Home assignment 2

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

May 20, 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, 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:

```
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)
```

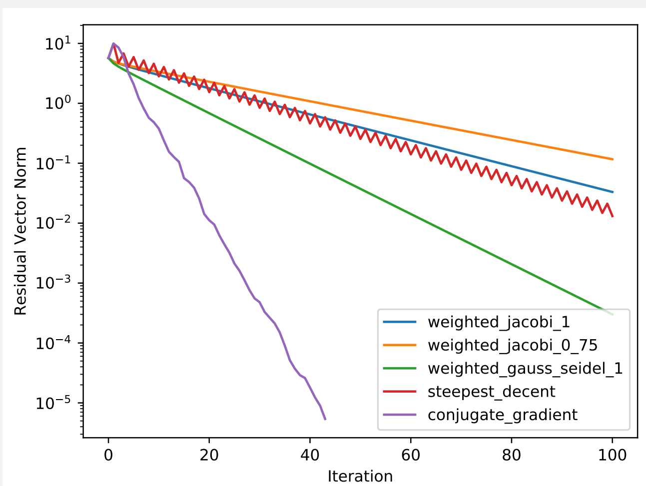
Conjugate Gradient

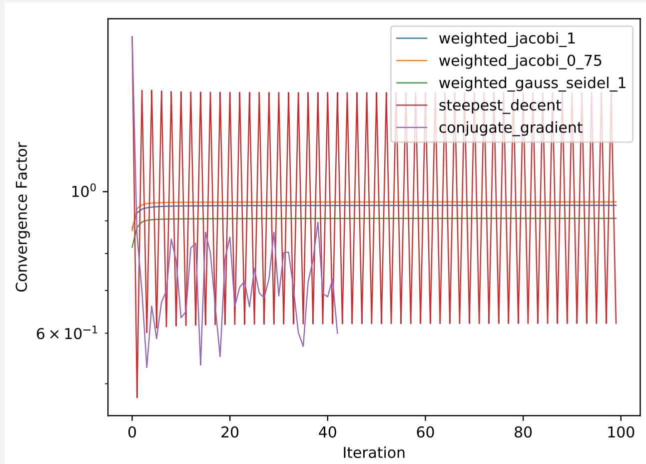
```python
from numpy import matmul, array, dot, copy
from numpy.linalg import norm
from py_files.utils import is_pos_def


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
```

```
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)
x, res['weighted_jacobi_0_75'] = weighted_jacobi(A, b, x_0, maxIter,epsilon,0.75)
x,res['weighted_gauss_seidel_1']=weighted_gauss_seidel(A,b,x_0,maxIter,epsilon,1)
x, res['steepest_decent'] = steepest_decent(A, b, x_0, maxIter, epsilon)
x, res['conjugate_gradient'] = conjugate_gradient(A, b, x_0, maxIter, epsilon)

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

plt.figure()
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()
for alg_con in convergence_factor:
    plt.semilogy(convergence_factor[alg_con], label=alg_con, linewidth=0.8)
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:

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

We will show the values of $|1 - \alpha\lambda_{min}|$ and $|1 - \alpha\lambda_{max}|$:

$$\alpha\lambda_{max} < 2 \Rightarrow 1 - \alpha\lambda_{max} > -1$$

$$\alpha\lambda_{max} > 0 \Rightarrow 1 - \alpha\lambda_{max} < 1$$

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

And,

$$2\frac{\lambda_{min}}{\lambda_{max}} \leq 2 \Rightarrow -1 \leq 1 - 2\frac{\lambda_{min}}{\lambda_{max}}$$

$$\alpha < \frac{2}{\lambda_{max}} \Rightarrow \alpha\lambda_{min} < \frac{2\lambda_{min}}{\lambda_{max}} \Rightarrow 1 - \alpha\lambda_{min} > \frac{2\lambda_{min}}{\lambda_{max}}$$

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

$$\alpha\lambda_{min} > 0 \Rightarrow 1 - \alpha\lambda_{min} < 1$$

Thus,

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

So we get that:

$$\rho(I - \alpha A) < 1$$

□

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

$$\rho(I - \alpha A) < 1$$

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)

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

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

$$
\begin{aligned}
f(\mathbf{x}^{(k+1)}) &= f(\mathbf{x}^{(k)}) + \alpha\mathbf{r}^{(k)} \\
&= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \alpha\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle + \frac{1}{2}\alpha^2\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle \\
&= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} + \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle^2} \qquad * \, \alpha = \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} \\
&= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} + \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} \\
&= \frac{1}{2}\langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)}\rangle - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle} \\
&= f(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{\langle \mathbf{r}^{(k)}, A\mathbf{e}^{(k)}\rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)}\rangle}
\end{aligned}
$$

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,

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

(iii) First, we will proof that $C^{(k)} \leq 1 - \frac{\lambda_{min}}{\lambda_{max}}$

$$1 - \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle^2}{\langle \mathbf{r}^{(k)}, A\mathbf{r}^{(k)} \rangle \langle \mathbf{e}^{(k)}, A\mathbf{e}^{(k)} \rangle} \leq 1 - \frac{\lambda_{min}}{\lambda_{max}}$$

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

$$= \lambda_{min} \cdot \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{r}^{(k)}, A^{-1}\mathbf{r}^{(k)} \rangle}$$

For each λ_i in A we get that the corresponding eigenvalue in A^{-1} is $\frac{1}{\lambda_i}$. Thus we get that the minimum eigenvalue for $A^{-1} = \frac{1}{\lambda_{max}}$ where λ_{max} is with respect to A. Therefore:

$$= \lambda_{min} \cdot \frac{\langle \mathbf{r}^{(k)}, \mathbf{r}^{(k)} \rangle}{\langle \mathbf{r}^{(k)}, A^{-1}\mathbf{r}^{(k)} \rangle} \leq \frac{\lambda_{min}}{\lambda_{max}}$$

Now, we will show that $1 - \frac{\lambda_{min}}{\lambda_{max}} < 1$
A is symmetric positive definite, Thus all eigenvalues are larger than 0 and clearly $0 < \frac{\lambda_{min}}{\lambda_{max}} \leq 1$

$$-1 < \frac{\lambda_{min}}{\lambda_{max}} - 1$$

$$1 - \frac{\lambda_{min}}{\lambda_{max}} < 1$$

(iv)

$$f(x^k) = f(x^0) \prod_{i=1}^{k} C^{(i)}$$

Since $C^{(k)} < 1$ we get that $\lim_{k\to\infty} f(\mathbf{x}^{(k)}) = 0$

$$f(x^{(k)}) = |b - A\mathbf{x}^{(k)}| = |A\mathbf{x}^* - A\mathbf{x}^{(k)}| = |A(\mathbf{x}^* - \mathbf{x}^{(k)})|$$

Since A is fully ranked $|A(\mathbf{x}^* - \mathbf{x}^{(k)})| = 0$ only if $\mathbf{x}^{(k)} = \mathbf{x}^*$, hence $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$

3 GMRES(1) method

(a)

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

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

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

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

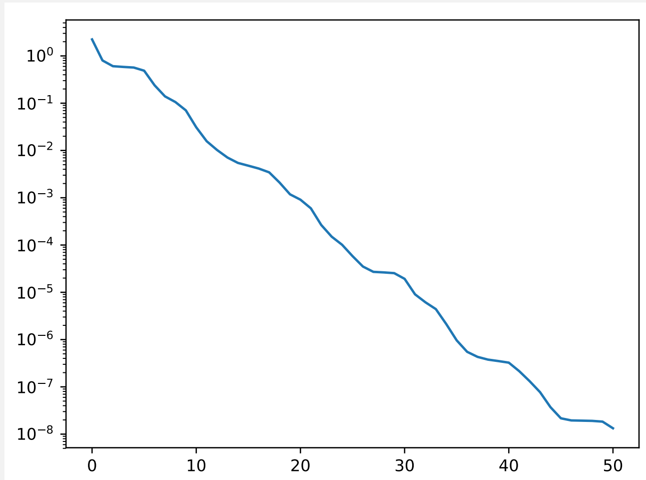
$$g'(\alpha) = -(\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)} + \alpha(A\mathbf{r}^{(k)})^T A\mathbf{r}^{(k)} = 0$$

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

(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()
plt.semilogy(all_r)
plt.savefig("myplot3.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot3.pdf}")
```



(d) At each step we choose $\alpha^{(k)}$ that minimizes the expression $||r^{(k+1)}||_2$. At the "worst case", when we couldn't decrease the norm of the residuals, $\alpha^{(k)}$ can be chosen to be 0, otherwise $\alpha^{(k)}$ will be chosen such that $||r^{(k+1)}||_2 < ||r^{(k+1)}||$, thus the graph is monotone.

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

$$\begin{aligned}
g(\alpha) &\triangleq f(\mathbf{x}^{(k)} + R^{(k)}\vec{\alpha}^{(k)}) \\
&= \frac{1}{2}||\mathbf{b} - A(\mathbf{x}^{(k)} + R^{(k)}\vec{\alpha}^{(k)})||_2 \\
&= \frac{1}{2}||\mathbf{b} - A\mathbf{x}^{(k)} - AR^{(k)}\vec{\alpha}^{(k)}||_2 \\
&= \frac{1}{2}||\mathbf{r}^{(k)} - AR^{(k)}\vec{\alpha}^{(k)}||_2 \\
&= \frac{1}{2}(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T AR^{(k)}\vec{\alpha}^{(k)} + \frac{1}{2}(AR^{(k)}\vec{\alpha}^{(k)})^T(AR^{(k)}\vec{\alpha}^{(k)}) \\
&= \frac{1}{2}(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T AR^{(k)}\vec{\alpha}^{(k)} + \frac{1}{2}(\vec{\alpha}^{(k)})^T(AR^{(k)})^T(AR^{(k)}\vec{\alpha}^{(k)}) \\
&= \frac{1}{2}(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)} - (\mathbf{r}^{(k)})^T AR^{(k)}\vec{\alpha}^{(k)} + \frac{1}{2}(\vec{\alpha}^{(k)})^T(R^{(k)})^T A^T AR^{(k)}\vec{\alpha}^{(k)}
\end{aligned}$$

Now, we derive the function by $\vec{\alpha}^{(k)}$ in order to find $\vec{\alpha}^{(k)}$ that minimize the expression. we get that

$$(\mathbf{r}^{(k)})^T AR^{(k)} = \frac{1}{2}(\vec{\alpha}^{(k)})^T(((R^{(k)})^T A^T AR^{(k)})^T + (R^{(k)})^T A^T AR^{(k)})$$

$$(\vec{\alpha}^{(k)})^T = 2(\mathbf{r}^{(k)})^T AR^{(k)}(((R^{(k)})^T A^T AR^{(k)})^T + (R^{(k)})^T A^T AR^{(k)})^{-1}$$

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:

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

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:

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

Now, let

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

therefore

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

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

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

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

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

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, diag, 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)
    hes = (1/m)*matmul(matmul(X, diag(sig*(1-sig))), X.T)
    return obj, grad, hes
```

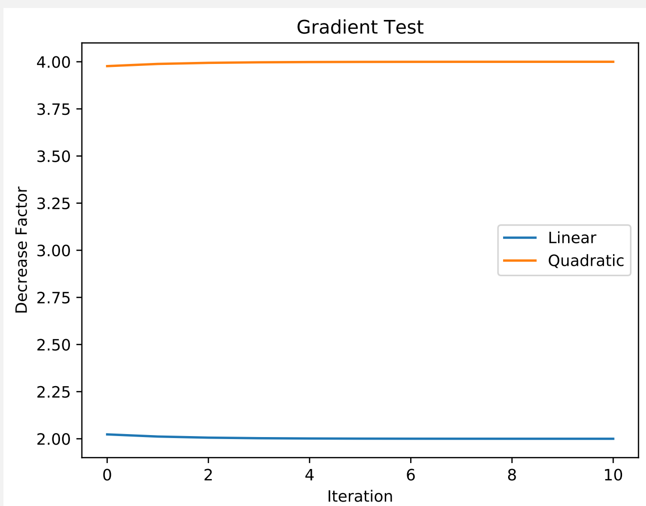
(b) Implementation of Gradient and Jacobian verification:

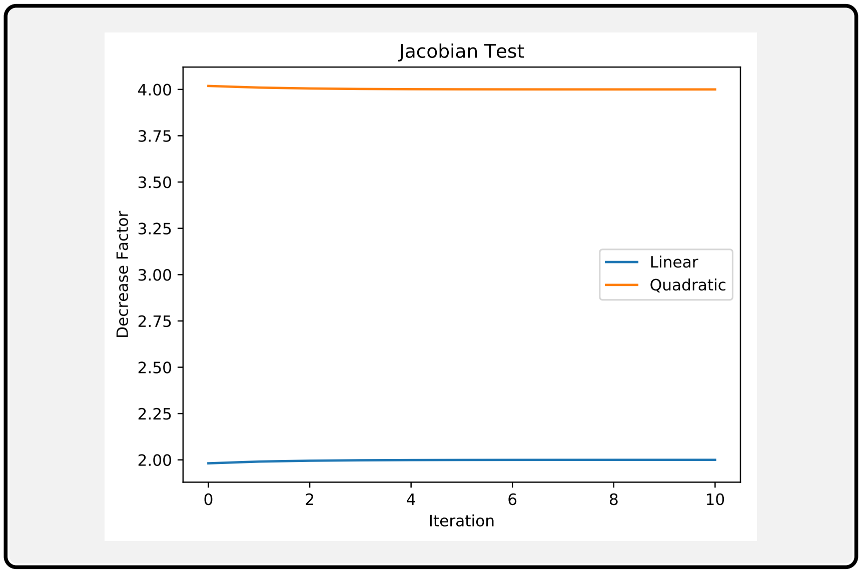
```python
from numpy.random import rand
from numpy import matmul
from numpy.linalg import norm
from py_files.part_5_a import reg_obj_grad_hes


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 array
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

# creating test example
X = array([[1, 0],
        [0, 1],
```

```
        [1, 1]])
y = array([0, 1])
w = array([1, 1, 0])

# running gradient test
res1, res2 = gradient_test_reg_obj_grad_hes(X, y, w, 1e-1, 12)

# plotting the decrease factor
plt.figure()
plt.plot([x[0]/x[1] for x in zip(array(res1)[:-1],array(res1)[1:])], label="Linear")
plt.plot([x[0]/x[1] for x in zip(array(res2)[:-1],array(res2)[1:])], label="Quadratic")
plt.legend()
plt.title("Gradient Test")
plt.ylabel("Decrease Factor")
plt.xlabel("Iteration")

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

# running jacobian test
res1, res2 = jacobian_test_reg_obj_grad_hes(X, y, w, 1e-1, 12)

# plotting the decrease factor
plt.figure()
plt.plot([x[0]/x[1] for x in zip(array(res1)[:-1],array(res1)[1:])], label="Linear")
plt.plot([x[0]/x[1] for x in zip(array(res2)[:-1],array(res2)[1:])], label="Quadratic")
plt.legend()
plt.title("Jacobian Test")
plt.ylabel("Decrease Factor")
plt.xlabel("Iteration")

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





(c) Implementation of Gradient Descent, and exact Newton methods:

```
from numpy import matmul, copy, mean, round, eye
from numpy.linalg import inv
from py_files.part_5_a import reg_obj_grad_hes, sigmoid
from py_files.part_5_a_helper import reg_obj


def steepest_decent(X, y, w_0, max_iter, alph_0, beta=0.5, c=1e-4):
    w = copy(w_0)
    w_l = [w]
    for _ in range(max_iter):
        obj, grad, _ = reg_obj_grad_hes(X, y, w)
        alph = armijo_line_search(X, y, w, obj, grad, -grad, alph_0, beta, c)
        if alph is None:
            break
        w = w - alph * grad
        w_l.append(w)
    return w_l


def newton(X, y, w_0, max_iter, epsilon):
    w = copy(w_0)
    f = []
    w_l = [w]
    for _ in range(max_iter):
        obj, grad, hes = reg_obj_grad_hes(X, y, w)
        w = w - matmul(inv(hes+eye(hes.shape[0])), grad)
        w_l.append(w)
        f.append(obj)
        if len(f) > 1 and f[-2] - obj < epsilon:
            break
    return w_l


def armijo_line_search(X, y, w, obj, grad, d, alpha, beta=0.5, c=1e-4, max_iter=100):
    for _ in range(max_iter):
        n_obj = reg_obj(X, y, w + alpha * d)
        if n_obj <= obj + c * alpha * matmul(grad, d):
            return alpha
        else:
            alpha = beta * alpha
    return None


def accuracy(X, y, w):
    return mean(y == round(sigmoid(matmul(X.T, w))))
```

Running over the MNIST data sets for the digits 0,1 and 8,9 and shwing the results:

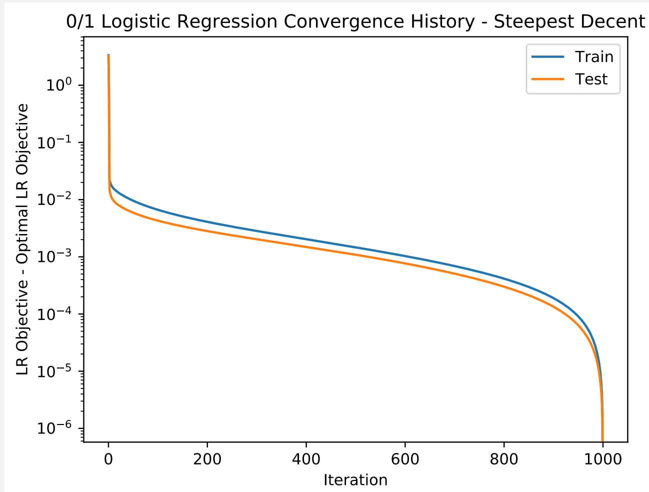
```
from numpy import asarray
from numpy.random import rand
from mnist import MNIST
import matplotlib.pyplot as plt
from py_files.part_5_c import steepest_decent, newton, accuracy
from py_files.part_5_a_helper import reg_obj

# loading data
mndata = MNIST('py_files/data')
mndata.gz = True
images_train, labels_train = mndata.load_training()
images_test, labels_test = mndata.load_testing()

# parameters definition
max_iter = 1000
```

```python
epsilon = 1e-4
alpha = 1

# 0/1

# train data pre processing
images_train = asarray(images_train)
labels_train = asarray(labels_train)
X_train = images_train[labels_train <= 1].T
X_train = X_train / X_train.max()
y_train = labels_train[labels_train <= 1].T

# test data pre processing
images_test = asarray(images_test)
labels_test = asarray(labels_test)
X_test = images_test[labels_test <= 1].T
X_test = X_test / X_test.max()
y_test = labels_test[labels_test <= 1].T

# initializing weights
w_0 = rand(X_train.shape[0]) * 2 - 1

# running sd an newton
w_sd = steepest_decent(X_train, y_train, w_0, max_iter, alpha)
w_n = newton(X_train, y_train, w_0, max_iter, epsilon)

# computing sd accuracy
print(r"0,1:\\")
print(r"Steepest Decent Accuracy:\\")
print(r"Train:", accuracy(X_train, y_train, w_sd[-1]), r"\\")
print(r"Test:", accuracy(X_test, y_test, w_sd[-1]), r"\\")

# computing convergence history - Steepest Decent
ch_sd_train = []
ch_sd_test = []
for w in w_sd:
    obj = reg_obj(X_train, y_train, w)
    ch_sd_train.append(obj)
    obj = reg_obj(X_test, y_test, w)
    ch_sd_test.append(obj)

# plotting convergence history - Steepest Decent
plt.figure()
plt.semilogy([abs(x-ch_sd_train[-1]) for x in ch_sd_train], label="Train")
plt.semilogy([abs(x-ch_sd_test[-1]) for x in ch_sd_test], label="Test")
plt.legend()
plt.title("0/1 Logistic Regression Convergence History - Steepest Decent")
plt.ylabel("LR Objective - Optimal LR Objective")
plt.xlabel("Iteration")

plt.savefig("myplot6.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot6.pdf}")

# computing newton accuracy
print(r"Newton Accuracy:\\")
print(r"Train:", accuracy(X_train, y_train, w_n[-1]), r"\\")
print(r"Test:", accuracy(X_test, y_test, w_n[-1]), r"\\")

# computing convergence history - newton
ch_n_train = []
ch_n_test = []
for w in w_n:
    obj = reg_obj(X_train, y_train, w)
    ch_n_train.append(obj)
    obj = reg_obj(X_test, y_test, w)
    ch_n_test.append(obj)
```

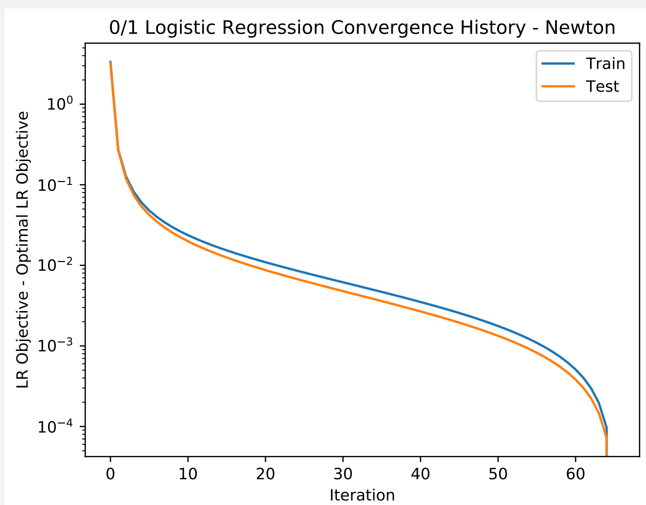
```python
# plotting convergence history - Steepest Decent
plt.figure()
plt.semilogy([abs(x-ch_n_train[-1]) for x in ch_n_train], label="Train")
plt.semilogy([abs(x-ch_n_test[-1]) for x in ch_n_test], label="Test")
plt.legend()
plt.title("0/1 Logistic Regression Convergence History - Newton")
plt.ylabel("LR Objective - Optimal LR Objective")
plt.xlabel("Iteration")

plt.savefig("myplot7.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot7.pdf}")

# plotting convergence train
plt.figure()
plt.plot(ch_sd_train, label="Steepest Decent")
plt.plot(ch_n_train, label="Newton")
plt.legend()
plt.title("0/1 Logistic Regression Train Set Convergence")
plt.ylabel("Logistic Regression Objective")
plt.xlabel("Iteration")

plt.savefig("myplot8.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot8.pdf}")

# 8/9

# train data pre processing
images_train = asarray(images_train)
labels_train = asarray(labels_train)
X_train = images_train[labels_train >= 8].T
X_train = X_train / X_train.max()
y_train = labels_train[labels_train >= 8].T
y_train -= 8

# test data pre processing
images_test = asarray(images_test)
labels_test = asarray(labels_test)
X_test = images_test[labels_test >= 8].T
X_test = X_test / X_test.max()
y_test = labels_test[labels_test >= 8].T
y_test -= 8

# initializing weights
w_0 = rand(X_train.shape[0]) * 2 - 1

# running sd an newton
w_sd = steepest_decent(X_train, y_train, w_0, max_iter, alpha)
w_n = newton(X_train, y_train, w_0, max_iter, epsilon)

# computing sd accuracy
print(r"8,9:\\")
print(r"Steepest Decent Accuracy:\\")
print(r"Train:", accuracy(X_train, y_train, w_sd[-1]), r"\\")
print(r"Test:", accuracy(X_test, y_test, w_sd[-1]), r"\\")

# computing convergence history - Steepest Decent
ch_sd_train = []
ch_sd_test = []
for w in w_sd:
    obj = reg_obj(X_train, y_train, w)
    ch_sd_train.append(obj)
    obj = reg_obj(X_test, y_test, w)
    ch_sd_test.append(obj)

# plotting convergence history - Steepest Decent
plt.figure()
```

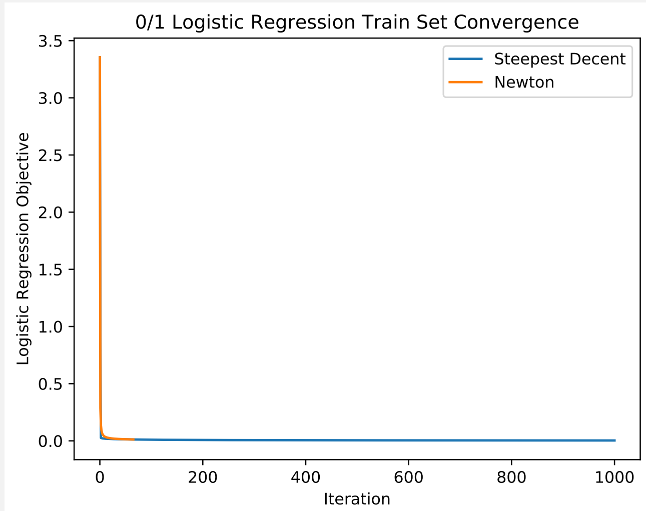
```
plt.semilogy([abs(x-ch_sd_train[-1]) for x in ch_sd_train], label="Train")
plt.semilogy([abs(x-ch_sd_test[-1]) for x in ch_sd_test], label="Test")
plt.legend()
plt.title("8/9 Logistic Regression Convergence History - Steepest Decent")
plt.ylabel("LR Objective - Optimal LR Objective")
plt.xlabel("Iteration")

plt.savefig("myplot9.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot9.pdf}")

# computing newton accuracy
print(r"Newton Accuracy:\\")
print(r"Train:", accuracy(X_train, y_train, w_n[-1]), r"\\")
print(r"Test:", accuracy(X_test, y_test, w_n[-1]), r"\\")

# computing convergence history - newton
ch_n_train = []
ch_n_test = []
for w in w_n:
    obj = reg_obj(X_train, y_train, w)
    ch_n_train.append(obj)
    obj = reg_obj(X_test, y_test, w)
    ch_n_test.append(obj)

# plotting convergence history - Steepest Decent
plt.figure()
plt.semilogy([abs(x-ch_n_train[-1]) for x in ch_n_train], label="Train")
plt.semilogy([abs(x-ch_n_test[-1]) for x in ch_n_test], label="Test")
plt.legend()
plt.title("8/9 Logistic Regression Convergence History - Newton")
plt.ylabel("LR Objective - Optimal LR Objective")
plt.xlabel("Iteration")

plt.savefig("myplot10.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot10.pdf}")

# plotting convergence train
plt.figure()
plt.plot(ch_sd_train, label="Steepest Decent")
plt.plot(ch_n_train, label="Newton")
plt.legend()
plt.title("8/9 Logistic Regression Train Set Convergence")
plt.ylabel("Logistic Regression Objective")
plt.xlabel("Iteration")

plt.savefig("myplot11.pdf", bbox_inches="tight")
print(r"\saveandshowplot{myplot11.pdf}")
```

0,1:
Steepest Decent Accuracy:
Train: 0.999131464666
Test: 0.999054373522

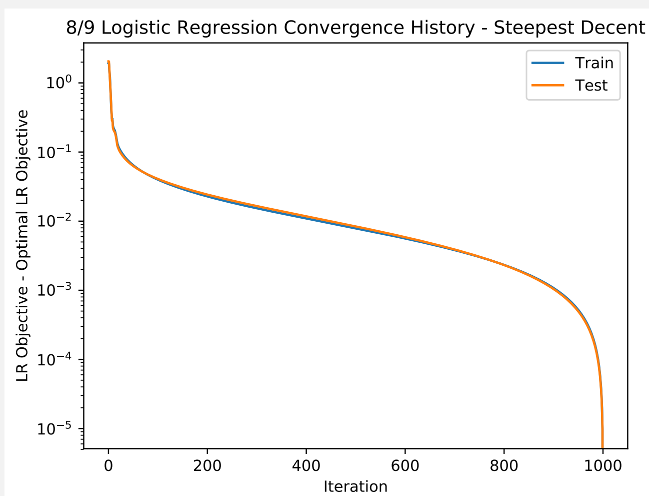


Newton Accuracy:
Train: 0.996604816423
Test: 0.998108747045

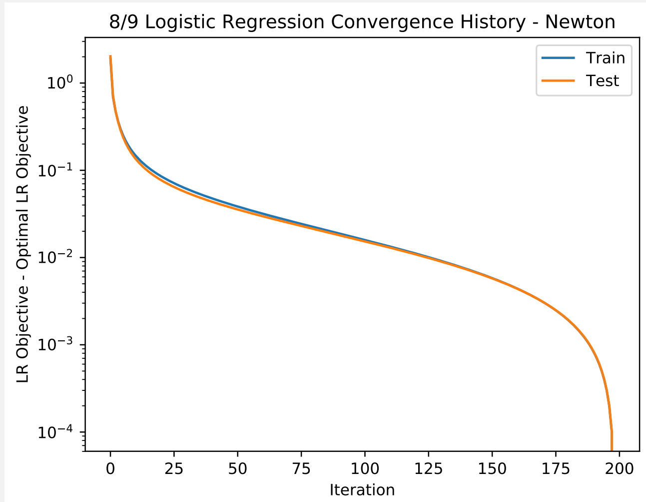


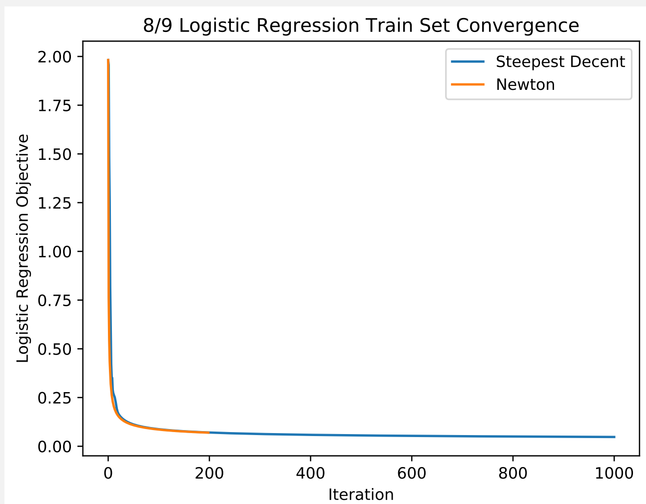


8,9:
Steepest Decent Accuracy:
Train: 0.985084745763
Test: 0.977811396873



Newton Accuracy:
Train: 0.976949152542
Test: 0.972768532526


8/9 Logistic Regression Convergence History - Newton
Train
Test
LR Objective - Optimal LR Objective
Iteration


8/9 Logistic Regression Train Set Convergence
Steepest Decent
Newton
Logistic Regression Objective
Iteration