Numerical Algorithms - 2023/24

Assignment 2

Fabian Gobet

1. Gauss-Seidel Method

1.1 Matrix definition

We first start by defining the matrix A with which we'll work with. For this exercise lets consider the nxn matrix A with entries:

- 6 on the main diagonal, i.e., $a_{i,i} = 6,\ i = 1, \dots, n;$
- −2 on the super- and sub-diagonal, i.e., $a_{i,i+1} = a_{i+1,i} = -2,\ i = 1, \dots, n-1;$
- 1 on the antidiagonal, i.e., $a_{i,n+1-i} = 1,\ i = 1, \dots, (n-1)/2, (n+3)/2, \dots, n.$

We would then have for $n = 7$ the following matrix A

$$A = \begin{bmatrix} 6 & -2 & 0 & 0 & 0 & 0 & 1 \\ -2 & 6 & -2 & 0 & 0 & 1 & 0 \\ 0 & -2 & 6 & -2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 6 & -2 & 0 & 0 \\ 0 & 0 & 1 & -2 & 6 & -2 & 0 \\ 0 & 1 & 0 & 0 & -2 & 6 & -2 \\ 1 & 0 & 0 & 0 & 0 & -2 & 6 \end{bmatrix}$$

1.2 Equations

Let's consider the the linear system

$$Ax = b, \quad where\ x \neq \vec{0}$$

By unfolding the equation into a system of linear equations we end up with

$$\begin{cases} x_1 = \frac{1}{6}(b_1 + 2x_2 - x_n) \\ x_i = \frac{1}{6}(b_1 + 2(x_{i-1} + x_{i+1}) - x_{n-i+1}),\ i \in [n-1] \setminus \{0, \lceil \frac{n}{2} \rceil\} \\ x_{\lceil \frac{n}{2} \rceil} = \frac{1}{6}(b_{\lceil \frac{n}{2} \rceil} + 2(x_{\lceil \frac{n}{2} \rceil - 1} + x_{\lceil \frac{n}{2} \rceil + 1})) \\ x_n = \frac{1}{6}(b_n + 2x_{n-1} - x_1) \end{cases} \tag{1}$$

For the Gauss-Seidel methold to be well defined within the contraints of the system when attributing superscripts to x_i, we would have to further unfold the case for $i \in [n-1] \setminus \{0, \lceil \frac{n}{2} \rceil\}$ into two.

But since we'll use the same vector x to store all the new calculated values upon computations inside an iterarion, we may use the system (1) as our definition, meaning that for each computed equation i we'll be using the latest values computed into x.

2. Python

2.1 Defining functions and libraries

Functions get_x(n) and get_b(n) return, respectively, a vector x initialized with all zeros and a vector b with all 3's, except in the middle, having 2, and on the edges, having 5. For this exercise we're considering n as odd, so vector b will always have a middle point assigned with 2.

Note:

- The case for $n = 1$ is trivial, for which we'll consider the assignment for higher order n.

```
def get_b(n):
  b = []
  for i in range(n):
    if i==0 or i == n-1:
      b.append(5.0)
    elif i== n//2:
      b.append(2.0)
    else:
      b.append(3.0)
  return b

def get_x_0(n):
  x = []
  for i in range(n):
    x.append(0.0)
  return x

print("For n=7 we have")
print("b vector: ",get_b(7))
print("x vector: ",get_x_0(7))
```

```
For n=7 we have
b vector:  [5.0, 3.0, 3.0, 2.0, 3.0, 3.0, 5.0]
x vector:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Taking into account the considerations of (1) in chapter 1.2, we can define a function gauss_seidel_step that performs a full Gauss-Seidel iteration. In order to achieve this, we iterate through the n equations, by order, that result from (1) to take advantage of the already computed prior equations (principle of the Gauss-Seidel method).

All computed values are directly updated in the input x vector. We also pass as input the vector b, containing all i_{th} constants, and n as a reference for the dimensional load used.

```
def gauss_seidel_step(b, x, n):
  m = n//2

  x[0] = (1/6)*(b[0]+2*x[1]-x[n-1])

  for i in range(1,n-1,1):
    if i != m:
      x[i] = (1/6)*(b[i]+2*(x[i-1]+x[i+1])-x[n-1-i])
    else:
      x[i] = (1/6)*(b[i]+2*(x[i-1]+x[i+1]))

  x[n-1] = (1/6)*(b[n-1]+2*x[n-1]-x[0])

  return x
```

For this exercise it is required that we perform Gauss-Seidel iterations until we reach a treshold regarding the biggest element-wise absolute difference between x_k and x_{k+1} . For this, we define a function called is_distance_below, which admits as inputs the two vectors and returns true if such treshold is reached, false otherwise

```
def is_distance_below(x_k,x_k_1):
  dif = 0;
  for i in range(len(x_k)):
    t = abs(x_k[i]-x_k_1[i])
    if t > dif:
      dif = t
  if dif < 1e-16:
    return True
  return False
```

At last, we define a function iterate_gs which takes as input $b, x, n, break\_limit$, which counts the number of steps k needed to reach the target treshhold, whilst maintaining a safeguard number of steps, $break\_limit$, so it doesn't go on longer then expected. The function then returns the last computed x and the number of iterations k.

```
def iterate_gs(b, x, n, break_limit):
```

```
def iterate_gs(b,x,n,break_limit):
  k = 0
  while(True and k < break_limit):
    x_k = x.copy()
    x = gauss_seidel_step(b,x,n)
    k += 1
    if(is_distance_below(x_k,x)):
      break
  return k,x
```

2.2 Exercise computation

We are now in conditions of solving the proposed exercise by defining $n = 99999$, generating b and x, defining a break limit of 10000 iterations and executing the algorithm.

```
n = 99999
b = get_b(n)
x = get_x_0(n)
break_limit = 10000

k,x_k_1 = iterate_gs(b,x,n,break_limit)
print("After "+str(k)+" interations reached vector:")


# The following piece of code takes the first and last 5 elements of x_k_1 and builds a string representation.
result = "[ "
for i in range(5):
  result += str(x_k_1[i])+" "
result += " ... "
for i in range(-5,0,1):
  result += str(x_k_1[i])+" "
result += "]"
print(result)
```

```
After 101 interations reached vector:
[ 1.0 1.0 1.0 1.0 1.0  ... 1.0 1.0 1.0 1.0 1.0 ]
```

As seen above, after 101 iterations the algorithm converges to the solution $x = \vec{1}$. **(For the sake of showing the vector x_k_1, we only show the first and last 5 elements)**