Universidad ESAN

FACULTAD DE INGENIERÍA

Trabajo Final - Álgebra Lineal

Flores Pamo, Ian [23100067]
Villarruel Meza, Alexis [20200057]
Avalos Sotelo, Jeremy [23100174]

Ingeniería de Tecnologías de Información y Sistemas

Resumen

El siguiente trabajo abarca la implementación de cuatro técnicas clave: descomposición de Cholesky, triangulación de Householder, ortogonalización de Gram-Schmidt y descomposición en valores singulares (SVD). Cada método se describe junto con su formulación matemática, algoritmos de implementación en Python y su complejidad computacional. Además, se discuten aspectos de estabilidad numérica y recomendaciones para el uso de cada técnica en distintos escenarios. Estos métodos permiten abordar problemas de regresión lineal y optimización con distintas ventajas en cuanto a estabilidad y eficiencia, dependiendo de las características de la matriz de entrada. La comparación entre los métodos resalta el enfoque práctico de cada uno en términos de precisión y adecuación al problema analizado.

Descripción: En regresión lineal se utiliza la notación de matrices para describir la relación entre una variable dependiente y varias variables independientes. Es útil cuando se trabaja con variables predictoras complejas. Modelo:

1) $y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip}$

para $i = 1, 2, \ldots, n$.

$\hat{\beta}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_k X_{ki}$

2) $y = X\beta + e \longrightarrow \hat{y} = X\hat{\beta}$

3) $\hat{\beta} = \arg_b \min \left[\|y - \hat{y}\|^2 \right]$

$\longrightarrow (y - \hat{y})\,(y - \hat{y})$

$y^t.y - y^t.\hat{y} - \hat{y}.y^t + \hat{y}^t.\hat{y}$

$y^t.y - 2y^t X\hat{\beta} + \hat{\beta}^t X^t X\hat{\beta}$

Derivamos: $\left(\dfrac{\partial}{\partial\hat{\beta}}\right)$

$-2X^t y + 2X^t X\hat{\beta} = 0$

$$\boxed{\left(X^t X\right)\hat{\beta} = X^t y}$$

Planteamiento: Considere el problema general de regresión lineal en el que se predice una variable target $\mathbf{y} \in \mathbb{R}^n$ utilizando una matriz de diseño $X \in \mathbb{R}^{nxp}$, cuyo problema consiste en estimar el vector $\hat{\beta} \in \mathbb{R}^p$ de coeficientes. Este problema se puede resolver mediante múltiples algoritmos, algunos de los cuales se listan a continuación.

I. Descomposición de Choleski

Una matriz A simétrica ($A = A^t$) y positiva puede ser factorizada de muchas formas por medio de una matriz triangular inferior y una matriz triangular superior.

Para una matriz no singular la descomposición LU nos lleva a considerar una descomposición de tal tipo $A = LU$; dadas las condiciones de A, simétrica y definida positiva, no es necesario hacer pivoteo, por lo que ésta factorización se hace eficientemente y en un número de operaciones la mitad de LU tomando la forma $A = LL^T$. Donde se puede imponer que $l_{ii} > 0$ para todo $i = 1, \ldots, n$.

Para resolver un sistema lineal $AX = b$ con A simétrica definida positiva y dada su factorizaciòn de Cholesky $A = LL^T$, primero debemos resolver $LY = b$ y entonces resolver $L^T X = y$

Definición: Sea A $= (a_{ij})$ una matriz $n \times n$ de coeficientes reales. Diremos que A es definida **positiva** si:

$$\sum_{i,j=1}^{n} a_{ij} u_i u_j > 0, \text{ para todo } u \in \mathbb{R}^n \setminus \{0\}$$

Definición (Criterio de Sylvester): Otro método para determinar que una matriz sea **positiva**, es que sea A una matriz real simétrica $n \times n$. Dicha matriz es definida positiva si y solo si todos sus menores principales son positivos.

Sea $A \in M_n(\mathbb{R}), k \in \{1, \ldots, n\}$. El menor principal líder de k-ésimo orden de A se define como el menor principal que está en la intersección de los primeros k renglones y las primeras k columnas de la matriz A.

$$\Delta_k(A) = \delta_{1,\ldots,k}(A) = det\left(A_{\{1,\ldots,k\},\{1,\ldots,k\}}\right).$$

Cálculo efectivo factorización de Choleski: Para el cálculo efectivo de la factorizacion de Cholesky, se parte de la igualdad:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \vdots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \times \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \vdots & \vdots & l_{n3} \\ 0 & 0 & \cdots & l_{nn} \end{pmatrix}$$

Así se obtiene lo siguiente:

$$a_{11} = l_{11}^2 \longrightarrow l_{11} = \sqrt{a_{11}}$$
$$a_{21} = l_{21} l_{11} \longrightarrow l_{22} = a_{21}/l_{11}$$
$$a_{22} = l_{21}^2 + l_{22}^2 \longrightarrow \sqrt{\left(a_{22} - l_{21}^2\right)}$$
$$a_{32} = l_{31} l_{21} + l_{32} l_{22} \longrightarrow l_{32} = \left(a_{32} - l_{31} l_{21}\right)/l_{22}$$

Y de manera general, para $i = 1, \ldots, n$ y $j = i+1, \ldots, n$, entonces se deduce que:

$$l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)}$$

$$l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{l_{ii}}}$$

Mínimos Cuadrados: Para el problema se quiere solucionar la ecuación normal (para estimar $\hat{\beta}$):

$$\boxed{\left(X^t X\right)\hat{\beta} = X^t y}$$

Si la matriz $X^t X$ es simétrica y definida positiva (lo cual sucede si las columnas de X son linealmente independientes), si se cumple con estas dos cosas, podemos usar la descomposición de Cholesky para escribir:

$$\boxed{X^t X = LL^t}$$

Después, se hará lo siguiente:

1. $Lz = X^t y$ (Resolvemos para z)
2. $L^t \hat{\beta} = z$ (Resolvemos para $\hat{\beta}$)

Codificación en Python: A continuación, se mostrará cómo la descomposición de Choleski se utiliza para encontrar la solución de mínimos cuadrados:

```python
import numpy as np

def Cholesky(A):
    n = A.shape[0]
    L = np.zeros_like(A)
    # Inicializa L como matriz compuesta por ceros

    # Verificar que A es simétrica
    if not np.allclose(A, A.T):
        raise ValueError("La matriz no es
        simétrica.")

    for i in range(n):
        for j in range(i + 1):
            sum_k = sum(L[i][k] * L[j][k] for k
            in range(j))
```

```
            # Suma parcial
            if i == j:
                val = A[i][i] - sum_k  # Calcular el
                valor del elemento diagonal

                # Verificar que la matriz es
                definida positiva
                if val <= 0:
                    raise ValueError("La matriz
                    no es definida positiva.")
                L[i][j] = np.sqrt(A[i][i] -
                sum_k)
                # Elementos de la diagonal de L
            else:
                L[i][j] = (A[i][j] - sum_k)
                / L[j][j]
                # No pertenecen a la diagonal
    return L

def regresion_cholesky(X, y):
    XtX = X.T @ X
    # Matriz de correlación de X
    Xty = X.T @ y
    # Producto de X transpuesta y y
    L = Cholesky(XtX)
    # Descomposición de Cholesky de XtX
    z = np.linalg.solve(L, Xty)
    # Resuelve Lz = Xty
    beta = np.linalg.solve(L.T, z)
    # Resuelve L.T * beta = z
    return beta
```

Complejidad Big O:

1. Cálculo de X^TX y X^Ty: Multiplicar X de tamaño $n \times p$ para obtener X^TX toma $O(np^2)$.

2. Descomposición de Cholesky de X^TX: Descomponer esto con Cholesky toma $O(p^3)$.

3. Resolución de sistemas triangulares: Resolver $Lz = X^T$ y $L^T\beta = z$ toma $O(p^2)$.

Complejidad Total: El costo principal proviene de los pasos 1 y 2, por lo que la complejidad total es $O(np^2 + p^3)$

Estabilidad Numérica y Número de condición O:

- Funciona bien cuando la matriz X^TX es simétrica definida positiva.

- Puede fallar en problemas mal condicionados (cuando la matriz es cercana a ser singular), lo que amplifica los errores de redondeo.

- Si la matriz es mal condicionada, se recomienda usar SVD o regularización para mejorar la estabilidad.

II. Triangulación Householder

Transformaciones de Householder: Se denomina transformación de Householser a una transformación lineal de $\mathbb{R}^n$ en $\mathbb{R}^n$ caracterizada por una matriz $H^{n\times n}$. Estas matrices de Householder son ortogonales y son simétricas, en consecuencia son iguales a su inversa, por la tanto: $H^{-1} = H^T = H$.
Para la construcción de estas matrices se necesita lo siguiente:

$$u = x \pm \|x\|_2 \, e_1.$$

$$H = H(u) = I - 2\frac{uu^T}{\|u\|_2^2}.$$

$$H = I - \frac{uu^T}{\beta}$$

$$\beta = \frac{\|u\|_2^2}{2}.$$

Construcción de la matriz de Householder: Sea $\vec{x} \in \mathbb{R}^n$, queremos construir una matriz de Householder H tal que $Hx = ae_1$ (todos los componentes de Hx son ceros; excepto la primera).
Escogemos el vector $\vec{u} = x \pm \|x\|_2 \, e_1$.
Es fácil de comprobar que, construyendo H con este vector $\vec{u}$, Hx tiene todos sus componentes iguales a 0, excepto la primera.

Descomposición QR basada en Householder: Para una matriz genérica A, $n \times n$, no singular, las propiedades que se acaban de describir permiten construir una sucesión de $n-1$ matrices de Householder tales que: $H_{n-1}, \ldots, H_2H_1A = R$ (una matriz no singular y triangular superior).

Procedimiento:

1. Construimos la matriz H_1 que transforma a_1 en un multiplo de e_1, es decir hacer ceros en los

componentes 2 hasta n del vector a_1.

$$H_1 a_1 = \left(I - \frac{u_1 u_1^T}{\beta_1}\right) = \pm \|a_1\|_2 e_1 = \begin{pmatrix} \gamma_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Donde $\gamma_{11} = \|a_1\|_2$, y su signo puede ser positivo o negativo, usualmente se escoge el signo opuesto de a_{11} : $Sign(\gamma_{11}) = -Sign(a_{11})$.

$$u = \begin{pmatrix} a_{11} - \gamma_{11} \\ a_{21} \\ \vdots \\ a_{n_1} \end{pmatrix}$$

2. Aplicamos la primera transformación de Householder H_1 en la matriz A, donde obtenemos la matriz parcialmente reducida $A^{(2)} = H_1 A_1$, donde la primera columna es múltiplo de e_1.

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

$$H_1 A = A^{(2)} = \begin{pmatrix} \gamma_{11} & a_{12}^{(2)} & \dots & a_{1n}^{(2)} \\ 0 & a_{12}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix}$$

3. Nos damos cuenta que para el vector u_i es múltiplo de e_i donde $u_{ii} = a_{ii}^{(i)} - \gamma_{ii}$.

4. A partir del H_2 en adelante se cumple lo siguiente:

$$H_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & \hat{H}_k \end{pmatrix}$$

5. Si A es una matriz no singular, se pueden efectuar $n-1$ pasos de reducción Householder, para obtener:

$$R = H_{n-1} \dots H_2 H_1 A$$

6. Se denota con Q^T la matriz ortogonal $n \times n$:

$$Q^T = H_{n-1} \dots H_2 H_1 \Rightarrow Q = H_1 H_2 \dots H_{n-1}$$

7. Entonces la descomposición QR de la matriz A es:

$$Q^T A = R \quad \vee \quad A = QR$$

Mínimos Cuadrados: Para el problema se quiere solucionar la ecuación normal (para estimar $\hat{\beta}$):

$$\boxed{\left(X^t X\right) \hat{\beta} = X^t y}$$

Si la matriz X es de $m \times n$ dimensiones y Q es una matriz ortogonal de dimensión $m \times m$, es decir que $Q^T Q = I$ y R una matriz triangular superior de dimensión $m \times n$, al momento de reemplazar en la ecuación de mínimos cuadrados, se obtiene lo siguiente:

$$\boxed{R\hat{\beta} = Q^T y}$$

Codificación en Python: A continuación, se mostrará cómo la descomposición de Householder se utiliza para encontrar la solución de mínimos cuadrados de un sistema de ecuaciones lineales sobredeterminado.

```python
import numpy as np

def householder_reflection(A):
    m, n = A.shape
    if m < n:
        raise ValueError("La matriz debe
        tener al menos tantas filas como
        columnas (m >= n).")

    Q = np.eye(m)
    R = A.copy()

    for i in range(n):

        x = R[i:, i]

        norm_x = np.linalg.norm(x)

        if norm_x == 0:
            continue

        e1 = np.zeros_like(x)
        e1[0] = norm_x

        v = x - e1
        v = v / np.linalg.norm(v)
```

```
27
28          H_i = np.eye(m)
29
30          H_i[i:, i:] -= 2.0 * np.outer(v, v)
31
32          R = H_i @ R
33          Q = Q @ H_i
34
35      return Q, R
```

Complejidad Big O: Cada iteración del método de Householder aplica una reflexión para triangularizar una columna de la matriz X. Esto requiere alrededor de n operaciones por columna, y al aplicarse en n columnas, el costo es aproximadamente $O(n^2)$.

Factor m: La matriz tiene m filas, lo que implica que cada operación de reflexión afecta m elementos. La multiplicación de una matriz de reflexión $m \times m$ on una submatriz requiere m operaciones.

Por lo tanto, el tiempo total de ejecución está dominando por el proceso de ortogonalización y la complejidad total es $O(n^2 m)$.

Estabilidad Numérica y Número de condición O: Para asegurar que el sistema de ecuaciones sea numéricamente estable, el número de condición de la matriz R debe ser menor a 1×10^{10}. Un número de condición alto implica que la matriz está cerca de ser singular y podría generar grandes errores de redondeo. En casos de mala condición, el método de Householder podría no ser estable para regresión lineal, y se recomendaría el uso de métodos de regularización u otra técnicas.

III. Ortogonalización Gram-Schmidt

El proceso se fundamenta en el principio de la geometría euclidiana que establece que la diferencia entre $\vec{v}$ y su proyección sobre $\vec{u}$ es $\perp \vec{u}$. Este principio permite construir un nuevo conjunto de vectores perpendiculares a partir de un par de vectores que no sean paralelos entre sí.
Dado un conjunto de vectores linealmente independientes, $\{v_1, v_2, \ldots, v_n\}$ que expanden un espacio Euclidiano de dimensión finita E^n. Entonces siempre se puede construir un conjunto ortogonal de vectores, $\{v_1, v_2, \ldots, v_n\}$ que también expandan E^n de la siguiente forma:

$$\boxed{v - \frac{\langle v, u\rangle}{\langle u, u\rangle} u \cong v - proy_u(v)}$$

Es un vector ortogonal a u. Entonces, dados los vectores $v_1, \ldots, v_n$ se define:

1. $u_1 = v_1$.
2. $u_2 = v_2 - \frac{\langle v_2, u_1\rangle}{\langle u_1, u_1\rangle} u_1$
3. $u_3 = v_3 - \frac{\langle v_3, u_1\rangle}{\langle u_1, u_1\rangle} u_1 - \frac{\langle v_3, u_2\rangle}{\langle u_2, u_2\rangle} u_2$

Generalizando en k:

$$\boxed{u_k = v_k - \sum_{j=1}^{k-1} \frac{\langle v_k, u_j\rangle}{\langle u_j, u_j\rangle} u_j}$$

Así siempre es posible construir una base ortonormal a partir de un conjunto de vectores linealmente independientes. Esta base ortogonal será única en E^n, si existe otra sus vectores serán proporcionales. Más aún, cada espacio vectorial V^n de dimensión finita tendrá una base ortogonal asociada,
Para formar la base ortonormal se tendrá que hacer lo siguiente:

$$\boxed{e_1 = \frac{u_j}{\langle u_j, u_j\rangle^{1/2}} = \frac{u_j}{\|u_j\|}}$$

Mínimos Cuadrados: Para el problema se quiere solucionar la ecuación normal (para estimar $\hat{\beta}$):

$$\boxed{\left(X^t X\right) \hat{\beta} = X^t y}$$

El proceso de Gram-Schmidt convierte un conjunto de vectores en un conjunto ortogonal (y eventualmente ortonormal). En el contexto de la regresión lineal, se transforma la matriz X en el producto de dos matrices: una matriz ortogonal Q y una matriz triangular superior $R \longrightarrow X = QR$.

Donde X tiene columnas $x_1, x_2, \ldots, x_n$. Nuestro objetivo es transformar estas columnas en vectores ortogonales, a las que llamaremos $u_1, u_2, \ldots, u_n$ y luego normalizarlos para obtener $q_1, q_2, \ldots, q_n$

las columnas de Q. La matriz R se construye utilizando los coeficientes de proyección calculados anteriormente. En otras palabras, R es una matriz triangular superior que contiene los valores $r_{ij} = q^T x_j$.

Es similar al caso que se vio antes (Householder), resolvemos lo siguiente:

$$\boxed{R\hat{\beta} = Q^T y}$$

Codificación en Python: Se mostrará cómo la ortogonalización Gram-Schmidt se utiliza para encontrar la solución de mínimos cuadrados.

```python
import numpy as np

def gram_schmidt_linear_regression(X, y):
    # Obtener la cantidad de muestras y
    #características
    n, p = X.shape

    # Ortogonalización de Gram-Schmidt
    Q = np.zeros((n, p))
    R = np.zeros((p, p))
    for j in range(p):
        v = X[:, j]
        for i in range(j):
            R[i, j] = np.dot(Q[:, i], X[:, j])
            v = v - R[i, j] * Q[:, i]
        R[j, j] = np.linalg.norm(v)
        Q[:, j] = v / R[j, j]

    # Resolver el sistema de ecuaciones
    #R * beta_hat = Q^T * y
    beta_hat = np.linalg.solve(R, Q.T @ y)

    return beta_hat

    return Q, R
```

Complejidad Big O:

1. Validación de entrada: La función verifica las dimensiones y compatibilidad de X y y. Estas operaciones tienen una complejidad constante, $O(1)$.
2. Ortogonalización de Gram-Schmidt:
   - El bucle externo se ejecuta p veces (una por cada columna de X).
   - En cada iteración j se inicializa el vector v y luego se ajusta en función de las columnas previas de Q.
   - El bucle interno se ejecuta j cada vez calculando un producto punto (costo $O(n)$), y ajustando v (también $O(n)$).
   - La complejidad total de esta sección es $O(np^2)$ debido a los bucles anidados y los cálculos en cada paso
3. Resolución del sistema: La resolución de $R\hat{\beta} = Q^T y$ tiene una complejidad de $O(p^2)$, ya que R es una matriz triangular.

Complejidad Total: La parte dominante es la ortogonalización, por lo que la complejidad total es $O(np^2)$

Estabilidad Numérica y Número de condición O:

- La estabilidad de la ortogonalización de Gram-Schmidt es sensible a problemas de colinealidad. Si alguna columna de X se aproxima a una combinación lineal de otras, el método puede ser inestable debido a la amplificación de errores numéricos en la normalización de vectores casi dependientes. Esto es una debilidad del Gram-Schmidt clásico.

IV. Descomposición de valores singulares

La descomposición en Valores singulares (*Singular Value Descomposition* en inglés) es una técnica que permite descomponer una matriz de la siguiente manera:

$$\boxed{A = U\Sigma V^T}$$

Donde la matriz $A \in \mathbb{R}^{m\times n}, m \geq n$ (sea cuadrada o no, simétrica o no y semi-definida positiva). La matriz $U \in \mathbb{R}^{m\times m}$, la matriz $\Sigma \in \mathbb{R}^{m\times n}$ y por último la matriz $V \in \mathbb{R}^{n\times n}$. A continuación se muestra como serían las dimensiones de las matrices para esta descomposición:

$$\boxed{A_{m\times n} = U_{m\times m} \times \Sigma_{m\times n} \times V^t_{n\times n}}$$

Recordemos que U y V son matrices ortogonales:

$$U^t \times U = U \times U^t = I_{n\times n}$$
$$V^t \times V = V \times V^t = I_{m\times m}$$

Descomposición: Los siguientes resultados demuestran que toda matriz tiene una descomposición en valores singulares como la descrita anteriormente.
Para formar la matriz V se hace lo siguiente:

1. $det\,(A - \lambda I) = 0 \cong$ cuando $i = j$, $(a_{ij}) - \lambda$.
2. $(A - \lambda_i I)\vec{x} = \vec{0}$.
3. $\vec{v_i} = \hat{x} \cong \frac{\vec{x}}{\|x\|}$.
4. $V = [v_1|v_2|\ldots|v_n]$.
5. Transponer la matriz V.

Como AA^T es simétrica, diagonaliza de forma ortogonal y sus autovalores son reales. Sea $(v_1, \ldots, v_n)$ una base ortonormal de $\mathbb{R}^n$ formada de autovectores de A^TA y sean $\lambda_1, \ldots, \lambda_n$ los autovalores asociados. Como A^TA es semidefinida positiva, es decir no negativos, todos sus autovalores son mayores o iguales que 0. Sin pérdida de generalidad consideramos $v_1, \ldots, v_n$ ordenados de modo que $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Por ser r el rango de A y por lo tanto, el rango de A^TA debe cumplirse que $\lambda_r > 0$ y $\lambda_{r+1} = \lambda_{r+2} = \ldots = \lambda_n = 0$. Por lo tanto $v_1, \ldots, v_n$ están en el núcleo de A^TA y, por ello, en el núcleo de A, es decir, $Av_i = 0$ para $i = r+1, \ldots, n$. Para $1 \leq i \leq r$ definimos σ_i y u_i de la siguiente forma

- $\boxed{\sigma = \sqrt{\lambda_i}} \qquad \wedge \qquad \boxed{u_i = \frac{1}{\sigma_i} A v_i}$

Para formar la matriz Σ:

$$\Sigma = \begin{bmatrix} \sigma_1 & & & 0 \\ & \ddots & & \vdots \\ & & \sigma_i & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix}$$

Con $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_i > 0$; valores singulares de A.
Para formar la matriz U:

1. $u_i = \frac{Av_i}{\sigma_i} \quad \forall_i \in \{1 \ldots r\}$
2. $U = [u_1|u_2|\ldots|u_n]$

Mínimos Cuadrados: Para el problema se quiere solucionar la ecuación normal (para estimar $\hat{\beta}$):

$$\boxed{\left(X^tX\right)\hat{\beta} = X^ty}$$

La descomposición en valores singulares, descompone una matriz en tres matrices: $X = U\Sigma V^T$, donde Σ es diagonal y U y V son ortogonales. Al momento de reemplazar se obtiene lo siguiente:

$$\boxed{\hat{\beta} = V\Sigma^{-1}U^Ty}$$

Codificación en Python: Se mostrará cómo la Descomposición en valores singulares (SVD) se utiliza para encontrar la solución de mínimos cuadrados:

```python
import numpy as np

def svd(X, tol=1e-10, max_iter=100):
    m, n = X.shape
    U = np.eye(m)  # Matriz ortogonal U
    V = np.eye(n)  # Matriz ortogonal V
    A = X.copy()  # Copia de X

    for _ in range(max_iter):
        for i in range(n):
            for j in range(i + 1, n):
                # Ángulo de rotación
                a = A[:, i]
                b = A[:, j]
                theta = 0.5 * np.arctan2(2 *
                np.dot(a, b), np.dot(a, a)
                - np.dot(b, b))

                # Matriz de rotación
                c, s = np.cos(theta), np.sin
                (theta)
                R = np.eye(n)
                R[i, i] = c
                R[i, j] = s
                R[j, i] = -s
                R[j, j] = c

                A = A @ R  # Aplica rotación a A
                V = V @ R  # Actualiza V

        if np.allclose(np.triu(A, 1), 0, atol=tol):
        # Verifica si A es diagonal
            break

```

```
    Sigma = np.diag(np.sqrt(np.sum(A**2, axis=0)))
    # Calcula valores singulares
    U = X @ V / Sigma
    # Calcula U a partir de X y V

    return U, Sigma, V.T

def regresion_SVD(X, y):
    U, Sigma, VT = svd(X)  # Descomposición SVD
    de X
    Sigma_inv = np.linalg.inv(Sigma)  # Sigma^(-1)
    beta = VT.T @ Sigma_inv @ U.T @ y
    # Calcula coeficientes de regresión
    return beta
```

Complejidad Big O:

1. Validación de matriz de diseño X:
   - Comprobación de dimensiones y rango completo de X.
   - El cálculo del rango de X tiene una complejidad de $O(np^2)$ debido a la factorización.
   - Complejidad: $O(np^2)$.
2. Descomposición en valores singulares (SVD):
   - En cada iteración de la aproximación SVD, se realizan rotaciones en cada par de columnas de X hasta que A sea diagonal.
   - Aproximadamente toma $O(n.m.max\_iter)$
3. Calcular β con SVD:
   - La pseudoinversa de Σ y el cálculo de $\hat{\beta} = V\Sigma^{-1}U^T y$ son dominados por el tamaño de X y la dimensión de Σ.
   - Complejidad: $O(n.p)$.

Complejidad Total: La complejidad principal está en el paso 2 de la SVD, dando un total de $O(n.m.max\_iter)$.

Estabilidad Numérica y Número de condición O:

- Es más estable numéricamente que Cholesky y no requiere que la matriz sea definida positiva.
- Ideal para problemas mal condicionados o con colinealidad, ya que permite gestionar valores singulares pequeños.
- SVD es más robusto, aunque computacionalmente más costoso que Cholesky.

Referencias

[1] L.N. Trefethen and D. Bau, *Numerical Linear Algebra,* Society for Industrial and Applied Mathematics (SIAM), 1997.

[2] C.D. Meyer, *Matrix Analysis and Applied Linear Algebra,* Society for Industrial and Applied Mathematics (SIAM), 2000.

[3] D.S. Watkins, *Fundamentals of Matrix Computations,* Wiley, 2004.

[4] S. Lipschutz and M. Lipson, *Linear Algebra, 4th Ed.,* McGraw-Hill, 2009.

[5] S. Mancho and M.P. Calvo Cabrero, *La descomposición en valores singulares,* Universidad de Valladolid, Facultad de Ciencias, 2013.

[6] R.I. Burden, R.L. Burden, and J.D. Faires, *Análisis numérico,* Grupo Editorial Iberoamérica, 1985.

[7] M. Otárola Lagos, I. Soto Muñoz, and F. Vallejos Améstica, *Descomposición LU y de Cholesky,* Universidad del Bío-Bío, Facultad de Educación y Humanidades, Escuela de Pedagogía en Educación Matemática, 2022.