Descomposición LU e inversión

Lección 10

Dr. Pablo Alvarado Moya

CE3102 Análisis Numérico para Ingeniería
Área de Ingeniería en Computadores
Tecnológico de Costa Rica

II Semestre 2017

Contenido

Objetivo de la descomposición **LU**

- Eliminación Gauss-Jordan permite encontrar solución a varios sistemas $\mathbf{A}\underline{\mathbf{x}}_i = \underline{\mathbf{b}}_k$ simultaneamente, pero todos vectores $\underline{\mathbf{b}}_i$ deben ser conocidos con anterioridad
- Si el vector $\underline{\mathbf{b}}_k$ solo se conoce uno a la vez, el proceso de eliminación debe repetirse
- Descomposición LU permite reducir complejidad en este caso.
- Permite además calcular la matriz $\mathbf{A}^{-1}$

Repaso de eliminación de Gauss

- Eliminación de Gauss tiene dos pasos:

Repaso de eliminación de Gauss

- Eliminación de Gauss tiene dos pasos:
    1. Eliminación

Repaso de eliminación de Gauss

- Eliminación de Gauss tiene dos pasos:
  1. Eliminación
  2. Sustitución hacia atrás

Repaso de eliminación de Gauss

- Eliminación de Gauss tiene dos pasos:
  1. Eliminación
  2. Sustitución hacia atrás
- Eliminación requiere mayor parte del cómputo $(\mathcal{O}(n^3))$

Repaso de eliminación de Gauss

- Eliminación de Gauss tiene dos pasos:
  1. Eliminación
  2. Sustitución hacia atrás
- Eliminación requiere mayor parte del cómputo ($\mathcal{O}\left(n^3\right)$)
- Sustitución es menos costoza ($\mathcal{O}\left(n^2\right)$)

Uso de la descomposición **LU** (1)

Asúmase que **A** se puede descomponer en

1. Una matriz triangular superior **U** (*upper*)
2. Una matriz triangular inferior **L** (*lower*)

$$\mathbf{A} = \mathbf{L}\mathbf{U}$$

Uso de la descomposición **LU** (2)

Se cumple que

$$\begin{aligned} \mathbf{A}\underline{\mathbf{x}} &= \underline{\mathbf{b}} \\ \mathbf{L}\mathbf{U}\underline{\mathbf{x}} &= \underline{\mathbf{b}} \\ \mathbf{L}\underbrace{(\mathbf{U}\underline{\mathbf{x}})}_{\underline{\mathbf{y}}} &= \underline{\mathbf{b}} \end{aligned}$$

que se puede resolver en dos pasos:

1. Encontrar un vector $\underline{\mathbf{y}}$ tal que

$$\mathbf{L}\underline{\mathbf{y}} = \underline{\mathbf{b}}$$

por medio de sustitución hacia adelante ($\mathcal{O}\left(n^2\right)$)

Uso de la descomposición **LU** (3)

2. Encontrar vector $\underline{\mathbf{x}}$ que

$$\mathbf{U}\underline{\mathbf{x}} = \underline{\mathbf{y}}$$

por medio de sustitución hacia atrás $(\mathcal{O}\left(n^2\right))$

Uso de la descomposición **LU** (4)

Con la matriz triangular inferior

$$\mathbf{L} = \begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix}$$

la descomposición hacia adelante de $\mathbf{L}\underline{\mathbf{y}} = \underline{\mathbf{b}}$ se resuelve con

$$y_1 = \frac{b_1}{l_{11}}$$

$$y_i = \frac{1}{l_{ii}} \left[b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right] \qquad i = 2, 3, \ldots, n$$

Uso de la descomposición LU (5)

Con la matriz triangular superior

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & u_{33} & \cdots & u_{3n} \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{bmatrix}$$

la descomposición hacia atrás de $\mathbf{U}\underline{\mathbf{x}} = \underline{\mathbf{y}}$ se resuelve con

$$\begin{aligned} x_n &= \frac{y_n}{u_{nn}} \\ x_i &= \frac{1}{u_{ii}} \left[y_i - \sum_{j=1+1}^{n} u_{ij} x_j \right] \qquad i = n-1, n-2, \ldots, 1 \end{aligned}$$

Descomposición **LU** con algoritmo de Doolittle (1)

Con la eliminación de Gauss se obtuvo la matriz

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ & & a''_{33} & \cdots & a''_{3n} \\ & & & \ddots & \vdots \\ & & & & a_{nn}^{(n-1)} \end{bmatrix}$$

en donde para obtener los ceros bajo la diagonal se utilizaron factores

$$f_{ij} = \frac{a_{ij}^{(j-1)}}{a_{jj}^{(j-1)}} \qquad i > j$$

Si estos factores f_{ij} se almacenan, el procesamiento de $\underline{\mathbf{b}}$ se puede realizar *a posteriori*.

Descomposición **LU** con algoritmo de Doolittle (2)

Se puede demostrar que

$$\mathbf{U} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ & & a''_{33} & \cdots & a''_{3n} \\ & & & \ddots & \vdots \\ & & & & a_{nn}^{(n-1)} \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} 1 & & & & \\ f_{21} & 1 & & & \\ f_{31} & f_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ f_{n1} & f_{n2} & f_{n3} & \cdots & 1 \end{bmatrix}$$

Descomposición **LU** con algoritmo de Doolittle (3)

Se pueden aprovechar los nuevos espacios en blanco de la matriz creada, para almacenar los factores:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ f_{21} & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ f_{31} & f_{32} & a''_{33} & \cdots & a''_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{n1} & f_{n2} & f_{n3} & \cdots & a^{(n-1)}_{nn} \end{bmatrix}$$

De esta forma, la descomposición se realiza *in situ* ahorrando memoria de almacenamiento

Descomposición **LU** con algoritmo de Doolittle (4)

- Este algoritmo tiene la misma complejidad algorítmica que la eliminación de Gauss
- Los dos pasos de sustituciones hacia adelante y hacia atrás se justifican solo si se requiere resolver varios sistemas de ecuaciones para varios vectores $\underline{\mathbf{b}}_k$
- Pivoteo es necesario para asegurar estabilidad del algoritmo

Ejemplo: Descomposición LU en LAPACK (1)

```
      SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
*     .. Scalar Arguments ..
      INTEGER            INFO, LDA, M, N
*     ..
*     .. Array Arguments ..
      INTEGER            IPIV( * )
      DOUBLE PRECISION   A( LDA, * )
*     ..
*  Purpose
*  =======
*
*  DGETRF computes an LU factorization of a general M-by-N matrix
*  A using partial pivoting with row interchanges.
*
*  The factorization has the form
*     A = P * L * U
*  where P is a permutation matrix, L is lower triangular with
*  unit diagonal elements (lower trapezoidal if m > n), and U
*  is upper triangular (upper trapezoidal if m < n).
```

Ejemplo: Descomposición LU en LAPACK (2)

```
*
*  This is the right-looking Level 3 BLAS version of the
*  algorithm.
*
*  Arguments
*  =========
*
*  M       (input) INTEGER
*          The number of rows of the matrix A.  M >= 0.
*
*  N       (input) INTEGER
*          The number of columns of the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension
*          (LDA,N)
*          On entry, the M-by-N matrix to be factored.
*          On exit, the factors L and U from the factorization
*          A = P*L*U; the unit diagonal elements of L are not
*          stored.
*
```

Ejemplo: Descomposición LU en LAPACK (3)

```
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,M).
*
*  IPIV    (output) INTEGER array, dimension (min(M,N))
*          The pivot indices; for 1 <= i <= min(M,N), row i of
*          the matrix was interchanged with row IPIV(i).
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal
*                value
*          > 0:  if INFO = i, U(i,i) is exactly zero. The
*                factorization has been completed, but the
*                factor U is exactly singular, and division by
*                zero will occur if it is used to solve a system
*                of equations.
```

Descomposición **LU** con algoritmo de Crout (1)

Obsérvese el producto en $\mathbf{A} = \mathbf{LU}$:

$$
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}
\end{bmatrix}
=
$$

$$
\begin{bmatrix}
l_{11} & & & & \\
l_{21} & l_{22} & & & \\
l_{31} & l_{32} & l_{33} & & \\
\vdots & \vdots & \vdots & \ddots & \\
l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn}
\end{bmatrix}
\begin{bmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
 & u_{22} & u_{23} & \cdots & u_{2n} \\
 & & u_{33} & \cdots & u_{3n} \\
 & & & \ddots & \vdots \\
 & & & & u_{nn}
\end{bmatrix}
$$

Descomposición **LU** con algoritmo de Crout (2)

- Se observa que por los ceros en ambas matrices

$$a_{ij} = \sum_{k=1}^{\min(i,j)} l_{ik} u_{ki} = \begin{cases} l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{ij} & i < j \\ l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{jj} & i = j \\ l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ij}u_{jj} & i > j \end{cases}$$

  que son n^2 ecuaciones para $n^2 + n$ incógnitas l y u.
- Para hacer solucionable este sistema se elige arbitrariamente $u_{ii} = 1$ para $i = 1 \ldots n$, así que restan n^2 incógnitas

Descomposición **LU** con algoritmo de Crout (3)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{21} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & 1 & \cdots & u_{3n} \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}$$

Descomposición **LU** con algoritmo de Crout (4)

¿Cómo podemos encontrar l_{ij} y u_{ij} a partir de los a_{ij}?

Proponga algún algoritmo

Descomposición **LU** con algoritmo de Crout (5)

Utilizando la interpretación del producto matricial (**columna 1**):

$$\begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} l_{11} \\ l_{21} \\ l_{31} \\ \vdots \\ l_{n1} \end{bmatrix}$$

y además (**fila 1**)

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix} = l_{11} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \end{bmatrix}$$

de donde se despejan $l_{\cdot 1}$ y $u_{1 \cdot}$.

Descomposición **LU** con algoritmo de Crout (6)

Además

$$\begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \\ \vdots \\ a_{n2} \end{bmatrix} = u_{12} \begin{bmatrix} l_{11} \\ l_{21} \\ l_{31} \\ \vdots \\ l_{n1} \end{bmatrix} + 1 \begin{bmatrix} \\ l_{22} \\ l_{32} \\ \vdots \\ l_{n2} \end{bmatrix}$$

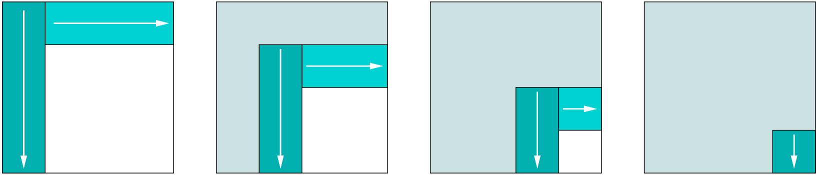
y adicionalmente

$$\begin{aligned} \begin{bmatrix} a_{21} & a_{22} & \cdots & a_{2n} \end{bmatrix} = & l_{21} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \end{bmatrix} + \\ & l_{22} \begin{bmatrix} 0 & 1 & \cdots & u_{2n} \end{bmatrix} \end{aligned}$$

de donde se despejan $l_{\cdot 2}$ y $u_{2 \cdot}$.

Descomposición **LU** con algoritmo de Crout (7)

El algoritmo de Crout soluciona el sistema utilizando los coeficientes de **A** en el orden ilustrado



$$l_{i1} = a_{i1} \qquad i = 1 \dots n$$

$$u_{1j} = \frac{a_{1j}}{l_{11}} \qquad j = 2 \dots n$$

Descomposición **LU** con algoritmo de Crout (8)

Y para $j = 2 \dots n-1$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \qquad i = j \dots n$$

$$u_{jk} = \frac{a_{jk} - \sum_{i=1}^{j-1} l_{ji} u_{ik}}{l_{jj}} \qquad k = j+1 \dots n$$

$$l_{nn} = a_{nn} - \sum_{k=1}^{n-1} l_{nk} u_{kn}$$

- Observe que a_{ij} se utiliza solo una vez, y por tanto los resultados se pueden almacenar *in-situ*

Cálculo de columnas de matriz inversa (1)

- Sea $\underline{\mathbf{i}}_{\cdot j}$ la j-ésima columna de la matriz identidad $\mathbf{I}$.
- Si se multiplica la ecuación $\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$ por la matriz inversa $\mathbf{A}^{-1}$ se obtiene

$$\begin{aligned}\mathbf{A}\underline{\mathbf{x}} &= \underline{\mathbf{b}}\\ \mathbf{A}^{-1}\mathbf{A}\underline{\mathbf{x}} &= \mathbf{A}^{-1}\underline{\mathbf{b}}\\ \underline{\mathbf{x}} &= \mathbf{A}^{-1}\underline{\mathbf{b}}\end{aligned}$$

Cálculo de columnas de matriz inversa (2)

- Asúmase que

$$\mathbf{A}^{-1} = \begin{bmatrix} y_{11} & y_{12} & y_{13} & \cdots & y_{1n} \\ y_{21} & y_{22} & y_{23} & \cdots & y_{2n} \\ y_{31} & y_{32} & y_{33} & \cdots & y_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & y_{n3} & \cdots & y_{nn} \end{bmatrix}$$

Cálculo de columnas de matriz inversa (3)

- Si se multiplica esta matriz por $\underline{\mathbf{i}}_{\cdot j}$ se obtiene (aquí con ejemplo $j = 3$)

$$\begin{bmatrix} y_{11} & y_{12} & y_{13} & \cdots & y_{1n} \\ y_{21} & y_{22} & y_{23} & \cdots & y_{2n} \\ y_{31} & y_{32} & y_{33} & \cdots & y_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & y_{n3} & \cdots & y_{nn} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} y_{13} \\ y_{23} \\ y_{33} \\ \vdots \\ y_{n3} \end{bmatrix}$$

  es decir, solucionar $\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{i}}_{\cdot j}$ brinda como solución la j-ésima columna de $\mathbf{A}^{-1}$
- La descomposición **LU** permite eficientemente con complejidad $\mathcal{O}\left(n^2\right)$ calcular cada columna de la matriz inversa

Condicionamiento y matriz inversa

Tres caminos para probar condicionamiento de sistema de ecuaciones:

1. Escalar $\mathbf{A}$ de modo que máximo elemento por fila sea 1. Invertir matriz escalada. Si existen elementos en $\mathbf{A}^{-1}$ varios órdenes de magnitud mayores que 1 entonces el sistema está mal condicionado.
2. Multiplicar $\mathbf{A}$ por $\mathbf{A}^{-1}$ y verificar qué tan cerca se encuentra el resultado de $\mathbf{I}$
3. Invertir $\mathbf{A}^{-1}$ y comparar con $\mathbf{A}$

Para realizar las comparaciones se necesita el concepto de **norma**.

Normas vectoriales

La **norma** p de un vector $\underline{\mathbf{x}}$, o norma de Minkowski, de define como

$$\|\underline{\mathbf{x}}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

donde el caso $p = 2$ se conoce como norma **euclidea**, $p = 1$ se conoce como la norma de **cuadras de ciudad**, o con $p \to \infty$ la norma **magnitud-máxima** es

$$\|\underline{\mathbf{x}}\|_\infty = \max_{1 \le i \le n} |x_i|$$

Normas matriciales (1)

- Para matrices las normas se definen de dos maneras:
  1. considerando todos los elementos de la matriz por igual, o
  2. un cálculo para las filas (o columnas) y luego combinandolos
- La norma de **Frobenius** es

$$\|\mathbf{A}\|_2 = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}|^2}$$

Normas matriciales (2)

- La norma **columna-suma** se define como

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}| = \max_{1\leq j\leq n} \|\underline{\mathbf{a}}_{\cdot\mathbf{j}}\|_1$$

es decir, como la máxima norma de cuadras de ciudad de los vectores columna de la matriz.

- La norma **fila-suma** es

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| = \max_{1\leq i\leq n} \|\underline{\mathbf{a}}_{\mathbf{i}\cdot}\|_1$$

que es la máxima norma de cuadras de ciudad de los vectores fila de la matriz.

Normas matriciales (3)

- La norma 2 o norma **espectral** de una matriz $\mathbf{A}$ es

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\text{máx}}}$$

donde $\lambda_{\text{máx}}$ es el mayor valor propio de $\mathbf{A}^T\mathbf{A}$, es decir el mayor valor λ_i asociado a un vector propio $\underline{\mathbf{e}}_i$ que satisface

$$\left(\mathbf{A}^T\mathbf{A}\right)\underline{\mathbf{e}}_i = \lambda_i \underline{\mathbf{e}}_i$$

Número de condición de una matriz

- El número de condición de una matriz $\mathbf{A}$ se define como

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$$

- Para la matriz normalizada y usando $\|\cdot\|_\infty$ este número es siempre mayor que uno.
- Para el sistema $\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$, se cumple para la precisión $\Delta\underline{\mathbf{x}}$ de la solución

$$\frac{\|\Delta\underline{\mathbf{x}}\|}{\|\underline{\mathbf{x}}\|} \leq \text{cond}(\mathbf{A})\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}$$

donde $\Delta\mathbf{A}$ es el error de los coeficientes de la matriz.

Si los coeficientes de $\mathbf{A}$ tienen t dígitos de precisión y $\text{cond}(\mathbf{A}) = 10^c$ la solución $\underline{\mathbf{x}}$ es válida para $t - c$ dígitos.

Ejemplo: Número de condición (1)

Ejemplo

Evalúe la condición de la matriz de Hilbert dada por

$$\begin{bmatrix} 1 & 1/2 & 1/3 & \cdots & 1/n \\ 1/2 & 1/3 & 1/4 & \cdots & 1/(n+1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1/n & 1/(n+1) & 1/(n+2) & \cdots & 1/(2n-1) \end{bmatrix}$$

para el caso $n = 3$

Ejemplo: Número de condición (2)

Solución: Para el caso $n = 3$ la matriz de Hilbert es

$$\mathbf{A} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

la cual normalizada para que el máximo elemento por fila sea 1 se modifica en

$$\mathbf{A} = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1 & 2/3 & 1/2 \\ 1 & 3/4 & 3/5 \end{bmatrix}$$

Ejemplo: Número de condición (3)

Utilizando la norma fila-suma, se calculan primero las normas de cada fila:

$$
\begin{aligned}
\|\underline{\mathbf{a}}_{1\cdot}\|_1 &= 11/6 \approx 1,8333 \\
\|\underline{\mathbf{a}}_{2\cdot}\|_1 &= 13/6 \approx 2,1667 \\
\|\underline{\mathbf{a}}_{3\cdot}\|_1 &= 47/20 = 2,35
\end{aligned}
$$

y finalmente se toma el máximo para la norma $\|\mathbf{A}\|_\infty = 2,35$.
La inversa de la matriz escalada se calcula como

$$
\mathbf{A}^{-1} = \begin{bmatrix} 9 & -18 & 10 \\ -36 & 96 & -60 \\ 30 & -90 & 60 \end{bmatrix}
$$

Ejemplo: Número de condición (4)

y se obtiene que la norma la brinda la segunda fila:

$$\|\mathbf{A}^{-1}\|_\infty = |-36| + |96| + |-60| = 192$$

por lo que el número de condición es

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\|_\infty \|\mathbf{A}^{-1}\|_\infty = 2,35 \cdot 192 = 451,2$$

Puesto que $\text{cond}(\mathbf{A}) \gg 1$ la matriz es mal condicionada. Puesto que $\log 451,2 = 2,65$ esto indica que se pierden alrededor de 3 cifras significativas de la precisión en los coeficientes de $\mathbf{A}$ en el cálculo de $\underline{\mathbf{x}}$.

Refinamiento iterativo (1)

- Para reducir los errores, se utiliza un procedimiento iterativo que mejora la solución.
- Supóngase que $\underline{\mathbf{x}}$ es la solución exacta de

$$\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$$

sin embargo, con la solución numérica se obtiene

$$\underline{\tilde{\mathbf{x}}} = \underline{\mathbf{x}} + \delta\underline{\mathbf{x}}$$

con el error desconocido $\delta\underline{\mathbf{x}}$.

Refinamiento iterativo (2)

- Si se aplica $\underline{\tilde{\mathbf{x}}}$ al sistema se obtiene

$$\begin{aligned} \mathbf{A}\underline{\tilde{\mathbf{x}}} &= \underline{\tilde{\mathbf{b}}} \\ \mathbf{A}\left(\underline{\mathbf{x}} + \delta\underline{\mathbf{x}}\right) &= \underline{\mathbf{b}} + \delta\underline{\mathbf{b}} \\ \mathbf{A}\underline{\mathbf{x}} + \mathbf{A}\delta\underline{\mathbf{x}} &= \underline{\mathbf{b}} + \delta\underline{\mathbf{b}} \end{aligned}$$

  por lo que

$$\mathbf{A}\delta\underline{\mathbf{x}} = \delta\underline{\mathbf{b}} = \mathbf{A}\left(\underline{\mathbf{x}} + \delta\underline{\mathbf{x}}\right) - \underline{\mathbf{b}}$$

- Es decir, se puede calcular $\delta\underline{\mathbf{b}}$ y luego con él encontrar $\delta\underline{\mathbf{x}}$ para mejorar la solución

$$\underline{\mathbf{x}} = \underline{\tilde{\mathbf{x}}} - \delta\underline{\mathbf{x}}$$

Refinamiento iterativo

Resumen

1. Para resolver $\mathbf{A}\underline{\mathbf{x}} = \underline{\mathbf{b}}$, descomponga $\mathbf{A} = \mathbf{LU}$
2. Resuelva $\mathbf{LU}\underline{\mathbf{x}} = \underline{\mathbf{b}}$ con $\mathbf{L}\underline{\mathbf{y}} = \underline{\mathbf{b}}$ y luego $\mathbf{U}\underline{\tilde{\mathbf{x}}} = \underline{\mathbf{y}}$
3. Calcule $\underline{\tilde{\mathbf{b}}} = \mathbf{A}\underline{\tilde{\mathbf{x}}}$
4. Calcule $\delta\underline{\mathbf{b}} = \underline{\tilde{\mathbf{b}}} - \underline{\mathbf{b}}$
5. Si $\|\delta\underline{\mathbf{b}}\| < \tau$ con el umbral τ, termine
6. Resuelva $\mathbf{LU}\delta\underline{\mathbf{x}} = \delta\underline{\mathbf{b}}$ con $\mathbf{L}\delta\underline{\mathbf{y}} = \delta\underline{\mathbf{b}}$ y luego $\mathbf{U}\delta\underline{\mathbf{x}} = \delta\underline{\mathbf{y}}$
7. Refine el valor $\underline{\tilde{\mathbf{x}}} \leftarrow \underline{\tilde{\mathbf{x}}} - \delta\underline{\mathbf{x}}$
8. Repetir desde paso 3

Resumen

Este documento ha sido elaborado con software libre incluyendo LaTeX, Beamer, GNUPlot, GNU/Octave, XFig, Inkscape, LTI-Lib-2, GNU-Make y Subversion en GNU/Linux