Home assignment 2

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

May 12, 2019

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

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

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


def weighted_jacobi(A, b, x_0, maxIter, epsilon, w):
    D = diag(diag(A))
    x = x_0
    res = [norm(matmul(A, x) - b)]
    for i in range(maxIter):
        x = x + w * matmul(inv(D), b - matmul(A, x))
        res.append(norm(matmul(A, x) - b))
        if res[-1] / norm(b) < epsilon:
            break
    return x, array(res)
```

Gauss-Seidel:

```python
from numpy import matmul, tril, array
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
    res = [norm(matmul(A, x) - b)]
    for i in range(maxIter):
        x = x + w * matmul(inv(L_D), b - matmul(A, x))
        res.append(norm(matmul(A, x) - b))
        if res[-1] / norm(b) < epsilon:
            break
    return x, array(res)
```

Steepest Descent:

```python
from numpy import matmul, array, dot, copy
from numpy.linalg import norm
from py_files.utils import is_pos_def


def steepest_decent(A, b, x_0, max_iter, epsilon):
    if not is_pos_def(A):
        print("matrix is not SPD, can't solve using steepest decent...")
        return None, None
    x = copy(x_0)
    r = b - matmul(A, x)
    all_r = [norm(r)]
    for k in range(max_iter):
        alph = dot(r, r) / dot(r, matmul(A, r))
        x = x + alph * r
        # res.append(norm(matmul(A, x) - b))
        r = b - matmul(A, x)
        all_r.append(norm(r))
        if norm(r) / norm(b) < epsilon:
            break
    return x, array(all_r)
```

Conjugate Gradient

```python
from numpy import matmul, array, dot, copy
from numpy.linalg import norm
from py_files.utils import is_pos_def


def conjugate_gradient(A, b, x_0, max_iter, epsilon):
    if not is_pos_def(A):
        print("matrix is not SPD, can't solve using steepest decent...")
        return None, None
    x = copy(x_0)
    r = b - matmul(A, x)
    p = r
    all_r = [norm(r)]
    for k in range(max_iter):
        alph = dot(r, p) / dot(p, matmul(A, p))
        x = x + alph * p
        # res.append(norm(matmul(A, x) - b))
        r_prev = copy(r)
        r = b - matmul(A, x)
        all_r.append(norm(r))
        if norm(r) / norm(b) < epsilon:
            break
        beta = dot(r, r) / dot(r_prev, r_prev)
        p = r + beta * p
    return x, array(all_r)
```

(b) Following are the system and parameters definition, methods calls, residual vector norm and convergence factor plotting:

```python
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
from py_files.part_1_sd import steepest_decent
from py_files.part_1_cg import conjugate_gradient

n = 100
```

```python
# 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, epsilon, 1)
#print('weighted_gauss_seidel_1 result:', x)
x, res['steepest_decent'] = steepest_decent(A, b, x_0, maxIter, epsilon)
#print('steepest_decent result:', x)
x, res['conjugate_gradient'] = conjugate_gradient(A, b, x_0, maxIter, epsilon)
#print('conjugate_gradient result:', x)

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

plt.figure(figsize=(20, 25))
for alg_res in res:
    plt.semilogy(res[alg_res], label=alg_res)
plt.legend()
plt.xlabel("Iteration")
plt.ylabel("Residual Vector Norm")

plt.savefig("myplot1.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot1.pdf}")

plt.figure(figsize=(20, 25))
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.savefig("myplot2.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot2.pdf}")
```
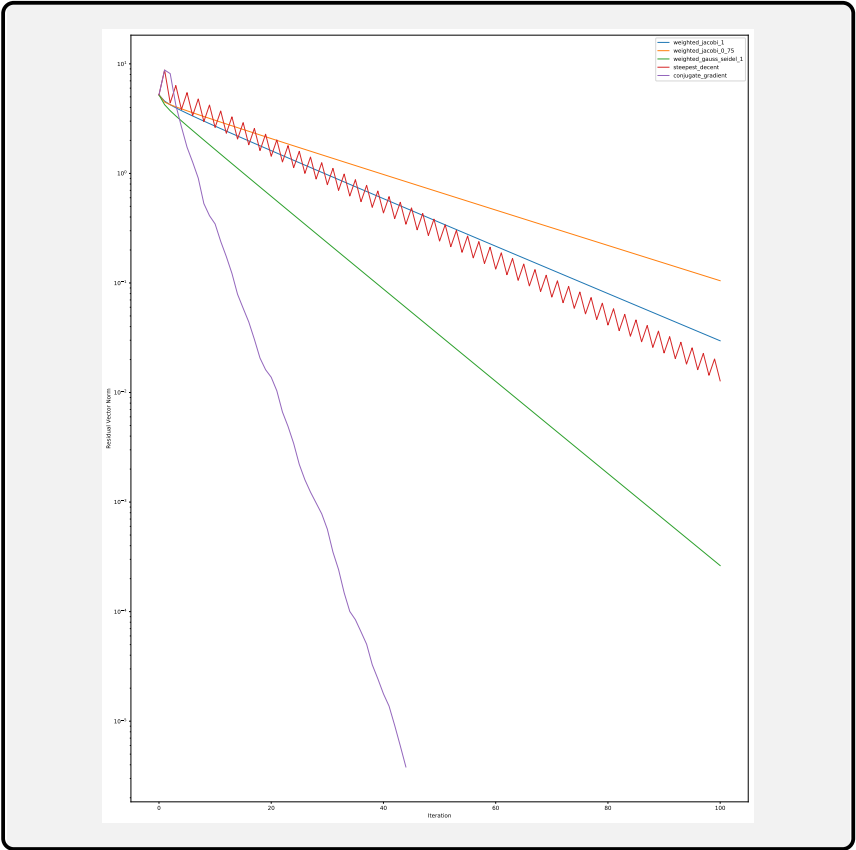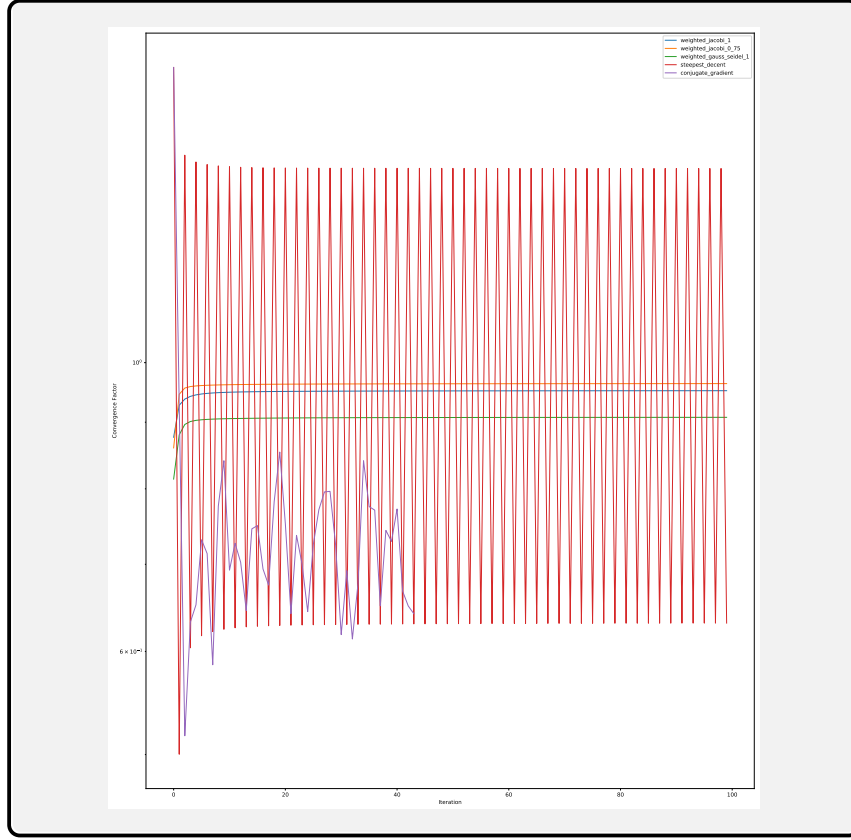
2 Convergence properties

(a)

Lemma 1.
$$0 < \alpha < \frac{2}{\lambda_{max}} \Rightarrow \rho(I - \alpha A) < 1$$

Proof. A is symmetric positive definite matrix, thus:

$$0 < \lambda_{min} \leq \cdots \leq \lambda_{max}$$

therefore, we get that:

$$\rho(I - \alpha A) = max(|1 - \alpha\lambda_{min}|, |1 - \alpha\lambda_{max}|)$$

$|1 - \alpha\lambda_{max}|$, then we get that:

$$-1 < 1 - \alpha\lambda_{max} < 1 \Rightarrow |1 - \alpha\lambda_{max}| < 1$$

5

And,
$$-1 < 1 - 2\frac{\lambda_{min}}{\lambda_{max}} < 1 - \alpha\lambda_{min} < 1 \Rightarrow |1 - \alpha\lambda_{min}| < 1$$

thus,
$$max(|1 - \alpha\lambda_{min}|, |1 - \alpha\lambda_{max}|) < 1$$

so we get that:
$$\rho(I - \alpha A) < 1$$

$\square$

In our case:
$$\alpha = \frac{1}{||A||}$$

we know that for any induced norm:
$$||A|| > \rho(A) = \lambda_{max}$$

thus,
$$\frac{1}{||A||} < \frac{1}{\lambda_{max}} < \frac{2}{\lambda_{max}}$$

therefore, by **Lemma 1** we get that
$$\rho(I - \alpha A) \leq 1$$

and the method converges.

(b) In the case A is indefinite, we a negative eigenvalue, therefore:
$$\rho(I - \alpha A) \geq |1 - \alpha\lambda_{min}|$$

$\lambda_{min} < 0$, by definition, therefore
$$\rho(I - \alpha A) \geq |1 - \alpha\lambda_{min}| > 1$$

(c) (i)
$$f(\mathbf{x}) = \frac{1}{2}||\mathbf{x}^* - \mathbf{x}||_A^2$$

$$\begin{aligned} f(\mathbf{x}^{(k)}) &= \frac{1}{2}||\mathbf{x}^* - \mathbf{x^{(k)}}||_A^2 \\ &= \frac{1}{2}||\mathbf{e}^{(k)}||_A^2 \\ &= \frac{1}{2}((\mathbf{e}^{(k)})^T A \mathbf{e}^{(k)}) \\ &= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)} \rangle \end{aligned}$$

$$f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) + \alpha \mathbf{r}^{(k)}$$

$$= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \alpha\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle + \frac{1}{2}\alpha^2\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle$$

$$= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} + \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle^2} \quad * \alpha = \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

$$= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} + \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

$$= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

$$= f(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

We get that:

$$f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

A is symmetric positive definite matrix, thus

$$\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} > 0$$

and therefore,

$$f(\mathbf{x}^{(k+1)}) = f(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} < f(\mathbf{x}^{(k)})$$

(ii) From previous section:

$$f(\mathbf{x}^{(k+1)}) = C^{(k)}f(\mathbf{x}^{(k)}) = f(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}$$

thus,

$$C^{(k)} = 1 - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle f(\mathbf{x}^{(k)})}$$

finally,

$$C^{(k)} = 1 - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle}$$

(iii) t

(iv) t

3 GMRES(1) method

(a)

$$||\mathbf{r}^{(k+1)}||_2 = ||\mathbf{b} - A\mathbf{x}^{(k+1)}||_2$$

we define the following scalar function $g(\alpha)$:

$$g(\alpha) \triangleq f(\mathbf{x}^{(k)}) + \alpha \mathbf{r}^{(k)}$$
$$= \frac{1}{2}||\mathbf{b} - A\mathbf{x}^{(k)} - \alpha A\mathbf{r}^{(k)}||_2$$
$$= \frac{1}{2}||\mathbf{r}^{(k)} - \alpha A\mathbf{r}^{(k)}||_2$$
$$= \frac{1}{2}(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)} - \alpha (\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)} + \frac{1}{2}\alpha^2 (A\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)}$$

And the minimization of g with respect to α is done by:

$$g'(\alpha) = -(\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)} + \alpha(A\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)} = 0$$

$$\Rightarrow \alpha_{opt} = \frac{(\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)}}{(A\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)}} = \frac{(\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T A^T A\mathbf{r}^{(k)}}$$

(b) (non-mandatory)

(c) a:

```python
from numpy import matmul, array, dot, copy, transpose, vectorize
from numpy.linalg import norm
from py_files.utils import is_pos_def
import matplotlib.pyplot as plt


def steepest_decent(A, b, x_0, max_iter, epsilon):
    if not is_pos_def(A):
        print("matrix is not SPD, can't solve using steepest decent...")
        return None, None
    x = copy(x_0)
    r = b - matmul(A, x)
    all_r = [norm(r)]
    for k in range(max_iter):
        alph = dot(r, matmul(A, r)) / matmul(r, matmul(transpose(A), matmul(A, r)))
        x = x + alph * r
        r = b - matmul(A, x)
        all_r.append(norm(r))
        if norm(r) / norm(b) < epsilon:
            break
    return x, array(all_r)


A = array([
    [5, 4, 4, -1, 0],
    [3, 12, 4, -5, -5],
    [-4, 2, 6, 0, 3],
    [4, 5, -7, 10, 2],
    [1, 2, 5, 3, 10]
])
b = array([1, 1, 1, 1, 1])
```
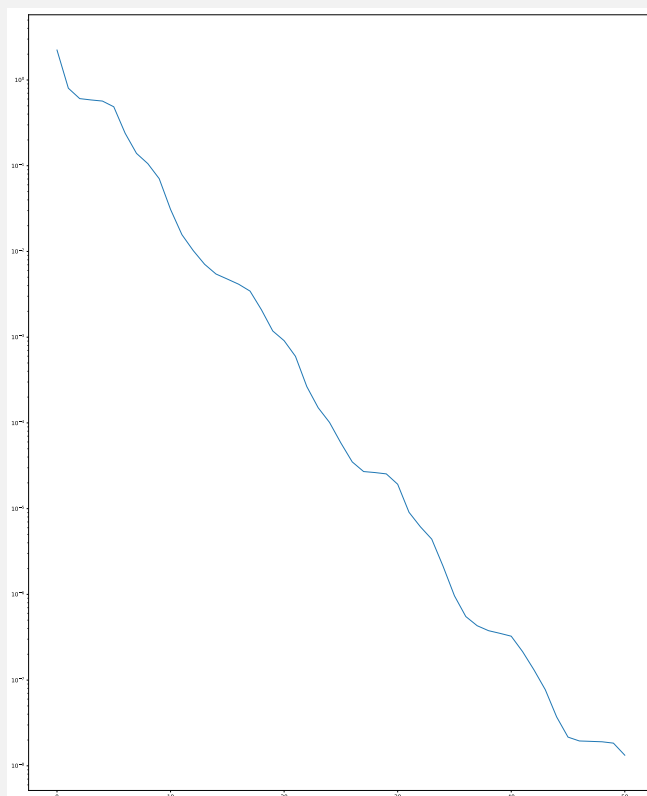
```
x_0 = array([0, 0, 0, 0, 0])
x, all_r = steepest_decent(A, b, x_0, 50, 0.00000000001)
plt.figure(figsize=(20, 25))
plt.semilogy(all_r)
plt.savefig("myplot3.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot3.pdf}")
```


(d) t

(e) t

4 Convexity

(a)   i. e^{ax} is convex:
$$(e^{ax})'' = a^2 e^{ax} \geq 0 \quad \forall x$$

    ii. $-log(x)$ is convex:

$$(-log(x))'' = \frac{1}{x^2} > 0 \quad \forall x > 0$$

iii. $log(x)$ is concave:

$$(log(x))'' = -\frac{1}{x^2} < 0 \quad \forall x > 0$$

iv. $|x|^a, \quad a \geq 1$ is convex:

$$
\begin{aligned}
f(\alpha x + (1-\alpha)y) &= |\alpha x + (1-\alpha)y|^a \\
&\leq (|\alpha x| + |(1-\alpha)y|)^a \quad \text{*triangle inequality} \\
&= (\alpha|x| + (1-\alpha)|y|)^a \\
&\leq \alpha|x|^a + (1-\alpha)|y|^a \quad \ast\, \alpha \leq 1 \\
&= \alpha f(x) + (1-\alpha)f(y)
\end{aligned}
$$

v. x^3 is none of those:

Not convex:

for $x = -1, y = 0, \alpha = 0.5$:

$$f(\alpha x + (1-\alpha)y) = f(0.5(-1) + (1-0.5)0) = f(-0.5) = (-0.5)^3 = -0.125$$

$$\alpha f(x) + (1-\alpha)f(y) = 0.5f(-1) + (1-0.5)f(0) = 0.5(-1)^3 = -0.5$$

we get that

$$f(\alpha x + (1-\alpha)y) > \alpha f(x) + (1-\alpha)f(y)$$

Not concave:

for $x = 0, y = 1, \alpha = 0.5$:

$$f(\alpha x + (1-\alpha)y) = f(0.5(0) + (1-0.5)1) = f(0.5) = (0.5)^3 = 0.125$$

$$\alpha f(x) + (1-\alpha)f(y) = 0.5f(0) + (1-0.5)f(1) = 0.5(1)^3 = 0.5$$

we get that

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y)$$

(b) Let

$$f(\mathbf{x}) = \mathbf{x}^T A \mathbf{x} + \mathbf{b}^T \mathbf{x} + \mathbf{c}$$

computing the Hessian of $f(\mathbf{x})$

$$\nabla f(\mathbf{x}) = 2A\mathbf{x} + \mathbf{b}$$

$$\nabla^2 f(\mathbf{x}) = J(\nabla f(\mathbf{x})) = 2A$$

$f(\mathbf{x})$ is a convex function over a convex region Ω if and only if $2A \succeq 0$, is positive semi definite.

(c) Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable in a convex domain Ω. We'll show the following:

$$f \quad \text{is convex} \Leftrightarrow f(y) > f(x) + \nabla f(x)^T(y - x) \quad , \forall x, y \in \Omega$$

$\Rightarrow$:
f is convex. Then, according to the fundamental definition of convex functions, the following inequality condition must be satisfied:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \qquad \forall x, y \in \Omega \land \alpha \in [0, 1]$$
$$\Rightarrow f(x + \alpha(y - x)) \leq f(x) + \alpha(f(y) - f(x))$$
$$\Rightarrow f(x + \alpha(y - x)) - f(x) \leq \alpha(f(y) - f(x))$$
$$\Rightarrow \frac{f(x + \alpha(y - x)) - f(x)}{\alpha} \leq f(y) - f(x)$$
$$\Rightarrow f(y) \geq f(x) + \frac{f(x + \alpha(y - x)) - f(x)}{\alpha}$$

Now, let
$$g(\alpha) = f(x + \alpha(y - x))$$

therefore
$$f(y) \geq f(x) + \frac{g(\alpha) - g(0)}{\alpha} \quad * g(0) = f(x)$$

Now taking the limit as $\alpha \to 0$ we get

$$f(y) \geq f(x) + \lim_{\alpha \to 0} \frac{g(\alpha) - g(0)}{\alpha}$$
$$\Rightarrow f(y) \geq f(x) + g'(0)$$

In order to find $g'(0)$, we'll compute the more general $g'(t)$:

$$g'(t) = \nabla_x f(x + t(y - x))^T(y - x)$$

assigning $t = 0$, we get

$$g'(0) = \nabla_x f(x)^T(y - x)$$

finally, by substituting $g'(0)$ we get

$$f(y) \geq f(x) + \nabla_x f(x)^T(y - x)$$

therefore

$$f \quad \text{is convex} \Rightarrow f(y) > f(x) + \nabla f(x)^T(y - x) \quad , \forall x, y \in \Omega$$

$\Leftarrow$:
Let
$$f(y) > f(x) + \nabla_x f(x)^T(y - x) \quad , \forall x, y \in \Omega$$

11

Now, consider

$$z = \alpha x + (1 - \alpha)y \quad \forall \alpha \in [0, 1]$$

Notice that, since Ω is a convex domain, $z \in \Omega$. Therefore:

$$f(x) > f(z) + \nabla_z f(z)^T (x - z) \tag{1}$$

$$f(y) > f(z) + \nabla_z f(z)^T (y - z) \tag{2}$$

Now multiplying the inequalities in Equations (1) and (2) with α and $(1 - \alpha)$ respectively and adding the results we get:

$$\alpha f(x) + (1 - \alpha)f(y) > f(z) + \nabla_z f(z)^T (\alpha x + (1 - \alpha)y - z)$$

by substituting $z = \alpha x + (1 - \alpha)y$, we get

$$\alpha f(x) + (1 - \alpha)f(y) > f(\alpha x + (1 - \alpha)y) + \nabla_z f(z)^T (\alpha x + (1 - \alpha)y - \alpha x + (1 - \alpha)y)$$
$$\Rightarrow \alpha f(x) + (1 - \alpha)f(y) > f(\alpha x + (1 - \alpha)y)$$

Observe that this is exactly the inequality that $f(x)$ must satisfy in order to be considered as a convex function, hence

$$f \quad \text{is convex} \Leftarrow f(y) > f(x) + \nabla f(x)^T (y - x) \quad , \forall x, y \in \Omega$$

and combining the 2 sides, we showed that

$$f \quad \text{is convex} \Leftrightarrow f(y) > f(x) + \nabla f(x)^T (y - x) \quad , \forall x, y \in \Omega$$

5 Non Linear Optimization

(a) Following is the implementation for the function that, given data matrix X and labels, computes the logistic regression objective, its gradient, and its Hessian matrix:

```python
from numpy import matmul, exp, log, asarray, outer


def sigmoid(x):
    return 1 / (1 + exp(-x))


def reg_obj_grad_hes(X, y, w):
    m = len(y)
    c_1 = y
    c_2 = 1-y
    sig = sigmoid(matmul(X.T,w))
    obj = -(1/m)*(matmul(c_1.T,log(sig))+matmul(c_2.T,log(1-sig)))
    grad = (1/m)*matmul(X, sig-c_1)
    # TODO: maybe outer(sig,(1-sig)) should be replaced to diag(sig*(1-sig))
    hes = (1/m)*matmul(matmul(X, outer(sig,(1-sig))), X.T)
    return obj, grad, hes
```

12

(b) Implementation of Gradient and Jacobian verification:

```python
from numpy.random import rand
from numpy import matmul
from numpy.linalg import norm
from py_files.part_5_a import reg_obj_grad_hes


def gradient_test_reg_obj_grad_hes(X, y, w, epsilon, iterations):
    d = rand(w.shape[0])
    d = d/d.sum()
    res1 = []
    res2 = []
    obj_0, grad_0, _ = reg_obj_grad_hes(X, y, w)
    for _ in range(iterations):
        obj, grad, _ = reg_obj_grad_hes(X, y, w+(epsilon*d))
        res1.append(abs(obj-obj_0))
        res2.append(abs(obj-obj_0-epsilon*matmul(d.T,grad_0)))
        epsilon *= 0.5
    return res1, res2


def jacobian_test_reg_obj_grad_hes(X, y, w, epsilon, iterations):
    d = rand(w.shape[0])
    d = d/d.sum()
    res1 = []
    res2 = []
    obj_0, grad_0, hes_0 = reg_obj_grad_hes(X, y, w)
    for _ in range(iterations):
        obj, grad, _ = reg_obj_grad_hes(X, y, w+(epsilon*d))
        res1.append(norm(grad-grad_0))
        res2.append(norm(grad-grad_0-matmul(hes_0, epsilon*d.T)))
        epsilon *= 0.5
    return res1, res2
```

Testing our gradient and hessiam implementation"

```python
from numpy import asarray
from numpy.random import rand
from mnist import MNIST
import matplotlib.pyplot as plt
from py_files.part_5_b import gradient_test_reg_obj_grad_hes
from py_files.part_5_b import jacobian_test_reg_obj_grad_hes

try:
    mndata = MNIST('py_files/data')
    mndata.gz = True
    images, labels = mndata.load_training()
except FileNotFoundError:
    mndata = MNIST('data')
    mndata.gz = True
    images, labels = mndata.load_training()

images = asarray(images)
labels = asarray(labels)
X = images[labels <= 1][:100].T
X = X / X.max()
y = labels[labels <= 1][:100].T
w = rand(X.shape[0]) * 2 - 1

res1, res2 = gradient_test_reg_obj_grad_hes(X, y, w, 1e-4, 15)

plt.figure(figsize=(20, 25))
plt.plot(res1, label="Linear")
plt.plot(res2, label="Quadratic")
plt.legend()
plt.title("Gradient Test")
plt.xlabel("Iteration")

plt.savefig("myplot4.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot4.pdf}")

res1, res2 = jacobian_test_reg_obj_grad_hes(X, y, w, 1e-4, 15)

plt.figure(figsize=(20, 25))
plt.plot(res1, label="Linear")
plt.plot(res2, label="Quadratic")
plt.legend()
plt.title("Jacobian Test")
plt.xlabel("Iteration")

plt.savefig("myplot5.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot5.pdf}")
```
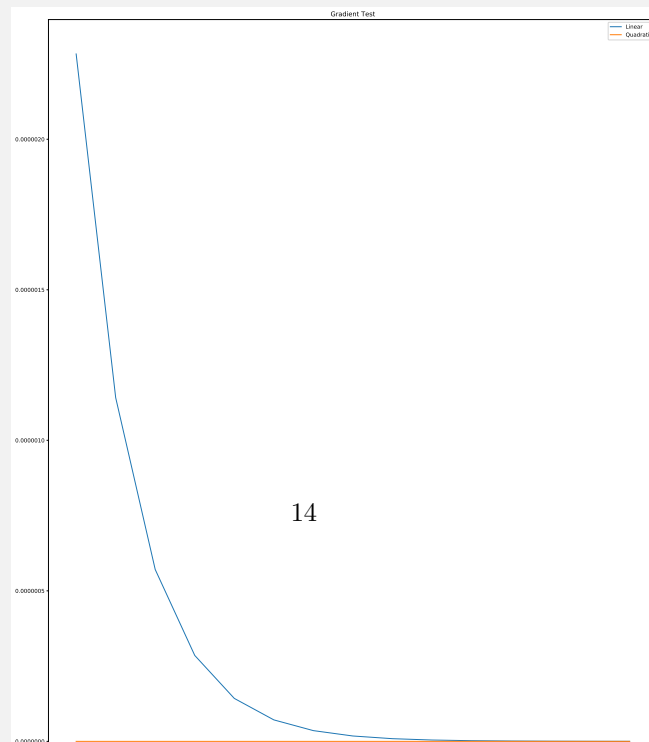

14

(c) t