se puede configurar la instalación de LAPACK para obtener las opciones correctas para la compilación. Lo primero es usar el archivo de configuración apropiado para LINUX

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

Antes de compilar LAPACK propiamente dicho, hay que compilar las BLAS:

cd ../BLAS/SRC
make

Si éstas funcionan sin ningún problema, se puede compilar la colección y los programas de prueba:

cd ../../
make

Lo primero que se 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 sí misma, cuyo código fuente se encuentra en el directorio SRC.

Luego se procederá a compilar los programas de prueba. Hay que tener 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 sólo desea compilar la colección, ejecute el comando make directamente en el directorio SRC.

3. ALGEBRA LINEAL NUMÉRICA

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 Álgebra 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.

Hay dos grandes familias de problemas en ALN:

a. Sistemas de ecuaciones algebraicas de la forma Ax=b, con A matriz y x, b vectores. Aquí se incluyen las soluciones por mínimos cuadrados.

b. Problemas de valores y vectores propios de la forma $Ax=\lambda x$, con A matriz, x vector y λ escalar (real o complejo).

Últimamente han aparecido buenos libros sobre el tema como (Demmel, 1997) y (Trefethen, 1997) que se unen a libros ya clásicos como (Golub, 1989) y (Stewart, 1973). Un libro de nivel intermedio y muy completo que se recomienda es (Noble, 1989).

Para dar una idea concreta acerca de este tema de estudio, se considera un ejemplo que sirve de motivación, no solo para el estudio somero de los conceptos y métodos que se ofrecen enseguida sino también para profundizar en algunos de ellos. Este ejemplo sirve para precisar la idea de discretización.

Se considera la ecuación diferencial ordinaria de segundo orden

$$y''(t) = f(t, y, y'), a \le t \le b,$$
$$y(a) = A, y(b) = B.$$

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 surge 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 se denotan $a = t_0 < t_1 < \ldots < t_{n+1} = b$. Enseguida para $j=1,2,\ldots,n$, 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

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

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. Se tiene 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.

Se supone que este sistema tiene solución para preservar sencilla esta exposición. Se advierte 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 este sistema se enuncia más fácilmente con notación vectorial. Denotamos por V al vector columna cuyas componentes son $v(1),v(2),...,v(n)$ y definimos una función de variable y valor vectorial G de la siguiente manera:

$$G : R^n \rightarrow R^n$$

con la componente j-ésima de GV igual al lado izquierdo de la ecuación j-ésima correspondiente. De esta forma, el sistema 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.

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.

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 Álgebra 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 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.

4. ALGUNAS RUTINAS DE LAPACK

En esta sección se presentan dos rutinas, de las muchas disponibles en LAPACK, con las que se ilustra la forma en que se pueden utilizar estos procedimientos.

4.1 Ecuaciones Lineales

En primer lugar, se considera el sistema de ecuaciones lineales $x=b$ 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.

A continuación se describe una subrutina, de las llamadas driver subroutines, que se encargan de resolver un problema completo de una matriz general.

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 $N \times N$, 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 factores L y U para resolver el sistema de ecuaciones propiamente dicho. Con este algoritmo, una matriz de coeficientes A se descompone en 3 matrices P, L y U, tales

que $A=PLU$, donde P es la matriz de permutaciones debidas al 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:

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

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 esto, tratar de resolver el sistema daría como resultado una división por cero.

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.

4.2 Valores y Vectores Propios

El problema estándar 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 no nulos tales que $Az = \lambda z$.

A los vectores z que cumplen esta ecuación 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 no nulos 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.

Se distinguen 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 no nulos 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, se distinguen dos problemas generalizados de valores propios, a saber:

1. Encontrar escalares (valores propios generalizados) λ y vectores propios generalizados correspondientes x no nulos tales que $Ax = \lambda Bx$.
2. Encontrar escalares (valores propios generalizados) ρ y vectores propios generalizados correspondientes y no nulos tales que $\rho Ay = Bx$.

Con fines ilustrativos, se presentan algunos detalles de la definición de una subrutina para resolver problemas de valores propios. Se trata de la subrutina SSYEV que 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.

Esta rutina es 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 sólo recientemente están llamando la atención de los investigadores como el algoritmo "Divide y vencerás".

La subrutina SSYEV 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 al epsilon de la máquina. Para los numerales 2 y 3 siguientes, se asume 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 $Q^T AQ = T$, 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 reescala 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 que tareas se desea realizar. Los valores posibles son:

N: Calcular sólo 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.

5. 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.

¿Por qué bloques?

Existen muchas razones por las cuales puede ser útil dividir las matrices en subbloques. 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:

1. Disco (Memoria Virtual)

2. Memoria Principal (física)

3. Memoria Cache L2

4. Memoria Cache L1

5. Registros

Por el momento, se hará énfasis principalmente en los 3 niveles intermedios. Cada nivel es aproximadamente un orden de magnitud más lento que el nivel inmediatamente superior, lo cual indica que hay gran potencial para incrementar el desempeño de los algoritmos si podemos mantener los datos sobre los que operan en el nivel más alto posible. Esto es conocido como el principio de localidad espacial.

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

Como ejemplo, considérese 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 x 1000.

Supóngase 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 esto, 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 se considera el algoritmo usual (manual) para multiplicar matrices, se notará 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:

a. Tamaño de Bloque (NB): para aprovechar el cache de la máquina al máximo, se necesita escoger un NB tal que el cache sea capaz de acomodar fácilmente todos los bloques sobre los que se está operando. Por ejemplo, con un tamaño de bloque de 64 bytes y un cache de 512KB, se podrían acomodar tres bloques completos de 8x8 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 ofrezca el algoritmo desarrollado.

b. 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. De poco sirve obtener resultados muy rápido si éstos son completamente erróneos.

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 se verá en detalle más adelante, tiene en cuenta esta información en la escogencia de NB.

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

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 se vio al principio, en operaciones de niveles 1 (vector-vector), 2 (matriz-vector) y 3 (matriz-matriz).

De manera análoga, las rutinas computacionales están 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 esto 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 esto, 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 sólo invocan funciones de nivel 1 ó 2 de las BLAS.

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:

a. Subrutina que solicita el valor de NB.

b. ¿Qué tan óptimo debe ser el valor retornado para NB por esta rutina?

El valor retornado por ILAENV es una constante predefinida 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, se obtendría un tamaño de bloque también óptimo.

6. CONCLUSIONES

LAPACK es una colección de rutinas que combina precisión y eficiencia. Usar las rutinas de LAPACK es fácil.

Se considera que los cursos de métodos numéricos que se dictan en las Universidades locales pueden complementarse con el uso de algunas de las rutinas de LAPACK. De esta forma los estudiantes, y futuros profesionales, estarán conscientes de la existencia de unas rutinas de alta calidad, de uso fácil y gratuitas que podrán utilizar más adelante cuando la ocasión lo amerite.

BIBLIOGRAFÍA

Anderson E. et al. (1992). LAPACK Users' Guide. USA: SIAM.

Demmel J.W. (1997). Applied Numerical Linear Algebra. USA: SIAM.

Golub G. and Van Loan C. (1989). Matrix Computations, 2a. ed. USA: Johns Hopkins University Press.

Hayes J.P. (1988). Computer Architecture and Organization. USA: McGraw Hill.

Noble B. and Daniel J. W. (1989). Algebra Lineal Aplicada, 3a. ed. USA: Prentice Hall.

Stewart G.W. (1973). Introduction to Matrix Computations. New York: Academic Press.

Stewart G.W. (1998). Matrix Algorithms - Volume I: Basic Decompositions", USA: SIAM.

Trefethen L.N. and Bau D. (1997). Numerical Linear Algebra. USA: SIAM.