MANUAL DE USUARIO

Sistemas de Ecuaciones Lineales

Métodos: Cholesky, Jacobi y Gauss-Seidel

ÍNDICE

---

1. INTRODUCCIÓN

Propósito del Programa

Este software implementa tres métodos numéricos avanzados para resolver sistemas de ecuaciones lineales **Ax = b**:

1. **Método de Cholesky**: Método directo para matrices simétricas definidas positivas
2. **Método de Jacobi**: Método iterativo para sistemas grandes
3. **Método de Gauss-Seidel**: Método iterativo mejorado

¿Por qué usar estos métodos?

Cholesky es ideal cuando:

- Tu matriz es simétrica ($A = A^T$)
- Tu matriz es definida positiva
- Necesitas máxima eficiencia (50% más rápido que Gauss estándar)

Jacobi es ideal cuando:

- Tienes sistemas muy grandes (miles de ecuaciones)
- La matriz es dispersa (muchos ceros)

- Quieres paralelizar el cálculo

Gauss-Seidel es ideal cuando:

- Mismas condiciones que Jacobi
- Quieres convergencia más rápida (típicamente 2x)
- No necesitas ejecución paralela

Aplicaciones Reales

- **Cholesky**: Análisis estadístico, problemas de mínimos cuadrados, portfolios de inversión
- **Jacobi/Gauss-Seidel**: Simulación de calor/fluidos, análisis de estructuras, circuitos eléctricos

---

2. REQUISITOS DEL SISTEMA

Hardware Mínimo

- Procesador: 1 GHz o superior
- RAM: 2 GB mínimo (4 GB recomendado)
- Espacio en disco: 100 MB

Software Requerido

Python

- **Versión mínima**: Python 3.7
- **Versión recomendada**: Python 3.9 o superior

Verificar instalación:

```
python --version
```

Bibliotecas Necesarias

- **NumPy**: Para cálculos numéricos
- **SciPy**: Para optimización (opcional, futuras extensiones)

Instalación de Dependencias

Windows:

```
pip install numpy scipy
```

Linux/Mac:

pip3 install numpy scipy

Verificar instalación:

python -c "import numpy; print('NumPy:', numpy.__version__)"

---

3. INSTALACIÓN Y EJECUCIÓN

Estructura del Proyecto

```
SistemasEcuacionesNumericos_2/
├── main.py                 # Programa principal
└── metodos/
    ├── cholesky.py         # Implementación de Cholesky
    └── iterativos.py       # Jacobi y Gauss-Seidel
```

Métodos de Ejecución

Método 1: Línea de Comandos (Recomendado)

```
cd SistemasEcuacionesNumericos_2
python main.py
```

Método 2: Usando un IDE

1. Abrir PyCharm, VSCode o Spyder
2. Abrir el archivo main.py
3. Presionar F5 o clic en "Run"

Método 3: Doble clic (Windows)

1. Crear un archivo ejecutar.bat con el contenido:
2. @echo offpython main.pypause
3. Doble clic en ejecutar.bat

---

4. DESCRIPCIÓN DETALLADA DE LOS MÉTODOS

4.1 MÉTODO DE CHOLESKY

Fundamento Matemático

El método de Cholesky descompone una matriz simétrica definida positiva A en:

$A = L \times L^T$

Donde L es una matriz triangular inferior.

Algoritmo Paso a Paso

1. **Verificación**: Comprobar que A es simétrica y definida positiva
2. **Descomposición**: Calcular L tal que $A = L \times L^T$
3. **Sustitución Forward**: Resolver $Ly = b$
4. **Sustitución Backward**: Resolver $L^Tx = y$

Ventajas

- 2x más rápido que eliminación de Gauss
- Más estable numéricamente
- Requiere menos operaciones: $O(n^3/3)$ vs $O(2n^3/3)$
- Garantiza solución si la matriz es definida positiva

Limitaciones

- Solo funciona con matrices simétricas definidas positivas
- Falla si hay eigenvalores negativos o cero

Condiciones de Uso

Matriz Simétrica: $A = A^T$

```
| 4  2  1 |     | 4  2  1 |
| 2  5  3 | =   | 2  5  3 | ✓
| 1  3  6 |     | 1  3  6 |
```

Matriz Definida Positiva: Todos los eigenvalores > 0

- Prueba rápida: Todos los determinantes de submatrices principales > 0

4.2 MÉTODO DE JACOBI

Fundamento Matemático

Método iterativo que actualiza cada variable usando:

$$x_i^{(k+1)} = (b_i - \sum_{j \neq i}(a_{ij} \cdot x_j^{(k)})) / a_{ii}$$

Algoritmo Paso a Paso

1. **Inicialización**: $x^{(0)} = [0, 0, ..., 0]$
2. **Iteración**: Calcular todas las nuevas componentes usando valores viejos
3. **Verificación**: Si $||x^{(k+1)} - x^{(k)}|| <$ tolerancia → CONVERGE
4. **Repetir** hasta convergencia o máximo de iteraciones

Ventajas

- Excelente para sistemas grandes y dispersos
- Fácil de paralelizar (cada ecuación es independiente)
- No modifica la matriz original
- Uso eficiente de memoria para matrices dispersas

Limitaciones

- Puede no converger si la matriz no es diagonalmente dominante
- Convergencia más lenta que Gauss-Seidel
- Requiere más iteraciones

Condición de Convergencia

Diagonal Dominante Estricta:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \text{ para todo } i$$

Ejemplo:

| 10 -1 2 | Fila 1: $\|10\| > \|-1\| + \|2\| = 3$ ✓
| -1 11 -1 | Fila 2: $\|11\| > \|-1\| + \|-1\| = 2$ ✓
| 2 -1 10 | Fila 3: $\|10\| > \|2\| + \|-1\| = 3$ ✓

4.3 MÉTODO DE GAUSS-SEIDEL

Fundamento Matemático

Similar a Jacobi, pero usa valores actualizados inmediatamente:

$$x_i^{(k+1)} = (b_i - \sum_{j<i} (a_{ij} \cdot x_j^{(k+1)}) - \sum_{j>i} (a_{ij} \cdot x_j^{(k)})) / a_{ii}$$

Diferencia Clave con Jacobi

- **Jacobi**: Usa todos los valores de la iteración anterior
- **Gauss-Seidel**: Usa valores nuevos tan pronto como estén disponibles

Ventajas

- Converge ~2x más rápido que Jacobi
- Requiere menos memoria (un solo vector)
- Mejor para matrices con patrón específico

Limitaciones

- Difícil de paralelizar (dependencia secuencial)
- Misma condición de convergencia que Jacobi

---

5. GUÍA DE USO COMPLETA

Inicio del Programa

Al ejecutar python main.py, verá:

=== SISTEMAS DE ECUACIONES LINEALES ===
1) Método de Cholesky
2) Método de Jacobi
3) Método de Gauss-Seidel
4) Salir
Seleccione una opción:

5.1 Usar Método de Cholesky

Paso 1: Seleccionar opción 1

Paso 2: Ingresar tamaño del sistema

Ingrese el tamaño n de la matriz A (n x n): 3

Paso 3: Ingresar matriz A

Ingrese las filas de A (separe los valores con espacios):
Fila 1: 4 2 2
Fila 2: 2 5 1
Fila 3: 2 1 6

Tips:

- Use espacios para separar valores
- Use punto (.) para decimales: 3.5 no 3,5
- Presione Enter después de cada fila

Paso 4: Ingresar vector b

Ingrese el vector b (3 valores separados por espacio): 4 3 2

Paso 5: Ver resultado

Solución encontrada:
[0.65517241 0.34482759 0.13793103]

Interpretación: $x_1 \approx 0.655$, $x_2 \approx 0.345$, $x_3 \approx 0.138$

5.2 Usar Método de Jacobi

Pasos 1-4: Igual que Cholesky

Paso 5: Configurar parámetros iterativos

Ingrese tolerancia (ejemplo: 1e-6): 1e-6

Ingrese el máximo de iteraciones (ejemplo: 1000): 1000

Parámetros explicados:

- **Tolerancia**: Criterio de parada. Valores típicos:
  - 1e-3: Baja precisión, rápido
  - 1e-6: Precisión estándar (recomendado)
  - 1e-10: Alta precisión, más iteraciones
- **Máximo de iteraciones**: Límite de seguridad
  - Sistemas pequeños (n < 10): 100-500
  - Sistemas medianos (10 < n < 100): 1000-5000
  - Sistemas grandes (n > 100): 10000+

Paso 6: Ver resultado

Convergió: True en 28 iteraciones.
Aproximación de x:
[0.65517543 0.34482876 0.13793054]

Información proporcionada:

- **Convergió**: True/False indica si alcanzó la tolerancia
- **Iteraciones**: Número de pasos realizados
- **Solución**: Vector resultante

5.3 Usar Método de Gauss-Seidel

Proceso idéntico a Jacobi (pasos 1-6).

Diferencia esperada:

- Gauss-Seidel converge en ~15 iteraciones
- Jacobi converge en ~28 iteraciones
- Ambos dan la misma solución final

---

6. PARÁMETROS Y CONFIGURACIÓN

Tolerancia (tol)

¿Qué es?

Define cuán "cerca" deben estar dos iteraciones consecutivas para considerar que el método convergió.

Valores recomendados:

Aplicación	Tolerancia	Descripción
Cálculos rápidos	1e-3	Precisión de 3 decimales
Uso general	1e-6	**Recomendado** - Precisión de 6 decimales
Alta precisión	1e-10	Para cálculos críticos
Máxima precisión	1e-15	Límite de precisión de máquina

Impacto en iteraciones:

tol = 1e-3 → 10 iteraciones
tol = 1e-6 → 30 iteraciones
tol = 1e-10 → 50 iteraciones

Máximo de Iteraciones (max_iter)

Propósito:

Evitar bucles infinitos cuando el método no converge.

Valores recomendados:

Tamaño del sistema	max_iter
n < 10	100-500
10 ≤ n < 50	1000
50 ≤ n < 100	5000
n ≥ 100	10000+

Señales de problemas:

- Si alcanza el máximo sin converger → matriz mal condicionada
- Si converge en muy pocas iteraciones (< 5) → tolerancia muy grande

---

7. INTERPRETACIÓN DE RESULTADOS

7.1 Resultado Exitoso (Cholesky)

Solución encontrada:
[2.5 3.1 -0.8]

Interpretación:

- $x_1 = 2.5$
- $x_2 = 3.1$
- $x_3 = -0.8$

Verificación manual: Sustituir en las ecuaciones originales y comprobar que se satisfacen.

7.2 Resultado Exitoso (Métodos Iterativos)

Convergió: True en 25 iteraciones.
Aproximación de x:
[2.50001234 3.09998876 -0.80000456]

Interpretación:

- **Convergió: True** → Solución confiable
- **25 iteraciones** → Convergencia normal
- Los valores tienen pequeños errores numéricos (normal en métodos iterativos)

7.3 No Convergencia

Convergió: False en 1000 iteraciones.
Aproximación de x:
[15.234 -8.123 22.456]

Posibles causas:

1. Matriz no es diagonalmente dominante
2. Sistema mal condicionado
3. Tolerancia demasiado estricta
4. Necesita más iteraciones

Soluciones:

- Verificar que la matriz cumple condiciones de convergencia
- Aumentar max_iter a 5000 o 10000
- Relajar tolerancia a 1e-4
- Intentar con otro método

7.4 Errores Comunes

Error: "La matriz A no es simétrica"

Error: La matriz A no es simétrica.

Causa: Intentaste usar Cholesky con matriz no simétrica **Solución:** Verifica que A[i,j] = A[j,i] para todo i,j

Error: "La matriz no es definida positiva"

Error: La matriz no es definida positiva.

Causa: La matriz tiene eigenvalores negativos o cero **Solución:** No puedes usar Cholesky. Usa Gauss o LU en su lugar

Error: "Cero en la diagonal"

ZeroDivisionError: Cero en la diagonal.

Causa: Hay un cero en la diagonal principal **Solución:** Reordena las ecuaciones o usa pivoteo

---

8. SOLUCIÓN DE PROBLEMAS COMUNES

Problema 1: El programa no inicia

Síntoma:

python: command not found

Soluciones:

1. Verificar que Python esté instalado: python --version
2. En algunos sistemas usar python3 en lugar de python
3. Reinstalar Python desde python.org

Problema 2: Módulo no encontrado

Síntoma:

ModuleNotFoundError: No module named 'numpy'

Solución:

```
pip install numpy
# o
pip3 install numpy
```

Problema 3: Resultados incorrectos

Síntoma: La solución no satisface las ecuaciones originales

Diagnóstico:

1. Verificar entrada de datos (error más común)
2. Calcular ||Ax - b|| manualmente
3. Verificar condición de la matriz

Verificación paso a paso:

```
import numpy as np
A = np.array([[...]])  # tu matriz
b = np.array([...])    # tu vector
x = np.array([...])    # solución obtenida
residuo = np.dot(A, x) - b
print("Residuo:", np.linalg.norm(residuo))
```

```
# Debe ser < 1e-6 para ser correcto
```

Problema 4: Convergencia muy lenta

Síntoma: Más de 1000 iteraciones sin converger

Soluciones:

1. **Reescalar el sistema**: Divide cada ecuación por su coeficiente más grande
2. **Mejorar dominancia diagonal**: Reordena ecuaciones
3. **Usar precondicionamiento**: Multiplica A por una matriz que mejore su condición
4. **Cambiar de método**: Usa Cholesky si es posible

Problema 5: Números muy grandes o muy pequeños

Síntoma:

```
[1.23e+15  -4.56e+14  7.89e+15]
```

Causa: Sistema mal escalado o mal condicionado

Soluciones:

1. Normalizar ecuaciones
2. Usar pivoteo parcial
3. Verificar que los datos sean correctos

---

9. CASOS DE USO Y APLICACIONES

Caso 1: Análisis de Portafolio de Inversiones (Cholesky)

Problema: Optimizar portafolio minimizando riesgo

Por qué Cholesky:

- La matriz de covarianza es simétrica
- Definida positiva por construcción
- Necesitas alta precisión

Sistema típico:

```
matriz de covarianza × pesos = retorno esperado
```

Caso 2: Simulación de Temperatura en Placa (Jacobi/Gauss-Seidel)

Problema: Ecuación de calor en 2D discretizada

Por qué métodos iterativos:

- Sistema muy grande (100x100 = 10,000 ecuaciones)
- Matriz muy dispersa (solo 5 elementos por fila)
- Diagonalmente dominante por naturaleza del problema

Ventaja: Converge en pocas iteraciones incluso para sistemas enormes

Caso 3: Análisis de Circuitos Eléctricos (Gauss-Seidel)

Problema: Leyes de Kirchhoff en circuito complejo

Por qué Gauss-Seidel:

- Convergencia rápida
- Sistema pequeño-mediano
- No necesita paralelización

---

10. PREGUNTAS FRECUENTES

¿Cuál método es más rápido?

Para matrices simétricas definidas positivas: Cholesky > Gauss > Gauss-Seidel > Jacobi

Para matrices diagonalmente dominantes: Gauss-Seidel > Jacobi > Cholesky (no aplica)

¿Qué hago si ningún método converge?

1. Verificar que el sistema tiene solución única (det(A) ≠ 0)
2. Usar método directo (Gauss del Trabajo 1)
3. Mejorar condicionamiento del sistema
4. Consultar con un experto en análisis numérico

¿Puedo resolver sistemas no cuadrados?

No, este programa solo resuelve sistemas cuadrados (n ecuaciones, n incógnitas). Para sistemas rectangulares, usar mínimos cuadrados.

¿Cuánta precisión puedo esperar?

- **Cholesky**: Precisión de máquina (~1e-15)
- **Jacobi/Gauss-Seidel**: Depende de la tolerancia configurada

---

Versión: 1.0
Fecha: Octubre 2025
Compatibilidad: Python 3.7+, NumPy 1.19+, SciPy 1.5+