EECS 16A HW14

Bryan Ngo

2019-12-07

1 How Much is Too Much?

1.a

Yes, the higher-degree polynomials are affected by noise. Around degree 15, the best fit polynomial starts to deviate heavily from the set.

1.b

According to the graph, it is beneficial to pick polynomial fits greater than first-order. However, this is clearly *not* the trend that is indicated in the data set, even if a degree 15 polynomial is the most beneficial.

2 OMP Exercise

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}}_{\boldsymbol{M}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\boldsymbol{x}} \approx \underbrace{\begin{bmatrix} 4 \\ 6 \\ 3 \end{bmatrix}}_{\boldsymbol{b}} \tag{1}$$

Finding the greatest inner product,

i	$\langle \boldsymbol{m}_i, \boldsymbol{b} \rangle$
1	10
2	7
3	-3
4	-1

we see that $\boldsymbol{m}_1$ has the largest inner product. The rejection of $\boldsymbol{b}$ onto $\boldsymbol{m}_1$ is now

$$\boldsymbol{b}' = \boldsymbol{b} - \frac{\langle \boldsymbol{m}_1, \boldsymbol{b} \rangle}{\langle \boldsymbol{m}_1, \boldsymbol{m}_1 \rangle} \boldsymbol{m}_1 = \begin{bmatrix} 4 \\ 6 \\ 3 \end{bmatrix} - 5 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix} \tag{2}$$

Finding the largest inner product with $\boldsymbol{b}'$,

i	$\langle \boldsymbol{m}_i, \boldsymbol{b}' \rangle$
1	0
2	2
3	2
4	4

Using least squares to find x_1, x_4,

$$[\boldsymbol{A}]^\top \boldsymbol{A} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \overset{\boldsymbol{A}^{-1}}{\Rightarrow} \frac{1}{3} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{3}$$

$$[\boldsymbol{A}]^\top \boldsymbol{b} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 10 \\ -1 \end{bmatrix} \tag{4}$$

$$\hat{\boldsymbol{x}} = \frac{1}{3} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 10 \\ -1 \end{bmatrix} = \begin{bmatrix} 19/3 \\ 8/3 \end{bmatrix} \tag{5}$$

$$\boldsymbol{x} = \begin{bmatrix} 19/3 \\ 0 \\ 0 \\ 8/3 \end{bmatrix} \tag{6}$$

3 Greedy Algorithm for Calculating Matrix Eigenvalues

3.a

$$\boldsymbol{Q} = [\boldsymbol{A}]^\top \boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 5 & 11 \\ 11 & 25 \end{bmatrix} = [\boldsymbol{Q}]^\top \tag{7}$$

Theorem 1. *For some matrix $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$, $\boldsymbol{Q} = [\boldsymbol{A}]^\top \boldsymbol{A}$ is symmetric.*

Proof.

$$\boldsymbol{Q} = [\boldsymbol{A}]^\top \boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} = \begin{bmatrix} a_{11}^2 + a_{12}^2 & a_{11}a_{21} + a_{12}a_{22} \\ a_{21}a_{11} + a_{22}a_{12} & a_{21}^2 + a_{22}^2 \end{bmatrix} \tag{8}$$

By inspection, it is clear that the off-diagonal entries are equal, so $[\boldsymbol{Q}]^\top = \boldsymbol{Q}$. □

3.b

Theorem 2. *For some matrix $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \in \mathbb{R}^{N\times N}$ such that $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = 0, i \neq j$, then $[\boldsymbol{V}]^\top \boldsymbol{V} = \boldsymbol{I}$.*

Proof.

$$[\boldsymbol{V}]^\top \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} = \begin{bmatrix} \langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_1, \boldsymbol{v}_N \rangle \\ \langle \boldsymbol{v}_2, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_2, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_2, \boldsymbol{v}_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{v}_N, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_N, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_N, \boldsymbol{v}_N \rangle \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} \|\boldsymbol{v}_1\|^2 & \langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_1, \boldsymbol{v}_N \rangle \\ \langle \boldsymbol{v}_2, \boldsymbol{v}_1 \rangle & \|\boldsymbol{v}_2\|^2 & \cdots & \langle \boldsymbol{v}_2, \boldsymbol{v}_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{v}_N, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_N, \boldsymbol{v}_2 \rangle & \cdots & \|\boldsymbol{v}_N\|^2 \end{bmatrix} \tag{10}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \boldsymbol{I} \tag{11}$$

□

3.c

Proof. In order to prove col($\boldsymbol{V}$) is a basis on $\mathbb{R}^N$, we must prove

- linear independence
- full span of $\mathbb{R}^N$

If col($\boldsymbol{V}$) were to be linearly independent, then null($\boldsymbol{V}$) = $\{\boldsymbol{0}\}$. By definition of the nullspace, there is some vector $\boldsymbol{x}$ such that

$$\boldsymbol{V}\boldsymbol{x} = \boldsymbol{0} \tag{12}$$

$$[\boldsymbol{V}]^\top \boldsymbol{V}\boldsymbol{x} = [\boldsymbol{V}]^\top \boldsymbol{0} \tag{13}$$

$$\boldsymbol{I}\boldsymbol{x} = \boldsymbol{x} = \boldsymbol{0} \tag{14}$$

where we use the result from **3.b**.
In order to prove full span of $\mathbb{R}^N$, any vector must be represented as a linear combination of col($\boldsymbol{V}$). Again, suppose some $\boldsymbol{x}$ such that

$$\boldsymbol{V}\boldsymbol{x} = \boldsymbol{y} \tag{15}$$

Since we proved earlier that col($\boldsymbol{V}$) has linearly independent columns, $\boldsymbol{V}$ is invertible. This means that for any $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{x} = \boldsymbol{V}^{-1}\boldsymbol{y}$. □

3.d

$$\langle \boldsymbol{v}_i, \boldsymbol{b} \rangle = [\boldsymbol{v}_i]^\top (\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_N \boldsymbol{v}_N) \tag{16}$$

$$= \alpha_1 \cancel{[\boldsymbol{v}_i]^\top \boldsymbol{v}_1} + \alpha_2 \cancel{[\boldsymbol{v}_i]^\top \boldsymbol{v}_2} + \cdots + \alpha_i [\boldsymbol{v}_i]^\top \boldsymbol{v}_i + \cdots + \alpha_N \cancel{[\boldsymbol{v}_i]^\top \boldsymbol{v}_N} \tag{17}$$

$$= \alpha_i \|\boldsymbol{v}_i\|^2 = \alpha_i \tag{18}$$

3.e

The key here is representing $\boldsymbol{b}$ as

$$\boldsymbol{b} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_N \boldsymbol{v}_N = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = \boldsymbol{V}\boldsymbol{\alpha} \tag{19}$$

The least squares solution is

$$[\boldsymbol{V}]^\top \boldsymbol{V} = \boldsymbol{I} \overset{\boldsymbol{V}^{-1}}{\Rightarrow} \boldsymbol{I} \tag{20}$$

$$[\boldsymbol{V}]^\top \boldsymbol{b} = [\boldsymbol{V}]^\top \boldsymbol{V}\boldsymbol{\alpha} = \boldsymbol{I}\boldsymbol{\alpha} = \boldsymbol{\alpha} \tag{21}$$

$$\hat{\boldsymbol{x}} = ([\boldsymbol{V}]^\top \boldsymbol{V})^{-1} [\boldsymbol{V}]^\top \boldsymbol{b} = \boldsymbol{I}\boldsymbol{\alpha} = \boldsymbol{\alpha} \tag{22}$$

$$\|\boldsymbol{V}\hat{\boldsymbol{x}} - \boldsymbol{b}\| = \|\boldsymbol{V}\boldsymbol{\alpha} - \boldsymbol{b}\| = 0 \tag{23}$$

3.f

$$[\boldsymbol{V}_2]^\top \boldsymbol{V}_2 = \begin{bmatrix} \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} = \begin{bmatrix} \langle \boldsymbol{v}_2, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_2, \boldsymbol{v}_N \rangle \\ \vdots & \ddots & \vdots \\ \langle \boldsymbol{v}_N, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_N, \boldsymbol{v}_N \rangle \end{bmatrix} = \boldsymbol{I}_{N-1} \tag{24}$$

$$[\boldsymbol{V}_2]^\top \boldsymbol{b} = \begin{bmatrix} \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} \langle \boldsymbol{v}_2, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_2, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_2, \boldsymbol{v}_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{v}_N, \boldsymbol{v}_1 \rangle & \langle \boldsymbol{v}_N, \boldsymbol{v}_2 \rangle & \cdots & \langle \boldsymbol{v}_N, \boldsymbol{v}_N \rangle \end{bmatrix} \boldsymbol{\alpha} \tag{25}$$

$$= \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \boldsymbol{\alpha} = \begin{bmatrix} \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} \tag{26}$$

$$\hat{\boldsymbol{x}} = ([\boldsymbol{V}_2]^\top \boldsymbol{V}_2)^{-1} [\boldsymbol{V}_2]^\top \boldsymbol{b} = \boldsymbol{I}_{N-1} \begin{bmatrix} \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} \tag{27}$$

$$\|\boldsymbol{V}_2 \hat{\boldsymbol{x}} - \boldsymbol{b}\| = \left\| \begin{bmatrix} \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} - \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} \right\| \tag{28}$$

$$= \|(\alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_N) - (\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_N)\| = \| - \alpha_1 \boldsymbol{v}_1 \| = \alpha_1 \tag{29}$$

3.g

Step 1 is justified since

$$[\boldsymbol{V}]^\top \boldsymbol{v}_1 = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_N \end{bmatrix} \boldsymbol{v}_1 = \begin{bmatrix} \langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle \\ \langle \boldsymbol{v}_2, \boldsymbol{v}_1 \rangle \\ \vdots \\ \langle \boldsymbol{v}_N, \boldsymbol{v}_1 \rangle \end{bmatrix} = \begin{bmatrix} \|\boldsymbol{v}_1\|^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{30}$$

Step 2 is justified since

$$\begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{31}$$

by simple matrix-vector multiplication. Step 3 is justified since

$$\begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_N \end{bmatrix} \begin{bmatrix} \lambda_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \lambda_1 \boldsymbol{v}_1 \tag{32}$$

by simple matrix-vector multiplication.

3.h

Theorem 3. $\boldsymbol{v}_1 \in \text{null}(\boldsymbol{Q}_2)$ *where* $\boldsymbol{Q} - \lambda_1 \boldsymbol{v}_1 [\boldsymbol{v}_1]^\top$.

Proof.

$$\begin{aligned} \boldsymbol{Q}_2 \boldsymbol{v}_1 &= (\boldsymbol{Q} - \lambda_1 \boldsymbol{v}_1 [\boldsymbol{v}_1]^\top) \boldsymbol{v}_1 &(33)\\ &= \boldsymbol{Q}\boldsymbol{v}_1 - \lambda_1 \boldsymbol{v}_1 [\boldsymbol{v}_1]^\top \boldsymbol{v}_1 &(34)\\ &= \lambda_1 \boldsymbol{v}_1 - \lambda_1 \boldsymbol{v}_1 \langle \boldsymbol{v}_1, \boldsymbol{v}_1 \rangle &(35)\\ &= \lambda_1 \boldsymbol{v}_1 - \lambda_1 \boldsymbol{v}_1 = \boldsymbol{0} &(36) \end{aligned}$$

□

Theorem 4. $\boldsymbol{v}_2, \ldots, \boldsymbol{v}_N$ *are the eigenvectors of* $\boldsymbol{Q}_2$.

Proof.

$$\begin{aligned} \boldsymbol{Q}_2 &= \boldsymbol{Q} - \lambda_1 \boldsymbol{v}_1 [\boldsymbol{v}_1]^\top &(37)\\ &= \sum_{i=1}^{N} \lambda_i \boldsymbol{v}_i [\boldsymbol{v}_i]^\top - \lambda_1 \boldsymbol{v}_1 [\boldsymbol{v}_1]^\top &(38)\\ &= \sum_{i=2}^{N} \lambda_i \boldsymbol{v}_i [\boldsymbol{v}_i]^\top &(39) \end{aligned}$$

Note that this is simply the definition of $\boldsymbol{Q}$ shifted one index up. This indicates that all λ_i for $i \in \{2, \ldots, N\}$ are eigenvalues of $\boldsymbol{Q}_2$. Using the proof in **3.g**, it is elementary to prove that $\boldsymbol{v}_i$ are the eigenvectors associated with the eigenvalues, respectively. □

3.i

1. Perform $f(\boldsymbol{Q})$. Put the eigenvalue in the list.
2. Let $\boldsymbol{Q} = \boldsymbol{Q} - \lambda_{max} \boldsymbol{v}_{max} [\boldsymbol{v}_{max}]^\top$. This step effectively ”removes” the largest eigenvalue so far.
3. Repeat Step 1 until there are no eigenvalues left in $\boldsymbol{Q}$.

4 Sparse Imaging

4.a

See Jupyter Notebook.

4.b

The image is the Cal logo.[1]

4.c

The algorithm fails to recover sparse images once the number of measurements approaches the sparsity. This is because least squares becomes less and less useful with a more and more square matrix.

5 Trolls Revisited

5.a

No.

5.b

$$\alpha = \frac{\langle \boldsymbol{l}, \boldsymbol{r} \rangle}{\|\boldsymbol{l}\|^2} \tag{40}$$

$$\boldsymbol{n} = \boldsymbol{r} - \alpha \boldsymbol{l} \tag{41}$$

5.c

We have 1 million equations to solve in this system. Since the system is overdetermined, we can simply find the least-squares solution, in effect projecting $\boldsymbol{r}$ onto $\boldsymbol{n}_i$. The resultant ”lecture” is still noisy.

5.d

The algorithm finally works, and Prof. Ranade is discussing the potential uninvertibility of $[\boldsymbol{A}]^\top \boldsymbol{A}$.

7 Homework Process and Study Group

I worked on this homework by myself.

[1] Go Bears!

EECS16A Homework 14

Question 1: How Much Is Too Much?

Some Setup Code

You do not need to understand how the following code works.

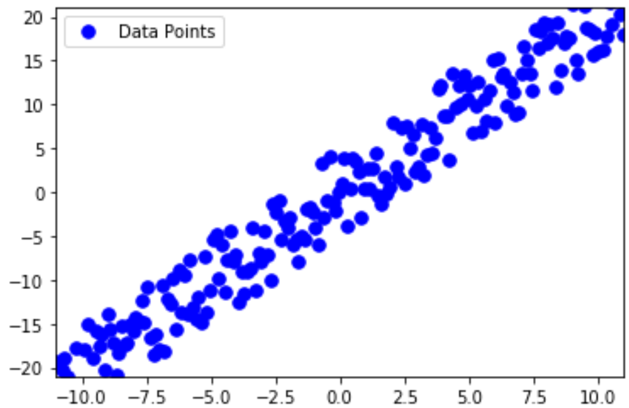
In [1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

%matplotlib inline

"""Function that constructs a polynomial curve for a set of
coefficients that multiply the polynomial terms and the x range."""
def poly_curve(params,x_input):
    # params contains the coefficients that multiply the polynomial terms, in d
egree of lowest degree to highest degree
    degree=len(params)-1
    x_range=[x_input[1], x_input[-1]]
    x=np.linspace(x_range[0],x_range[1],1000)
    y=x*0

    for k in range(0,degree+1):
        coeff=params[k]
        y=y+list(map(lambda z:coeff*z**k,x))
    return x,y

"""Function that defines a data matrix for some input data."""
def data_matrix(input_data,degree):
    # degree is the degree of the polynomial you plan to fit the data with
    Data=np.zeros((len(input_data),degree+1))

    for k in range(0,degree+1):
        Data[:,k]=(list(map(lambda x:x**k ,input_data)))

    return Data

"""Function that computes the Least Squares Approximation"""
def leastSquares(D,y):
    return np.linalg.lstsq(D,y)[0]

np.random.seed(10)
```

Part a)

Some setup code to create our resistor test data points and plot them.

In [2]:
```
R = 2
x_a = np.linspace(-11,11,200)
y_a = R*x_a + (np.random.rand(len(x_a))-0.5)*10
fig = plt.figure()
ax=fig.add_subplot(111,xlim=[-11,11],ylim=[-21,21])
ax.plot(x_a,y_a, '.b', markersize=15)
ax.legend(['Data Points'])
```

Out[2]:
```
<matplotlib.legend.Legend at 0x7f0336cd7ac8>
```

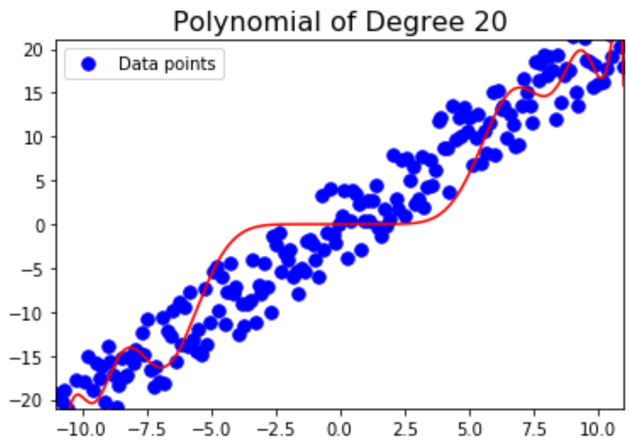


Let's calculate a polynomial approximation of the above device.

In [3]:
```
#Play around with degree here to try and fit different degree polynomials
degree=20 # change the degree here
D_a = data_matrix(x_a,degree)
p_a = leastSquares(D_a, y_a)

fig=plt.figure()
ax=fig.add_subplot(111,xlim=[-11,11],ylim=[-21,21])
x_a_,y_a_=poly_curve(p_a,x_a)
ax.plot(x_a,y_a,'.b',markersize=15)
ax.plot(x_a_, y_a_, 'r')
ax.legend(['Data points'])
plt.title('Polynomial of Degree %d' %(len(p_a)-1),fontsize=16)
```

```
/home/bngo/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:33: Futu
reWarning: `rcond` parameter will change to the default of machine precision ti
mes ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=Non
e`, to keep using the old, explicitly pass `rcond=-1`.
```

Out[3]:
```
Text(0.5, 1.0, 'Polynomial of Degree 20')
```



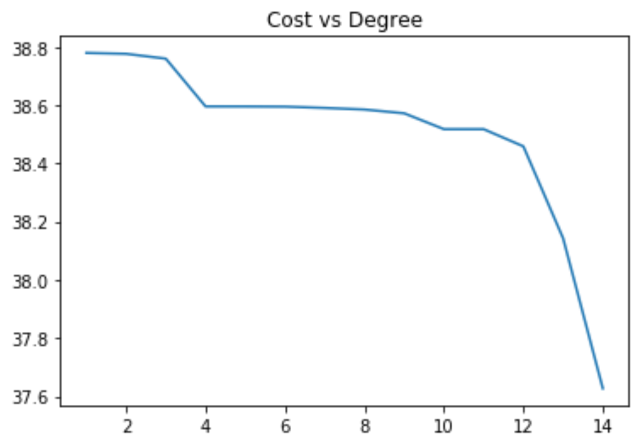
Part b)

In [4]:
```
def cost(x, y, start_deg, end_deg):
    """Given a set of x and y points, and a range of polynomial degrees to try,
    this function calculates polynomial fits to the data for polynomials
    of different degrees. It returns the "cost", i.e. the magnitude of the erro
r vector for each fit.
   The output is an array of the cost corresponding to each degree.
   """
    c = []
    for degree in range(start_deg, end_deg):
        D = data_matrix(x,degree)
        params = leastSquares(D,y)
        error = np.linalg.norm(y-np.dot(D,params))
        c.append(error)
    return c
```

In [5]:
```
start = 1
end = 15
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(range(start, end), cost(x_a,y_a,start,end))
plt.title('Cost vs Degree')
```

```
/home/bngo/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:33: Futu
reWarning: `rcond` parameter will change to the default of machine precision ti
mes ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=Non
e`, to keep using the old, explicitly pass `rcond=-1`.
```

Out[5]:
```
Text(0.5, 1.0, 'Cost vs Degree')
```



Question 4: Sparse Imaging

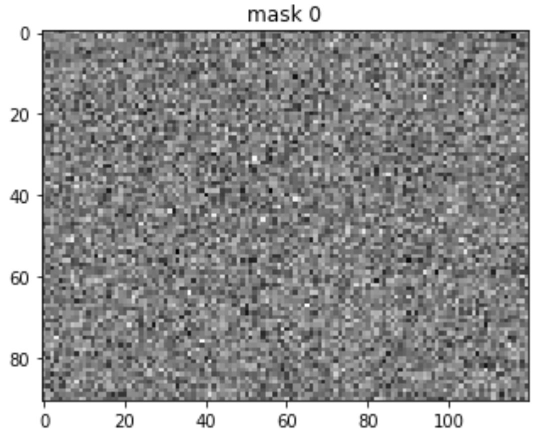
This example tries to reconstruct an image using the Orthogonal Matching Pursuit algorithm.

In [6]:
```
# imports
import matplotlib.pyplot as plt
import numpy as np
from scipy import misc
from IPython import display
import sys
import imageio
%matplotlib inline

def randMasks(numMasks, numPixels):
    randNormalMat = np.random.normal(0,1,(numMasks,numPixels))
    # make the columns zero mean and normalize
    for k in range(numPixels):
        # make zero mean
        randNormalMat[:,k] = randNormalMat[:,k] - np.mean(randNormalMat[:,k])
        # normalize to unit norm
        randNormalMat[:,k] = randNormalMat[:,k] / np.linalg.norm(randNormalMat
[:,k])
    A = randNormalMat.copy()
    Mask = randNormalMat - np.min(randNormalMat)
    return Mask,A

def simulate():
    # read the image in grayscale
    I = np.load('helper.npy')
    sp = np.sum(I)
    numMeasurements = 6500
    numPixels = I.size
    Mask, A = randMasks(numMeasurements,numPixels)
    full_signal = I.reshape((numPixels,1))
    measurements = np.dot(Mask,full_signal)
    measurements = measurements - np.mean(measurements)
    return measurements, A
```

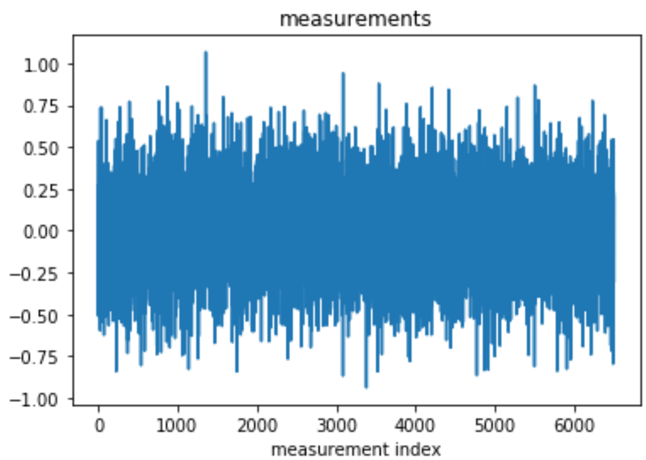
Part (a)

In [7]:
```
measurements, A = simulate()

# THE SETTINGS FOR THE IMAGE - PLEASE DO NOT CHANGE
height = 91
width = 120
sparsity = 476
numPixels = len(A[0])
```

In [8]:
```
# CHOOSE DIFFERENT MASKS TO PLOT
chosenMaskToDisplay = 0

M0 = A[chosenMaskToDisplay].reshape((height,width))
plt.title('mask %d'%chosenMaskToDisplay)
plt.imshow(M0, cmap=plt.cm.gray, interpolation='nearest');
```



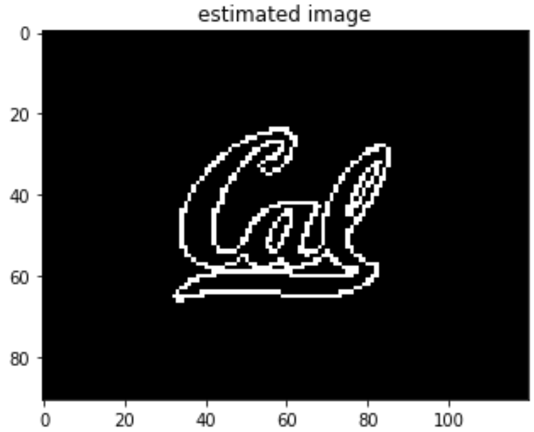
In [9]:
```
# measurements
plt.title('measurements')
plt.plot(measurements)
plt.xlabel('measurement index')
plt.show()
```



In [10]:
```
# OMP algorithm
# THERE ARE MISSING LINES THAT YOU NEED TO FILL
def OMP(imDims, sparsity, measurements, A):
    r = measurements.copy()
    indices = []

    # Threshold to check error. If error is below this value, stop.
    THRESHOLD = 0.1

    # For iterating to recover all signal
    i = 0

    while i < sparsity and np.linalg.norm(r) > THRESHOLD:
       # Calculate the inner products of r with columns of A
        print('%d - '%i,end="",flush=True)
        simvec = A.T.dot(r)

        # Choose pixel location with highest inner product and add to collectio
n
        # COMPLETE THE LINE BELOW
        best_index = np.argmax(np.abs(simvec))
        indices.append(best_index)

        # Build the matrix made up of selected indices so far
        # COMPLETE THE LINE BELOW
        Atrunc = A[:,indices]

        # Find orthogonal projection of measurements to subspace
        # spanned by recovered codewords
        b = measurements
        # COMPLETE THE LINE BELOW
        xhat = np.linalg.lstsq(Atrunc, b)[0]

        # Find component orthogonal to subspace to use for next measurement
        # COMPLETE THE LINE BELOW
        r = b - Atrunc.dot(xhat)

        # This is for viewing the recovery process
        if i % 10 == 0 or i == sparsity-1 or np.linalg.norm(r) <= THRESHOLD:
            recovered_signal = np.zeros(numPixels)
            for j, x in zip(indices, xhat):
                recovered_signal[j] = x
            Ihat = recovered_signal.reshape(imDims)
            plt.title('estimated image')
            plt.imshow(Ihat, cmap=plt.cm.gray, interpolation='nearest')
            display.clear_output(wait=True)
            display.display(plt.gcf())

        i = i + 1

    display.clear_output(wait=True)

    # Fill in the recovered signal
    recovered_signal = np.zeros(numPixels)
    for i, x in zip(indices, xhat):
        recovered_signal[i] = x

    return recovered_signal
```

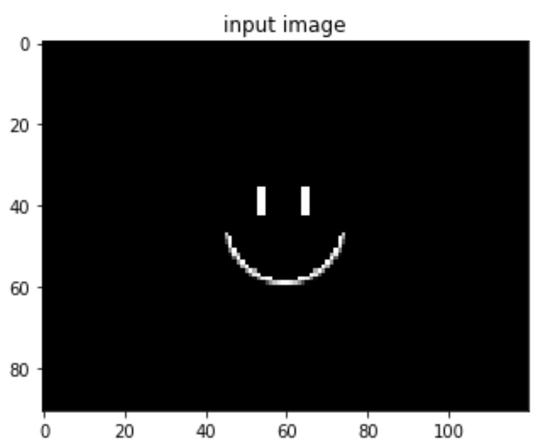
Part (b)

In [11]:
```
rec = OMP((height,width), sparsity, measurements, A)
```



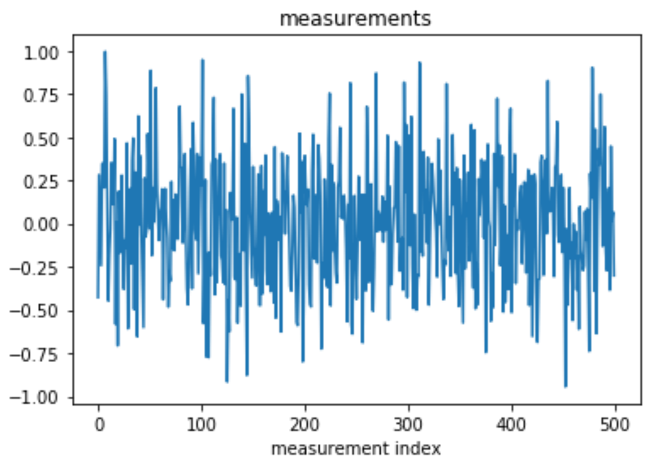
PRACTICE: Part (c)

In [12]:
```
# the setting

# file name for the sparse image
fname = 'figures/smiley.png'
# number of measurements to be taken from the sparse image
numMeasurements = 500
# the sparsity of the image
sparsity = 400

# read the image in black and white
I = imageio.imread(fname, as_gray=True)
# normalize the image to be between 0 and 1
I = I/np.max(I)

# shape of the image
imageShape = I.shape
# number of pixels in the image
numPixels = I.size

plt.title('input image')
plt.imshow(I, cmap=plt.cm.gray, interpolation='nearest');
```

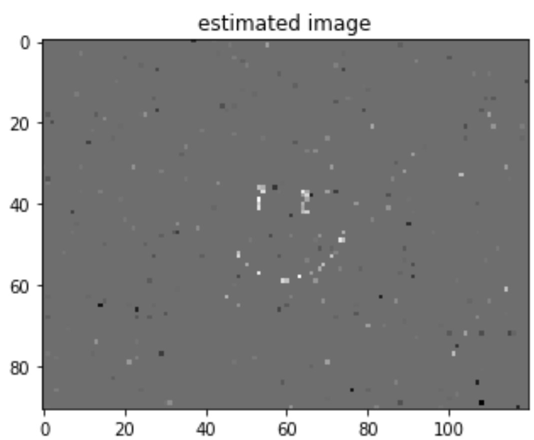


In [13]:
```
# generate your image masks and the underlying measurement matrix
Mask, A = randMasks(numMeasurements,numPixels)
# vectorize your image
full_signal = I.reshape((numPixels,1))
# get the measurements
measurements = np.dot(Mask,full_signal)
# remove the mean from your measurements
measurements = measurements - np.mean(measurements)
```

In [14]:
```
# measurements
plt.title('measurements')
plt.plot(measurements)
plt.xlabel('measurement index')
plt.show()
```



In [15]:
```
smiley = OMP((height, width), sparsity, measurements, A)
```



Question 6: Noise Cancelling Headphones (PRACTICE)

troll

December 8, 2019

0.0.1 Part (a)

Listen to the recording you made, stored in the file `recording.wav`. You can load recordings using the `load_recording` function that we have written for you and imported. You can play recordings using the `play` function that we have also written and imported.

[1]:
```
import numpy as np
from utils import load_recording, play, save_recording

RECORDING_FILE = "recording.wav"

r = load_recording(RECORDING_FILE)
play(r)
```

```
<IPython.lib.display.Audio object>
```

0.0.2 Part (b)

Let $\vec{r}$ be your recording. Let us say you have access to the true lecture given by $\vec{l}$. You know that your received vector and the lecture have the relationship

$$\vec{r} = \alpha\vec{l} + \vec{n},$$

where α is an unknown constant. Estimate $\vec{n}$ by projecting $\vec{r}$ ontol $\vec{l}$ to recover α. What remains is $\vec{n}$. Assume that $\vec{l}$ is orthogonal to $\vec{n}$.

[2]:
```
# Note that l and r are 1D arrays, not 2D arrays, so calling np.linalg.lstsq␣
↪will give an error here. How else can you project one vector onto another?
def projection(l, r):
    # YOUR CODE HERE
    return (np.dot(l, r) / np.dot(l, l)) * l
```

[3]:
```
def recover_noise(r, l):
    return r - projection(l,r)
```

[4]:
```
#We use the technique above to recover candidate interference signals.

#noisy_lectures contains the lecture recordings with interference
```

```
noisy_lectures = [load_recording("noisy_lecture_{}.wav".format(i+1)) for i in␣
↪range(4)]

# lectures contains the clean lectures that you played to understand the␣
↪possible noises
lectures = [load_recording("lecture_{}.wav".format(i+1)) for i in range(4)]

# interferences is a matrix whose columns contain the possible interference␣
↪sequences
interferences = np.column_stack([recover_noise(r_i, l_i) for r_i, l_i in␣
↪zip(noisy_lectures, lectures)])

#you can change the index 0 below to play different lectures and recordings and␣
↪the extracted interferences. There are four of each.
play(lectures[0])
play(noisy_lectures[0])
play(interferences[:, 0])
```

```
<IPython.lib.display.Audio object>

<IPython.lib.display.Audio object>

<IPython.lib.display.Audio object>
```

0.0.3 Part (c)

Now, given $\vec{r}$ and the $\vec{n}_i$, and the model

$$\vec{r} = \vec{l} + \sum_{i=1}^{s} \beta_i \vec{n}_i,$$

use least squares to recover $\vec{l}$. The $\vec{n}_i$ are computed from the $\vec{r}_i$ using your function from the previous part.

[5]:
```
#r is the signal you have recorded
r = load_recording(RECORDING_FILE)

# Project r onto the interference signals to recover the component of r␣
↪explained by the interference.
# What remains must be the lecture.

A = interferences
b = r

# Hint, use least squares
```

```
betas = np.linalg.lstsq(A, b, None)[0]

# This is the recovered lecture. Have you successfully recovered a
# noise-free signal? Or is it still noisy?
l = b - A.dot(betas)

play(l)
```

<IPython.lib.display.Audio object>

0.0.4 Part (d)

Now, we will include the effect of the travel time of the noise signals, using the model

$$\vec{r} = \vec{l} + \sum_{i=1}^{s} \beta_i \vec{n}_i^{(k_i)}.$$

Recover $\vec{l}$ using this new model, using OMP, by filling in the blanks in the below code block.

[6]:
```
from utils import cross_correlate

r = load_recording(RECORDING_FILE)
interferences = [recover_noise(r_i, l_i) for r_i, l_i in zip(noisy_lectures,␣
 ↪lectures)]

k = np.zeros(4, "int")

vecs = []

# the initial residual for OMP
residual = r

for _ in range(4):
    best_corr = float("-inf")
    best_vec = None
    # We first iterate over all the interferences n_i
    for i, n_i in enumerate(interferences):
        # for each interference, we look through its correlation with the␣
 ↪residual at every possible delay

        # Fill in the arguments to cross_correlate
        for k_i, corr in enumerate(cross_correlate(
            residual,
            n_i
        ) # This function returns a vector of cross correlation values of
          # the residual/received signal with every possible delay of the␣
 ↪signatures (interferences in this case)
```

```
        ):
            # we find the (noise, shift) pair that maximizes the correlation␣
↪with the residual
            if corr > best_corr:
                best_corr = corr
                best_vec = (i, k_i)
    i, k_i = best_vec
    k[i] = k_i

    # we shift the best noise by the best shift and add it to our list of␣
↪columns
    vecs.append(np.roll(interferences[i], k[i]))

    A = np.column_stack(vecs) # this is the matrix that captures all the␣
↪interferences we have identified so far

    # Use least squares to update the residual
    residual = r - np.dot(A, np.linalg.lstsq(A, r, None)[0])

l = residual
play(l)
```

```
<IPython.lib.display.Audio object>
```