Escuela Politecnica Nacional

Metodos Numericos

Tarea 09

Fecha: 13/07/2025

Nombre: Joseph Jimenez

Enlace al repositorio : https://github.com/Josefu-zero/Metodos-Numericos-Tareas/tree/main/Tarea09

1. Para cada uno de los siguientes sistemas lineales, obtenga, de ser posible, una solución con métodos gráficos.

Explique los resultados desde un punto de vista geométrico.

Esta librería tiene una función para resolver un sistema de ecuaciones mediante matrices.

In [2]:
```python
import numpy as np
```

a)

$$x_1 + 2x_2 = 0$$
$$x_1 - x_2 = 0$$

In [3]:
```python
A = [
    [1, 2],
    [1, -1]
]

b = [0, 0]

x = np.linalg.solve(A, b)
print(x)
```

```
[ 0. -0.]
```

b)

$$x_1 + 2x_2 = 3$$
$$-2x_1 - 4x_2 = 6$$

In [4]:
```python
A = [
    [1, 2],
```

```
    [-2, -4]
]

b = [3, 6]

x = np.linalg.solve(A, b)
print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[4], line 8
      1 A = [
      2     [1, 2],
      3     [-2, -4]
      4 ]
      6 b = [3, 6]
----> 8 x = np.linalg.solve(A, b)
      9 print(x)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:410, in solve(a, b)
    407 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    408 with errstate(call=_raise_linalgerror_singular, invalid='call',
    409               over='ignore', divide='ignore', under='ignore'):
--> 410     r = gufunc(a, b, signature=signature)
    412 return wrap(r.astype(result_t, copy=False))

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:104, in _raise_linalgerror_singular(err, flag)
    103 def _raise_linalgerror_singular(err, flag):
--> 104     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

R: No existe solución.

c)

$$2x_1 + x_2 = -1$$

$$x_1 + x_2 = 2$$

$$x_1 - 3x_2 = 5$$

In [ ]:

```
A = [
    [2, 1],
    [1, 1],
    [1, -3]
]

b = [-1, 2, 5]

x = np.linalg.solve(A, b)
print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[5], line 9
      1 A = [
      2     [2, 1],
      3     [1, 1],
      4     [1, -3]
      5 ]
      7 b = [-1, 2, 5]
----> 9 x = np.linalg.solve(A, b)
     10 print(x)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:457, in solve(a, b)
    384 """
    385 Solve a linear matrix equation, or system of linear scalar equations.
    386
   (...)    454
    455 """
    456 a, _ = _makearray(a)
--> 457 _assert_stacked_square(a)
    458 b, wrap = _makearray(b)
    459 t, result_t = _commonType(a, b)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:264, in _assert_stacked_square(*arrays)
    261     raise LinAlgError('%d-dimensional array given. Array must be '
    262             'at least two-dimensional' % a.ndim)
    263 if m != n:
--> 264     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

R: Existen soluciones infinitas

d)

$$2x_1 + x_2 + x_3 = 1$$

$$2x_1 + 4x_2 - x_3 = -1$$

In [ ]:

```
A = [
    [2, 1, 1],
    [2, 4, -1]
]

b = [1, -1]

x = np.linalg.solve(A, b)
print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[6], line 8
      1 A = [
      2     [2, 1, 1],
      3     [2, 4, -1]
      4 ]
      6 b = [1, -1]
----> 8 x = np.linalg.solve(A, b)
      9 print(x)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:457, in solve(a, b)
    384 """
    385 Solve a linear matrix equation, or system of linear scalar equations.
    386
   (...)    454
    455 """
    456 a, _ = _makearray(a)
--> 457 _assert_stacked_square(a)
    458 b, wrap = _makearray(b)
    459 t, result_t = _commonType(a, b)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:264, in _assert_stacked_square(*arrays)
    261     raise LinAlgError('%d-dimensional array given. Array must be '
    262             'at least two-dimensional' % a.ndim)
    263 if m != n:
--> 264     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

Soluciones infinitas

2. Utilice la eliminación gaussiana con sustitución hacia atrás y aritmética de redondeo de dos dígitos para resolver los siguientes sistemas lineales. No reordene las ecuaciones. (La solución exacta para cada sistema es $x_1 = -1$, $x_2 = 2$, $x_3 = 3$.)

a)

$$-x_1 + 4x_2 + x_3 = 8$$

$$\frac{5}{3}x_1 + \frac{2}{3}x_2 - \frac{2}{3}x_3 = 1$$

$$2x_1 + x_2 + 4x_3 = 11$$

In [5]:
```
A = [
    [-1, 4, 1],
    [round(5/3, 2), round(2/3, 2), round(2/3, 2)],
    [2, 1, 4]
]

b = [8, 1, 11]
```

```python
x = np.linalg.solve(A, b)
print(x)
```

```
[-1.00651042  0.99739583  3.00390625]
```

b)

$$4x_1 + 2x_2 - x_3 = -5$$

$$\frac{1}{9}x_1 + \frac{1}{9}x_2 - \frac{1}{3}x_3 = -1$$

$$x_1 + 4x_2 + 2x_3 = 9$$

In [8]:
```python
A = [
    [4, 2, -1],
    [round(1/9, 2), round(1/9, 2), round(1/3, 2)],
    [1, 4, 2]
]

b = [-5, -1, 9]

x = np.linalg.solve(A, b)
print(x)
```

```
[-4.52993348  4.97117517 -3.17738359]
```

3. Utilice el algoritmo de eliminación gaussiana para resolver, de ser posible, los siguientes sistemas de ecuaciones lineales y muestre los intercambios de fila necesarios.

In [9]:
```python
def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    A = np.array(A)
    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1).'
    n = A.shape[0]

    for i in range(0, n - 1):  # loop por columna

        # --- encontrar pivote
        p = None  # default, first element
        for pi in range(i, n):
            if A[pi, i] == 0:
                # must be nonzero
                continue

            if p is None:
                # first nonzero element
                p = pi
                continue

            if abs(A[pi, i]) < abs(A[p, i]):
                p = pi

        if p is None:
            # no pivot found.
            raise ValueError("No existe solución única.")

        if p != i:
            # swap rows
```

```python
            print(f"Intercambiando filas {i} y {p}")
            _aux = A[i, :].copy()
            A[i, :] = A[p, :].copy()
            A[p, :] = _aux

        # --- Eliminación: loop por fila
        for j in range(i + 1, n):
            m = A[j, i] / A[i, i]
            A[j, i:] = A[j, i:] - m * A[i, i:]


    if A[n - 1, n - 1] == 0:
        raise ValueError("No existe solución única.")

        print(f"\n{A}")
    # --- Sustitución hacia atrás
    solucion = np.zeros(n)
    solucion[n - 1] = A[n - 1, n] / A[n - 1, n - 1]

    for i in range(n - 2, -1, -1):
        suma = 0
        for j in range(i + 1, n):
            suma += A[i, j] * solucion[j]
        solucion[i] = (A[i, n] - suma) / A[i, i]

    return solucion
```

a)

$$x_1 - x_2 + 3x_3 = 2$$

$$3x_1 - 3x_2 + 1x_3 = -1$$

$$x_1 + x_2 = 3$$

In [10]:

```python
A = [
    [1, -1, 3, 2],
    [3, -3, 1, -1],
    [1, 1, 0, 3]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
Intercambiando filas 1 y 2
[1.1875 1.8125 0.875 ]
```

b)

$$2x_1 - 1.5x_2 + 3x_3 = 1$$

$$-x_1 + 2x_3 = 3$$

$$4x_1 - 4.5x_2 + 5x_3 = 1$$

In [11]:

```python
A = [
    [2, -1.5, 3, 1],
    [-1, 0, 2, 3],
    [4, -4.5, 5, 1]
]
```

```python
x = eliminacion_gaussiana(A)
print(x)
```

```
Intercambiando filas 0 y 1
[-1. -0.  1.]
```

c)

$$2x_1 = 3$$

$$x_1 + 1.5x_2 = 4.5$$

$$-3x_2 - 0.5x_3 = -6.6$$

$$2x_1 - 2x_2 + x_3 + x_4 = 0.8$$

In [ ]:
```python
A = [
    [2, 0, 0, 0, 3],
    [1, 1.5, 0, 0, 4.5],
    [0, -3, 0.5, 0, -6.6],
    [2, -2, 1, 1, 0.8]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
Intercambiando filas 0 y 1
[ 1.5  2.  -1.2  3. ]
```

d)

$$x_1 + x_2 + x_4 = 2$$

$$2x_1 + x_2 - x_3 + x_4 = 1$$

$$4x_1 - x_2 - 2x_3 + 2x_4 = 0$$

$$3x_1 - x_2 - x_3 + 2x_4 = -3$$

In [12]:
```python
A = [
    [1, 1, 0, 1, 2],
    [2, 1, -1, 1, 1],
    [4, -1, -2, 2, 0],
    [3, -1, -1, 2, -3]
]

x = eliminacion_gaussiana(A)
print(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[12], line 8
      1 A = [
      2     [1, 1, 0, 1, 2],
      3     [2, 1, -1, 1, 1],
      4     [4, -1, -2, 2, 0],
      5     [3, -1, -1, 2, -3]
      6 ]
----> 8 x = eliminacion_gaussiana(A)
      9 print(x)

Cell In[9], line 41, in eliminacion_gaussiana(A)
     37         A[j, i:] = A[j, i:] - m * A[i, i:]
     40 if A[n - 1, n - 1] == 0:
---> 41     raise ValueError("No existe solución única.")
     43     print(f"\n{A}")
     44 # --- Sustitución hacia atrás

ValueError: No existe solución única.
```

4.Use el algoritmo de eliminación gaussiana y la aritmética computacional de precisión de 32 bits para resolver los siguientes sistemas lineales.

In [14]:

```python
def eliminacion_gaussiana(A: np.ndarray) -> np.ndarray:
    A = np.array(A)

    assert A.shape[0] == A.shape[1] - 1, "La matriz A debe ser de tamaño n-by-(n+1).'
    n = A.shape[0]

    for i in range(0, n - 1):  # Loop por columna

        # --- encontrar pivote
        p = None  # default, first element
        for pi in range(i, n):
            if A[pi, i] == 0:
                # debe ser distinto de cero
                continue

            if p is None:
                # first nonzero element
                p = pi
                continue

            if abs(A[pi, i]) < abs(A[p, i]):
                p = pi

        if p is None:
            # no pivot found.
            raise ValueError("No existe solución única.")

        if p != i:
            # swap rows
            print(f"Intercambiando filas {i} y {p}")
            _aux = A[i, :].copy()
            A[i, :] = A[p, :].copy()
            A[p, :] = _aux
```

```python
        # --- Eliminación: loop por fila
        for j in range(i + 1, n):
            m = A[j, i] / A[i, i]
            A[j, i:] = A[j, i:] - m * A[i, i:]

    if A[n - 1, n - 1] == 0:
        raise ValueError("No existe solución única.")

    # --- Sustitución hacia atrás
    solucion = np.zeros(n, dtype=np.float32)
    solucion[n - 1] = A[n - 1, n] / A[n - 1, n - 1]

    for i in range(n - 2, -1, -1):
        suma = 0
        for j in range(i + 1, n):
            suma += A[i, j] * solucion[j]
        solucion[i] = (A[i, n] - suma) / A[i, i]

    return solucion
```

a)

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 = 9$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = 8$$

$$\frac{1}{2}x_1 + x_2 + 2x_3 = 8$$

In [ ]:
```python
A = np.array([[1/4, 1/5, 1/6, 9],
              [1/3, 1/4, 1/5, 8],
              [1/2, 1, 2, 8]], dtype=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
[-227.07666  476.92264 -177.69217]
```

b)

$$3.333x_1 + 15920x_2 - 10.333x_3 = 15913$$

$$2.222x_1 + 16.71x_2 + 9.612x_3 = 28.544$$

$$1.5611x_1 + 5.1791x_2 + 1.6852x_3 = 8.4254$$

In [15]:
```python
A = np.array([[3.333, 15920, 10.333, 15913],
              [2.222, 16.71, 9.612, 28.544],
              [1.5611, 5.1791, 1.6852, 8.4254]], dtype=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 2
[1.0024955 0.9987004 1.0016824]
```

c)

$$x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 = \frac{1}{6}$$

$$\frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 + \frac{1}{5}x_4 = \frac{1}{7}$$

$$\frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 + \frac{1}{6}x_4 = \frac{1}{8}$$

$$\frac{1}{4}x_1 + \frac{1}{5}x_2 + \frac{1}{6}x_3 + \frac{1}{7}x_4 = \frac{1}{9}$$

In [16]:
```
A = np.array([[1, 1/2, 1/3, 1/4, 1/6],
              [1/2, 1/3, 1/4, 1/5, 1/7],
              [1/3, 1/4, 1/5, 1/6, 1/8],
              [1/4, 1/5, 1/6, 1/7, 1/9]], dtype=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 3
Intercambiando filas 1 y 2
[-0.03174075  0.5951853  -2.3808312   2.7777011 ]
```

d)

$$2x_1 + x_2 - x_3 + x_4 - 3x_5 = 7$$

$$x_1 + 2x_3 - x_4 + x_5 = 2$$

$$-2x_2 - x_3 + x_4 - x_5 = -5$$

$$3x_1 + x_2 - 4x_3 + 5x_5 = 6$$

$$x_1 - x_2 - x_3 - x_4 + x_5 = -3$$

In [17]:
```
A = np.array([[2, 1, -1, 1, -3, 7],
              [1, 0, 2, -1, 1, 2],
              [0, -2, -1, 1, -1, -5],
              [3, 1, -4, 0, 5, 6],
              [1, -1, -1, -1, 1, -3]], dtype=np.float32)

solucion = eliminacion_gaussiana(A)
print(solucion)
```

```
Intercambiando filas 0 y 1
Intercambiando filas 2 y 3
Intercambiando filas 3 y 4
[1.8830411  2.807017   0.73099416 1.4385967  0.09356727]
```

5.Dado el sistema lineal:

$$x_1 - x_2 + \alpha x_3 = -2$$

$$-x_1 + 2x_2 - \alpha x_3 = 3$$

$$\alpha x_1 + x_2 + x_3 = 2$$

a)

Encuentre el valor(es) de α para los que el sistema no tiene soluciones.

Se puede obtener α si calculamos el determinante de esta matriz y la igualamos a cero, entonces:

$$det(A) = (2+\alpha) + (-1+\alpha) + \alpha(-1-2\alpha) = 0$$

$$det(A) = 2\alpha^2 - \alpha + 1 = 0$$

$$\alpha = 1; \alpha = -\frac{1}{2}$$

Si α tiene alguno de los valores del conjunto: $\color{red}{\backslash\text{set}}{-\frac{1}{2}, 1}$, entonces el sistema no tiene solución.

b)

Encuentre el valor(es) de α para los que el sistema tiene un número infinito de soluciones.

Para encontrar el valor de α necesitamos reducir la matriz ampliada de este sistema de ecuaciones de manera que encontremos la última fila dependiendo de este escalar. Si α, para este momento es igual a cero, entonces existen soluciones infinitas.

$$\begin{pmatrix} 1 & -1 & \alpha & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & (1-\alpha^2) & \dots \end{pmatrix}$$

Esta es la matriz reducida, y el valor $(1-\alpha^2)$ es lo que nos interesa.

$$1 - \alpha^2 = 0$$

Si α tiene alguno de los valores del conjunto $\color{red}{\backslash\text{set}}{-1, 1}$, entonces el sistema tiene soluciones infinitas.

c)

Suponga que existe una única solución para una a determinada, encuentre la solución.

Asignemos de manera: $\alpha = 2$. Con esto, calculemos las soluciones mediante numpy.

In [ ]:
```
A = [
    [1, -1, 2],
    [-1, 2, -2],
    [2, 1, 1]
]

b = [-2, 3, 2]

x = np.linalg.solve(A, b)
print(x)
```

```
[ 1.  1. -1.]
```

El sistema si tiene solución.

6. Suponga que en un sistema biológico existen n especies de animales y m fuentes de alimento. Si x_j representa la población de las j-ésimas especies, para cada $j = 1, \ldots, n$,;b_i; representa el suministro diario disponible del i-ésimoalimento $a_{ij}j$ representa la cantidad del i-ésimo alimento

a)

Si:

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$x = (x_j) = (1000, 500, 350, 400)$ y $b = (b_i) = (3500, 2700, 900)$. ¿Existe suficiente alimento para satisfacer el promedio consumo diario? .

In [18]:
```
A = [
    [1, 2, 0, 3],
    [1, 0, 2, 2],
    [0, 0, 1, 1]
]

x = [1000, 500, 350, 400]

x = np.matmul(A, x)
print(x)
```

```
[3200 2500  750]
```

El alimento fue suficiente para satisfacer a las poblaciones de especies que conviven debido al suministro por donde se parte ($[3500, 2700, 900]$).

b) Cuál es el número máximo de animales de cada especie que se podría agregar de forma individual al sistema con el suministro de alimento que cumpla con el consumo?

In [19]:
```
A = [
    [1, 2, 0, 3],
    [1, 0, 2, 2],
    [0, 0, 1, 1]
]

b = [3500, 2700, 900]

x = np.linalg.solve(A, b)
print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[19], line 9
      1 A = [
      2     [1, 2, 0, 3],
      3     [1, 0, 2, 2],
      4     [0, 0, 1, 1]
      5 ]
      7 b = [3500, 2700, 900]
----> 9 x = np.linalg.solve(A, b)
     10 print(x)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:396, in solve(a, b)
    394 a, _ = _makearray(a)
    395 _assert_stacked_2d(a)
--> 396 _assert_stacked_square(a)
    397 b, wrap = _makearray(b)
    398 t, result_t = _commonType(a, b)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:202, in _assert_stacked_square(*arrays)
    200 m, n = a.shape[-2:]
    201 if m != n:
--> 202     raise LinAlgError('Last 2 dimensions of the array must be square')

LinAlgError: Last 2 dimensions of the array must be square
```

No se puede determinar de forma exacta ya que hay muchas distribuciones posibles

c) Si la especie 1 se extingue, ¿qué cantidad de incremento individual de las especies restantes se podría soportar?

Por lo tanto...

In [20]:
```
A = [
    [2, 0, 3],
    [0, 2, 2],
    [0, 1, 1]
]

b = [3500, 2700, 900]

x = np.linalg.solve(A, b)
print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[20], line 9
      1 A = [
      2     [2, 0, 3],
      3     [0, 2, 2],
      4     [0, 1, 1]
      5 ]
      7 b = [3500, 2700, 900]
----> 9 x = np.linalg.solve(A, b)
     10 print(x)

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:410, in solve(a, b)
    407 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    408 with errstate(call=_raise_linalgerror_singular, invalid='call',
    409               over='ignore', divide='ignore', under='ignore'):
--> 410     r = gufunc(a, b, signature=signature)
    412 return wrap(r.astype(result_t, copy=False))

File c:\Users\andyj\AppData\Local\Programs\Python\Python313\Lib\site-packages\numpy\l
inalg\_linalg.py:104, in _raise_linalgerror_singular(err, flag)
    103 def _raise_linalgerror_singular(err, flag):
--> 104     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

no parece existir una distribución exacta.

d) Si la especie 2 se extingue, ¿qué cantidad de incremento individual de las especies restantes se podría soportar?

In [21]:
```
A = [
    [1, 0, 3],
    [1, 2, 2],
    [0, 1, 1]
]

b = [3500, 2700, 900]

x = np.linalg.solve(A, b)
print(x)
```

```
[900.          33.33333333 866.66666667]
```

En este caso se determinó la población que pueden tener cada una de las especies (Especie 1: 900, especie 2: 33, especie 3: 866)