Home assignment 2

Numerical Optimization and its Applications - Spring 2019
Gil Ben Shalom, 301908877
Tom Yaacov, 305578239

May 1, 2019

1 The efficiency of different iterative methods for solving a linear system

(a) Following are the implementation for the four methods:
Jacobi:

```
from numpy import diag, matmul
from numpy.linalg import inv, norm


def weighted_jacobi(A, b, x_0, maxIter, epsilon, w):
    D = diag(diag(A))
    L_U = A - D
    x = x_0
    for i in range(maxIter):
        x = x + w * matmul(inv(D), b - matmul(A, x))
        if norm(matmul(A, x) - b) / norm(b) < epsilon:
            break
    return x
```

Gauss-Seidel:

```python
from numpy import matmul, tril
from numpy.linalg import inv, norm


def weighted_gauss_seidel(A, b, x_0, maxIter, epsilon, w):
    L_D = tril(A, k=0)
    x = x_0
    for i in range(maxIter):
        x = x + w * matmul(inv(L_D), b - matmul(A, x))
        if norm(matmul(A, x) - b) / norm(b) < epsilon:
            break
    return x
```

```python
import numpy as np
A = np.array([[1, 2, 3, 4],
              [2, 4, -4, 8]])
print(A)
```

```
[[ 1 2 3 4] [ 2 4 -4 8]]
```

(b) t

```
import numpy as np
from scipy.sparse import spdiags
import matplotlib.pyplot as plt
from py_files.part_1_gauss_seidel import weighted_gauss_seidel
from py_files.part_1_jacobi import weighted_jacobi

n = 100
# TODO: not sure if .toarray() is the right approach
A = spdiags(np.array([-np.ones(n), 2.1 * np.ones(n), -np.ones(n)]),
            np.array([-1, 0, 1]), n, n).toarray()
x_0 = np.zeros(n)
b = np.random.rand(n)
maxIter = 100
epsilon = 1e-6

res = dict()
x, res['weighted_jacobi_1'] = weighted_jacobi(A, b, x_0, maxIter, epsilon, 1)
#print('weighted_jacobi_1 result:', x)
x, res['weighted_jacobi_0_75'] = weighted_jacobi(A, b, x_0, maxIter, epsilon, 0.75)
#print('weighted_jacobi_0_75 result:', x)
x, res['weighted_gauss_seidel_1'] = weighted_gauss_seidel(A, b, x_0, maxIter, epsilo
#print('weighted_gauss_seidel_1 result:', x)

convergence_factor = dict()
for alg_res in res:
    convergence_factor[alg_res] = res[alg_res][1:] / res[alg_res][:-1]

for alg_res in res:
    plt.semilogy(res[alg_res], label=alg_res)
plt.legend()
plt.xlabel("Iteration")
plt.ylabel("Residual Vector Norm")
plt.show()

# # Save the plot as a PDF file
# plt.savefig("myplot1.pdf", bbox_inches="tight")
# # Include the plot in the current LaTeX document
# print(r"\begin{center}")
# print(r"\includegraphics[width=0.85\textwidth]{myplot1.pdf}")
# print(r"\end{center}")

for alg_con in convergence_factor:
    plt.semilogy(convergence_factor[alg_con], label=alg_con)
plt.legend()
plt.xlabel("Iteration")
plt.ylabel("Convergence Factor")
plt.show()
```



```
# # Save the plot as a PDF file
# plt.savefig("myplot2.pdf", bbox_inches="tight")
# # Include the plot in the current LaTeX document
# print(r"\begin{center}")
# print(r"\includegraphics[width=0.85\textwidth]{myplot2.pdf}")
# print(r"\end{center}")
```

2 Convergence properties

(a) t

(b) t

(c) t

3 GMRES(1) method

(a) t

(b) t

(c) t

(d) t

(e) t

4 Convexity

(a) t

(b) t

(c) t

5 Non Linear Optimization

(a) t

(b) t

(c) t