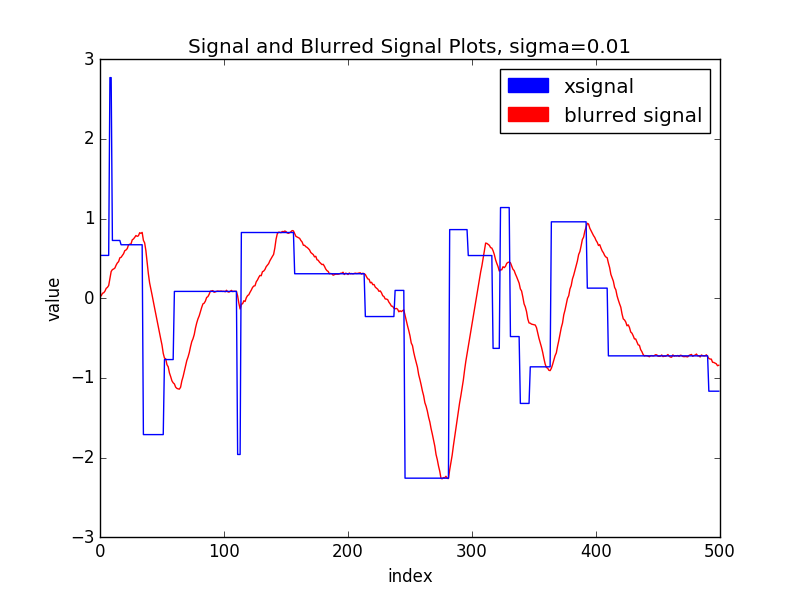
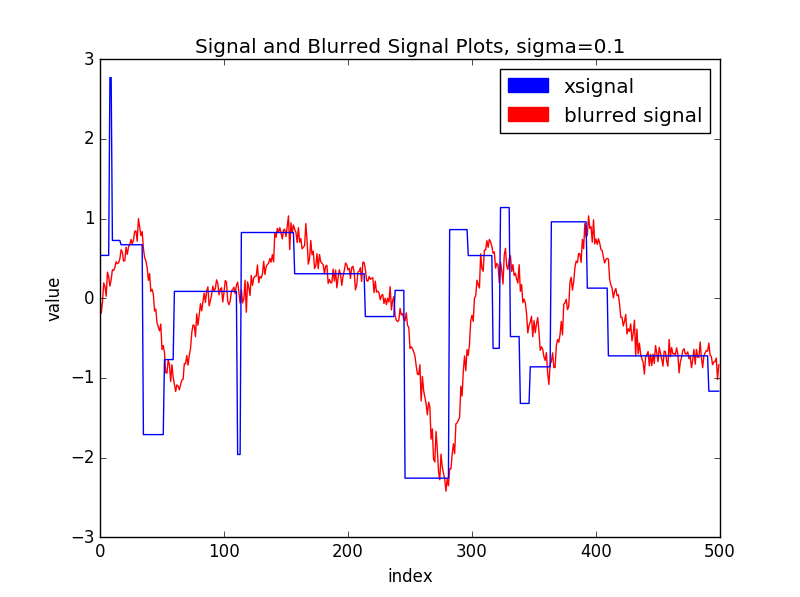
HW6 Part 3 Answers

# Part A Ans

**'''  
bi = (1/k)(xi + xi-1 + xi-2 + ... + xi-k+1) + wi is a blur equation.  
This equation makes a matrix A (nxn) for b = Ax + w.  
'''  
def** generateBlurMatrix(n, k):  
 A = np.zeros((n,n))  
  
 **for** row **in** range(0, n):  
 kind = row  
 **for** writeIndex **in** range(kind, row - k, -1):  
 **if** writeIndex < 0:  
 **break  
 else**:  
 A[row, writeIndex] = 1  
  
 **return** (1/float(k))\*A

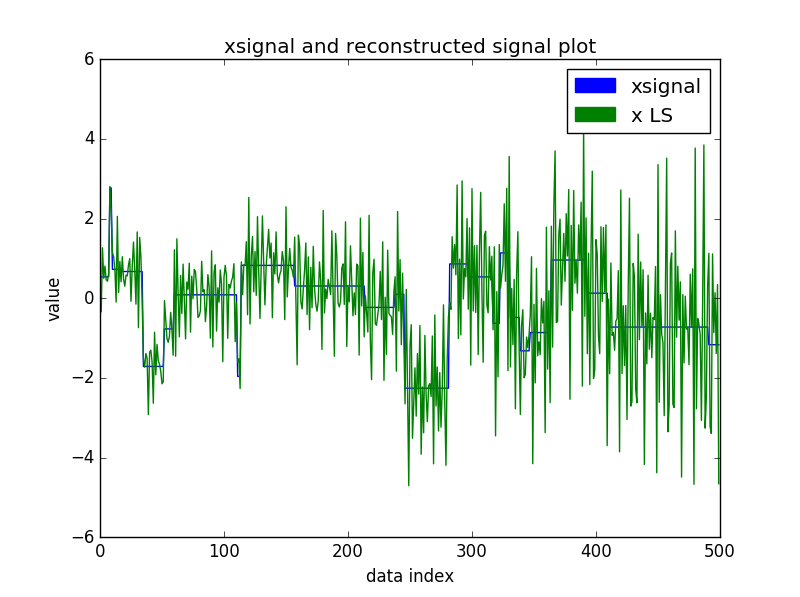
# Part B Ans



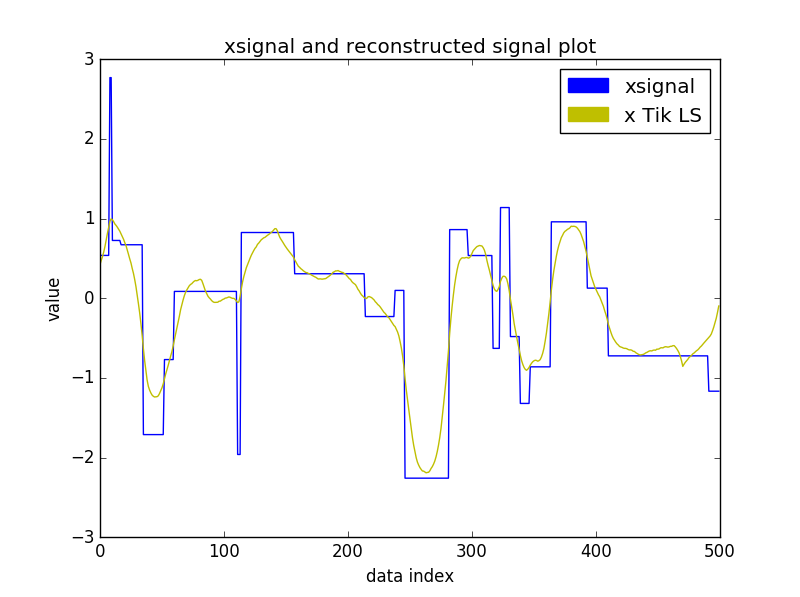


# Part C Ans

sigma = 0.01

Least Squares Plot

Truncated SVD Plot

Tikhonov Regularization Plot

# Part D Ans

As the noise level increases, the best lambda regularization parameter for the model also increases in value. As the noise level increases, the best number of singular values to use decreases. As the blurring level k increases, the best lambda regularization parameter for the model decreases. As the blurring level k increases, the best number of singular values to use doesn’t change all that much.

# Full Code

**import** numpy **as** np  
**import** numpy.linalg **as** linalg  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.patches **as** mpatches  
**import** math  
**import** random **as** rand  
**import** scipy **as** sp  
  
  
  
**def** main():  
 **global** xsignal  
 xsignal = np.loadtxt(**'./xsignal.csv'**, delimiter=**','**)  
 xsignal = xsignal.reshape((xsignal.size, 1))  
  
 *# Part a* A = generateBlurMatrix(xsignal.shape[0], 30)  
 *# Part b* b0 = partb(xsignal, A, 0.01, plot=**False**)  
 b1 = partb(xsignal, A, .1, plot=**False**)  
  
 *# Part c* partc(xsignal, A, b0, msing=30, lambd=.1)  
 partc(xsignal, A, b1, msing=20, lambd=.05)  
  
  
**def** partc(xsignal, A, b, msing, lambd, plot=**True**):  
 xLS = 0  
 xSVDLS = 0  
 xLSReg = 0  
  
 *# Least Squares Solution.* U, S, V = np.linalg.svd(A, full\_matrices=**False**)  
 xLS = V.T.dot(np.linalg.inv(np.diag(S))).dot(U.T).dot(b)*#np.linalg.inv(A.T.dot(A)).dot(A.T).dot(b)  
  
 # SVD Reduced Least Squares Solution.* Sr = np.diag(S[:msing])  
 *# Use the pseudo inverse equation from the reduced matrix and calculate the corresponding x.* xSVDLS = V.T[:, :msing].dot(np.linalg.inv(Sr)).dot(U.T[:msing, :]).dot(b)  
  
 *# Least Squares with Tikhonov Regularization.* B = A.T.dot(A) + (lambd\*np.eye(xsignal.shape[0]))  
 xLSReg = np.linalg.inv(B).dot(A.T).dot(b)  
  
 **if** plot:  
 plt.plot(list(range(0, xsignal.shape[0])), xsignal, color=**'b'**)  
 plt.plot(list(range(0, xLS.shape[0])), xLS, color=**'g'**)  
 blue\_patch = mpatches.Patch(color=**'b'**, label=**'xsignal'**)  
 green\_patch = mpatches.Patch(color=**'g'**, label=**'x LS'**)  
 plt.legend(handles=[blue\_patch, green\_patch])  
 plt.title(**'xsignal and reconstructed signal plot'**)  
 plt.xlabel(**'data index'**)  
 plt.ylabel(**'value'**)  
 plt.show()  
  
 plt.plot(list(range(0, xsignal.shape[0])), xsignal, color=**'b'**)  
 plt.plot(list(range(0, xSVDLS.shape[0])), xSVDLS, color=**'r'**)  
 blue\_patch = mpatches.Patch(color=**'b'**, label=**'xsignal'**)  
 red\_patch = mpatches.Patch(color=**'r'**, label=**'x SVD LS'**)  
 plt.legend(handles=[blue\_patch, red\_patch])  
 plt.title(**'xsignal and reconstructed signal plot'**)  
 plt.xlabel(**'data index'**)  
 plt.ylabel(**'value'**)  
 plt.show()  
  
 plt.plot(list(range(0, xsignal.shape[0])), xsignal, color=**'b'**)  
 plt.plot(list(range(0, xLSReg.shape[0])), xLSReg, color=**'y'**)  
 blue\_patch = mpatches.Patch(color=**'b'**, label=**'xsignal'**)  
 yellow\_patch = mpatches.Patch(color=**'y'**, label=**'x Tik LS'**)  
 plt.legend(handles=[blue\_patch, yellow\_patch])  
 plt.title(**'xsignal and reconstructed signal plot'**)  
 plt.xlabel(**'data index'**)  
 plt.ylabel(**'value'**)  
 plt.show()  
  
**def** partb(xsignal, A, msigma, plot=**False**):  
 *# Part b* w = np.ones((xsignal.shape[0], 1))  
 **for** wi **in** range(0, w.shape[0]):  
 w[wi] \*= rand.normalvariate(0, sigma=msigma)  
 *# Calculate the blur.* b = A.dot(xsignal) + w  
 *# Plots for msigma = 0.01 and 0.1.* **if** plot:  
 plt.plot(list(range(0, b.shape[0])), b, color=**'r'**)  
 plt.plot(list(range(0, xsignal.shape[0])), xsignal, color=**'b'**)  
 blue\_patch = mpatches.Patch(color=**'b'**, label=**'xsignal'**)  
 red\_patch = mpatches.Patch(color=**'r'**, label=**'blurred signal'**)  
 plt.legend(handles=[blue\_patch, red\_patch])  
 plt.xlabel(**'index'**)  
 plt.ylabel(**'value'**)  
 plt.title(**'Signal and Blurred Signal Plots, sigma=%s'** % str(msigma))  
 plt.show()  
  
 **return** b  
  
**'''  
bi = (1/k)(xi + xi-1 + xi-2 + ... + xi-k+1) + wi is a blur equation.  
This equation makes a matrix A (nxn) for b = Ax + w.  
'''  
def** generateBlurMatrix(n, k):  
 A = np.zeros((n,n))  
  
 **for** row **in** range(0, n):  
 kind = row  
 **for** writeIndex **in** range(kind, row - k, -1):  
 **if** writeIndex < 0:  
 **break  
 else**:  
 A[row, writeIndex] = 1  
  
 **return** (1/float(k))\*A  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()