1. Preliminaries

1.1. Data Construction

X is the "design matrix" with M*D(150*75)

w_true is the true weight vector $w_{true} \in \mathbf{R}^{d \times 1}$

y is the response with M*1

Dataset is split by taking 80 points for training, the next 2-00 points for validation and the last 50 points for testing.

In [1]:
```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import timeit
%matplotlib inline
```

In [2]:
```
m = 150
d = 75
```

In [3]:
```
np.random.seed(1738)
### Design Matrix ###
X = np.random.rand(m,d)
### Weight Vector ###
w_true = np.zeros(d)
w_true[:10] = (np.random.choice(2,10)-0.5)*2
### Response ###
sigma = 0.1
mu = 0
epsilon = sigma*np.random.randn(m) + mu
y = np.dot(X, w_true)+epsilon
```

In [4]:
```
X_train = X[:80,:]
X_validation = X[80:100,:]
X_test = X[-50:150,:]
```

In [5]:
```
y_train = y[:80]
y_validation = y[80:100]
y_test = y[-50:150]
```

1.2. Ridge Regression

Run ridge regression on this dataset. Choose the λ that minimizes the square loss on the validation set.

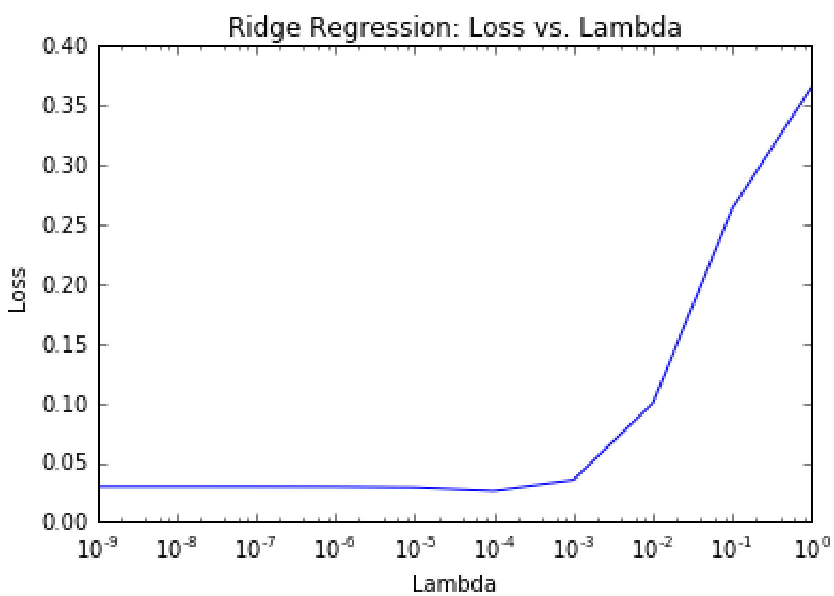
In [6]:

```
def ridge(X, y, Lambda):
    (N, D) = X.shape
    def ridge_obj(theta):
        return ((np.linalg.norm(np.dot(X, theta) - y))**2) / (2 *
N) +\
            Lambda * (np.linalg.norm(theta))**2
    return ridge_obj

def compute_loss(X, y, theta):
    (N, D) = X.shape
    return ((np.linalg.norm(np.dot(X, theta) - y))**2) / (2 * N)

def ridge_regression(Lambda=0):
    """
    Args:
        lambda - if not given loop through differen lambdas and fin
d the best
    Returns:
        theta_opt - the optimization of parameter vector, 1D numpy
array of size (num_features)
        lambda_opt - the history optimization of parameter, 1D nump
y array of size (num_lambdas)
        loss_opt - the history of regularized loss value, 1D numpy
array of size (num_lambdas)
    """
    (N, D) = X.shape
    w = np.random.rand(D, 1)
    Lambda_history = []
    loss_history = []
    t = 0
    loss_min=lambda_min=w_min=np.nan
    if (Lambda==0):
        for i in range(-9, 1):
            Lambda = 10**i
            Lambda_history.append(Lambda)
            w_opt = minimize(ridge(X_train, y_train, Lambda), w)
            loss = compute_loss(X_validation, y_validation, w_opt.
x)

            loss_history.append(loss)
            print(Lambda, loss)
            if (t==0 or loss<loss_min):
                loss_min = loss
                lambda_min = Lambda
                w_min = w_opt.x.copy()
                t=t+1
    else:
        w_opt = minimize(ridge(X_train, y_train,Lambda), w)
        loss_opt = compute_loss(X_validation, y_validation, w_opt.
x)
        return w_opt.x.copy(), Lambda, loss_opt

    print "Best Lambda",lambda_min
    print "Best Loss", loss_min
    return w_min, Lambda_history, loss_history
```

```
w_min, Lambda_hist, loss_hist = ridge_regression()
```

```
(1e-09, 0.029819016409124582)
(1e-08, 0.029817999953792869)
(1e-07, 0.029807937055520251)
(1e-06, 0.029708705105347 66)
(1e-05, 0.029149523109953529)
(0.0001, 0.026048744308054129)
(0.001, 0.035628138031012738)
(0.01, 0.1000448583973369)
(0.1, 0.26339067097969077)
(1, 0.3654767260665357)
Best Lambda 0.0001
Best Loss 0.0260487443081
```

In [7]:
```
plt.plot(Lambda_hist, loss_hist)
plt.xlabel('Lambda')
plt.ylabel('Loss')
plt.xscale('log')
plt.title('Ridge Regression: Loss vs. Lambda')
plt.show()
```



In [8]:
```
### Choose 1e-4 as lambda, test model coefficients ###
Lambda = 10**-4
w_min, Lambda, loss_hist = ridge_regression(Lambda)

### Coefficient Report under tolerence=0.0 ###

tolerance = 0.0
false_nonzeros = sum((w_true==0) & (abs(w_min)>=tolerance))
false_zeros = sum((w_true!=0) & (abs(w_min)<tolerance))
true_nonzeros = sum((w_true!=0) & (abs(w_min)>=tolerance))
true_zeros = sum((w_true==0) & (abs(w_min)<tolerance))
print "With tolerence of {0}".format(tolerance)
print "True Value Zero estimated as Nonzero: {0}".format(false_nonz
eros)
print "True Value Nonzero estimated as Zero: {0}".format(false_zero
s)
print "True Value Zero estimated as Zero: {0}".format(true_zeros)
print "True Value Nonzero estimated as Nonzero: {0}".format(true_no
nzeros)

### Coefficient Report under tolerence=0.05 ###

tolerance = 0.05
false_nonzeros = sum((w_true==0) & (abs(w_min)>=tolerance))
false_zeros = sum((w_true!=0) & (abs(w_min)<tolerance))
true_nonzeros = sum((w_true!=0) & (abs(w_min)>=tolerance))
true_zeros = sum((w_true==0) & (abs(w_min)<tolerance))
print "With tolerence of {0}".format(tolerance)
print "True Value Zero estimated as Nonzero: {0}".format(false_nonz
eros)
print "True Value Nonzero estimated as Zero: {0}".format(false_zero
s)
print "True Value Zero estimated as Zero: {0}".format(true_zeros)
print "True Value Nonzero estimated as Nonzero: {0}".format(true_no
nzeros)
```

```
With tolerence of 0.0
True Value Zero estimated as Nonzero: 65
True Value Nonzero estimated as Zero: 0
True Value Zero estimated as Zero: 0
True Value Nonzero estimated as Nonzero: 10
With tolerence of 0.05
True Value Zero estimated as Nonzero: 41
True Value Nonzero estimated as Zero: 0
True Value Zero estimated as Zero: 24
True Value Nonzero estimated as Nonzero: 10
```

2. Coordinate Descent for Lasso (a.k.a. The Shooting Algo)

2.1 Experiment with the Shooting Algorithm

(1) Write a function that computes the Lasso solution for a given λ

In [9]:
```
def soft(a, delta):
    return np.sign(a)*max( abs(a) - delta, 0)

def lasso_shooting(X, y, Lambda=10, tolerance=1e-4):
    (N, D) = X.shape
    maxIt = 1000
    ### Shooting Algo ###
    it = 1
    converged = False
    ### Initialize w ###
    w = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) + np.identity
(D)), X.T), y)
    while (not converged) & (it<maxIt):
        w_old = w.copy()
        for j in range(D):
            aj=cj=0
            for i in range(1,N):
                aj=aj+2*X[i,j]**2
                cj=cj+2*X[i,j]*(y[i]-np.dot(w,X[i,:])+ w[j]*X[i,j])

            w[j] = soft(cj/aj,Lambda/aj)
        it = it + 1
        converged = (sum(abs(w-w_old)) < tolerance)
    #print "Converged:",converged,"Iterations:",it
    return w, converged

### sanity test ###
%timeit lasso_shooting(X_train, y_train, 10)
```

```
Converged: True Iterations: 69
Converged: True Iterations: 69
Converged: True Iterations: 69
Converged: True Iterations: 69
1 loop, best of 3: 1.14 s per loop
```

In [10]:
```
def timeme(func,iterations=1,*args,**kwargs):
    """
    Timer wrapper.  Runs a given function, with arguments,
    100 times and displays the average time per run.
    """
    def wrapper(func, *args, **kwargs):
        def wrapped():
            return func(*args, **kwargs)
        return wrapped
    wrapped = wrapper(func,*args,**kwargs)
    run_time = float(timeit.timeit(wrapped, number=iterations))/ite
rations
    print "Avg time to run %s after %i trials: %i seconds per tria
l" %(func,iterations,run_time)
```

In [11]:
```
def lambda_search():
    t=0
    Lambdas=[]
    loss_hist=[]
    loss_min = lambda_min=w_opt=np.nan
    print "Start Searching"
    Lambda_max = np.linalg.norm(np.dot(X.T,y),np.inf)
    log_lambda_max = np.log10(Lambda_max)
    for i in np.linspace(-2, log_lambda_max, 15):
        Lambda = 10**i
        print "Lambda",Lambda
        w, converged = lasso_shooting(X_train, y_train, Lambda)
        Lambdas.append(Lambda)
        loss = compute_loss(X_validation, y_validation, w)
        loss_hist.append(loss)
        if t==0:
            loss_min = loss
            lambda_min = Lambda
            w_min = w.copy()
        elif converged:
            if loss<=loss_min:
                loss_min = loss
                lambda_min = Lambda
                w_min = w.copy()
        t=t+1
    best_lost = lasso_shooting(X_test, y_test, lambda_min)
    print "Best Lambda:",lambda_min
    print "Square Loss on Test Data:", loss_min

    return w_opt,Lambdas,loss_hist


timeme(lambda_search,3)
```

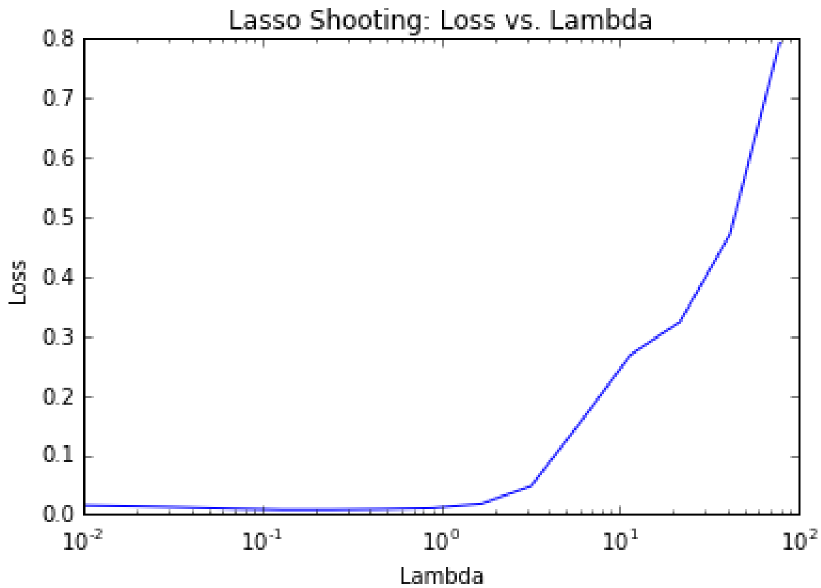
```
Start Searching
Lambda 0.01
Converged: False Iterations: 1000
Lambda 0.018977868643
Converged: False Iterations: 1000
Lambda 0.0360159498232
Converged: False Iterations: 1000
Lambda 0.06835059648
Converged: False Iterations: 1000
Lambda 0.129714864167
Converged: False Iterations: 1000
Lambda 0.246171165321
Converged: True Iterations: 927
Lambda 0.467180403917
Converged: True Iterations: 458
Lambda 0.886608833814
Converged: True Iterations: 254
Lambda 1.68259459859
Converged: True Iterations: 186
Lambda 3.19320592715
Converged: True Iterations: 169
Lambda 6.06002426356
Converged: True Iterations: 142
Lambda 11.5006344447
Converged: True Iterations: 54
Lambda 21.8257529804
Converged: True Iterations: 47
Lambda 41.4206273097
Converged: True Iterations: 17
Lambda 78.6075224196
Converged: True Iterations: 7
Converged: True Iterations: 775
Best Lambda: 0.246171165321
Square Loss on Test Data: 0.008095764033
Start Searching
Lambda 0.01
Converged: False Iterations: 1000
Lambda 0.018977868643
Converged: False Iterations: 1000
Lambda 0.0360159498232
Converged: False Iterations: 1000
Lambda 0.06835059648
Converged: False Iterations: 1000
Lambda 0.129714864167
Converged: False Iterations: 1000
Lambda 0.246171165321
Converged: True Iterations: 927
Lambda 0.467180403917
Converged: True Iterations: 458
Lambda 0.886608833814
Converged: True Iterations: 254
Lambda 1.68259459859
Converged: True Iterations: 186
Lambda 3.19320592715
```

```
Converged: True Iterations: 169
Lambda 6.06002426356
Converged: True Iterations: 142
Lambda 11.5006344447
Converged: True Iterations: 54
Lambda 21.8257529804
Converged: True Iterations: 47
Lambda 41.4206273097
Converged: True Iterations: 17
Lambda 78.6075224196
Converged: True Iterations: 7
Converged: True Iterations: 775
Best Lambda: 0.246171165321
Square Loss on Test Data: 0.008095764033
Start Searching
Lambda 0.01
Converged: False Iterations: 1000
Lambda 0.018977868643
Converged: False Iterations: 1000
Lambda 0.0360159498232
Converged: False Iterations: 1000
Lambda 0.06835059648
Converged: False Iterations: 1000
Lambda 0.129714864167
Converged: False Iterations: 1000
Lambda 0.246171165321
Converged: True Iterations: 927
Lambda 0.467180403917
Converged: True Iterations: 458
Lambda 0.886608833814
Converged: True Iterations: 254
Lambda 1.68259459859
Converged: True Iterations: 186
Lambda 3.19320592715
Converged: True Iterations: 169
Lambda 6.06002426356
Converged: True Iterations: 142
Lambda 11.5006344447
Converged: True Iterations: 54
Lambda 21.8257529804
Converged: True Iterations: 47
Lambda 41.4206273097
Converged: True Iterations: 17
Lambda 78.6075224196
Converged: True Iterations: 7
Converged: True Iterations: 775
Best Lambda: 0.246171165321
Square Loss on Test Data: 0.008095764033
Avg time to run <function lambda_search at 0x115e98758> after 3 tr
ials: 114 seconds per trial
```

In [12]:
```
w_opt,Lambdas,loss_hist_lasso = lambda_search()
```

```
Start Searching
Lambda 0.01
Converged: False Iterations: 1000
Lambda 0.018977868643
Converged: False Iterations: 1000
Lambda 0.0360159498232
Converged: False Iterations: 1000
Lambda 0.06835059648
Converged: False Iterations: 1000
Lambda 0.129714864167
Converged: False Iterations: 1000
Lambda 0.246171165321
Converged: True Iterations: 927
Lambda 0.467180403917
Converged: True Iterations: 458
Lambda 0.886608833814
Converged: True Iterations: 254
Lambda 1.68259459859
Converged: True Iterations: 186
Lambda 3.19320592715
Converged: True Iterations: 169
Lambda 6.06002426356
Converged: True Iterations: 142
Lambda 11.5006344447
Converged: True Iterations: 54
Lambda 21.8257529804
Converged: True Iterations: 47
Lambda 41.4206273097
Converged: True Iterations: 17
Lambda 78.6075224196
Converged: True Iterations: 7
Converged: True Iterations: 775
Best Lambda: 0.246171165321
Square Loss on Test Data: 0.008095764033
```

In [13]:
```
plt.plot(Lambdas,loss_hist_lasso)
plt.xlabel('Lambda')
plt.ylabel('Loss')
plt.xscale('log')
plt.title('Lasso Shooting: Loss vs. Lambda')
plt.show()
```



(2) Analyze the sparsity of your solution, reporting how many components with true value zero have been estimated to be non-zero, and vice-versa.

> There are 3 cases where true value zero were estimated as nonzero. Vice versa never happens

In [14]:
```
### Choose 0.246171165321 as lambda, test model coefficients ###
Lambda = 0.246171165321
w_opt, _ = lasso_shooting(X_train, y_train, Lambda)

### Coefficient Report under tolerence=0.05 ###
tolerance = 0.05
false_nonzeros = sum((w_true==0) & (abs(w_opt)>=tolerance))
false_zeros = sum((w_true!=0) & (abs(w_opt)<tolerance))
true_nonzeros = sum((w_true!=0) & (abs(w_opt)>=tolerance))
true_zeros = sum((w_true==0) & (abs(w_opt)<tolerance))
print "With tolerence of {0}".format(tolerance)
print "True Value Zero estimated as Nonzero: {0}".format(false_nonz
eros)
print "True Value Nonzero estimated as Zero: {0}".format(false_zero
s)
print "True Value Zero estimated as Zero: {0}".format(true_zeros)
print "True Value Nonzero estimated as Nonzero: {0}".format(true_no
nzeros)
```

```
Converged: True Iterations: 927
With tolerence of 0.05
True Value Zero estimated as Nonzero: 3
True Value Nonzero estimated as Zero: 0
True Value Zero estimated as Zero: 62
True Value Nonzero estimated as Nonzero: 10
```

(3) Implement the homotopy method described above. Compare the runtime for computing the full regularization path (for the same set of λ's you tried in the first question above) using the homotopy method compared to the basic shooting algorithm.

In [15]:
```
def soft(a, delta):
    return np.sign(a)*max( abs(a) - delta, 0)

def homotopy_lasso_shooting(X, y, Lambda=1, tolerance=1e-4, w='unde
fined'):
    (N, D) = X.shape
    maxIt = 1000
    ### Shooting Algo ###
    it = 1
    converged = False
    ### Initialize w ###
    if w=='undefined':
        w=np.zeros(D)
    while (not converged) & (it<maxIt):
        w_old = w.copy()
        for j in range(D):
            aj=cj=0
            for i in range(1,N):
                aj=aj+2*X[i,j]**2
                cj=cj+2*X[i,j]*(y[i]-np.dot(w,X[i,:])+ w[j]*X[i,j])

            w[j] = soft(cj/aj,Lambda/aj)
        it = it + 1
        converged = (sum(abs(w-w_old)) < tolerance)
    print "Converged:",converged,"Iterations:",it
    return w, converged
#homotopy_lasso_shooting(X_train, y_train, 1)
```

In [16]:
```
def homotopy_lambda_search():
    t=0
    Lambdas=[]
    loss_hist=[]
    loss_min = lambda_min=w_opt=np.nan
    (N, D) = X.shape
    Lambda_max = np.linalg.norm(np.dot(X.T,y),np.inf)
    log_lambda_max = np.log10(Lambda_max)
    print "Start Searching"
    w_old = w = np.zeros(D)
    for i in np.linspace(log_lambda_max, -2, 15):
        Lambda = 10**i
        print "Lambda = {0}. Warm Starting.....".format(Lambda)

        w_old = w
        w, converged = homotopy_lasso_shooting(X_train, y_train, La
mbda=Lambda, w=w_old)

        Lambdas.append(Lambda)
        loss = compute_loss(X_validation, y_validation, w)
        loss_hist.append(loss)
        if t==0:
            loss_min = loss
            lambda_min = Lambda
            w_min = w.copy()
        elif converged:
            if loss<=loss_min:
                loss_min = loss
                lambda_min = Lambda
                w_min = w.copy()
        t=t+1
    best_lost = homotopy_lasso_shooting(X_test, y_test, lambda_min)
    print "Best Lambda:",lambda_min
    print "Square Loss on Test Data:", loss_min

    return w_opt,Lambdas,loss_hist

timeme(homotopy_lambda_search)
#%timeit w_lasso,lambdas_lasso,loss_hist_lasso = homotopy_lambda_se
arch()
```

```
Start Searching
Lambda = 78.6075224196. Warm Starting.....
Converged: True Iterations: 3
Lambda = 41.4206273097. Warm Starting.....
Converged: True Iterations: 22
Lambda = 21.8257529804. Warm Starting.....
Converged: True Iterations: 44
Lambda = 11.5006344447. Warm Starting.....
Converged: True Iterations: 44
Lambda = 6.06002426356. Warm Starting.....
Converged: True Iterations: 140
Lambda = 3.19320592715. Warm Starting.....
Converged: True Iterations: 171
Lambda = 1.68259459859. Warm Starting.....
Converged: True Iterations: 165
Lambda = 0.886608833814. Warm Starting.....
Converged: True Iterations: 192
Lambda = 0.467180403917. Warm Starting.....
Converged: True Iterations: 302
Lambda = 0.246171165321. Warm Starting.....
Converged: True Iterations: 647
Lambda = 0.129714864167. Warm Starting.....
Converged: True Iterations: 875
Lambda = 0.06835059648. Warm Starting.....
Converged: False Iterations: 1000
Lambda = 0.0360159498232. Warm Starting.....
Converged: True Iterations: 886
Lambda = 0.018977868643. Warm Starting.....
Converged: False Iterations: 1000
Lambda = 0.01. Warm Starting.....
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Best Lambda: 0.129714864167
Square Loss on Test Data: 0.00802342043097
Avg time to run <function homotopy_lambda_search at 0x115e1b398> a
fter 1 trials: 105 seconds per trial

/usr/local/lib/python2.7/site-packages/ipykernel/__main__.py:11: F
utureWarning: elementwise comparison failed; returning scalar inst
ead, but in the future will perform elementwise comparison
```

(4) Implement the matrix expressions and measure the speedup to compute the regularization path.

In [17]:
```
def soft(a, delta):
    return np.sign(a)*max( abs(a) - delta, 0)

def vect_homotopy_lasso_shooting(X, y, Lambda=1, tolerance=1e-4, w
='undefined'):
    (N, D) = X.shape
    maxIt = 1000
    ### Shooting Algo ###
    it = 1
    converged = False
    ### Initialize w ###
    if w=='undefined':
        w=np.zeros(D)
    while (not converged) & (it<maxIt):
        w_old = w.copy()
        for j in range(D):
            aj = 2*np.dot(X[:,j].T,X[:,j])
            cj = 2*(X[:,j].dot(y) - (w.T.dot(X.T)).dot(X[:,j]) + w
[j]*(X[:,j].T.dot(X[:,j])))
            w[j] = soft(cj/aj,Lambda/aj)
        it = it + 1
        converged = (sum(abs(w-w_old)) < tolerance)
    print "Converged:",converged,"Iterations:",it
    return w, converged
#homotopy_lasso_shooting(X_train, y_train, 1)
```

In [18]:
```
def vect_homotopy_lambda_search():
    t=0
    Lambdas=[]
    loss_hist=[]
    loss_min = lambda_min=w_opt=np.nan
    (N, D) = X.shape
    Lambda_max = np.linalg.norm(np.dot(X.T,y),np.inf)
    log_lambda_max = np.log10(Lambda_max)
    print "Start Searching"
    w_old = w = np.zeros(D)
    for i in np.linspace(log_lambda_max, -2, 15):
        Lambda = 10**i
        print "Lambda = {0}. Warm Starting.....".format(Lambda)

        w_old = w
        w, converged = vect_homotopy_lasso_shooting(X_train, y_trai
n, Lambda=Lambda, w=w_old)

        Lambdas.append(Lambda)
        loss = compute_loss(X_validation, y_validation, w)
        loss_hist.append(loss)
        if t==0:
            loss_min = loss
            lambda_min = Lambda
            w_min = w.copy()
        elif converged:
            if loss<=loss_min:
                loss_min = loss
                lambda_min = Lambda
                w_min = w.copy()
        t=t+1
    best_lost = vect_homotopy_lasso_shooting(X_test, y_test, lambda
_min)
    print "Best Lambda:",lambda_min
    print "Square Loss on Test Data:", loss_min

    return w_opt,Lambdas,loss_hist

timeme(vect_homotopy_lambda_search)
#%timeit w_lasso,lambdas_lasso,loss_hist_lasso = homotopy_lambda_se
arch()
```

```
Start Searching
Lambda = 78.6075224196. Warm Starting.....
Converged: True Iterations: 3
Lambda = 41.4206273097. Warm Starting.....
Converged: True Iterations: 22
Lambda = 21.8257529804. Warm Starting.....
Converged: True Iterations: 44
Lambda = 11.5006344447. Warm Starting.....
Converged: True Iterations: 44
Lambda = 6.06002426356. Warm Starting.....
Converged: True Iterations: 146
Lambda = 3.19320592715. Warm Starting.....
Converged: True Iterations: 175
Lambda = 1.68259459859. Warm Starting.....
Converged: True Iterations: 168
Lambda = 0.886608833814. Warm Starting.....
Converged: True Iterations: 196
Lambda = 0.467180403917. Warm Starting.....
Converged: True Iterations: 313
Lambda = 0.246171165321. Warm Starting.....
Converged: True Iterations: 666
Lambda = 0.129714864167. Warm Starting.....
Converged: True Iterations: 884
Lambda = 0.06835059648. Warm Starting.....
Converged: False Iterations: 1000
Lambda = 0.0360159498232. Warm Starting.....
Converged: True Iterations: 803
Lambda = 0.018977868643. Warm Starting.....
Converged: True Iterations: 631
Lambda = 0.01. Warm Starting.....
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Best Lambda: 0.129714864167
Square Loss on Test Data: 0.00787268083111
Avg time to run <function vect_homotopy_lambda_search at 0x115ede2
30> after 1 trials: 5 seconds per trial
```

```
/usr/local/lib/python2.7/site-packages/ipykernel/__main__.py:11: F
utureWarning: elementwise comparison failed; returning scalar inst
ead, but in the future will perform elementwise comparison
```

2.2 Deriving the Coordinate Minimizer for lasso

(1) First let's get a trival case out of the way. If $x_{ij} = 0$ for $i = 1, \ldots, n$, what is the coordinate minimizer w_j?

> **ANSWER**
>
> $w_j = 0$

(2) Write the derivative of $f(w_j)$.

> **ANSWER**
>
> $$\begin{aligned} \partial_{w_j} f(w_j) &= (a_j w_j - c_j) + \lambda \partial_{w_j} \|w_j\|_1 \\ &= a_j w_j - c_j + sign(w_j)\lambda \end{aligned}$$

(3) If $w_j > 0$ and minimizes f, then show that $w_j = -\frac{1}{a_j}(\lambda - c_j)$. Similarly, if $w_j < 0$ and minimizes f, show that $w_j = \frac{1}{a_j}(\lambda + c_j)$. Give conditions on c_j that imply the minimizer $w_j > 0$ and $w_j < 0$, respectively.

> **ANSWER**
>
> The magnitude of c_j is an indication of relevance feature j is for predicting y.
>
> If $c_j > \lambda$, then feature j is strongly possitively correlated with the residual, so $w_j > 0$.
>
> If $c_j < -\lambda$, then feature j is strongly negatively correlated with the residual, so $w_j < 0$.

(4) Derive expressions for the two one-sided derivative at $f(0)$, and show that $c_j \in [-\lambda, \lambda]$ implies that $w_j = 0$ is a minimizer.

> **ANSWER**
>
> $$a = \lim_{w \to 0} \frac{\|w\| - \|0\|}{w} = 1$$
> $$b = \lim_{w \to 0} \frac{\| - w\|) - \|0\|)}{w} = -1$$
>
> The derivative for the $f(w_j)$ is
>
> $$[-c_j + \lambda b, -c_j + \lambda a] = [-c_j - \lambda, -c_j + \lambda]$$
>
> $\therefore$ 0 is the minimizer.

(5) Conclude the minimizer.

> If $c_j > \lambda$, then feature j is strongly possitively correlated with the residual, so $w_j > 0$ and $w_j = -\frac{1}{a_j}(\lambda - c_j)$.
>
> if $c_j \in [-\lambda, \lambda]$, $w_j = 0$ is a minimizer.
>
> If $c_j < -\lambda$, then feature j is strongly negatively correlated with the residual, so $w_j < 0$ and $w_j = \frac{1}{a_j}(\lambda + c_j)$.

3 Lasso Properties

3.1 Deriving λ_{max}

(1) Compute $L\prime(0; v)$.

ANSWER

$$\begin{aligned} L'(0;v) &= \lim_{h\to 0} \tfrac{1}{h}[\|Xhv-y\|_2^2 + \lambda\|hv\|_1 - \|y\|^2 - \lambda\|0\|] \\ &= -2(Xv)^T y + \lambda\|v\| \end{aligned}$$

(2) Since the Lasso objective is convex, for w^* to be a minimizer of $L(w)$ we must have that the directional derivative $L'(w^*;v) \geq 0$ for all v. Starting form the condition $L'(0;v) \geq 0$, rearrange terms to get a lower bounds on λ.

ANSWER

$$\begin{aligned} 0 &\leq L'(0;v) \\ &\leq -2vX^T y + \lambda\|v\| \\ \lambda &\geq \frac{2vX^T y}{\|v\|_1} \quad for \quad every \quad v \end{aligned}$$

λ's lower bound is $\frac{2vX^T y}{\|v\|_1}$

(3) Since our lower bounds on λ for all v, we want to compute the maximum lower bound. Compute the maximum lower bound of λ by maximizing the expression over v. Show that this expression is equivalent to $\lambda_{max} = 2\|X^T y\|_\infty$.

ANSWER

$$\begin{aligned} L'(w,v) &= \lim_{h\to 0} \frac{L(w+hv)-L(w)}{h} \\ &= \lim_{h\to 0} \frac{(X(w+hv)-y)^T(X(w+hv)-y)+\lambda(\|w+hv\|_1-\|w\|_1)}{h} \\ &= 2X^T(Xw-y)v + \lambda(\sum_{j,w_j\neq 0} sign(w_j)v_j + \sum_{j,w_j=0} |v_j|) \end{aligned}$$

Lower bounds for all v and w should be,

$$\lambda_{max} \geq \frac{2X^T(y-Xw)v}{\sum_{j,w_j\neq 0} sign(w_j)v_j + \sum_{j,w_j=0} |v_j|} = 2\|X^T y\|_\infty$$

(4) Show that for $L(w) = \|Xw + b - y\|_2^2 + \lambda\|w\|_1$,$\lambda_{max} = 2\|X^T(y - \bar{y})\|_\infty$ where $\bar{y}$ is the mean of valus in the vector y.

ANSWER

$$L(w) = \sum_{i=1}^{n}(w^T x_i + b - y_i) + \lambda\|w\|_1$$

$$\begin{aligned} 0 &\leq L'(0; v) \\ &\leq -2vX^T(y - b) + \lambda\|v\| \end{aligned}$$

Therefore,

$$\begin{aligned} \lambda &\geq 2\frac{v}{\|v\|_1}X^T(y - b) \\ &= 2\|X^T(y - b)\|_\infty \\ &= 2\|X^T(y - \bar{y} + \bar{y} - b)\| \\ &\geq 2\|X^T(y - \bar{y})\|_\infty \end{aligned}$$

Hence,

$$\lambda_{max} = 2\|X^T(y - \bar{y})\|_\infty$$

3.2 Feature Correlation

(1) Derive the relation between $\hat{\theta}_i$ and $\hat{\theta}_j$, the i^{th} and the j^{th} components of the optimal weight vector obtained by solving the Lasso optimization problem.

ANSWER

Assume $\hat{\theta}_i = a$ and $\hat{\theta}_j = b$

The lasso objective function, below, must minimize both loss and regularization.

$$\sum_{k=1}^{n}(h(x_k) - y_k)^2 + \lambda\|w\|_1$$

First consider the loss, $\sum_{k=1}^{n}(h(x_k) - y_k)^2$, where $h(x_k) = \mathbf{w}^T x_k$

The loss due to x_i , x_j is $\sum_{k=1}^{n}(\hat{\theta}_i X_{ik} + \hat{\theta}_j X_{jk} - y_k)$

Since $X_i = X_j$, this simplifies to $\sum_{k=1}^{n}((\hat{\theta}_i + \hat{\theta}_j)X_{ik} - y_k)$

Thus the optimal values a and b must sum to another value c that minimizes this expression $\sum_{k=1}^{n}(cX_{ik} - y_k)$

Next we minimize the lasso regularization component, $\lambda\|w\|_1 = \lambda \sum_{l=1}^{d} |w_l|$

The regularization penalty due to x_i , x_j is $|a| + |b|$. If a and b are of opposite sign, and both are nonzero, then $|a| + |b| > |c| + |0|$, and the regularization penalty is not minimized. Therefore a and b must be of the same sign and are constrained by optimal value c such that $a + b = c$

(2) Derive the relation between $\hat{\theta}_i$ and $\hat{\theta}_j$, the i^{th} and j^{th} components of the optimal weight vector obtained by solving the ridge regression optimization problem.

> The ridge regression objective function, below, must minimize both loss and regularization.
>
> $$\sum_{k=1}^{n}(h(x_k) - y_k)^2 + \lambda\|w\|_2^2$$
>
> The loss component is the same as with lasso, so we must minimize the regularization penalty subject to the same constraint as in lasso, which is that $a + b = c$
>
> The regularization penalty due to x_i, x_j is $a^2 + b^2$. Next I will show that $a^2 + b^2 \geq (\frac{c}{2})^2 + (\frac{c}{2})^2$, and therefore a and b must be equal to $\frac{c}{2} = \frac{a+b}{2}$ under these conditions.
>
> Claim: $a^2 + b^2 \geq 2(\frac{c}{2})^2$
>
> Proof:
>
> $$\begin{aligned} a^2 + b^2 &= a^2 + (c-a)^2 \\ &= a^2 + c^2 - 2ac + a^2 \\ &= 2a^2 + c^2 - 2ac \\ &= \tfrac{1}{2}(4a^2 - 4ac + 2c^2) \\ &= \tfrac{1}{2}(2a-c)^2 + \tfrac{c^2}{2} \\ &= \tfrac{1}{2}(2a-c)^2 + 2(\tfrac{c}{2})^2 \\ &= \tfrac{1}{2}(2a-c)^2 \geq 0 \end{aligned}$$
>
> therefore $\frac{1}{2}(2a-c)^2 + 2(\frac{c}{2})^2 \geq 2(\frac{c}{2})^2$
>
> Thus a and b must be equal.

4 The Ellipsoids in the ℓ_1/ℓ_2 regularization picture

(1) Let $\hat{w} = (X^T X)^{-1} X^T y$. Show that $\hat{w}$ has empirical risk given by

$$\hat{R}_n(\hat{w}) = \frac{1}{n}(-y^T X \hat{w} + y^T y)$$

ANSWER

$$\begin{aligned}
\hat{R}_n(\hat{w}) &= \tfrac{1}{n}(Xw - y)^T(Xw - y) \\
&= \tfrac{1}{n}(w^T X^T X w - 2w^T X^T y + y^T y) \\
&= \tfrac{1}{n}[w^T X^T (Xw - 2y) + y^T y] \\
&= \tfrac{1}{n}[w^T X^T (-2y + X(X^T X)^{-1} X^T y) + y^T y] \\
&= \tfrac{1}{n}[-w^T X^T y + y^T y] \\
&= \tfrac{1}{n}\{-[(X^T X)^{-1} X^T y]^T X^T y + y^T y\} \\
&= \tfrac{1}{n}[-y^T X[(X^T X)^{-1} X^T y] + y^T y] \\
&= \tfrac{1}{n}[-y^T X \hat{w} + y^T y]
\end{aligned}$$

(2) Show that for any w we have

$$\hat{R}_n(w) = \frac{1}{n}(w - \hat{w})^T X^T X (w - \hat{w}) + \hat{R}_n(\hat{w})$$

ANSWER

$$\begin{aligned}
\hat{R}_n(w) &= \tfrac{1}{n}(Xw - y)^T(Xw - y) \\
&= \tfrac{1}{n}[w^T (X^T X) w - 2(X^T y)^T w + y^T y] \\
&= \tfrac{1}{n}\{[w - (X^T X)^{-1} X^T y]^T (X^T X)[w - (X^T X)^{-1} X^T y] - (X^T y)^T (X^T X)^{-1} X^T y + y^T \\
&= \tfrac{1}{n}[(w - \hat{w})^T (X^T X)(w - \hat{w}) - y^T X \hat{w} + y^T y] \\
&= \tfrac{1}{n}(w - \hat{w})^T (X^T X)(w - \hat{w}) + \hat{R}_n(\hat{w})
\end{aligned}$$

(3) Using the expression in (2), give a very short proof that $\hat{w} = (X^T X)^{-1} X^T y$ is the empirical risk minimizer. That is:

$$\hat{w} = \underset{w}{\operatorname{argmin}} \hat{R}_n(w)$$

ANSWER

$X^T X$ is positive semidefinite

$\therefore \forall (w - \hat{w}) \in \mathbf{R}^d,$

$\Phi(w) = (w - \hat{w})^T X^T X (w - \hat{w}) \geq 0$

Hence $\hat{w}$ is the minimizer of $\Phi(w)$

$\hat{R}_n(w) = \frac{1}{n}\Phi(w) + \hat{R}_n(\hat{w}) \geq \hat{R}_n(\hat{w})$

Therefore, $\hat{w}$ is also the minimizer fo the empirical risk $\hat{R}_n(w)$

(4) Give an expression for the set of w for which the empirical risk exceeds the minimum empirical risk $\hat{R}_n(\hat{w})$ by an amount $c > 0$. This set is an ellipse - what is its center?

ANSWER

$\{w \quad | \quad (w - \hat{w})^T X^T X (w - \hat{w}) = c, c > 0\}$ is an ellipsoid centered at $\hat{w}$

5 Projected SGD via Variable Splitting

(1) Implement projected SGD to solve the above optimization problem for the same value λ's as used with the shooting algorithm.

In [19]:
```
def compute_sgd_loss(X, y, w_p, w_n, Lambda):
    m = y.size
    hX = (w_p - w_n).dot(X.T)
    #loss = ((np.square(hX - y))/(m)).sum() + Lambda*((w_p+w_n).sum
())
    loss = ((np.square(hX - y))/(2*m)).sum()
    return loss
```

In [20]:
```
def compute_sgd_gradient(X, y, i, w_p, w_n, Lambda, component):
    if component is 'p':
        grad = (X[i].dot(w_p-w_n)-y[i])*(X[i]) + Lambda
    elif component is 'n':
        grad = -(X[i].dot(w_p-w_n)-y[i])*(X[i]) + Lambda
    return grad
```

```
In [21]: def projected_sgd(X,y,validation_X, validation_y, Lambda=0.3, alpha
=0.1, beta=0.1, num_iter=5000):
    m,d = X.shape
    w = np.random.rand(d,1) #initialize weight
    w_p = np.zeros(d)
    w_n = np.zeros(d)
    for i in range(d):
        if w[i] > 0:
            w_p[i] = w[i]
        else:
            w_n[i] = w[i]

    opt_loss=1000
    opt_w = w
    step=0
    epoch=0

    order = range(m)

    for j in range(num_iter + 1):
        np.random.shuffle(order)

        for k in range(m):
            loss = compute_sgd_loss(validation_X, validation_y, w_
p, w_n, Lambda)
            grad_p = compute_sgd_gradient(X, y, order[k], w_p, w_n,
Lambda, component='p')
            grad_n = compute_sgd_gradient(X, y, order[k], w_p, w_n,
Lambda, component='n')

            if loss < opt_loss:
                opt_loss = loss
                opt_w = w_p - w_n
                step = j*m+k+1
                epoch= j+1

            if alpha is "1/t":
                alpha=beta*(1./(j*m+k+1))
            elif alpha is "1/sqrt(t)":
                alpha=beta*(1./np.sqrt(j*m+k+1))
            else:
                alpha=alpha

            w_p = w_p - alpha*grad_p
            w_n = w_n - alpha*grad_n

            for i in range(d):
                if w_p[i] < 0:
                    w_p[i] = 0
                elif w_n[i] < 0:
                    w_n[i] = 0
    return opt_loss, opt_w, w_p, w_n
```

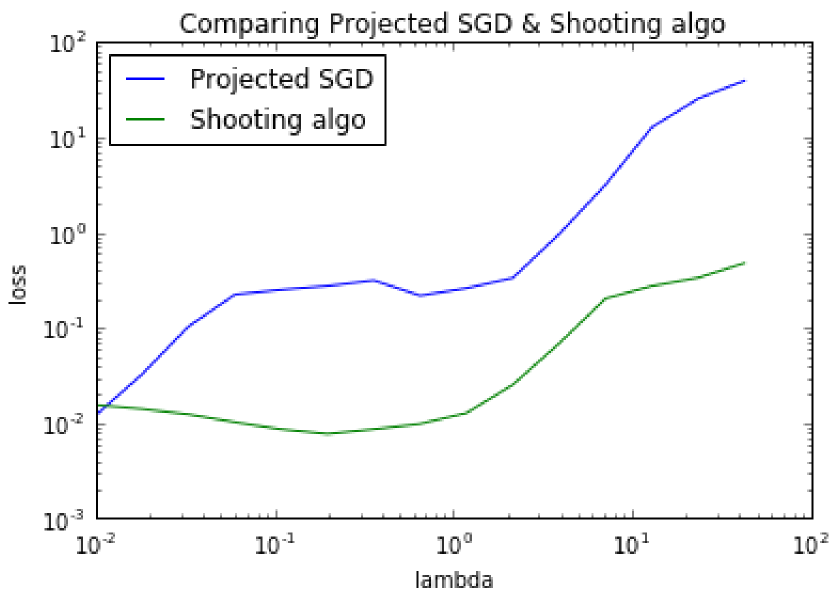
In [22]:

```
loss = []
w = []
loss1=[]
w1=[]
Lambda_max = np.linalg.norm(np.dot(X_train.T,y_train),np.inf)
log_lambda_max = np.log10(Lambda_max)
print "Start Searching"

for i in np.linspace( -2,log_lambda_max, 15):
    Lambda = 10**i;
    opt_loss, opt_w, w_p, w_n = projected_sgd(X_train,y_train,X_val
idation, y_validation, Lambda=Lambda, alpha=0.005, num_iter=5000)
    opt_w1, converged = lasso_shooting(X_train, y_train, Lambda)
    opt_loss1 = compute_loss(X_validation, y_validation, opt_w1)
    loss.append(opt_loss)
    w.append(opt_w)
    w1.append(opt_w1)
    loss1.append(opt_loss1)
```

```
Start Searching
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Converged: False Iterations: 1000
Converged: True Iterations: 587
Converged: True Iterations: 310
Converged: True Iterations: 213
Converged: True Iterations: 178
Converged: True Iterations: 157
Converged: True Iterations: 129
Converged: True Iterations: 52
Converged: True Iterations: 47
Converged: True Iterations: 22
```

In [23]:
```
plt.plot([10**i for i in np.linspace( -2,log_lambda_max, 15)], los
s, label="Projected SGD")
plt.plot([10**i for i in np.linspace( -2,log_lambda_max, 15)], loss
1, label="Shooting algo")
plt.gca().set_xscale("log")
plt.gca().set_yscale("log")
plt.legend(loc="best")
plt.xlabel('lambda')
plt.ylabel('loss')
plt.title('Comparing Projected SGD & Shooting algo')
```

Out[23]: <matplotlib.text.Text at 0x115dc8910>



In [24]:
```
#Analyze the sparsity of the projected SGD solution
loss_array = np.array(loss)
indx = np.where(loss_array ==loss_array.min())[0][0]
w_opt = w[indx]
### Coefficient Report under tolerence=0.05 ###
tolerance = 0.05
false_nonzeros = sum((w_true==0) & (abs(w_opt)>=tolerance))
false_zeros = sum((w_true!=0) & (abs(w_opt)<tolerance))
true_nonzeros = sum((w_true!=0) & (abs(w_opt)>=tolerance))
true_zeros = sum((w_true==0) & (abs(w_opt)<tolerance))
print "With tolerence of {0}".format(tolerance)
print "True Value Zero estimated as Nonzero: {0}".format(false_nonz
eros)
print "True Value Nonzero estimated as Zero: {0}".format(false_zero
s)
print "True Value Zero estimated as Zero: {0}".format(true_zeros)
print "True Value Nonzero estimated as Nonzero: {0}".format(true_no
nzeros)
```

```
With tolerence of 0.05
True Value Zero estimated as Nonzero: 0
True Value Nonzero estimated as Zero: 0
True Value Zero estimated as Zero: 65
True Value Nonzero estimated as Nonzero: 10
```

In [25]:
```
#Analyze the sparsity of the projected SGD solution
loss_array = np.array(loss1)
indx = np.where(loss_array ==loss_array.min())[0][0]
w_opt = w1[indx]
### Coefficient Report under tolerence=0.05 ###
tolerance = 0.05
false_nonzeros = sum((w_true==0) & (abs(w_opt)>=tolerance))
false_zeros = sum((w_true!=0) & (abs(w_opt)<tolerance))
true_nonzeros = sum((w_true!=0) & (abs(w_opt)>=tolerance))
true_zeros = sum((w_true==0) & (abs(w_opt)<tolerance))
print "With tolerence of {0}".format(tolerance)
print "True Value Zero estimated as Nonzero: {0}".format(false_nonz
eros)
print "True Value Nonzero estimated as Zero: {0}".format(false_zero
s)
print "True Value Zero estimated as Zero: {0}".format(true_zeros)
print "True Value Nonzero estimated as Nonzero: {0}".format(true_no
nzeros)
```

```
With tolerence of 0.05
True Value Zero estimated as Nonzero: 5
True Value Nonzero estimated as Zero: 0
True Value Zero estimated as Zero: 60
True Value Nonzero estimated as Nonzero: 10
```

Compare Projected SGD with Lasso Shooting, the Projected SGD solution has more sparsity.