Tarea5
Métodos Numéricos

Benjamin Rivera

4 de octubre de 2020

Índice

1. Tarea 5

Tarea 5 de *Benjamín Rivera* para el curso de **Métodos Numéricos** impartido por *Joaquín Peña Acevedo*. Fecha limite de entrega **4 de Octubre de 2020**.

1.0.1. Como ejecutar

Requerimientos Este programa se ejecuto en mi computadora con la version de **Python 3.8.2** y con estos requerimientos

Jupyter En caso de tener acceso a un *servidor jupyter* ,con los requerimientos antes mencionados, unicamente basta con ejecutar todas las celdas de este *notebook*. Probablemente no todas las celdas de *markdown* produzcan el mismo resultado por las *Nbextensions*.

Consola Habrá archivos e instrucciones para poder ejecutar cada uno de los ejercicios desde la consola.

Si todo sale mal En caso de que todo salga mal, tratare de dejar una copia disponible en **GoogleColab** que se pueda ejecutar con la versión de **Python** de *GoogleColab*

[138]:
```
usage = """
Programa correspondiente a la Tarea 5 de Metodos Numericos.
Este programa espera leer los archivos de tipo npy

Alumno: Benjamin Rivera

Usage:
  Tarea5.py ejercicio1 <matA> <vecB> <N>[--path=<path>]
  Tarea5.py -h | --help

Options:
  -h --help       Show this screen.
  -v --version    Show version.
  --path=<path>   Directorio para buscar archivos [default: data/].
"""
import sys
import scipy
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_triangular

if __name__ == "__main__":
    import doctest
    from docopt import docopt
    doctest.testmod()
    args = docopt(usage, version='Tarea4, prb')


    if args['ejercicio3']:
        Ejercicio3(args['<matA>'], args['<vecB>'], args['<N>'],␣
 ↪args['--path'])
```

1.1. Ejercicio 1

Considere la matriz

$$A = \begin{pmatrix} a^2 & a & a/2 & 1 \\ a & -9 & 1 & 0 \\ a/2 & 1 & 10 & 0 \\ 1 & 0 & 0 & a \end{pmatrix}$$

Da un rango de valores para a de manera que garantice la convergencia del método de Jacobi.

1.1.1. Respuesta

Por las notas del curso (ppt clase 9, diapositiva 8-41), sabemos que el método de Jacobi converge cuando la matriz A es **estrictamente diagonal dominante**. Y esto es cierto cuando

$$\forall i \in [1, \ldots, n], |a_{i,i}| > \sum_{j=1, j \neq i}^{n} |a_{i,j}|$$

si extendemos estas desigualdades para la matriz A nos queda que

$$sol = \begin{cases} |a^2| & > |a| + |a/2| + |1| \\ |-9| & > |a| + |1| + |0| \\ |10| & > |a/2| + |1| + |0| \\ |a| & > |1| + |0| + |0| \end{cases} \tag{1}$$

$$\text{despues de simplificar queda que} \tag{2}$$

$$= \begin{cases} a^2 & > |a| + |a/2| + 1 \\ 8 & > |a| \\ 9 & > |a/2| \\ |a| & > 1 \end{cases} \tag{3}$$

$$\tag{4}$$

$$= \begin{cases} a^2 & > |a| + |a/2| + 1 \\ 64 & > a^2 \\ 4 * 91 & > a^2 \\ a^2 & > 1 \end{cases} \tag{5}$$

$$\text{realcionamos 5.4 con 5.3 y 5.2} \tag{6}$$

$$= \begin{cases} a^2 & > |a| + |a/2| + 1 \\ 8^2 & > a^2 > 1 \\ 4 * 9^2 & > a^2 > 1 \end{cases} \tag{7}$$

$$\text{de esto solo nos importa 7.1 y 7.2} \tag{8}$$

$$= \begin{cases} a^2 & > 3|a|/2 + 1 \\ 8 & > a > 1 \end{cases} \tag{9}$$

$$\tag{10}$$

Podemos calcular los intervalos de soluci'on de 9. Estos quedan

$$\begin{cases} a^2 > 3|a|/2 + 1 \Rightarrow & (-\infty, -2) \cup (2, \infty) \\ 8 > a > 1 \Rightarrow & (1, 8) \end{cases} \tag{11}$$

Y la solucion que buscamos es la interseccion de 11. De manera que, para que la matriz A converja con el metodo de Jacobi, se necesita que $x \in (2, 8)$.

1.2. Ejercicio 2

Sea $A \in \mathbb{R}^{n\times n}$ una matriz tridiagonal y que las tres diagonales de interes se almacenan en un arreglo $B_{n\times 3}$.

Escribe las expresiones para calcular las actualizaciones de las componentes del vector $x^{i+1} = \left(x_0^{i+1}, \ldots, x_{n-1}^{i+1}\right)$ de acuerdo con *Gauss-Seidel*. Especificamente escribir la expresion para actualizar x_0^{i+1}, x_i^{i+1} para $i = 1, 2, \ldots, n-2$; ademas de x_{n-1}^{i+1} usando los coeficientes $a_{i,j}$ de A y b_{ij} de B.

1.2.1. Respuesta

Sea A una matriz tridiagonal con elementos $a_{i,j}$, B el arreglo descrito anteriormente con elementos $b'_{i,j}$, b el vector de terminos independientes con elementos b_i y x el vector solucion con $x_i^{(t)}$ su elemento i de la iteraci'on t.

Se da que en el m'etodo de *Gauss-Seidel* original tenemos que los componentes $x^{(t+1)}$ se calculan siguiendo `forwardSubstitution`

$$x_i^{(t+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=0}^{i-1} a_{j,j}x_j^{(t+1)} - \sum_{j=i+1}^{n-1} a_{i,j}x_j^{(t)}\right) \tag{12}$$

pero como en este ejercicio estamos trabajando con una matriz tridiagonal, lo que implica que solo habr'a elementos distitnos de cero en las tres diagonales de interes; entonces podemos reescribir la ecuaci'on~12, lo que queda como

$$\begin{aligned} x_i^{(t+1)} &= \frac{1}{a_{i,i}}\left(b_i - a_{i,j-1}x_{i-1}^{t+1} - a_{i,j+1}x_{i+1}^{t}\right) \\ &= \frac{1}{b'_{1,i}}\left(b_i - b'_{i,0}x_{i-1}^{t+1} - b'_{i,2}x_{i+1}^{t}\right) \qquad \text{Usando el arreglo B} \end{aligned}$$

esto se puede usar $\forall i = 0, 1, \ldots, n-1, n$ sobre el arreglo B.

Espec'ificamente podemos definir al elemento x_0^{t+1} como

$$\begin{aligned} x_0^{t+1} &= \frac{1}{a_{0,0}}\left(b_0 - a_{0,1}x_1^{t}\right) \\ &= \frac{1}{b_{0,1}}\left(b_0 - b'_{0,2}x_1^{t}\right) \end{aligned}$$

y para el elemento x_{n-1}^{i+1} queda que

$$\begin{aligned} x_{n-1}^{i+1} &= \frac{1}{a_{n-1,n-1}}\left(b_{n-1} - a_{n-1,n-2}x_{n-2}^{t+1}\right) \\ &= \frac{1}{a_{n-1,1}}\left(b_{n-1} - b'_{n-1,0}x_{n-2}^{t+1}\right) \end{aligned}$$

1.3. Ejercicio 3

Programa el metodo de *Gauss-Seidel* para resolver sistemas tridiagonales.

[65]:
```python
# Extras

def data4mFile(n_file,/,path='datos/npy/', ext='npy', dtype=np.
 ↪float64):
    """ Cargar matrices y vectores de memoria

    Funcion para cargar los archivos en memoria. El nombre del
    archivo no espera path, ni la extension, solo el nombre. Por
    default trata de leer los archivos .npy, pero numpy soporta
    leer de otros formatos.

        Input:
            n_file := nombre del archivo sin extension
            path := directorio para buscar el archivo
            ext := extension del archivo a buscar (sin punto)
            dtype := tipo de dato para guardar los valores
        Output:
            Regresa el una instancia np.matrix con los datos
                obtenidos del archivo cargado.
    """




def show1D(vec,/, max_sz=8, show=True):
    """ Implementacion para pprint vector 1D.

    Funcion para generar string para poder imporimir un
    vector de manera reducida, dando un maximo de elementos
    a imprimir. Lo puede imprimir directamente si se quiere

    Input:
        vec := vector de informacion a imprimir.
        [opcionales]
        max_sz := Maximo de elementos a imprimir.
        show := Imprimir vector

    _Doctest:
        >>> show1D([1,2,3,4], show=False)
        '1, 2, 3, 4'

        >>> show1D([1,2,3,4,5,6,7,8,9], show=False)
        '1, 2, 3, 4, ..., 6, 7, 8, 9'
    """
```

[143]:

```
# Parte 1

def diagonalesRelevantes(A, dtype=np.float64):
    """ Funcion que otiene las diagonales relevantes de A.

    Esta funcion, con A una matriz tridiagonal cuadrada(n) extrae_
    ra las diagonales relevantes y las pondra en un arreglo B de
    3xn, donde la columna 0 correspondera a la diagonal -1, la col_
    umna 1 a la diagonal de A y la columna 2 a la diagonal +1 de
    la matriz A.
    Se espera, y corrobora, que A sea instancia de np.matrix para
    usar su metodos

    Input:
        A := Matriz tridiagonal cuadrada instancia de np.matrix
    Output:
        B := Arreglo de valores relevantes de A

    """
    if isinstance(A, (np.matrix)):
        return B
    else:
        raise Exception("A no es instancia de np.matrix")
```

[190]:

```
# Parte 2

def error_GS(B, xt, b,/, dtype=np.float64):
    """ Funcion para calular el error || Ax^t - b|| desde B """
    n = len(xt) # esperamos que las dimensiones coincidan
    vec = np.asmatrix(np.zeros((n,1)),
                      dtype=dtype)
    # En vec generaremos Ax^t

    vec[0,0] = B[0,1]*xt[0,0] + B[0,2]*xt[1,0]

    # Calculamos hasta el penultimo
    for i in range(1, n-1):
        vec[i,0] = B[i,0]*xt[i-1,0] + B[i,1]*xt[i,0] +␣
 ↪B[i,2]*xt[i+1,0]

    n = n-1 # Calculamos el ultimo
    vec[n,0] = B[n,0]*xt[n-1,0] + B[n,1]*xt[n,0]

    return np.linalg.norm(vec - b)
```

[190]:

```

def GaussSeidel_tridiagonal(B, xt, b, N,/, t=None, dtype=np.float64):
    """ Implementacion de GaussSeidel para matrices tridiagonales.

    Esta funcion trata de resolver un sistema de ecuaciones Ax = b
```

```
    con A una matriz (nxn) cuadrada tridiagonal y estas diagonales
    almacenadas en el arreglo B (3xn).

    Respecto a la tolerancia t del metodo, en caso de ser None, se
    tomara el epsilon para el tipo de dato dtype que se le pase a
    la funcion (calculado por numpy)

    Input:
        B := arreglo (3xn) de la diagonales relecantes para el metodo
        x0 := vector (nx1) inicial de aproximacion de respuestas
        b :=  vector (nx1) de terminos independientes
        N := maximo numero de iteraciones del metodo

        t := Tolerancia del metodo (default: None)
        dtype := Tipop de dato para trabajar con el metodo

    Output:
        x, n, e
        x := vector respuesta en la iteracion en que se detenga
        n := iteracion en la que se detuvo el metodo
        e := error al momento de detenerse
    """
    # Inicializacion
    if t == None: t = np.finfo(dtype).eps   # Correcion tolerancia
    sz = len(b)
    e = float('inf')                        # Error inicial es infinito
    n = 0                                            # Iteracion inicial

    while n < N:

        # Primer elemento de iteracion
        i = 0
        xt[i,0] = (b[i,0] - B[i,2]*xt[i+1,0])/B[i,1]
        # Iteracion del metodo
        for i in range(1, sz-1):
            xt[i,0] = (b[i,0] - B[i,0]*xt[i-1,0] - B[i,2]*xt[i+1,0])/
↪B[i,1]
        # Ultimo elemento de iteracion
        i = sz-1
        xt[i,0] = (b[i,0] - B[i,0]*xt[i-1,0])/B[i,1]

        # avance bucle verificacion tolerancia
        e = error_GS(B, xt, b, dtype=dtype)
        if e < t:
            break
        n += 1

    return xt, n, e
```

[191]:
```python
# Parte 3

def Ejercicio3(mat, vecb, N,/, path='datos/npy/', show=True):
    """ Funcion para ejecutar la parte 3 de la tarea

    Esta funcion usara las funciones diagonalesRelevantes, error_GS,
    GaussSeidel_tridiagonal, data4mFile y show1D para tratar de
    resolver un sistema Ax = b mediante la variante del metodo de
    Gauss-Seidel para matrices tridiagonales cuadradas

    Input:
        mat := nombre del archivo que contiene una matriz
            tridiagonal
        vecb := nombre del archivo con el vector de terminos
            independientes
        N := numero maximo de iteraciones para el metodo

        path := directorio para buscar los archivos
        show := Indica si se desea imprimir los detalles
    """
    dtype = np.float64
    t = (np.finfo(dtype).eps)**(1/2)

    A = data4mFile(mat,  dtype=dtype)
    b = data4mFile(vecb, dtype=dtype).transpose()

    x0 = np.zeros(b.shape, dtype=dtype)

    # Suponemos que A si es tridiagonal
    B = diagonalesRelevantes(A, dtype=dtype)

    xt, n, e = GaussSeidel_tridiagonal(B, x0, b, N, t=t, dtype=dtype)
    conv = True if e < t else False

    if show:
        # Segunda solucion
        x = np.linalg.solve(A, b)
        # Print
        __ = f'Matriz de "{mat}" con el vector de "{vecb}"'
        __ += f'\n\tIteraciones: {n}'
        __ += f'\n\tError: {e}'
        __ += f'\n\tSol: {show1D(xt,show=False)}\n'
        __ += ('El metodo converge' if e < t else 'El metodo no␣
↪converge')
        __ += f'\nLa diferencia entre soluciones es {np.linalg.
↪norm(x - xt)}'
        print(__)

    return e, n, conv
```

[240]:

```
# Parte 4

if NOTEBOOK:
    sizes = ['6', '20', '500']
    data = {}
    for sz in sizes:
        data[sz] = [[],[],[]]

    itr = [0, 5, 10, 15, 20, 25, 35, 50]

    for sz in sizes:
        for N in itr:
            e, n, conv = Ejercicio3('matrizA'+sz, 'vecb'+sz, N,␣
 ↪show=True)
            data[sz][0].append(e)
            data[sz][1].append(n)
            data[sz][2].append(conv)
```

```
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 0
        Error: inf
        Sol: 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
El metodo no converge
La diferencia entre soluciones es 2.449489742783178
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 5
        Error: 0.00022626705409276946
        Sol: 0.9999715782516367, 1.000046095638286, 0.
 ↪9999528204288628,
1.000020636310501, 0.9999923716262922, 1.0000011924468593
El metodo no converge
La diferencia entre soluciones es 7.51264727492206e-05
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
```

```
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA6" con el vector de "vecb6"
        Iteraciones: 9
        Error: 1.8517803127045294e-09
        Sol: 1.0000000003609595, 0.9999999996391488, 1.
 ↪0000000003090268,
0.9999999998701277, 1.0000000000473834, 0.9999999999925931
El metodo converge
La diferencia entre soluciones es 6.125110654833705e-10
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 0
        Error: inf
        Sol: 0.0, 0.0, 0.0, 0.0, ..., 0.0, 0.0, 0.0, 0.0
El metodo no converge
La diferencia entre soluciones es 44.72135954999579
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 5
        Error: 5.2135922638126534e+17
        Sol: 10510715.21822592, 1872849342.4323115, 335161007203.7978,
30692230637880.035, ..., 4.099992817956197e+16, 4.332425697334634e+17,
3.365085604102188e+17, 6.572227164204795e+17
El metodo no converge
La diferencia entre soluciones es 8.573144572254433e+17
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 10
        Error: 2.1453562634774947e+30
        Sol: -4.325493091370833e+19, -7.706464998382885e+21,
-1.3791289121479942e+24, -1.2629315580338113e+26, ..., -1.
 ↪687103028015392e+29,
-1.782760425966517e+30, -1.3847085054535812e+30, -2.
 ↪704424173623813e+30
```

```
El metodo no converge
La diferencia entre soluciones es 3.527786386918865e+30
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 15
        Error: 8.827765445581566e+42
        Sol: 1.7798646821865153e+32, 3.171075432413957e+34,
5.674874034154363e+36, 5.196742264243895e+38, ..., 6.
 ↪942133605708313e+41,
7.335747052314032e+42, 5.69783308475235e+42, 1.1128232022110846e+43
El metodo no converge
La diferencia entre soluciones es 1.451622338717728e+43
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 20
        Error: 3.6324709362165934e+55
        Sol: -7.323831572435901e+44, -1.3048420255161528e+47,
-2.3351113169951127e+49, -2.1383684641646873e+51, ..., -2.
 ↪8565664452140027e+54,
-3.0185314876376157e+55, -2.3445585643800182e+55, -4.
 ↪5790726589847025e+55
El metodo no converge
La diferencia entre soluciones es 5.97317179331505e+55
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 25
        Error: 1.4946981978392363e+68
        Sol: 3.0136284762679543e+57, 5.3691965008129655e+59,
9.608574269562953e+61, 8.799011873257374e+63, ..., 1.
 ↪1754270832835685e+67,
1.242072862774836e+68, 9.647448038654873e+67, 1.8842082349289435e+68
El metodo no converge
La diferencia entre soluciones es 2.4578556226939094e+68
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 35
        Error: 2.5307907379234587e+93
        Sol: 5.102610711852639e+82, 9.091007665622408e+84,
1.6269030631878133e+87, 1.4898297050140288e+89, ..., 1.
 ↪990207775575571e+92,
2.1030509714140467e+93, 1.6334850858936836e+93, 3.190300728381666e+93
El metodo no converge
La diferencia entre soluciones es 4.1615881079264364e+93
Matriz de "matrizA20" con el vector de "vecb20"
        Iteraciones: 50
        Error: 1.7632355142337908e+131
        Sol: -3.5550566420321894e+120, -6.3338257628323356e+122,
-1.1334849682523423e+125, -1.0379841393747967e+127, ...,
-1.3866041858041658e+130, -1.4652235388231731e+131, -1.
 ↪1380707508761582e+131,
-2.2227248824152343e+131
El metodo no converge
La diferencia entre soluciones es 2.8994336977579156e+131
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 0
```

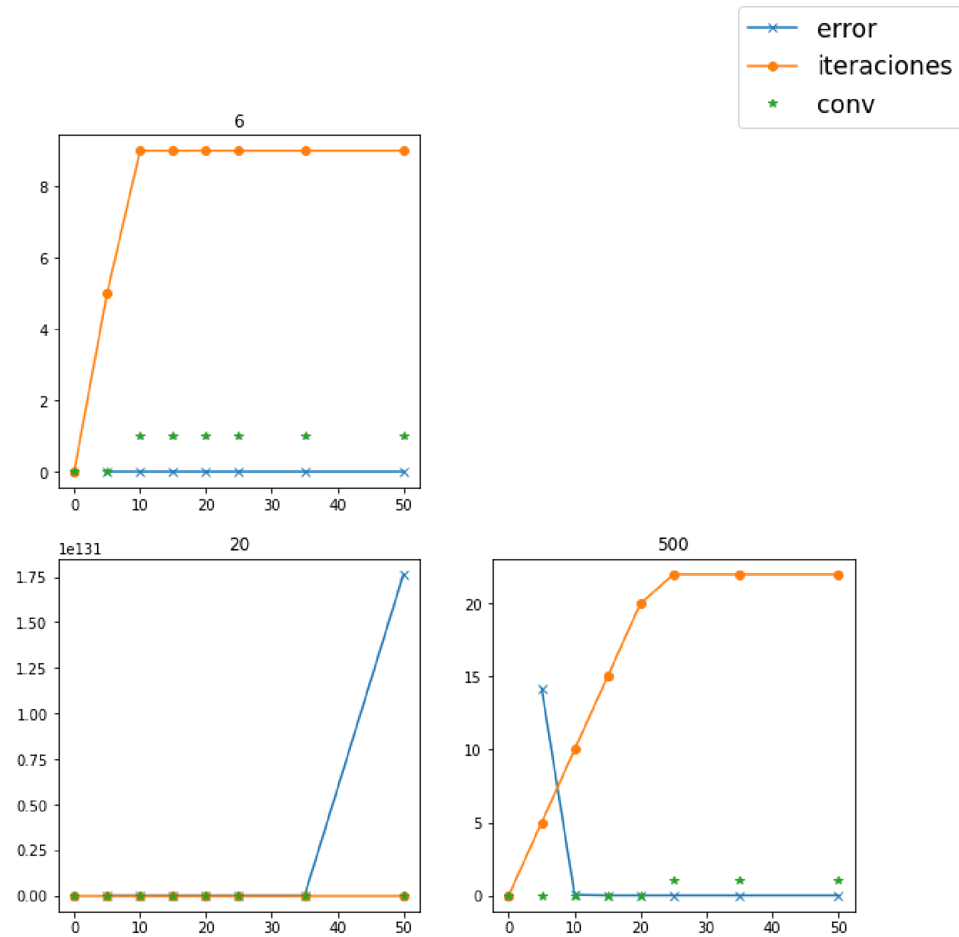
```
        Error: inf
        Sol: 0.0, 0.0, 0.0, 0.0, ..., 0.0, 0.0, 0.0, 0.0
El metodo no converge
La diferencia entre soluciones es 2236.06797749979
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 5
        Error: 14.176187474228898
        Sol: -99.63958099545093, 99.47440006331433, -99.
 ↪59286032243953,
99.70598909314587, ..., -99.91668866583471, 99.9348818313726,
-99.96581153196503, 99.99173629461029
El metodo no converge
La diferencia entre soluciones es 5.874263087048714
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 10
        Error: 0.038973335267423345
        Sol: -99.99917521831776, 99.99893028154601, -99.
 ↪99925001852736,
99.99948437479507, ..., -99.99998299574862, 99.99998753009527,
-99.99999350241926, 99.9999984294677
El metodo no converge
La diferencia entre soluciones es 0.015979939541367193
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 15
        Error: 0.00010199731641958948
        Sol: -99.99999855757989, 99.99999805927155, -99.
 ↪99999854223792,
99.99999893159874, ..., -99.99999999646232, 99.99999999750894,
-99.99999999870863, 99.99999999968786
El metodo no converge
La diferencia entre soluciones es 4.2453048360324766e-05
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 20
        Error: 2.970999512048245e-07
        Sol: -99.99999999713852, 99.99999999618126, -99.
 ↪99999999708542,
99.99999999772496, ..., -99.9999999999992, 99.99999999999949,
-99.99999999999974, 99.99999999999993
El metodo no converge
La diferencia entre soluciones es 1.2607776366756404e-07
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 22
        Error: 9.286484050814949e-09
        Sol: -99.99999999992775, 99.9999999998994, -99.99999999991924,
99.99999999993344, ..., -99.99999999999999, 100.0, -100.0, 100.0
El metodo converge
La diferencia entre soluciones es 3.9575573020619025e-09
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 22
        Error: 9.286484050814949e-09
        Sol: -99.99999999992775, 99.9999999998994, -99.99999999991924,
```

```
99.99999999993344, ..., -99.99999999999999, 100.0, -100.0, 100.0
El metodo converge
La diferencia entre soluciones es 3.9575573020619025e-09
Matriz de "matrizA500" con el vector de "vecb500"
        Iteraciones: 22
        Error: 9.286484050814949e-09
        Sol: -99.99999999992775, 99.9999999998994, -99.99999999991924,
99.99999999993344, ..., -99.99999999999999, 100.0, -100.0, 100.0
El metodo converge
La diferencia entre soluciones es 3.9575573020619025e-09
```

[249]:

```
PLOT = True
if PLOT:
    rng = itr
    fig, ax = plt.subplots(2, 2, figsize=(10,10))
    ax[0,1].axis('off')

    for sz in sizes:
        if sz == '6': i,j = 0,0
        elif sz == '20': i,j = 1,0
        elif sz == '500': i,j = 1,1

        ax[i,j].set_title(sz)
        a = ax[i,j].plot(rng, data[sz][0], '-x')
        b = ax[i,j].plot(rng, data[sz][1], '-o')
        c = ax[i,j].plot(rng, data[sz][2], '*')

    labels = ['error', 'iteraciones', 'conv']
    fig.legend([e, i, c],         # The line objects
           labels=labels,         # The labels for each line
           loc="upper right",     # Position of legend
           borderaxespad=0.1,     # Small spacing around legend box
           fontsize='xx-large'
           )

    plt.show()
```

```
<ipython-input-249-735f8276e7b2>:18: UserWarning: You have mixed␣
↪positional and
keyword arguments, some input may be discarded.
  fig.legend([e, i, c],         # The line objects
```



En la figura anterior podemos ver como convergen, o no, el metodo para los distintos datos proporcionados. Se grafica el error, de manera directa; las iteraciones con las que termina el metodo; y si converge o no, donde 0 es no y 1 es si. Todos estos datos se grafican contra el limite superior de iteraciones que se le pasa al metodo.

```
[ ]:
```