Name: Philip Pham

AMATH 515 Homework Set 2

(1) Recall that

$$\operatorname{prox}_{tf}(y) = \arg\min_x \frac{1}{2t}\|x-y\|^2 + f(x)$$
$$f_t(y) = \min_x \frac{1}{2t}\|x-y\|^2 + f(x).$$

Suppose f is convex.

(a) Prove that f_t is convex.

Proof. Let $h(x,y) = \frac{1}{2t}\|x-y\|^2 + f(x)$, so $f_t(y) = \min_x h(x,y)$. h is a convex function of x as it is the sum of a ℓ_2-norm and a convex function. Only the first term depends on y, which is a ℓ_2-norm, which is convex, so h is convex as a function of y.

Now, using the convexity, we have that

$$\begin{aligned} f_t\left(\lambda y_1 + (1-\lambda)y_2\right) &= \min_x h\left(x, \lambda y_1 + (1-\lambda)y_2\right) \\ &\leq h\left(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2\right) \\ &\leq \lambda h\left(x_1, y_1\right) + (1-\lambda)h\left(x_2, y_2\right). \end{aligned}$$

We can choose x_1 and x_2 to be anything. For instance, we could choose $x_1 = \arg\min_x h\left(x, y_1\right)$ and $x_2 = \arg\min_x h\left(x, y_2\right)$. In this case, we'd have that

$$f_t\left(\lambda y_1 + (1-\lambda)y_2\right) \leq \lambda h\left(x_1, y_1\right) + (1-\lambda)h\left(x_2, y_2\right) = \lambda f_t\left(y_1\right) + (1-\lambda) f_t\left(y_2\right),$$

so f_t is convex. □

(b) Prove that prox_{tf} is a single-valued mapping.

Proof. In h, the first term is strongly convex with $\alpha = t^{-1} \gneq 0$. Since h is the sum of convex functions, h will also be strongly convex with $\alpha \gneq 0$. Therefore, h has a unique global minimizer, so prox_{tf} is a single-valued mapping. □

(c) Compute prox_{tf} and f_t, where $f(x) = \|x\|_1$.

Proof. We can rewrite the objective as a sum of positive terms

$$\frac{1}{2t}\|x-y\|^2 + \|x\|_1 = \sum_{i=1}^{n}\left[\frac{1}{2t}\left(x_i - y_i\right)^2 + |x_i|\right].$$

Each term is independent of each other, so we can minimize each term separately. The derivative of each term is undefined at $x_i = 0$, but otherwise,

$$\frac{\partial}{\partial x_i}\left(\frac{1}{2t}(x_i - y_i)^2 + |x_i|\right) = \frac{x_i - y_i}{t} + \operatorname{sign}(x_i).$$

Solving for x_i when $|y_i| \geq t$, we find $x_i = y_i - \operatorname{sign}(y_i)\, t$. Otherwise, we note that the derivative is negative when $x_i < 0$ and positive when $x_i > 0$, so the solution must be $x_i = 0$. Thus, we have that

$$\left[\operatorname{prox}_{tf}(y)\right]_i = \begin{cases} y_i - \operatorname{sign}(y_i)\, t, & |y_i| \geq t; \\ 0, & \text{otherwise.} \end{cases}$$

□

(d) Compute prox_{tf} and f_t for $f = \delta_{\mathbb{B}_\infty}(x)$, where $\mathbb{B}_\infty = [-1, 1]^n$.

Proof. We can use a similar strategy as the previous problem and treat each term separately. In this case, we have that

$$\frac{1}{2t}\|x - y\|^2 + \delta_{\mathbb{B}_\infty}(x) = \sum_{i=1}^{n}\left[\frac{1}{2t}(x_i - y_i)^2 + \delta_{[-1,1]}(x_i)\right].$$

For each term, we have that

$$\frac{1}{2t}(x_i - y_i)^2 + \delta_{[-1,1]}(x_i) = \begin{cases} \frac{1}{2t}(x_i - y_i)^2, & x_i \in [-1, 1]; \\ \infty, & \text{otherwise.} \end{cases}$$

This can easily be seen to be minimized by $x_i = y_i$ when $y_i \in [-1, 1]$, $x_i = 1$ when $y_i > 1$ and $x_i = -1$, when $y_i < -1$, so we have that

$$\left[\operatorname{prox}_{tf}(y)\right]_i = \begin{cases} y_i, & y_i \in [-1, 1]; \\ -1, & y_i < -1; \\ 1, & y_i > 1. \end{cases}$$

□

(2) More prox identities.

(a) Suppose f is convex and let $g(x) = f(x) + \frac{1}{2}\|x - x_0\|^2$. Find formulas for prox_{tg} and g_t in terms of prox_{tf} and f_t.

Proof. We would like to minimize the objective

$$\frac{1}{2t}\|x - y\|^2 + f(x) + \frac{1}{2}\|x - x_0\|^2 = \frac{1}{2t}\|x - y\|^2 + \frac{1}{2}\|x\|^2 + f(x) - \langle x_0, x\rangle + \frac{1}{2}\|x_0\|^2.$$

We can drop the last term when calculating $\operatorname{prox}_{tg}(y)$ since it does not depend on x. We have that

$$\operatorname{prox}_{tg}(y) = \arg\min_x \frac{1}{2t}\|x-y\|^2 + \frac{1}{2}\|x\|^2 + f(x) - \langle x_0, x\rangle .$$

From Problem (2)(c), we'll have that

$$\operatorname{prox}_{\frac{t}{2}\|\cdot\|^2}(y) = \frac{1}{1+t}y.$$

$x \mapsto -\langle x_0, x\rangle$ is differentiable, so we can differentiate $x \mapsto \frac{1}{2t}\|x-y\|^2 - \langle x_0, x\rangle$, set it equal to 0, and solve for x to get

$$\operatorname{prox}_{-t\langle x_0,\cdot\rangle}(y) = y + tx_0.$$

We can decompose $\operatorname{prox}_{tg}(y) = \operatorname{prox}_{t\left(\frac{\|\cdot\|^2}{2}+f+\langle x_0,\cdot\rangle\right)}(y)$. Yu's *On Decomposing the Proximal Map* tells us how to relate this decomposition to the individual proximal operators.

Because $x \mapsto -\langle x_0, x\rangle$ is affine, we can apply Theorem 3, which gives us

$$\begin{aligned}\operatorname{prox}_{tg}(y) &= \left(\operatorname{prox}_{t\left(\frac{\|\cdot\|^2}{2}+f\right)} \circ \operatorname{prox}_{-t\langle x_0,\cdot\rangle}\right)(y)\\ &= \operatorname{prox}_{t\left(\frac{\|\cdot\|^2}{2}+f\right)}(y+tx_0)\end{aligned}$$

by our earlier calculation.

Next, we note that $\operatorname{prox}_{\frac{t}{2}\|\cdot\|^2}(y) = \frac{1}{1+t}y$ and $\frac{1}{1+t} \in [0,1]$, so we can apply Theorem 4 to get

$$\begin{aligned}\operatorname{prox}_{tg}(y) &= \operatorname{prox}_{t\left(\frac{\|\cdot\|^2}{2}+f\right)}(y+tx_0)\\ &= \left(\operatorname{prox}_{t\frac{\|\cdot\|^2}{2}} \circ \operatorname{prox}_{tf}\right)(y+tx_0)\\ &= \operatorname{prox}_{t\frac{\|\cdot\|^2}{2}}\left(\operatorname{prox}_{tf}(y+tx_0)\right).\end{aligned}$$

Finally, we can apply our previous result to get

$$\operatorname{prox}_{tg}(y) = \frac{1}{1+t}\operatorname{prox}_{tf}(y+tx_0).$$

□

(b) The elastic net penalty is used to detect groups of correlated predictors:

$$g(x) = \beta\|x\|_1 + (1-\beta)\frac{1}{2}\|x\|^2, \quad \beta \in (0,1).$$

Write down the formula for prox_{tg} and g_t.

Proof. We can use the previous result (with $x_0 = 0$) along with Problem (1)(c) to get

$$\operatorname{prox}_{tg}(y) = \frac{1}{1 + t(1-\beta)} \operatorname{sign}(y) \left(|y| - t\beta\right)_+ ,$$

where

$$\operatorname{sign}(y)_i = \begin{cases} 1, & y_i > 0; \\ -1, & y_i < 0; \\ 0, & y_i = 0. \end{cases}$$

$$[x_+]_i = \begin{cases} x_i, & x_i \geq 0; \\ 0, & \text{otherwise}. \end{cases}$$

□

(c) Let $f(x) = \frac{1}{2}\|Cx\|^2$. Write $\operatorname{prox}_{tf}(y)$ in closed form.

Proof. By definition,

$$\operatorname{prox}_{tf}(y) = \arg\min_x \frac{1}{2t} \|x - y\|^2 + \frac{1}{2}\|Cx\|^2.$$

This is differentiable, so we can just take the derivative, set it to 0, and solve for x. This yields

$$\operatorname{prox}_{tf}(y) = \left(tC^T C + I\right)^{-1} y,$$

which is well defined since $tC^T C + I$ is always invertible. □

(d) Let $f(x) = \|x\|_2$. Write $\operatorname{prox}_{tf}(y)$ in closed form.

Proof. By definition,

$$\operatorname{prox}_{tf}(y) = \arg\min_x \frac{1}{2t} \|x - y\|^2 + \|x\|_2.$$

This is differentiable except at $x = 0$. For $x \neq 0$, we can just take the derivative, set it to 0, and solve for x. This yields

$$\frac{x - y}{t} + \frac{x}{\|x\|_2} = 0 \Rightarrow x + t\frac{x}{\|x\|_2} = y.$$

The second term of the objective only depends on the magnitude of x, so geometrically, we can see that we always want x to be in the same direction as y, that is, $x = ky$ for some $k \geq 0$, that is, the gradient can be rewritten

$$\frac{x - y}{t} + \frac{x}{\|x\|_2} = \left(k + \frac{t}{\|y\|_2} - 1\right) \frac{y}{t}$$

The derivative doesn't exist when $x = 0$, so we need to be careful in that neighborhood. If $\|y\|_2 < t$, for all values of k the gradient is pointing in the same direction as y, which implies the function decreases in the direction towards the origin. Solving for x (by solving for k) gives us a vector in the opposite direction ($k < 0$), so we want $x = 0$ in that case. Thus, we have

$$\mathrm{prox}_{tf}(y) = \begin{cases} \left(1 - \frac{t}{\|y\|_2}\right) y, & \|y\|_2 \geq t; \\ 0, & \text{otherwise.} \end{cases}$$

□

Coding Assignment

See attached pages.

AMATH 515 Homework 2

Due Date: 02/19/2020

Homework Instruction: Please follow order of this notebook and fill in the codes where commented as `TODO`.

In [1]:
```
UW_ID = "1772371"
FIRST_NAME = "Philip"
LAST_NAME = "Pham"
```

In [2]:
```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
```

Please complete the solvers in `solver.py`

In [3]:
```
import sys
sys.path.append('./')
from solvers import *
```

Problem 3: Compressive Sensing

Consier the optimization problem,

$$\min_x \ \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$$

In the following, please specify the f and g and use the proximal gradient descent solver to obtain the solution.

In [4]:
```
# create the data
np.random.seed(123)
m = 100  # number of measurements
n = 500  # number of variables
k = 10   # number of nonzero variables
s = 0.05 # measurements noise level
#
A_cs = np.random.randn(m, n)
x_cs = np.zeros(n)
x_cs[np.random.choice(range(n), k, replace=False)] = np.random.choice([-1.0, 1.0], k)
b_cs = A_cs.dot(x_cs) + s*np.random.randn(m)
#
lam_cs = 0.1*norm(A_cs.T.dot(b_cs), np.inf)
```

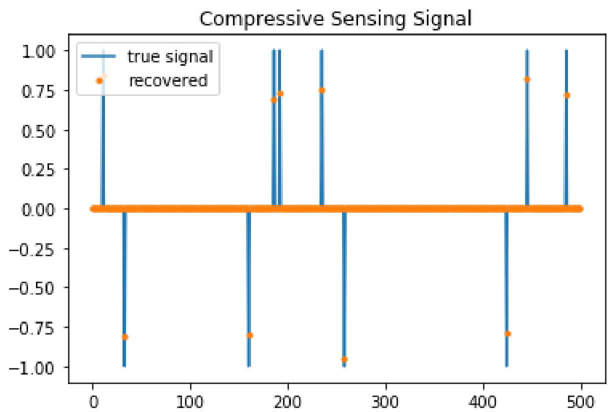
In [5]:
```
# define the function, prox and the beta constant
def func_f_cs(x):
    return np.sum(np.square(np.matmul(A_cs, x) - b_cs)) / 2.

def func_g_cs(x):
    return lam_cs * np.sum(np.abs(x))

def grad_f_cs(x):
    return np.matmul(A_cs.T, np.matmul(A_cs, x) - b_cs)

def prox_g_cs(x, t):
    delta = t * lam_cs
    return np.where(np.abs(x) >= delta, x - np.sign(x) * delta, 0.)
# Largest eigenvalue of Gramian matrix.
beta_f_cs = sorted(np.abs(np.linalg.eigvals(np.matmul(A_cs.T, A_cs))))[-1]
```

Proximal gradient descent on compressive sensing

In [6]:
```
# apply the proximal gradient descent solver
x0_cs_pgd = np.zeros(x_cs.size)
x_cs_pgd, obj_his_cs_pgd, err_his_cs_pgd, exit_flag_cs_pgd = \
    optimizeWithPGD(x0_cs_pgd, func_f_cs, func_g_cs, grad_f_cs, prox_g_cs, beta_f_cs)
```

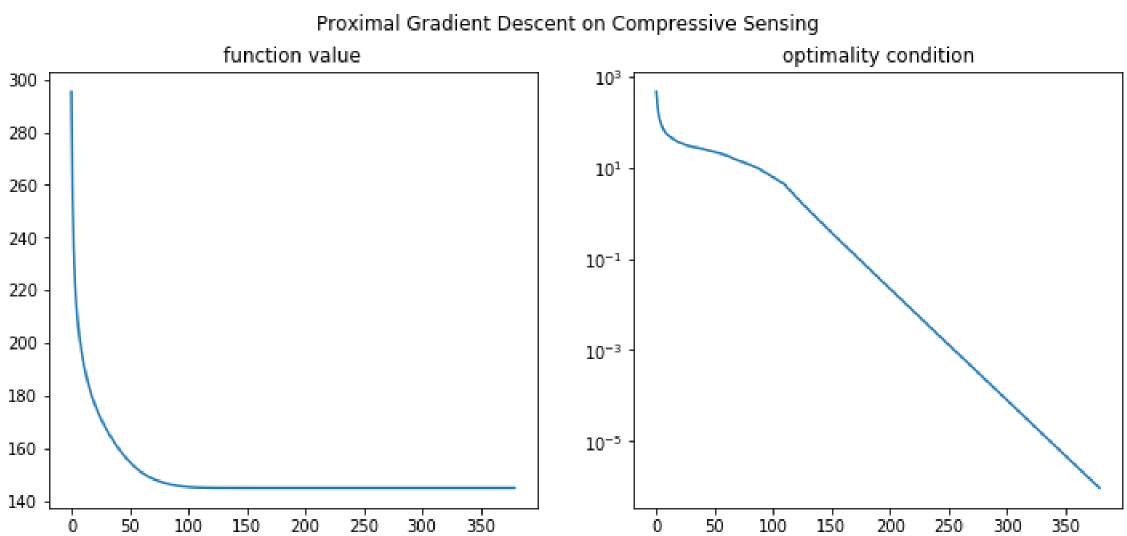
In [7]:
```
# plot signal result
plt.plot(x_cs)
plt.plot(x_cs_pgd, '.')
plt.legend(['true signal', 'recovered'])
plt.title('Compressive Sensing Signal')
```

Out[7]:
```
Text(0.5, 1.0, 'Compressive Sensing Signal')
```



In [8]:
```
# plot result
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].plot(obj_his_cs_pgd)
ax[0].set_title('function value')
ax[1].semilogy(err_his_cs_pgd)
ax[1].set_title('optimality condition')
fig.suptitle('Proximal Gradient Descent on Compressive Sensing')
```

Out[8]:
```
Text(0.5, 0.98, 'Proximal Gradient Descent on Compressive Sensing')
```

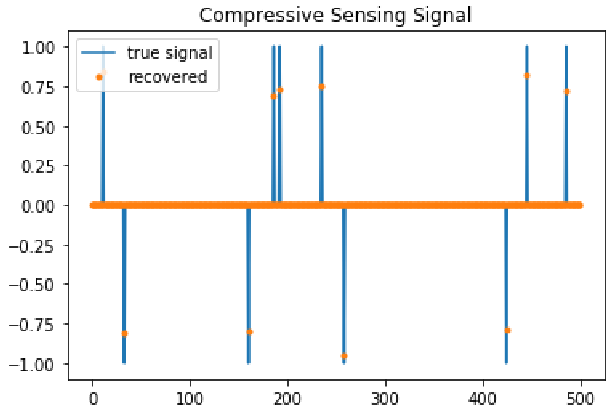


Accelerate proximal gradient descent on compressive sensing

In [9]:
```python
# apply the proximal gradient descent solver
x0_cs_apgd = np.zeros(x_cs.size)
x_cs_apgd, obj_his_cs_apgd, err_his_cs_apgd, exit_flag_cs_apgd = \
    optimizeWithAPGD(x0_cs_apgd, func_f_cs, func_g_cs, grad_f_cs, prox_g_cs, beta_f_cs)
```

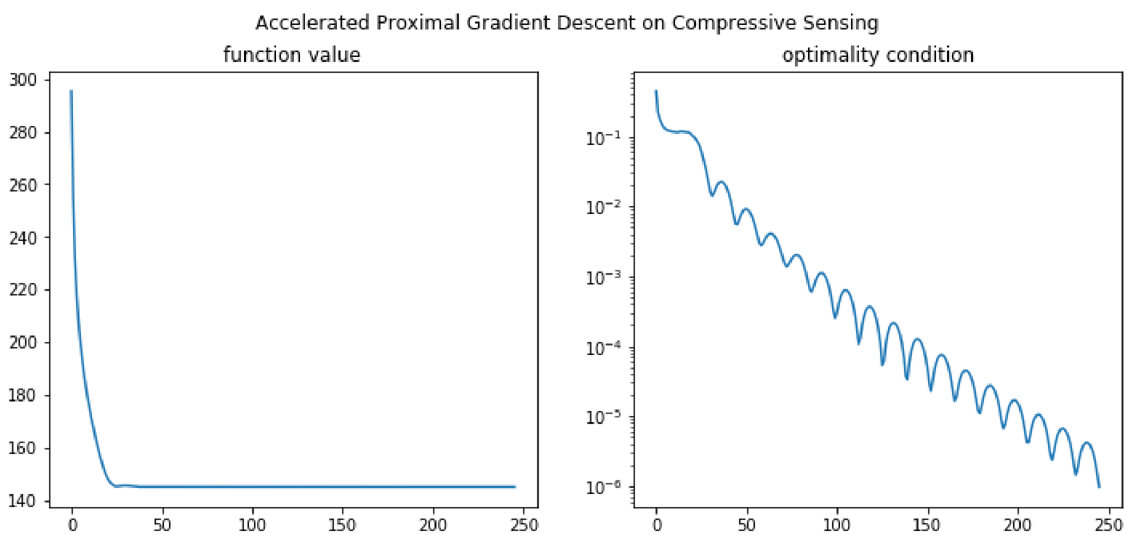
In [10]:
```python
# plot signal result
plt.plot(x_cs)
plt.plot(x_cs_apgd, '.')
plt.legend(['true signal', 'recovered'])
plt.title('Compressive Sensing Signal')
```

Out[10]: Text(0.5, 1.0, 'Compressive Sensing Signal')



In [11]:
```python
# plot result
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].plot(obj_his_cs_apgd)
ax[0].set_title('function value')
ax[1].semilogy(err_his_cs_apgd)
ax[1].set_title('optimality condition')
fig.suptitle('Accelerated Proximal Gradient Descent on Compressive Sensing')
```

Out[11]: Text(0.5, 0.98, 'Accelerated Proximal Gradient Descent on Compressive Sensing')



Problem 4: Logistic Regression on MINST Data

Now let's play with some real data, recall the logistic regression problem,

$$\min_x \sum_{i=1}^m \{\log(1+\exp(\langle a_i, x\rangle)) - b_i\langle a_i, x\rangle\} + \frac{\lambda}{2}\|x\|^2.$$

Here our data pair $\{a_i, b_i\}$, a_i is the image and b_i is the label. In this homework problem, let's consider the binary classification problem, where $b_i \in \{0, 1\}$.

In [12]:

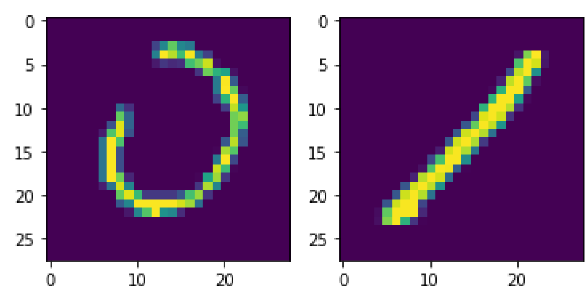
```
# import data
mnist_data = np.load('mnist01.npy', allow_pickle=True)
#
A_lgt = mnist_data[0]
b_lgt = mnist_data[1]
A_lgt_test = mnist_data[2]
b_lgt_test = mnist_data[3]
#
# set regularizer parameter
lam_lgt = 0.1
#
# beta constant of the function
beta_lgt = 0.25*norm(A_lgt, 2)**2 + lam_lgt
```

In [13]:

```
# plot the images
fig, ax = plt.subplots(1, 2)
ax[0].imshow(A_lgt[0].reshape(28,28))
ax[1].imshow(A_lgt[7].reshape(28,28))
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x11f3ad9b0>
```



In [14]:

```
# define function, gradient and Hessian
def lgt_func(x):
    logits = np.matmul(A_lgt, x)
    return np.sum(np.log(1 + np.exp(logits)) - b_lgt * logits) + 0.5 * lam_lgt * np.sum(np.squ
are(x))
#
def lgt_grad(x):
    logits = np.matmul(A_lgt, x)
    exp_logits = np.exp(logits)
    return np.sum(A_lgt.T * exp_logits / (1 + exp_logits), axis=1) - np.matmul(A_lgt.T, b_lgt)
+ lam_lgt * x
#
def lgt_hess(x):
    n_lgt = A_lgt.shape[-1]
    logits = np.matmul(A_lgt, x)
    exp_logits = np.exp(logits)
    rescaled_A_T = A_lgt.T * np.sqrt(exp_logits) / (1 + exp_logits)
    return np.matmul(rescaled_A_T, rescaled_A_T.T) + np.eye(n_lgt) * lam_lgt
```

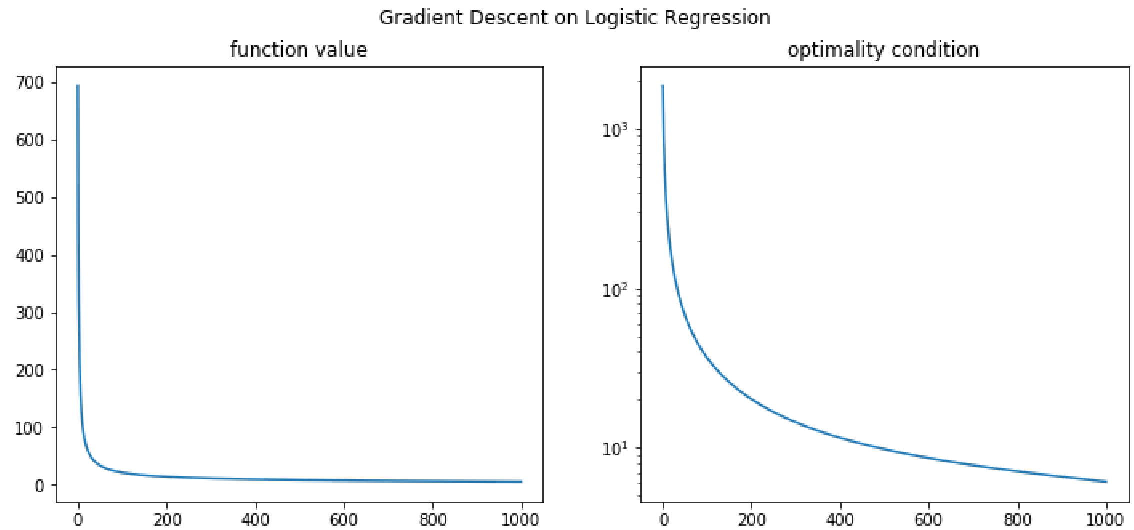
Gradient descent on logistic regression

In [15]:
```
# apply the gradient descent
x0_lgt_gd = np.zeros(A_lgt.shape[1])
x_lgt_gd, obj_his_lgt_gd, err_his_lgt_gd, exit_flag_lgt_gd = \
    optimizeWithGD(x0_lgt_gd, lgt_func, lgt_grad, beta_lgt)
```

```
Gradient descent reach maximum number of iteration.
```

In [16]:
```
# plot result
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].plot(obj_his_lgt_gd)
ax[0].set_title('function value')
ax[1].semilogy(err_his_lgt_gd)
ax[1].set_title('optimality condition')
fig.suptitle('Gradient Descent on Logistic Regression')
```

Out[16]:
```
Text(0.5, 0.98, 'Gradient Descent on Logistic Regression')
```



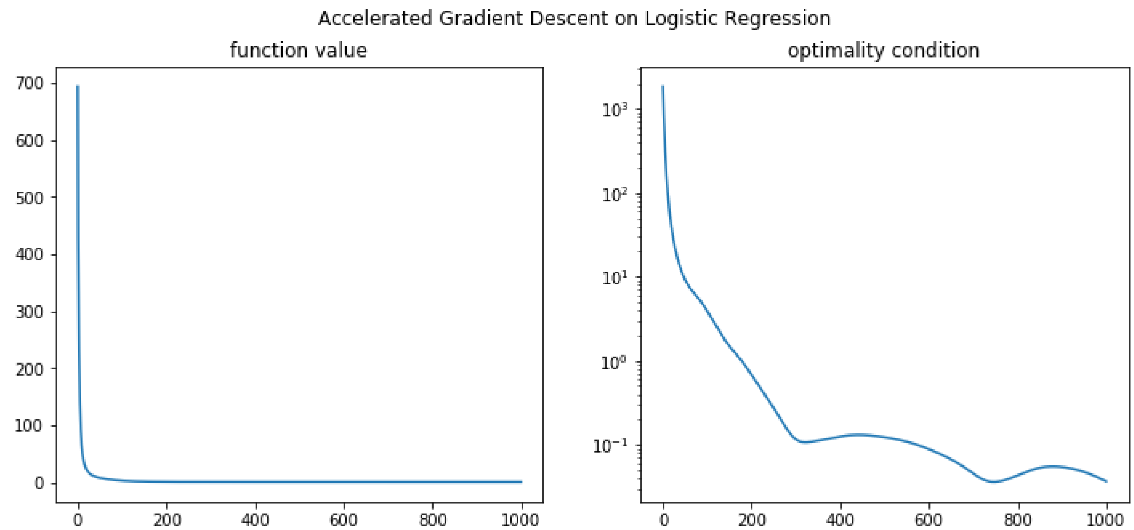
Accelerate Gradient descent on logistic regression

In [17]:
```
# apply the accelerated gradient descent
x0_lgt_agd = np.zeros(A_lgt.shape[1])
x_lgt_agd, obj_his_lgt_agd, err_his_lgt_agd, exit_flag_lgt_agd = \
    optimizeWithAGD(x0_lgt_agd, lgt_func, lgt_grad, beta_lgt)
```

```
Accelerated gradient descent reach maximum of iteration
```

In [18]:
```
# plot result
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].plot(obj_his_lgt_agd)
ax[0].set_title('function value')
ax[1].semilogy(err_his_lgt_agd)
ax[1].set_title('optimality condition')
fig.suptitle('Accelerated Gradient Descent on Logistic Regression')
```

Out[18]:
```
Text(0.5, 0.98, 'Accelerated Gradient Descent on Logistic Regression')
```

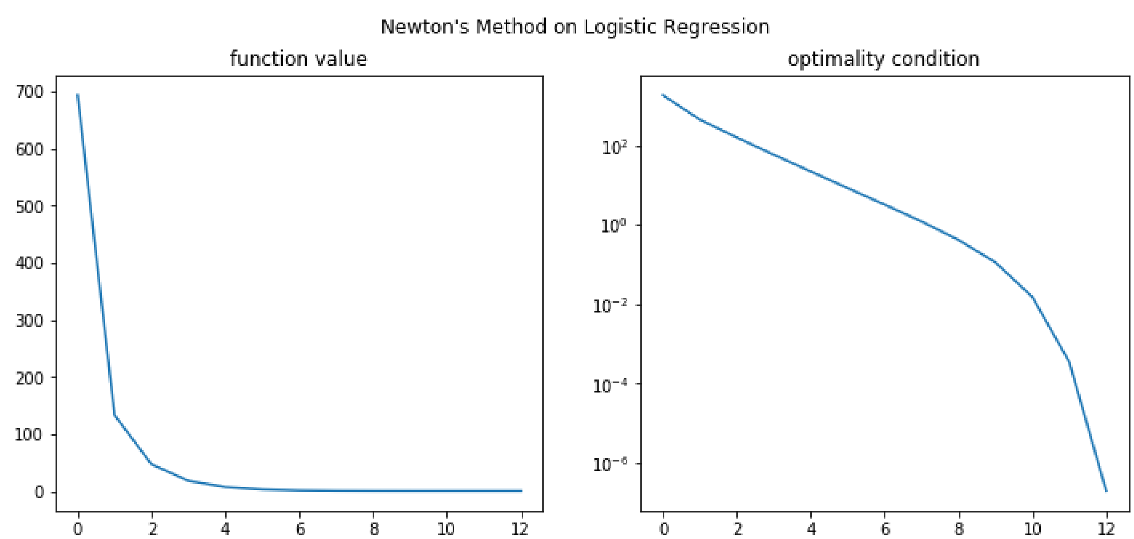


Newton's Method on logistic regression

In [19]:
```
# apply the accelerated gradient descent
x0_lgt_nt = np.zeros(A_lgt.shape[1])
x_lgt_nt, obj_his_lgt_nt, err_his_lgt_nt, exit_flag_lgt_nt = \
    optimizeWithNT(x0_lgt_nt, lgt_func, lgt_grad, lgt_hess)
```

In [20]:
```
# plot result
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].plot(obj_his_lgt_nt)
ax[0].set_title('function value')
ax[1].semilogy(err_his_lgt_nt)
ax[1].set_title('optimality condition')
fig.suptitle('Newton\'s Method on Logistic Regression')
```

Out[20]:
```
Text(0.5, 0.98, "Newton's Method on Logistic Regression")
```



Test Logistic Regression

In [21]:
```
# define accuracy function
def accuracy(x, A_test, b_test):
    r = A_test.dot(x)
    b_test[b_test == 0.0] = -1.0
    correct_count = np.sum((r*b_test) > 0.0)
    return correct_count/b_test.size
```

In [22]:
```
print('accuracy of the result is %0.3f' % accuracy(x_lgt_nt, A_lgt_test, b_lgt_test))
```

```
accuracy of the result is 1.000
```