# Top 10 algoritmos

Camarena Perez, Victor Daniel

IMCA
Universidad Nacional de Ingeniería

12 de septiembre del 2018

# Contenido

# Objetivo:

Mostrar la matemática por tras de la ciencia y tecnología.

# Objetivo:

**Mostrar la matemática por tras de la ciencia y tecnología.**


# Objetivo:

**Mostrar la matemática por tras de la ciencia y tecnología.**


Un problema que enfrenta cualquiera que compile tal lista es definir qué se entiende por "algoritmo". ¿Dónde se traza la línea entre un algoritmo y una técnica? Para un ejemplo simple,

**¿poner una función racional en forma de fracción parcial es un algoritmo?**

**El algoritmo de Newton**: Se pide hallar el cero de una función

$$x : f(x) = 0.$$

Un problema que enfrenta cualquiera que compile tal lista es definir qué se entiende por "algoritmo". ¿Dónde se traza la línea entre un algoritmo y una técnica? Para un ejemplo simple,

**¿poner una función racional en forma de fracción parcial es un algoritmo?**

**El algoritmo de Newton**: Se pide hallar el cero de una función

$$x\colon f(x) = 0.$$

Fije $a < x_0 < b$, para cada $n \geq 0$ calcule

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Un problema que enfrenta cualquiera que compile tal lista es definir qué se entiende por "algoritmo". ¿Dónde se traza la línea entre un algoritmo y una técnica? Para un ejemplo simple,

**¿poner una función racional en forma de fracción parcial es un algoritmo?**

**El algoritmo de Newton**: Se pide hallar el cero de una función

$$x\colon f(x) = 0.$$

Fije $a < x_0 < b$, para cada $n \geq 0$ calcule

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

# Línea temporal

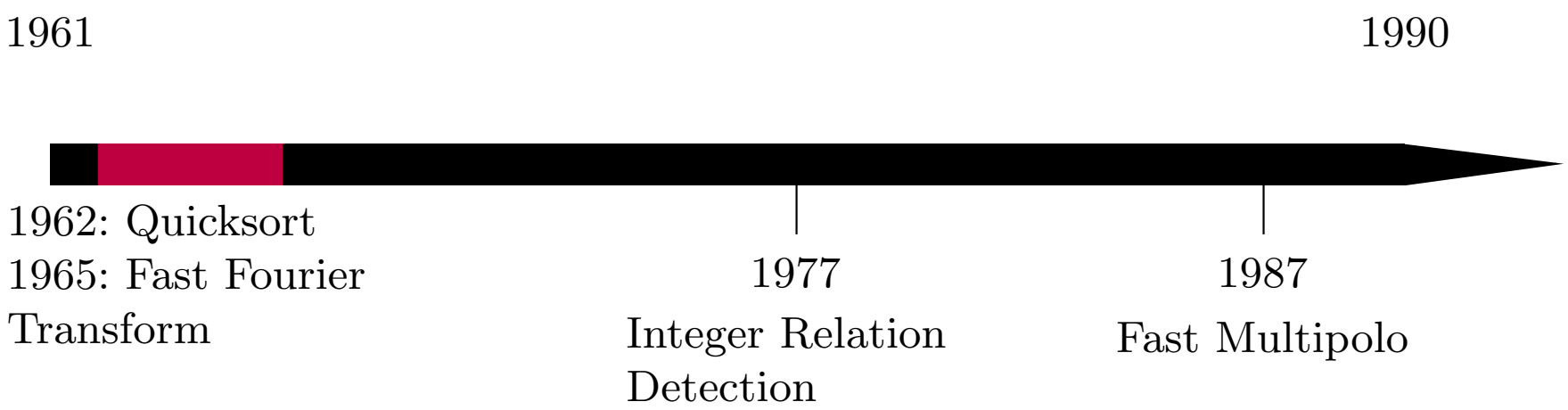
1940 1961

1946: Monte Carlo
1947: Simplex

1950: Krylov
1951: Matrix Decomposition

1957: Fortran
1959-61: QR-algorithm

# Línea temporal



# Simplex method (1946)

La programación lineal domina el mundo de la industria, donde la supervivencia económica depende de la capacidad de optimizar dentro de las limitaciones presupuestarias.

**El Problema de la Dieta**: Supongamos que hay

- 3 alimentos disponibles, maíz, leche y pan
- restricciones en el número de calorías (entre 2000 y 2250) y la cantidad de vitamina A (entre 5,000 y 50,000)
-

| Comida | costo por porción | Vitamina A | Calorías |
|---|---|---|---|
| Maíz | \$0,18 | 107 | 72 |
| 2 %Leche | \$0,23 | 500 | 121 |
| Pan | \$0,05 | 0 | 65 |

La programación lineal domina el mundo de la industria, donde la supervivencia económica depende de la capacidad de optimizar dentro de las limitaciones presupuestarias.

**El Problema de la Dieta**: Supongamos que hay

- 3 alimentos disponibles, maíz, leche y pan
- restricciones en el número de calorías (entre 2000 y 2250) y la cantidad de vitamina A (entre 5,000 y 50,000)
-

| Comida | costo por porción | Vitamina A | Calorías |
|---|---|---|---|
| Maíz | \$0,18 | 107 | 72 |
| 2 %Leche | \$0,23 | 500 | 121 |
| Pan | \$0,05 | 0 | 65 |

Formulación como problema de Programación Lineal (PL):

$$\begin{aligned} &\text{mín } c^t x \\ &s.a.\ Ax \gtrless b \\ &x \geq 0 \end{aligned}$$

La programación lineal domina el mundo de la industria, donde la supervivencia económica depende de la capacidad de optimizar dentro de las limitaciones presupuestarias.

**El Problema de la Dieta**: Supongamos que hay

- 3 alimentos disponibles, maíz, leche y pan
- restricciones en el número de calorías (entre 2000 y 2250) y la cantidad de vitamina A (entre 5,000 y 50,000)
-

| Comida | costo por porción | Vitamina A | Calorías |
|---|---|---|---|
| Maíz | \$0,18 | 107 | 72 |
| 2 %Leche | \$0,23 | 500 | 121 |
| Pan | \$0,05 | 0 | 65 |

Formulación como problema de Programación Lineal (PL):

$$\begin{aligned} &\text{mín } c^t x \\ &s.a.\ Ax \gtrless b \\ &x \geq 0 \end{aligned}$$

# Simplex method (1946)

Importancia Práctica:

- En los 40', muchas organizaciones estaban muy ávidas de soluciones a problemas LP.
- Las compañías petroleras y químicas: optimizar la combinación de productos de múltiples fuentes o sitios múltiples.
- Las compañías de transporte y el ejército: los problemas de transporte y logística.

# Simplex method (1946)

Importancia Práctica:

- En los 40', muchas organizaciones estaban muy ávidas de soluciones a problemas LP.
- Las compañías petroleras y químicas: optimizar la combinación de productos de múltiples fuentes o sitios múltiples.
- Las compañías de transporte y el ejército: los problemas de transporte y logística.
- LP incluso se puede aplicar para mantener la confidencialidad de las estadísticas del gobierno.

Importancia Práctica:

- En los 40', muchas organizaciones estaban muy ávidas de soluciones a problemas LP.
- Las compañías petroleras y químicas: optimizar la combinación de productos de múltiples fuentes o sitios múltiples.
- Las compañías de transporte y el ejército: los problemas de transporte y logística.
- LP incluso se puede aplicar para mantener la confidencialidad de las estadísticas del gobierno.

# Krylov subspace iteration methods (1950)

Planteemos la tarea, aparentemente simple, de resolver la ecuación

$$Ax = b$$

cuando A es una gran matriz.

Típicamente se usan métodos iterativos

$$Kx_{i+1} = Kx_i + b - Ax_i$$

donde $K$ es una matriz más simple que es idealmente "cercana" a $A$.

# Krylov subspace iteration methods (1950)

Planteemos la tarea, aparentemente simple, de resolver la ecuación

$$Ax = b$$

cuando A es una gran matriz.

Típicamente se usan métodos iterativos

$$Kx_{i+1} = Kx_i + b - Ax_i$$

donde $K$ es una matriz más simple que es idealmente "cercana" a $A$.

Los subespacios de Krylov están generados por las potencias de una matriz aplicada a un vector incial "resto"

$$r_0 = b - Ax_0.$$

# Krylov subspace iteration methods (1950)

Planteemos la tarea, aparentemente simple, de resolver la ecuación

$$Ax = b$$

cuando A es una gran matriz.

Típicamente se usan métodos iterativos

$$Kx_{i+1} = Kx_i + b - Ax_i$$

donde $K$ es una matriz más simple que es idealmente "cercana" a $A$.

Los subespacios de Krylov están generados por las potencias de una matriz aplicada a un vector incial "resto"

$$r_0 = b - Ax_0.$$

Alogitmos:

- **Lanczos**: Diseñó una forma ingeniosa de generar una base ortogonal para dicho subespacio cuando la matriz es simétrica.
- **Hestenes y Stiefel**: El método de gradiente conjugado, para sistemas (simétricos) definidos positivos.

| Método | Número de iteraciones |
|---|---|
| Gauss Seidel | 208 000 |
| Block succesive overrelax methods | 765 |
| Incomplete Cholesky conjugate gradients | 25 |

Cuadro : Resultados de Kershaw para el problema de fusión [4].

Alogitmos:

- **Lanczos**: Diseñó una forma ingeniosa de generar una base ortogonal para dicho subespacio cuando la matriz es simétrica.
- **Hestenes y Stiefel**: El método de gradiente conjugado, para sistemas (simétricos) definidos positivos.

| Método | Número de iteraciones |
|---|---|
| Gauss Seidel | 208 000 |
| Block succesive overrelax methods | 765 |
| Incomplete Cholesky conjugate gradients | 25 |

Cuadro : Resultados de Kershaw para el problema de fusión [4].

# Decompositional approach to matrix computations (1951)

1951: P. Dwyer publica el primer libro de álgebra lineal numérica.

1954: A. Householder publica Principies of numerical anaylisis.

# Decompositional approach to matrix computations (1951)

1951: P. Dwyer publica el primer libro de álgebra lineal numérica.
1954: A. Householder publica Principies of numerical anaylisis.



or a non-diagonal pivot, is used. The coefficient serving as a pivot should be different from zero.

By dividing the first equation by $a_{11}$ and letting $a_{1j}/a_{11} = b_{1j}$ as in (6.3.1), the equations (4.1.2) become

$$\begin{aligned} a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= a_{25} \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= a_{36} \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= a_{45} \\ x_1 + b_{12}x_2 + b_{13}x_3 + b_{14}x_4 &= b_{15}. \end{aligned} \tag{1}$$

Multiply the last equation by $a_{21}$ and subtract from the first equation, by $a_{31}$ and subtract from the second equation, by $a_{41}$ and subtract from the third equation, and get the three equations

$$\begin{aligned} g_{22\cdot1}x_2 + g_{23\cdot1}x_3 + g_{24\cdot1}x_4 &= g_{25\cdot1} \\ g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1} \\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1} \end{aligned} \tag{2}$$

with

$$g_{ij\cdot1} = a_{ij} - a_{i1}b_{1j}. \tag{3}$$

Now

$$\frac{g_{ij\cdot1}}{g_{i\ell\cdot1}} = \frac{a_{ij} - a_{i1}b_{1j}}{a_{i\ell} - a_{i1}b_{1\ell}} = \frac{\dfrac{a_{ij}}{a_{i\ell}} - \dfrac{a_{i1}}{a_{i\ell}}b_{1j}}{\dfrac{a_{i\ell}}{a_{i\ell}} - \dfrac{a_{i1}}{a_{i\ell}}b_{1\ell}} = \frac{b_{ij} - b_{i1}b_{1j}}{1 - b_{i1}b_{1\ell}} = b_{ij\cdot1} \tag{4}$$

as defined by (6.3.4). We divide the first equation of (2) by $g_{22\cdot1}$ and place the results in the bottom row to get

$$\begin{aligned} g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1} \\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1} \\ x_2 + b_{23\cdot1}x_3 + b_{24\cdot1}x_4 &= b_{25\cdot1}. \end{aligned} \tag{5}$$

We eliminate as before and obtain

$$\begin{aligned} g_{33\cdot12}x_3 + g_{34\cdot12}x_4 &= g_{35\cdot12} \\ g_{43\cdot12}x_3 + g_{44\cdot12}x_4 &= g_{45\cdot12} \end{aligned} \tag{6}$$

Figure 1. This page from *Linear Computations* shows that Paul Dwyer's approach begins with a system of scalar equations. Courtesy of John Wiley & Sons.



unknowns is equivalent to the operation of multiplying the system by a particular unit lower triangular matrix—a matrix, in fact, whose off-diagonal non-null elements are all in the same column. The product of all these unit lower triangular matrices is again a unit lower triangular matrix, and hence the entire process of elimination (as opposed to that of back substitution) is equivalent to that of multiplying the system by a suitably chosen unit lower triangular matrix. Since the matrix of the resulting system is clearly upper triangular, these considerations constitute another proof of the possibility of factorizing $A$ into a unit lower triangular matrix and an upper triangular matrix.

For the system

$$Ax = y,$$

after eliminating any one of the variables, the effect to that point is that of having selected a unit lower triangular matrix of the form

$$\begin{pmatrix} L_{11} & 0 \\ L_{12} & I_{22} \end{pmatrix}$$

where $L_{11}$ is itself unit lower triangular, in such a way that $A$ is factored

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix}\begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix}. \tag{2.21.1}$$

with $W_{11}$ upper triangular but $M_{22}$ not. Hence

$$M_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}. \tag{2.21.2}$$

The original system has at this stage been replaced by the system

$$\begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \tag{2.21.3}$$

where

$$\begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix}\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = y. \tag{2.21.4}$$

The matrices $L_{11}$ and $L_{21}$ are not themselves written down. The partial system

$$M_{22}x_2 = z_2$$

represents those equations from which further elimination remains to be done, and this can be treated independently of the other equations of the system, which fact explains why it is unnecessary to obtain the $L$ matrices explicitly.

If the upper left-hand element of $M_{22}$ vanishes, this cannot be used in the next step of the elimination, and it is not advantageous to use it when it is small. Hence rows or columns, or both, in $M_{22}$ must be

Figure 2. On this page from *Principles of Numerical Analysis*, Alston Householder uses partitioned matrices and LU decomposition. Courtesy of McGraw-Hill.

# Decompositional approach to matrix computations (1951)

1951: P. Dwyer publica el primer libro de álgebra lineal numérica.
1954: A. Householder publica Principies of numerical anaylisis.



or a non-diagonal pivot, is used. The coefficient serving as a pivot should be different from zero.

By dividing the first equation by $a_{11}$ and letting $a_{1j}/a_{11} = b_{1j}$ as in (6.3.1), the equations (4.1.2) become

$$\begin{aligned} a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= a_{25} \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= a_{36} \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= a_{45} \\ x_1 + b_{12}x_2 + b_{13}x_3 + b_{14}x_4 &= b_{15}. \end{aligned} \tag{1}$$

Multiply the last equation by $a_{21}$ and subtract from the first equation, by $a_{31}$ and subtract from the second equation, by $a_{41}$ and subtract from the third equation, and get the three equations

$$\begin{aligned} g_{22\cdot1}x_2 + g_{23\cdot1}x_3 + g_{24\cdot1}x_4 &= g_{25\cdot1} \\ g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1} \\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1} \end{aligned} \tag{2}$$

with

$$g_{ij\cdot1} = a_{ij} - a_{i1}b_{1j}. \tag{3}$$

Now

$$\frac{g_{ij\cdot1}}{g_{i\ell\cdot1}} = \frac{a_{ij} - a_{i1}b_{1j}}{a_{i\ell} - a_{i1}b_{1\ell}} = \frac{\dfrac{a_{ij}}{a_{i\ell}} - \dfrac{a_{i1}}{a_{i\ell}}b_{1j}}{\dfrac{a_{i\ell}}{a_{i\ell}} - \dfrac{a_{i1}}{a_{i\ell}}b_{1\ell}} = \frac{b_{ij} - b_{i1}b_{1j}}{1 - b_{i1}b_{1\ell}} = b_{ij\cdot1} \tag{4}$$

as defined by (6.3.4). We divide the first equation of (2) by $g_{22\cdot1}$ and place the results in the bottom row to get

$$\begin{aligned} g_{32\cdot1}x_2 + g_{33\cdot1}x_3 + g_{34\cdot1}x_4 &= g_{35\cdot1} \\ g_{42\cdot1}x_2 + g_{43\cdot1}x_3 + g_{44\cdot1}x_4 &= g_{45\cdot1} \\ x_2 + b_{23\cdot1}x_3 + b_{24\cdot1}x_4 &= b_{25\cdot1}. \end{aligned} \tag{5}$$

We eliminate as before and obtain

$$\begin{aligned} g_{33\cdot12}x_3 + g_{34\cdot12}x_4 &= g_{35\cdot12} \\ g_{43\cdot12}x_3 + g_{44\cdot12}x_4 &= g_{45\cdot12} \end{aligned} \tag{6}$$

Figure 1. This page from *Linear Computations* shows that Paul Dwyer's approach begins with a system of scalar equations. Courtesy of John Wiley & Sons.



unknowns is equivalent to the operation of multiplying the system by a particular unit lower triangular matrix—a matrix, in fact, whose off-diagonal non-null elements are all in the same column. The product of all these unit lower triangular matrices is again a unit lower triangular matrix, and hence the entire process of elimination (as opposed to that of back substitution) is equivalent to that of multiplying the system by a suitably chosen unit lower triangular matrix. Since the matrix of the resulting system is clearly upper triangular, these considerations constitute another proof of the possibility of factorizing $A$ into a unit lower triangular matrix and an upper triangular matrix.

For the system

$$Ax = y,$$

after eliminating any one of the variables, the effect to that point is that of having selected a unit lower triangular matrix of the form

$$\begin{pmatrix} L_{11} & 0 \\ L_{12} & I_{22} \end{pmatrix}$$

where $L_{11}$ is itself unit lower triangular, in such a way that $A$ is factored

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix}\begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix}. \tag{2.21.1}$$

with $W_{11}$ upper triangular but $M_{22}$ not. Hence

$$M_{22} = A_{22} - A_{21}A_{11}^{-1}A_{12}. \tag{2.21.2}$$

The original system has at this stage been replaced by the system

$$\begin{pmatrix} W_{11} & W_{12} \\ 0 & M_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \tag{2.21.3}$$

where

$$\begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{22} \end{pmatrix}\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = y. \tag{2.21.4}$$

The matrices $L_{11}$ and $L_{21}$ are not themselves written down. The partial system

$$M_{22}x_2 = z_2$$

represents those equations from which further elimination remains to be done, and this can be treated independently of the other equations of the system, which fact explains why it is unnecessary to obtain the $L$ matrices explicitly.

If the upper left-hand element of $M_{22}$ vanishes, this cannot be used in the next step of the elimination, and it is not advantageous to use it when it is small. Hence rows or columns, or both, in $M_{22}$ must be

Figure 2. On this page from *Principles of Numerical Analysis*, Alston Householder uses partitioned matrices and LU decomposition. Courtesy of McGraw-Hill.

# Decompositional approach to matrix computations (1951)

El cálculo de las órbitas planetarias (Gauss-Cholesky):

$$\begin{aligned}\varphi(x) &= x^t A x \\ &= \left(r_1^t x\right)^2 + \cdots + \left(r_n^t x\right)^2 \\ &= \rho_1(x)^2 + \cdots + \rho_n(x)^2\end{aligned}$$

1961-65: J. Wilkinson. El enfoque descomposicional estaba firmemente establecido.

# Decompositional approach to matrix computations (1951)

El cálculo de las órbitas planetarias (Gauss-Cholesky):

$$\begin{aligned}\varphi(x) &= x^t A x \\ &= \left(r_1^t x\right)^2 + \cdots + \left(r_n^t x\right)^2 \\ &= \rho_1(x)^2 + \cdots + \rho_n(x)^2\end{aligned}$$

1961-65: J. Wilkinson. El enfoque descomposicional estaba firmemente establecido.

Beneficios:

1. Resuelve no uno sino muchos problemas. A pesar de su costo computacional, puede reusarse para resolver nuevos problemas.
2. Facilita el análisis de errores de redondeo, uno de los grandes errores del álgebra lineal numérica.
3. Ha permitido a los desarrolladores de software producir paquetes de matriz flexibles y eficientes (LAPACK).

El cálculo de las órbitas planetarias (Gauss-Cholesky):

$$\begin{aligned}\varphi(x) &= x^t A x \\ &= \left(r_1^t x\right)^2 + \cdots + \left(r_n^t x\right)^2 \\ &= \rho_1(x)^2 + \cdots + \rho_n(x)^2\end{aligned}$$

1961-65: J. Wilkinson. El enfoque descomposicional estaba firmemente establecido.

Beneficios:

1. Resuelve no uno sino muchos problemas. A pesar de su costo computacional, puede reusarse para resolver nuevos problemas.
2. Facilita el análisis de errores de redondeo, uno de los grandes errores del álgebra lineal numérica.
3. Ha permitido a los desarrolladores de software producir paquetes de matriz flexibles y eficientes (LAPACK).

# La Descomposición Cholesky (1809-10)

Sea $A$ una matriz (simétrica) definida positiva de orden $n$.

**Descomposición Cholesky de $A$**

Existe una única matriz triangular superior $R$, con elementos en la diagonal positivos, tal que

$$A = R^t R.$$

Aplicaciones:

1. Sistemas lineales definidos positivos.
2. Cálculo de cantidades usuales en estadística.

# La Descomposición Cholesky (1809-10)

Sea $A$ una matriz (simétrica) definida positiva de orden $n$.

**Descomposición Cholesky de $A$**

Existe una única matriz triangular superior $R$, con elementos en la diagonal positivos, tal que

$$A = R^t R.$$

Aplicaciones:

1. Sistemas lineales definidos positivos.
2. Cálculo de cantidades usuales en estadística.

# La Descomposición LU (1857)

Sea $A$ una matriz de orden $n$ con rango máximo.

**Descomposición LU de $A$**

Existen unas matrices de permutación $P$ y $Q$ tales que

$$P^t AQ = LU$$

donde $L$ es una matriz triangular inferior y $U$ es una matriz triangular superior.

# La Descomposición LU (1857)

Sea $A$ una matriz de orden $n$ con rango máximo.

**Descomposición LU de $A$**

Existen unas matrices de permutación $P$ y $Q$ tales que

$$P^t AQ = LU$$

donde $L$ es una matriz triangular inferior y $U$ es una matriz triangular superior.

Aplicaciones:

1. Resolución de sistemas lineales.
2. Diversas otras: Cómputo del vector estado-estable de una cadena de Markov (PageRank).

# La Descomposición LU (1857)

Sea $A$ una matriz de orden $n$ con rango máximo.

**Descomposición LU de $A$**

Existen unas matrices de permutación $P$ y $Q$ tales que

$$P^t AQ = LU$$

donde $L$ es una matriz triangular inferior y $U$ es una matriz triangular superior.

Aplicaciones:

1. Resolución de sistemas lineales.
2. Diversas otras: Cómputo del vector estado-estable de una cadena de Markov (PageRank).

# La Descomposición QR (Schmidt 1907-Laplace 1820)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición QR de $A$**

Existen matriz ortogonal $Q$ tal que

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

donde $R$ es triangular superior con elementos en la diagonal no negativos (positivos si $A$ es de rango $n$).

# La Descomposición QR (Schmidt 1907-Laplace 1820)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición QR de $A$**

Existen matriz ortogonal $Q$ tal que

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

donde $R$ es triangular superior con elementos en la diagonal no negativos (positivos si $A$ es de rango $n$).

Si se particiona $Q = (Q_A, Q_1)$, de modo que $Q_A$ tenga sus $n$ primeras columnas, se tiene la Factorización QR de $A$ dada por

$$A = Q_A R.$$

# La Descomposición QR (Schmidt 1907-Laplace 1820)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición QR de $A$**

Existen matriz ortogonal $Q$ tal que

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

donde $R$ es triangular superior con elementos en la diagonal no negativos (positivos si $A$ es de rango $n$).

Si se particiona $Q = (Q_A, Q_1)$, de modo que $Q_A$ tenga sus $n$ primeras columnas, se tiene la Factorización QR de $A$ dada por

$$A = Q_A R.$$

Aplicaciones:

1. Si $A$ es de rango máximo entonces $Q_A$ forman una base ortonormal para el espacio columna de $A$.
2. Principalmente geométricas: mínimos cuadrados.

# La Descomposición QR (Schmidt 1907-Laplace 1820)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición QR de $A$**

Existen matriz ortogonal $Q$ tal que

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

donde $R$ es triangular superior con elementos en la diagonal no negativos (positivos si $A$ es de rango $n$).

Si se particiona $Q = (Q_A, Q_1)$, de modo que $Q_A$ tenga sus $n$ primeras columnas, se tiene la Factorización QR de $A$ dada por

$$A = Q_A R.$$

Aplicaciones:

1. Si $A$ es de rango máximo entonces $Q_A$ forman una base ortonormal para el espacio columna de $A$.
2. Principalmente geométricas: mínimos cuadrados.

# Descomposición Espectral (1829)

Sea $A$ una matriz simétrica de orden $n$.

**Descomposición espectral de $A$**

Existe una matriz ortogonal $V$ tal que

$$A = V \Lambda V^t$$

donde $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

# Descomposición Espectral (1829)

Sea $A$ una matriz simétrica de orden $n$.

**Descomposición espectral de $A$**

Existe una matriz ortogonal $V$ tal que

$$A = V\Lambda V^t$$

donde $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

Si $v_i$ denota la $i$-ésima columna de $V$ entonces

$$Av_i = \lambda_i v_i.$$

# Descomposición Espectral (1829)

Sea $A$ una matriz simétrica de orden $n$.

**Descomposición espectral de $A$**

Existe una matriz ortogonal $V$ tal que

$$A = V\Lambda V^t$$

donde $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

Si $v_i$ denota la $i$-ésima columna de $V$ entonces

$$Av_i = \lambda_i v_i.$$

Aplicaciones:

1. Donde se requiera una base de autovectores.
2. Importancia teórica.

# Descomposición Espectral (1829)

Sea $A$ una matriz simétrica de orden $n$.

**Descomposición espectral de $A$**

Existe una matriz ortogonal $V$ tal que

$$A = V\Lambda V^t$$

donde $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

Si $v_i$ denota la $i$-ésima columna de $V$ entonces

$$Av_i = \lambda_i v_i.$$

Aplicaciones:

1. Donde se requiera una base de autovectores.
2. Importancia teórica.

# Descomposición Schur (1909)

Sea $A$ una matriz compleja de orden $n$.

**Descomposición Schur de $A$**

Existe una matriz unitaria $U$ tal que

$$A = UTU^*$$

donde $T$ es triangular superior y los elementos de su diagonal son los autovalores de $A$.

# Descomposición Schur (1909)

Sea $A$ una matriz compleja de orden $n$.

**Descomposición Schur de $A$**

Existe una matriz unitaria $U$ tal que

$$A = UTU^*$$

donde $T$ es triangular superior y los elementos de su diagonal son los autovalores de $A$.

Aplicaciones:

1. Cálculo de autovalores y autovectores.
2. Donde se requiera tanto autovalores como autovectores: la ecuación de Sylvester.

# Descomposición Schur (1909)

Sea $A$ una matriz compleja de orden $n$.

**Descomposición Schur de $A$**

Existe una matriz unitaria $U$ tal que

$$A = UTU^*$$

donde $T$ es triangular superior y los elementos de su diagonal son los autovalores de $A$.

Aplicaciones:

1. Cálculo de autovalores y autovectores.
2. Donde se requiera tanto autovalores como autovectores: la ecuación de Sylvester.

# Descomposición Valores Singulares (1873-74)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición singular de $A$**

Existen matrices ortogonales $U$ y $V$ tales que

$$U^t A V = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

donde $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ con $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

# Descomposición Valores Singulares (1873-74)

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición singular de $A$**

Existen matrices ortogonales $U$ y $V$ tales que

$$U^t A V = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

donde $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ con $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

Si $U_A$ consiste de las primeras $n$ columnas de $U$ entonces

$$A = U_A \Sigma V^T$$

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición singular de $A$**

Existen matrices ortogonales $U$ y $V$ tales que

$$U^t A V = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

donde $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ con $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

Si $U_A$ consiste de las primeras $n$ columnas de $U$ entonces

$$A = U_A \Sigma V^T = \sum_{i=1}^{n} \sigma_i u^i v^{iT},$$

la cual es llamada Factorización Singular de $A$.

Sea $A$ una $m \times n$ matriz real con $m \geq n$.

**Descomposición singular de $A$**

Existen matrices ortogonales $U$ y $V$ tales que

$$U^t A V = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

donde $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ con $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

Si $U_A$ consiste de las primeras $n$ columnas de $U$ entonces

$$A = U_A \Sigma V^T = \sum_{i=1}^{n} \sigma_i u^i v^{iT},$$

la cual es llamada Factorización Singular de $A$.

Aplicaciones

1. $rank(A) = \text{máx}\{r : \sigma_r \neq 0\}$
2. $ker(A) = span\{v_{r+1}, \ldots, v_n\}$
3. $Im(A) = span\{u_1, \ldots, u_n\}$
4. $\|A\|_2 = \sigma_1 = \sigma_{\text{máx}}$
5. $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_n^2}$

**¡Cálculo de Tablas de Mortalidad!**

# The Fortran I compiler (1957)

Finalmente, los científicos podrían decirle a la computadora lo que querían que hiciera, sin tener que descender al **inframundo** del código máquina.

El equipo de IBM, liderado por Backus, no solo desarrolló el compilador sino que también diseñó el lenguaje Fortran. Él cual ha influenciado a los lenguajes actuales más populares: Matlab, C, Java.

# The Fortran I compiler (1957)

Finalmente, los científicos podrían decirle a la computadora lo que querían que hiciera, sin tener que descender al **inframundo** del código máquina.

El equipo de IBM, liderado por Backus, no solo desarrolló el compilador sino que también diseñó el lenguaje Fortran. Él cual ha influenciado a los lenguajes actuales más populares: Matlab, C, Java.

**El compilador Fortran 1** era bastante pequeño para los estándares actuales: Consistió en 23,500 instrucciones en lenguaje ensamblador. John Backus, mucho después, se refirió:

**"Produjo código de tal eficiencia que sus salidas sorprenderían a los programadores que lo estudiaron"**

Finalmente, los científicos podrían decirle a la computadora lo que querían que hiciera, sin tener que descender al **inframundo** del código máquina.

El equipo de IBM, liderado por Backus, no solo desarrolló el compilador sino que también diseñó el lenguaje Fortran. Él cual ha influenciado a los lenguajes actuales más populares: Matlab, C, Java.

**El compilador Fortran 1** era bastante pequeño para los estándares actuales: Consistió en 23,500 instrucciones en lenguaje ensamblador. John Backus, mucho después, se refirió:

**"Produjo código de tal eficiencia que sus salidas sorprenderían a los programadores que lo estudiaron"**

# The Fortran I compiler (1957)

Dos características del compilador:

- Los bloques básicos: “un tramo de programa que tiene un solo punto de entrada y un único punto de salida”
- Registro real simbólico: son nombres de variables que el compilador usa en una forma intermedia del código que se generará.

Ténicas de optimización:

1. Expresiones de análisis
2. DO optimizaciones de bucle y cálculos de subíndices
3. Asignación de registro

# The Fortran I compiler (1957)

Dos características del compilador:

- Los bloques básicos: “un tramo de programa que tiene un solo punto de entrada y un único punto de salida”
- Registro real simbólico: son nombres de variables que el compilador usa en una forma intermedia del código que se generará.

Ténicas de optimización:

1. Expresiones de análisis
2. DO optimizaciones de bucle y cálculos de subíndices
3. Asignación de registro

# QR algorithm (1961-62)

El problema del cálculo de autovalores de una matriz cuadrada:

1. Calcule los coeficientes del polinomio característico.
2. Halle las raíces del polinomio característico.

¡El Algoritmo QR no sigue estos pasos!

# QR algorithm (1961-62)

El problema del cálculo de autovalores de una matriz cuadrada:

1. Calcule los coeficientes del polinomio característico.
2. Halle las raíces del polinomio característico.

¡El Algoritmo QR no sigue estos pasos!

**Algoritmo QR Básico**
**Entrada**: Una matriz $A \in \mathbb{C}^{n \times n}$
**Salida**: Matrices $U$ y $T$ tales que $A = UTU^*$
  Sean $A_0 = A$ y $U_0 = I$
  **para** $k = 1, \ldots$ **hacer**
    Compute la factorización QR: $A_{k-1} =: Q_k R_k$
    Sean $A_k = R_k Q_k$ y $U_k = U_{k-1} Q_k$
  **fin para**
  Return: $T = A_\infty$ y $U = U_\infty$

El problema del cálculo de autovalores de una matriz cuadrada:

1. Calcule los coeficientes del polinomio característico.
2. Halle las raíces del polinomio característico.

¡El Algoritmo QR no sigue estos pasos!

**Algoritmo QR Básico**
**Entrada**: Una matriz $A \in \mathbb{C}^{n \times n}$
**Salida**: Matrices $U$ y $T$ tales que $A = UTU^*$
Sean $A_0 = A$ y $U_0 = I$
**para** $k = 1, \ldots$ **hacer**
Compute la factorización QR: $A_{k-1} =: Q_k R_k$
Sean $A_k = R_k Q_k$ y $U_k = U_{k-1} Q_k$
**fin para**
Return: $T = A_\infty$ y $U = U_\infty$

# QR algorithm (1961-62)

**Las dos fases del algoritmo**:

1. Reducción de Hessenberg: compute la forma Hessenberg $H_A$ de la matriz $A$.
2. Algoritmo QR Hessenberg: Aplique el Algorimto QR a la matriz $H_A$.

**Las dos fases del algoritmo**:

1. Reducción de Hessenberg: compute la forma Hessenberg $H_A$ de la matriz $A$.
2. Algoritmo QR Hessenberg: Aplique el Algorimto QR a la matriz $H_A$.

**Aceleración con shifts y deflación**:
Sea $\lambda$ un valor propio de la matriz de Hessenberg irreducible $H$ entonces

$$H - \lambda I = QR$$
$$RQ + \lambda I = Q^* H Q$$

**Las dos fases del algoritmo**:

1. Reducción de Hessenberg: compute la forma Hessenberg $H_A$ de la matriz $A$.
2. Algoritmo QR Hessenberg: Aplique el Algorimto QR a la matriz $H_A$.

**Aceleración con shifts y deflación**:
Sea $\lambda$ un valor propio de la matriz de Hessenberg irreducible $H$ entonces

$$H - \lambda I = QR$$
$$RQ + \lambda I = Q^* HQ$$

# Quicksort algorithm (1962)

Desafío: ordenar rápidamente $N$ objetos (números).

**El algoritmo de Hoare**:

1. Seleccione un elemento "pivote"
2. Separe el resto en pilas de elementos "menores" y "mayores", en comparación con el pivote.
3. Repita este procedimiento en cada pila.

# Quicksort algorithm (1962)

Desafío: ordenar rápidamente $N$ objetos (números).

**El algoritmo de Hoare**:

1. Seleccione un elemento "pivote"
2. Separe el resto en pilas de elementos "menores" y "mayores", en comparación con el pivote.
3. Repita este procedimiento en cada pila.

Quicksort se ejecuta en promedio con complejidad computacional $\mathcal{O}(N \log N)$.

# Quicksort algorithm (1962)

Desafío: ordenar rápidamente $N$ objetos (números).

**El algoritmo de Hoare**:

1. Seleccione un elemento "pivote"
2. Separe el resto en pilas de elementos "menores" y "mayores", en comparación con el pivote.
3. Repita este procedimiento en cada pila.

Quicksort se ejecuta en promedio con complejidad computacional $\mathcal{O}(N \log N)$.

**La Transformada Discreta de Fourier (DFT)**:

$$X = (X(0), \ldots, X(N-1)) \;\mapsto\; \hat{X} = (\hat{X}(0), \ldots, \hat{X}(N-1)),$$

donde

$$\hat{X}(k) = \sum_{j=0}^{N-1} X(j) W_N^{jk}, \quad k = 0, \ldots, N-1$$

con $W_N = \exp\left(\frac{2\pi}{N}\sqrt{-1}\right)$.

El cómputo directo de la DFT requeriría $N^2$ operaciones. En cambio, la **Transformada Rápida de Fourier (FFT)** es un algoritmo para calcular la DFT usando $\mathcal{O}(N \log N)$ operaciones.

**La Transformada Discreta de Fourier (DFT)**:

$$X = (X(0), \ldots, X(N-1)) \;\mapsto\; \hat{X} = (\hat{X}(0), \ldots, \hat{X}(N-1)),$$

donde

$$\hat{X}(k) = \sum_{j=0}^{N-1} X(j) W_N^{jk}, \quad k = 0, \ldots, N-1$$

con $W_N = \exp\left(\frac{2\pi}{N}\sqrt{-1}\right)$.

El cómputo directo de la DFT requeriría $N^2$ operaciones. En cambio, la **Transformada Rápida de Fourier (FFT)** es un algoritmo para calcular la DFT usando $\mathcal{O}(N \log N)$ operaciones.

**Aplicaciones**:
La FFT es quizás el algoritmo más omnipresente que se utiliza hoy en día para analizar y manipular datos digitales o discretos.

**La Transformada Discreta de Fourier (DFT)**:

$$X = (X(0), \ldots, X(N-1)) \;\mapsto\; \hat{X} = (\hat{X}(0), \ldots, \hat{X}(N-1)),$$

donde

$$\hat{X}(k) = \sum_{j=0}^{N-1} X(j) W_N^{jk}, \quad k = 0, \ldots, N-1$$

con $W_N = \exp\left(\frac{2\pi}{N}\sqrt{-1}\right)$.

El cómputo directo de la DFT requeriría $N^2$ operaciones. En cambio, la **Transformada Rápida de Fourier (FFT)** es un algoritmo para calcular la DFT usando $\mathcal{O}(N \log N)$ operaciones.

**Aplicaciones**:
La FFT es quizás el algoritmo más omnipresente que se utiliza hoy en día para analizar y manipular datos digitales o discretos.

# Integer Relation Detection Algorithm (1977)

Dado una cantidad de números reales, $x_1, x_2, \ldots, x_n$, existen enteros $a_1, a_2, \ldots, a_n$ (no todos 0) para los cuales

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0.$$

Para $n = 2$, el algoritmo Euclideano hace el trabajo, computando los términos de la fracción continua en la expansión de $x_1/x_2$. **Ferguson y Forcade generalizaron esto**.

# Integer Relation Detection Algorithm (1977)

Dado una cantidad de números reales, $x_1, x_2, \ldots, x_n$, existen enteros $a_1, a_2, \ldots, a_n$ (no todos 0) para los cuales

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0.$$

Para $n = 2$, el algoritmo Euclideano hace el trabajo, computando los términos de la fracción continua en la expansión de $x_1/x_2$. **Ferguson y Forcade generalizaron esto**.

**Aplicaciones:**

- Sistemas dinámicos discretos.
- Determinación de un número real como combinación lineal de $\pi$ y $\ln 2$.
- Simplificación de los cálculos con diagramas de Feynman diagrams en teoría cuántica de campos.

# Integer Relation Detection Algorithm (1977)

Dado una cantidad de números reales, $x_1, x_2, \ldots, x_n$, existen enteros $a_1, a_2, \ldots, a_n$ (no todos 0) para los cuales

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0.$$

Para $n = 2$, el algoritmo Euclideano hace el trabajo, computando los términos de la fracción continua en la expansión de $x_1/x_2$. **Ferguson y Forcade generalizaron esto**.

**Aplicaciones:**

- Sistemas dinámicos discretos.
- Determinación de un número real como combinación lineal de $\pi$ y $\ln 2$.
- Simplificación de los cálculos con diagramas de Feynman diagrams en teoría cuántica de campos.

# Fast multipole algorithm (1987)

El mayor dolor de cabeza de las simulaciones de $N$-cuerpos: los cálculos precisos de los movimientos de las $N$ partículas que interactúan mediante fuerzas gravitacionales o electrostáticas parecen requerir $\mathcal{O}(N^2)$ operaciones, por cada par de partículas.

**¡El algoritmo multipolo rápido se corre con $\mathcal{O}(N)$ operaciones!**

# Fast multipole algorithm (1987)

El mayor dolor de cabeza de las simulaciones de $N$-cuerpos: los cálculos precisos de los movimientos de las $N$ partículas que interactúan mediante fuerzas gravitacionales o electrostáticas parecen requerir $\mathcal{O}(N^2)$ operaciones, por cada par de partículas.

**¡El algoritmo multipolo rápido se corre con $\mathcal{O}(N)$ operaciones!**

Aplicaciones:

- Astrofísica: el sistema Tierra-Luna-Sol, la evolución de la estructura a gran escala del universo.
- Cosmología física: los procesos de formación de estructuras no lineales, la evolución dinámica de los cúmulos de estrellas.

# Fast multipole algorithm (1987)

El mayor dolor de cabeza de las simulaciones de $N$-cuerpos: los cálculos precisos de los movimientos de las $N$ partículas que interactúan mediante fuerzas gravitacionales o electrostáticas parecen requerir $\mathcal{O}(N^2)$ operaciones, por cada par de partículas.

**¡El algoritmo multipolo rápido se corre con $\mathcal{O}(N)$ operaciones!**

Aplicaciones:

- Astrofísica: el sistema Tierra-Luna-Sol, la evolución de la estructura a gran escala del universo.
- Cosmología física: los procesos de formación de estructuras no lineales, la evolución dinámica de los cúmulos de estrellas.

Determinando si una moneda es justa:

Fije

$$p = Pr(cara)$$

Sean los lanzamientos

$$X_1, X_2, \ldots, X_N \sim Ber(p)$$

Calcule la fracción de éxitos

$$\frac{1}{N} \sum_{i=1}^{N} X_i$$

Determinando si una moneda es justa:

Fije

$$p = Pr(cara)$$

Sean los lanzamientos

$$X_1, X_2, \ldots, X_N \sim Ber(p)$$

Calcule la fracción de éxitos

$$\frac{1}{N} \sum_{i=1}^{N} X_i$$

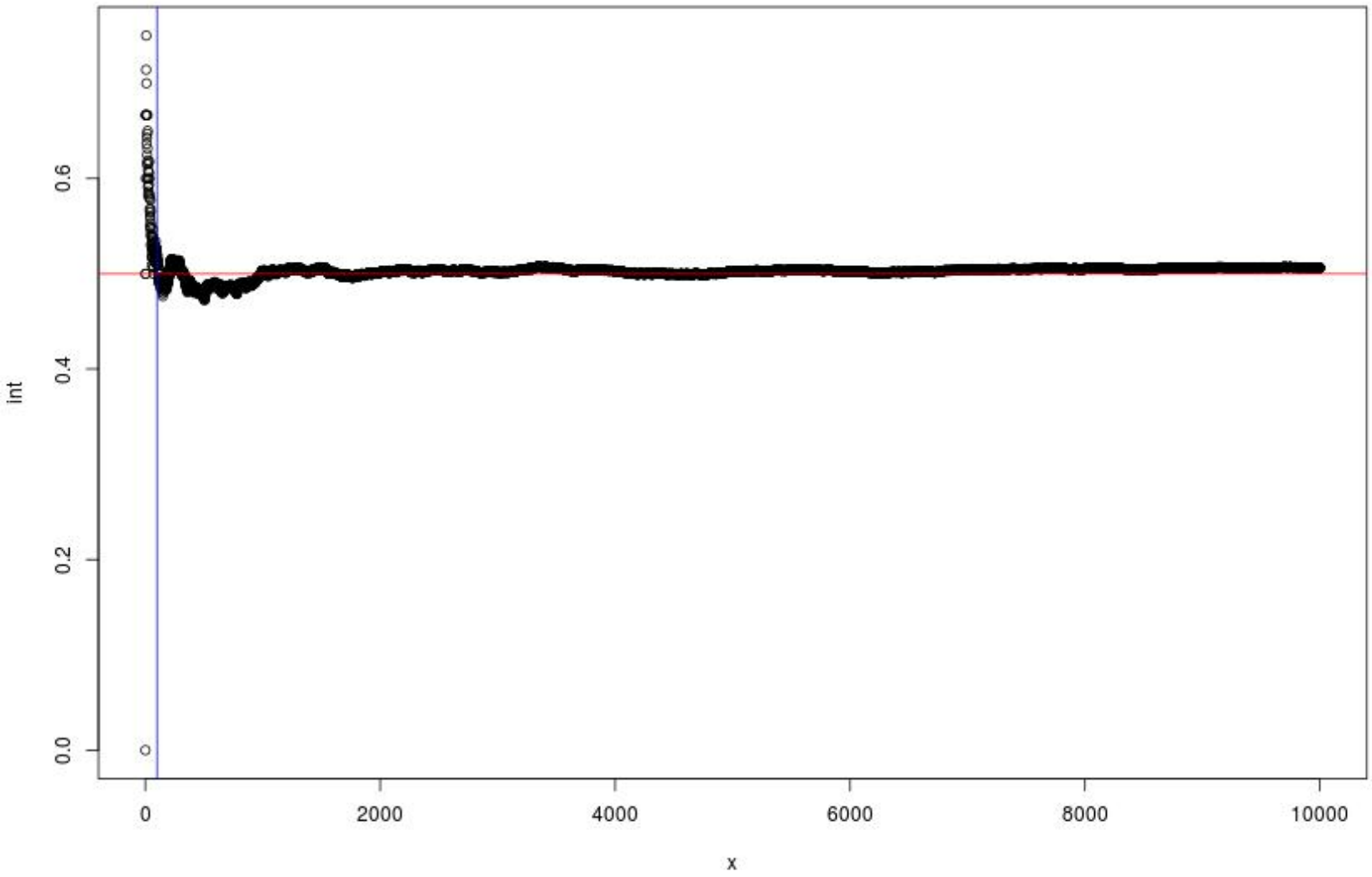



Figura : Lanzamientos de una moneda justa

Sea

$$\theta = \int_0^1 f(x)dx$$

Esto se reescribe

$$\theta = \mathbb{E}(f(X))$$

donde $X \sim Unif([0,1])$

**Estimador del parámetro**:

$$\hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

donde los $X_i \sim Unif([0,1])$

Sea

$$\theta = \int_0^1 f(x)dx$$

Esto se reescribe

$$\theta = \mathbb{E}(f(X))$$

donde $X \sim Unif([0,1])$

**Estimador del parámetro**:

$$\hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

donde los $X_i \sim Unif([0,1])$

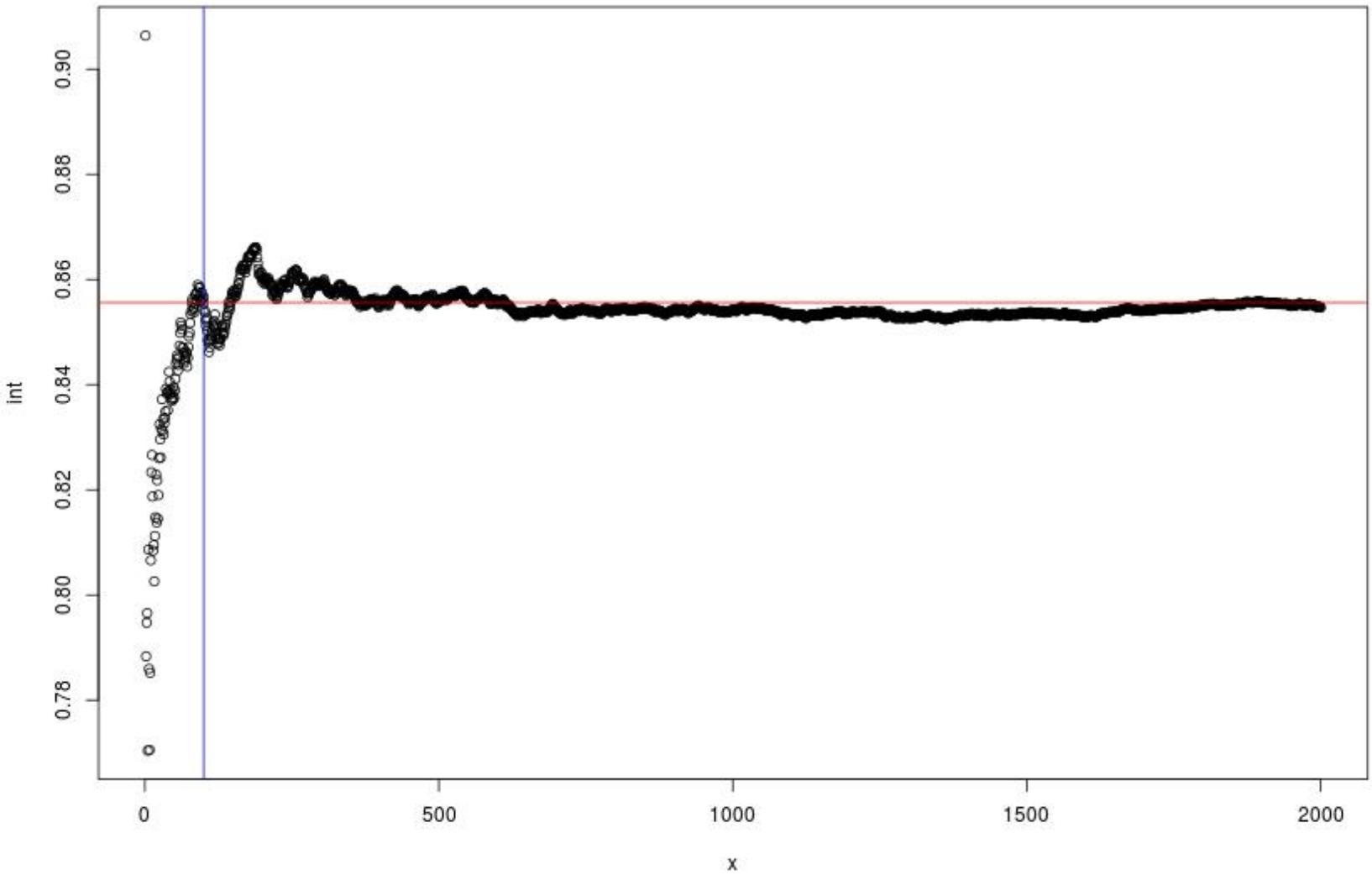



Figura : Integración por Monte Carlo

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0, 1)$.

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0,1)$.

   **si** $u \leq c\frac{\nu(x)}{\mu(x)}$
   **entonces** acepte $x$

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0,1)$.

   **si** $u \leq c\frac{\nu(x)}{\mu(x)}$
   **entonces** acepte $x$
   **sino** pruebe con otro $x$

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0,1)$.

   **si** $u \leq c\frac{\nu(x)}{\mu(x)}$
   **entonces** acepte $x$
   **sino** pruebe con otro $x$

Aquí se elije $c$ de modo que $c\frac{\nu(x)}{\mu(x)} < 1$ para todo $x$.

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0,1)$.

   **si** $u \leq c\frac{\nu(x)}{\mu(x)}$
   **entonces** acepte $x$
   **sino** pruebe con otro $x$

   Aquí se elije $c$ de modo que $c\frac{\nu(x)}{\mu(x)} < 1$ para todo $x$.

La probabilidad de seleccionar y luego aceptar algún $x$ es

$$c\frac{\nu(x)}{\mu(x)} \cdot \mu(x) = c\nu(x).$$

**Algoritmo rechazo**

1. Use $\mu$ para seleccionar una muestra, $x$.
2. Evalue $\nu(x)$. Esto debe ser fácil, una vez que se tiene $x$.
3. Genere una variable aleatoria uniforme $u \sim U[0,1)$.

   **si** $u \leq c\frac{\nu(x)}{\mu(x)}$
   **entonces** acepte $x$
   **sino** pruebe con otro $x$

   Aquí se elije $c$ de modo que $c\frac{\nu(x)}{\mu(x)} < 1$ para todo $x$.

La probabilidad de seleccionar y luego aceptar algún $x$ es

$$c\frac{\nu(x)}{\mu(x)} \cdot \mu(x) = c\nu(x).$$

Se desarrolló a principios del siglo XX como modelo de magnetización y fenómenos relacionados.

Cada configuración $\sigma$ tiene una **energía** asociada

$$E(\sigma) = \sum_{i \sim j} J_{ij} \sigma_i \sigma_j - B \sum_k \sigma_k.$$

Se desarrolló a principios del siglo XX como modelo de magnetización y fenómenos relacionados.

Cada configuración $\sigma$ tiene una **energía** asociada

$$E(\sigma) = \sum_{i \sim j} J_{ij} \sigma_i \sigma_j - B \sum_k \sigma_k.$$

La media $\langle f \rangle$ de un observable $f$ es

$$\langle f \rangle = \frac{1}{Z(T)} \sum_\sigma f(\sigma) \exp\left(-\frac{E(\sigma)}{\kappa T}\right).$$

La **función partición**

$$Z(T) = \sum_\sigma \exp\left(-\frac{E(\sigma)}{\kappa T}\right),$$

aquí $T$ es la temperatura y $\kappa$ es la constante de Boltzmann.

Se desarrolló a principios del siglo XX como modelo de magnetización y fenómenos relacionados.

Cada configuración $\sigma$ tiene una **energía** asociada

$$E(\sigma) = \sum_{i \sim j} J_{ij} \sigma_i \sigma_j - B \sum_k \sigma_k.$$

La media $\langle f \rangle$ de un observable $f$ es

$$\langle f \rangle = \frac{1}{Z(T)} \sum_\sigma f(\sigma) \exp\left(-\frac{E(\sigma)}{\kappa T}\right).$$

La **función partición**

$$Z(T) = \sum_\sigma \exp\left(-\frac{E(\sigma)}{\kappa T}\right),$$

aquí $T$ es la temperatura y $\kappa$ es la constante de Boltzmann.

Un enfoque de muestreo por importancia natural podría ser seleccionar configuraciones según la distribución

$$\mu(\sigma) = \frac{1}{Z(T)} \exp\left(-\frac{E(\sigma)}{\kappa T}\right)$$

de modo que la media muestral, de $M$ muestras,

$$\overline{f} = \frac{1}{M} \sum_{k=1}^{M} f(\sigma^k)$$

converja rápidamente a la media del observable, $\langle f \rangle$.

Un enfoque de muestreo por importancia natural podría ser seleccionar configuraciones según la distribución

$$\mu(\sigma) = \frac{1}{Z(T)} \exp\left(-\frac{E(\sigma)}{\kappa T}\right)$$

de modo que la media muestral, de $M$ muestras,

$$\overline{f} = \frac{1}{M} \sum_{k=1}^{M} f(\sigma^k)$$

converja rápidamente a la media del observable, $\langle f \rangle$.

**¡El problema es encontrar una forma de muestrear según $\mu$!**

Un enfoque de muestreo por importancia natural podría ser seleccionar configuraciones según la distribución

$$\mu(\sigma) = \frac{1}{Z(T)} \exp\left(-\frac{E(\sigma)}{\kappa T}\right)$$

de modo que la media muestral, de $M$ muestras,

$$\overline{f} = \frac{1}{M} \sum_{k=1}^{M} f(\sigma^k)$$

converja rápidamente a la media del observable, $\langle f \rangle$.

**¡El problema es encontrar una forma de muestrear según $\mu$!**

Construcción de una **cadena de Markov simétrica aperiódica** que converja a la distribución límite $\mu$. Las probabilidades de transición son tales que

$$\mu(\sigma) = \sum_{\xi} \mu(\xi) p_{\xi,\sigma}.$$

La suma es sobre todas las configuraciones $\xi$ que difieren de $\sigma$ por un spin, y

$$p_{\xi,\sigma} = \begin{cases} \frac{\mu(\sigma)}{\mu(\xi)} = \exp\left(-\frac{\Delta E}{kT}\right) & \text{si} \Delta E > 0 \\ 1 & \text{si} \Delta E < 0 \end{cases}$$

Construcción de una **cadena de Markov simétrica aperiódica** que converja a la distribución límite $\mu$. Las probabilidades de transición son tales que

$$\mu(\sigma) = \sum_{\xi} \mu(\xi) p_{\xi,\sigma}.$$

La suma es sobre todas las configuraciones $\xi$ que difieren de $\sigma$ por un spin, y

$$p_{\xi,\sigma} = \begin{cases} \frac{\mu(\sigma)}{\mu(\xi)} = \exp\left(-\frac{\Delta E}{kT}\right) & \text{si} \Delta E > 0 \\ 1 & \text{si} \Delta E < 0 \end{cases}$$

**Dinámica de Metropolis:**

1. Seleccione un sitio de manera uniforme
2. Acepte el cambio con probabilidad

$$\text{mín}\left\{\exp\left(-\frac{\Delta E}{kT}\right), 1\right\}$$

Construcción de una **cadena de Markov simétrica aperiódica** que converja a la distribución límite $\mu$. Las probabilidades de transición son tales que

$$\mu(\sigma) = \sum_{\xi} \mu(\xi) p_{\xi,\sigma}.$$

La suma es sobre todas las configuraciones $\xi$ que difieren de $\sigma$ por un spin, y

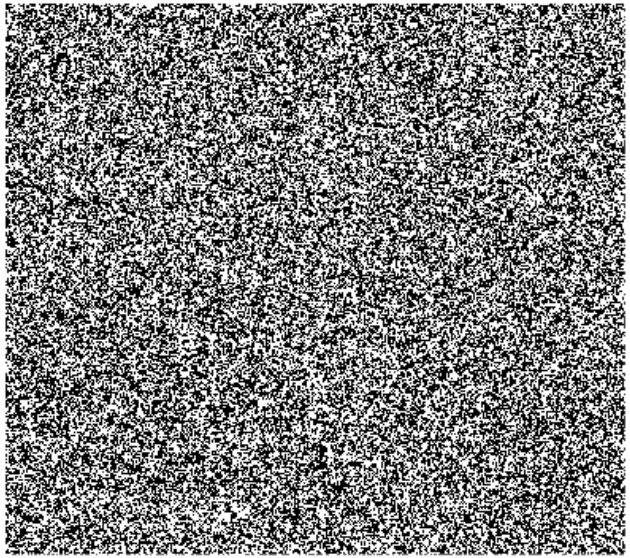
$$p_{\xi,\sigma} = \begin{cases} \frac{\mu(\sigma)}{\mu(\xi)} = \exp\left(-\frac{\Delta E}{kT}\right) & \text{si} \Delta E > 0 \\ 1 & \text{si} \Delta E < 0 \end{cases}$$

**Dinámica de Metropolis:**

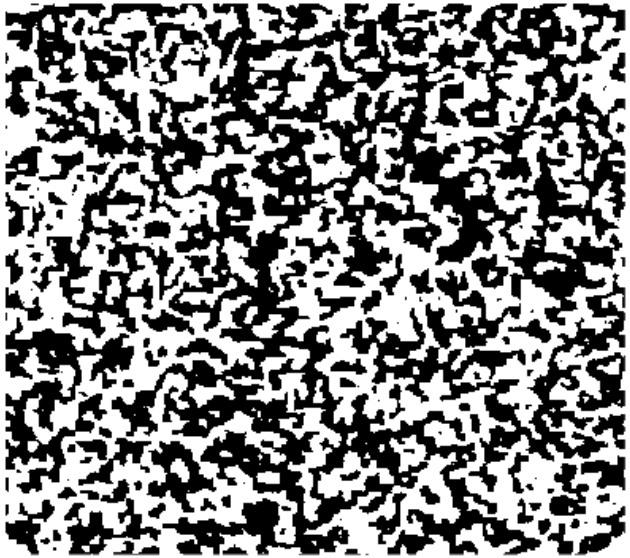
1. Seleccione un sitio de manera uniforme
2. Acepte el cambio con probabilidad

$$\text{mín}\left\{\exp\left(-\frac{\Delta E}{kT}\right), 1\right\}$$

**Video 1: T alta**
**No magnetización**



**Video 1: T baja**
**Magnetización**



# Nuevo Top 10 Algoritmos

Lista según el número de localizadores en el índice del libro **The Princeton Companion to Applied Mathematics (PCAM)**.

1. **Newton and quasi-Newton methods**
2. Matrix factorizations (LU, Cholesky, QR)
3. Singular value decomposition, QR and QZ algorithms
4. Monte-Carlo methods
5. Fast Fourier transform

# Nuevo Top 10 Algoritmos

Lista según el número de localizadores en el índice del libro **The Princeton Companion to Applied Mathematics (PCAM)**.

1. **Newton and quasi-Newton methods**
2. Matrix factorizations (LU, Cholesky, QR)
3. Singular value decomposition, QR and QZ algorithms
4. Monte-Carlo methods
5. Fast Fourier transform
6. Krylov subspace methods
7. **JPEG**
8. **PageRank**
9. Simplex algorithm
10. **Kalman filter**

# Nuevo Top 10 Algoritmos

Lista según el número de localizadores en el índice del libro **The Princeton Companion to Applied Mathematics (PCAM)**.

1. **Newton and quasi-Newton methods**
2. Matrix factorizations (LU, Cholesky, QR)
3. Singular value decomposition, QR and QZ algorithms
4. Monte-Carlo methods
5. Fast Fourier transform
6. Krylov subspace methods
7. **JPEG**
8. **PageRank**
9. Simplex algorithm
10. **Kalman filter**

Dongarra y Sullivan: “lo que sea que se nos ocurrió al final, sería controvertido”.

| **PCAM list** | **2000 list** |
|---|---|
| Newton and quasi-Newton methods | The Fortran Optimizing Compiler |
| JPEG | Quicksort algorithm for sorting |
| PageRank | Integer relation detection |
| Kalman filter | Fast multipole method |

Cuadro : Comparativa de listas [1].

**¡Hay un acuerdo notable entre las dos listas!**

Dongarra y Sullivan: “lo que sea que se nos ocurrió al final, sería controvertido”.

| **PCAM list** | **2000 list** |
|---|---|
| Newton and quasi-Newton methods | The Fortran Optimizing Compiler |
| JPEG | Quicksort algorithm for sorting |
| PageRank | Integer relation detection |
| Kalman filter | Fast multipole method |

Cuadro : Comparativa de listas [1].

**¡Hay un acuerdo notable entre las dos listas!**

¿Qué algoritmos listaría usted?

Dongarra y Sullivan: “lo que sea que se nos ocurrió al final, sería controvertido”.

| **PCAM list** | **2000 list** |
|---|---|
| Newton and quasi-Newton methods | The Fortran Optimizing Compiler |
| JPEG | Quicksort algorithm for sorting |
| PageRank | Integer relation detection |
| Kalman filter | Fast multipole method |

Cuadro : Comparativa de listas [1].

**¡Hay un acuerdo notable entre las dos listas!**

**¿Qué algoritmos listaría usted?**

# Referencias


The top 10 algorithms in applied mathematics.
https://nickhigham.wordpress.com/2016/03/29/the-top-10-algorithms-in-applied-mathematics/.
Accessed: 2018-09-10.


Barry A Cipra.
The best of the 20th century: Editors name top 10 algorithms.
*SIAM news*, 33(4):1–2, 2000.


Jack Dongarra and Francis Sullivan.
Guest editors' introduction: The top 10 algorithms.
*Computing in Science & Engineering*, 2(1):22–23, 2000.


David S Kershaw.
The incomplete cholesky—conjugate gradient method for the iterative solution of systems of linear equations.
*Journal of Computational Physics*, 26(1):43–65, 1978.