1. Recipe Reconnaissance

(a) 6; 6

Since $2 \cdot 3 = 6$, so we have 6 unknowns in the problem, and we need 6 linearly independent (assuming consistent) equations.

(b) $D_e = 0.2386$ eggs, $H_e = 0.0491$ eggs, $D_s = 5.3571$ g, $H_s = 10$ g; $D_b = 10$ g, $H_b = 10$ g

Let the six unknowns be $D_e, D_s, D_b, H_e, H_s, H_b$, where D_e represents the number of eggs in each Decadent Dwight (DD) cookie, D_s represents the amount of sugar (in grams) in each DD cookie, and D_b represents the amount of butter (in grams) in each DD cookie; H_e represents the number of eggs in each Heavenly Hearst (HH) cookie, H_s represents the amount of sugar (in grams) in each HH cookie, and H_b represents the amount of butter (in grams) in each HH cookie.

Since the bakery produces 40 Decadent Dwight and 50 Heavenly Hearst cookies each day, so we can calulate that the bakery produces $40 \cdot 7 = 280$ DD cookies and $50 \cdot 7 = 350$ HH cookies each week. Also, a 5-kg bag of sugar would cost $5\,kg \cdot \$5/kg = \$25$, and I'm basing units on dollars whenever dealing with money.

Thus, from Bob the Baker, who buys a dozen (= 12) eggs daily and a 5-kg (= 5000g) bag of sugar weekly, so we have two estimated (approximate) equations:

$$40D_e + 50H_e = 12$$

$$280D_s + 350H_s = 5000$$

Also, the master taster gives us two exact equations:

$$D_b = 10$$

$$H_b = 10$$

Then, with the fact that 12 eggs costs \$2, so each egg costs $\$\frac{1}{6}$, and similarly, each gram of sugar costs $\$\frac{5}{1000} = \$\frac{1}{200}$, each gram of butter costs $\$\frac{1}{100}$, and that each type of cookie are worth about \$0.2 in ingredients, so the approximate prices of each cookie's ingredients and the exact prices of the ingredients gives us two estimated equations:

$$\frac{1}{6}D_e + \frac{1}{200}D_s + \frac{1}{100}D_b = 0.2$$

$$\frac{1}{6}H_e + \frac{1}{200}H_s + \frac{1}{100}H_b = 0.2$$

Finally, by the grapevine info, we have one more approximate equation:

$$H_s = 10$$

Everything's setup as a least-squares problem, and we're estimating 4 variables D_e, H_e, D_s, H_s. Using iPython Notebook, we solve the equations. (For the purpose of iPython Notebook, we plug in the exact values of D_b, H_b to obtain the following two approximate equations: $\frac{1}{6}D_e + \frac{1}{200}D_s = 0.1$, and $\frac{1}{6}H_e + \frac{1}{200}H_s = 0.1$)

The result is:

$$D_e = 0.2386 \text{ eggs}$$

$$H_e = 0.0491 \text{ eggs}$$

$$D_s = 5.3571 \text{ g}$$

$$H_s = 10 \text{ g}$$

with Sum Squared Residuals = 0.002867 and Average error for data points = 0.000573

(c) Less accurate

With the facts that Decadent Dwight is about 25 grams, Heavenly Hearst is about 24 grams, and that 1 egg = 50 grams, so we now have two additional approximate equations:

$$50D_e + D_s + D_b = 25$$

$$50H_e + H_s + H_b = 24$$

Plug in the exactly amount of butter in both cookies, i.e. $D_b = H_b = 10$, we have that $50D_e + D_s = 15$, and $50H_e + H_s = 14$.

Again, using iPython Notebook, we can solve for the result:

$$D_e = 0.1946 \text{ eggs}$$

$$H_e = 0.0822 \text{ eggs}$$

$$D_s = 5.3572 \text{ g}$$

$$H_s = 10 \text{ g}$$

with Sum Squared Residuals = 0.033903 and Average error for data points = 0.004843

Here, the amount of eggs in each DD cookie is less than part (b), while the amount of eggs in each HH cookie is greater than part (b), while the amount of sugar in each type of cookie remains roughly the same. However, the new values are actually less accurate with a greater average error.

2. Constrained Least Squares Optimization

(a) Direct Proof

Consider the corresponding eigenvector, $\vec{v_1}$, of eigenvalue λ_1. By definition, we have that $\mathbf{A}^T\mathbf{A}\vec{v_1} = \lambda_1\vec{v_1}$, which gives that

Then, by definition of the 2-norm, we have that

$$\|\mathbf{A}\vec{v_1}\|^2 = \vec{v_1}^T\mathbf{A}^T\mathbf{A}\vec{v_1} = \vec{v_1}^T\lambda_1\vec{v_1} = \lambda_1\vec{v_1}^T\vec{v_1} = \lambda_1 \cdot \|\vec{v_1}\|^2$$

Since the matrix $\mathbf{A}$ is full rank, so it has a trivial nullspace, i.e. for non-trivial vector $\vec{v_1}$, we have that $\mathbf{A}\vec{v_1} \neq \vec{0}$ and $\|\vec{v_1}\| \neq$, and thus, $\|\mathbf{A}\vec{v_1}\| \neq 0$. By the nature of squares, so

$$\|\mathbf{A}\vec{v_1}\|^2 > 0, \text{and } \|\vec{v_1}\|^2 > 0$$

Therefore, we have that

$$\lambda_1 = \frac{\|\mathbf{A}\vec{v_1}\|^2}{\|\vec{v_1}\|^2} > 0$$

as desired, i.e. all the eigenvalues are strictly positive.

Q.E.D.

(b) Direct Proof

Given the two equations:

$$\mathbf{A}^T\mathbf{A}\vec{v_k} = \lambda_k\vec{v_k} \tag{1}$$

$$\vec{v_l}^T\mathbf{A}^T\mathbf{A} = \lambda_l\vec{v_l}^T \tag{2}$$

Premultiply Equation 1 with $\vec{v_l}^T$, and postmultiply Equation 2 with $\vec{v_k}$ gives us the two following equations:

$$\vec{v_l}^T\mathbf{A}^T\mathbf{A}\vec{v_k} = \vec{v_l}^T\lambda_k\vec{v_k}$$
$$\vec{v_l}^T\mathbf{A}^T\mathbf{A}\vec{v_k} = \lambda_l\vec{v_l}^T\vec{v_k}$$

Thus, we can conclude that:

$$\vec{v_l}^T\lambda_k\vec{v_k} = \vec{v_l}^T\mathbf{A}^T\mathbf{A}\vec{v_k} = \lambda_l\vec{v_l}^T\vec{v_k}$$

which, after rearranging the terms to put constant (eigenvalue) as first, would give us:

$$\lambda_k\vec{v_l}^T\vec{v_k} = \lambda_l\vec{v_l}^T\vec{v_k} \tag{3}$$

Since we have that $\lambda_k \neq \lambda_l$, and that we proved in part (a) that all eigenvalues are strictly positive, which gives us that $\lambda_k, \lambda_l \neq 0$, so, for Equation 3 to hold, it's necessary that:

$$\vec{v_l}^T\vec{v_k} = 0$$

$$\implies \langle\vec{v_l}, \vec{v_k}\rangle = 0$$

which gives us that $\vec{v_k}$ and $\vec{v_l}$ are orthogonal, as desired. Q.E.D.

(c) (i) $\alpha_n = \vec{v_n}^T\vec{x}$; (ii) Direct Proof

(i) To determine the n^th coefficient α_n, consider as we multiply $\vec{v_n}^T$ by $\vec{x}$. Given that $\vec{x} = \sum_{n=1}^{N} \alpha_n \vec{v_n}$, so we have that:

$$\vec{v_n}^T\vec{x} = \vec{v_n}^T \cdot \left(\sum_{k=1}^{N} \alpha_k \vec{v_k}\right) =$$

$$= \vec{v_n}^T\alpha_1\vec{v_1} + \vec{v_n}^T\alpha_2\vec{v_2} + \cdots + \vec{v_n}^T\alpha_n\vec{v_n} + \cdots + \vec{v_n}^T\alpha_N\vec{v_N} = \alpha_1\vec{v_n}^T\vec{v_1} + \alpha_2\vec{v_n}^T\vec{v_2} + \cdots + \alpha_n\vec{v_n}^T\vec{v_n} + \cdots + \alpha_N\vec{v_n}^T\vec{v_N}$$

Since $\vec{v_1}, \cdots, \vec{v_N}$ form an orthonormal basis and they're all unit length, so by definition, for any $i, j, n \in [1, N], i \neq j$, we have that $\langle \vec{v_i}, \vec{v_j} \rangle = 0$, and that $\langle \vec{v_n}, \vec{v_n} \rangle = \vec{v_n}^T\vec{v_n} = \|\vec{v_n}\|^2 = 1$. Thus, we have that:

$$\implies \vec{v_n}^T\vec{x} = 0 + 0 + \cdots \alpha_n \cdot 1 + 0 + \cdots + 0 = \alpha_n$$

Therefore, the n^th coefficient is:

$$\alpha_n = \vec{v_n}^T\vec{x}$$

(ii) Using the fact that $\vec{x} = \sum_{n=1}^{N} \alpha_n \vec{v_n}$, consider

$$\|\vec{x}\|^2 = \left(\sum_{n=1}^{N} \alpha_n \vec{v_n}\right)^2$$

Expand the square of the summation and we get:

$$\|\vec{x}\|^2 = (\alpha_1\vec{v_1} \cdot \alpha_1\vec{v_1} + \cdots + \alpha_1\vec{v_1} \cdot \alpha_N\vec{v_N}) + \cdots\cdots + (\alpha_N\vec{v_N} \cdot \alpha_1\vec{v_1} + \cdots + \alpha_N\vec{v_N} \cdot \alpha_N\vec{v_N})$$

$$\implies \|\vec{x}\|^2 = (\alpha_1^2\vec{v_1}\vec{v_1} + \alpha_1\alpha_2\vec{v_1}\vec{v_2} + \cdots + \alpha_1\alpha_N\vec{v_1}\vec{v_N}) + \cdots\cdots + (\alpha_N\alpha_1\vec{v_N}\vec{v_1} + \cdots + \alpha_N^2\vec{v_N}\vec{v_N})$$

As similarly demonstrated in part (i), for any $i, j, n \in [1, N], i \neq j$, we have that $\langle \vec{v_i}, \vec{v_j} \rangle = 0$, and that $\langle \vec{v_n}, \vec{v_n} \rangle = \vec{v_n}^T\vec{v_n} = \|\vec{v_n}\|^2 = 1$. Thus, we can simplify the result above to:

$$\|\vec{x}\|^2 = (\alpha_1^2 \cdot 1 + 0 + \cdots + 0) + (0 + \alpha_2^2 \cdot 1 + \cdots + 0) + (0 + \cdots + \alpha_N^2 \cdot 1) = \alpha_1^2 + \cdots + \alpha_N^2$$

Since $\vec{x}$ is a unit vector, so the left side of the equation can be simplified to $\|\vec{x}\|^2 = 1$, and the right side could be rewritten as $\sum_{n=1}^{N} \alpha_n^2$.

Therefore, we can conclude that:

$$\sum_{n=1}^{N} \alpha_n^2 = 1$$

as desired.

Q.E.D.

(d) $\|\mathbf{A}\vec{x}\|^2 = \sum\limits_{n=1}^{N} \alpha_n^2 \lambda_n$; $\vec{\tilde{x}} = \vec{v_1}$; $\|\mathbf{A}\vec{x}\| = \lambda_1$

Using the given equations, setup and results from previous parts, we have that:

$$\|\mathbf{A}\vec{x}\|^2 = \vec{x}^T \mathbf{A}^T \mathbf{A}\vec{x}$$

$$\mathbf{A}^T \mathbf{A}\, \vec{v_k} = \lambda_k \vec{v_k} \qquad \text{for all } k \in \{1, \dots, N\}$$

$$\vec{x} = \sum_{n=1}^{N} \alpha_n \vec{v_n}, \text{ and so } \vec{x}^T = \sum_{n=1}^{N} \alpha_n \vec{v_n}^T$$

Also, as similarly demonstrated in part (i), for any $i, j, n \in [1, N], i \neq j$, we have that $\langle \vec{v_i}, \vec{v_j} \rangle = 0$, and that $\langle \vec{v_n}, \vec{v_n} \rangle = \vec{v_n}^T \vec{v_n} = \|\vec{v_n}\|^2 = 1$.

Thus, we can rewrite $\|\mathbf{A}\vec{x}\|^2$ as:

$$\|\mathbf{A}\vec{x}\|^2 = \left(\sum_{n=1}^{N} \alpha_n \vec{v_n}^T \right) \cdot \mathbf{A}^T \mathbf{A} \cdot \left(\sum_{n=1}^{N} \alpha_n \vec{v_n} \right) \tag{4}$$

Now, since $\mathbf{A}^T \mathbf{A}\, \vec{v_k} = \lambda_k \vec{v_k}$ for all $k \in \{1, \dots, N\}$, so we have

$$\mathbf{A}^T \mathbf{A}\, \alpha_k \vec{v_k} = \alpha_k \lambda_k \vec{v_k}, \text{ for all } k \in \{1, \dots, N\}$$

Thus,

$$\mathbf{A}^T \mathbf{A} \cdot \left(\sum_{n=1}^{N} \alpha_n \vec{v_n} \right) = \sum_{n=1}^{N} \alpha_n \lambda_n \vec{v_n}$$

Thus, we can further simply Eq. (4) as:

$$\|\mathbf{A}\vec{x}\|^2 = \left(\sum_{n=1}^{N} \alpha_n \vec{v_n}^T \right) \cdot \sum_{n=1}^{N} \alpha_n \lambda_n \vec{v_n}$$

Expanding the product of the two summations into individual terms gives us that $\|\mathbf{A}\vec{x}\|^2 = (\alpha_1 \vec{v_1}^T \cdot \alpha_1 \lambda_1 \vec{v_1} + \alpha_1 \vec{v_1}^T \cdot \alpha_2 \lambda_2 \vec{v_2} + \cdots + \alpha_1 \vec{v_1}^T \cdot \alpha_N \lambda_N \vec{v_N}) + (\alpha_2 \vec{v_2}^T \cdot \alpha_1 \lambda_1 \vec{v_1} + \alpha_2 \vec{v_2}^T \cdot \alpha_2 \lambda_2 \vec{v_2} + \cdots + \alpha_2 \vec{v_2}^T \cdot \alpha_N \lambda_N \vec{v_N}) + \cdots\cdots + (\alpha_N \vec{v_N}^T \cdot \alpha_1 \lambda_1 \vec{v_1} + \alpha_N \vec{v_N}^T \cdot \alpha_2 \lambda_2 \vec{v_2} + \cdots + \alpha_N \vec{v_N}^T \cdot \alpha_N \lambda_N \vec{v_N})$, which, by pulling constants to the front, we could rewrite as:

$$\|\mathbf{A}\vec{x}\|^2 = (\alpha_1 \alpha_1 \lambda_1 \cdot \vec{v_1}^T \vec{v_1} + \cdots + \alpha_1 \alpha_N \lambda_N \cdot \vec{v_1}^T \vec{v_N}) + \cdots + (\alpha_N \alpha_1 \lambda_1 \cdot \vec{v_N}^T \vec{v_1} + \cdots + \alpha_N \alpha_N \lambda_N \cdot \vec{v_N}^T \vec{v_N})$$

Using the orthonormal property (details included above), so we can further simply our result to be:

$$\|\mathbf{A}\vec{x}\|^2 = \alpha_1 \alpha_1 \lambda_1 \cdot 1 + \alpha_2 \alpha_2 \lambda_2 \cdot 1 + \cdots + \alpha_N \alpha_N \lambda_N \cdot 1 = \sum_{n=1}^{N} \alpha_n^2 \lambda_n$$

To minimize $\|\mathbf{A}\vec{x}\|^2 = \alpha_1^2 \lambda_1 + \cdots + \alpha_N^2 \lambda_N$ with the constraint that $\sum\limits_{n=1}^{N} \alpha_n^2 = 1$, and also since we set the λ's such that $\lambda_1 < \cdots < \lambda_N$, so the minimum occurs when $\alpha_1^2 = 1$, $\alpha_2^2 = \cdots = \alpha_N^2 = 0$. Thus, in this case, the $\vec{\tilde{x}}$ we're looking for is:

$$\vec{\tilde{x}} = 1 \cdot \vec{v_1} + 0 \cdot \vec{v_2} + \cdots + 0 \cdot \vec{v_N} = \vec{v_1}$$

Therefore, with $\vec{v_1}$ being a unit vector, so we have:

$$\|\mathbf{A}\vec{x}\| = \|\lambda_1 \vec{v_1}\| = \lambda_1 \cdot \|\vec{v_1}\| = \lambda_1$$

3. Noise Cancelling Headphones

(a) $\vec{m} + \mathbb{R}\vec{\gamma} + \vec{n}$

The signal at the listener's ear is:

$$\vec{s} = \vec{m} + \mathbb{R}\vec{\gamma} + \vec{n}$$

(b) $\mathbb{R}\vec{\gamma} + \vec{n}$

In order to have a signal at the ear that matches the original music signal perfectly, we should aim to minimize:

$$\mathbb{R}\vec{\gamma} + \vec{n}$$

(c) Implemented based on formula $\vec{x} = (A^T A)^{-1} A^T \vec{b}$.

(d) $\vec{\gamma} = \begin{bmatrix} -0.0883 \\ -0.093 \\ -0.9184 \end{bmatrix}$

Using IPython, we have that:

$$\gamma_A = -0.0883$$

$$\gamma_B = -0.093$$

$$\gamma_C = -0.9184$$

which gives us that

$$\vec{\gamma} = \begin{bmatrix} \gamma_A \\ \gamma_B \\ \gamma_C \end{bmatrix} = \begin{bmatrix} -0.0883 \\ -0.093 \\ -0.9184 \end{bmatrix}$$

(e)

For the three loaded sounds, music_y is the song "Hallelujah"; noise1_y is full of static noise; noise2_y is static noise plus some other noise (laughter and train whistle).

Then, adding the first noise to the signal literally imposes the static noise upon the song; adding the second noise does the same thing, with the extra bit of laughters and train whistles.

Only the origianl music (music_y) is not full of static. Adding both noises would make the sound full of static, but the effect is especially obvious when adding the first noise.

4. Image Analysis

(a)

We can find the equation of a circle that surrounds the cell by performing least squares on the points and find the best-fit quadratic equation of the form of a circle.

We first set up a system of linear equations from our data:

$$(0.3^2+(-0.69)^2)a+0.3d+(-0.69)e=0.5661a+0.3d-0.69e\approx 1$$
$$(0.5^2+0.87^2)a+0.5d+0.87e=1.0069a+0.5d+0.87e\approx 1$$
$$(0.9^2+(-0.86)^2)a+0.9d+(-0.86)e=1.5496a+0.9d-0.86e\approx 1$$
$$(1^2+0.88^2)a+1\cdot d+0.88e=1.7744a+1\cdot d+0.88e\approx 1$$
$$(1.2^2+(-0.82)^2)a+1.2d+(-0.82)e=2.1124a+1.2d-0.82e\approx 1$$
$$(1.5^2+0.64^2)a+1.5d+0.64e=2.6596a+1.5d+0.64e\approx 1$$
$$(1.8^2+0^2)a+1.8d+0\cdot e=3.24a+1.8d+0\cdot e\approx 1$$

Thus, we can formulate a set of matrix equations: $\begin{bmatrix}0.5661 & 0.3 & -0.69\\ 1.0069 & 0.5 & 0.87\\ 1.5496 & 0.9 & -0.86\\ 1.7744 & 1 & 0.88\\ 2.1124 & 1.2 & -0.82\\ 2.6596 & 1.5 & 0.64\\ 3.24 & 1.8 & 0\end{bmatrix}\cdot\begin{bmatrix}a\\ d\\ e\end{bmatrix}=\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix}+\vec{e}$

(b)

Similar to part (a), we can find the equation of an ellipse that surrounds the cell by performing least squares on the points and find the best-fit quadratic equation of the form of an ellipse.

We first set up a system of linear equations from our data:

$$0.3^2a+0.3\cdot(-0.69)b+(-0.69)^2c+0.3d+(-0.69)e\approx 1$$
$$0.5^2a+0.5\cdot 0.87b+0.87^2c+0.5d+0.87e\approx 1$$
$$0.9^2a+0.9\cdot(-0.86)b+(-0.87)^2c+0.9d+(-0.86)e\approx 1$$
$$1^2a+1\cdot 0.88b+0.88^2c+1d+0.88e\approx 1$$
$$1.2^2a+1.2\cdot(-0.82)b+(-0.82)^2c+1.2d+(-0.82)e\approx 1$$
$$1.5^2a+1.5\cdot 0.64b+0.64^2c+1.5d+0.64e\approx 1$$
$$1.8^2a+1.8\cdot 0b+0^2c+1.8d+0\cdot e\approx 1$$

Thus, the set of matrix equations is: $\begin{bmatrix} 0.09 & -0.207 & 0.4761 & 0.3 & -0.69 \\ 0.25 & 0.435 & 0.7569 & 0.5 & 0.87 \\ 0.81 & -0.774 & 0.7396 & 0.9 & -0.86 \\ 1 & 0.88 & 0.7744 & 1 & 0.88 \\ 1.44 & -0.984 & 0.6724 & 1.2 & -0.82 \\ 2.25 & 0.96 & 0.4096 & 1.5 & 0.64 \\ 3.24 & 0 & 0 & 1.8 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \vec{e}$

(c) $\frac{\|\vec{e}\|}{N} = 0.1375$

Points and best-fit circle plotted on IPython.

(d) $\frac{\|\vec{e}\|}{N} = 0.01285$; Much smaller error; Ellipse is better.

Points and best-fit ellipse plotted on IPython.

This average error, $\frac{\|\vec{e}\|}{N} = 0.01285$ is much smaller than that of the circle's in part (c), which indicates that the method in part (d), the best-fit ellipse, is a better technique.

5. Pollster: Regularized Least Squares

(a) $\mathbf{A} = \mathbb{R}^{90\times 10}$, $\vec{x} = \mathbb{R}^{10\times 1}$, $\vec{b} = \mathbb{R}^{90\times 1}$

$\mathbf{A}$ is constructed as the first 90 examples in the dataset, i.e. the 90 feature vectors of the first 90 counties. It has dimensions 90×10.

$\vec{x}$ is the model vector we're trying to train and optimize, i.e. the weight vector that would optimize the predictions of the voting decisions of any county. It has dimensions 10×1.

$\vec{b}$ is constructed as the 90 actual voting decisions of the first 90 counties. It has dimensions 90×1.

(b)

Using IPython, I found the solved for $\vec{\tilde{x}}$ (vector indicated on IPython), and using this linear model, I evaluated that the total prediction error on the training data is: 0.8633; the total prediction error on the testing data is: 0.3203.

(c) Direct Proof

First, since the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are denoted as λ_i, so as we proved in HW earlier, we have that the eigenvalues of $\mathbf{A}^T\mathbf{A}^{-1}$ are $\frac{1}{\lambda_i}$.

The solution $\vec{\tilde{x}}$ to the least squares problem $\mathbf{A}\vec{x} = \vec{b}$ is, by our formula:

$$\vec{\tilde{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$$

Thus, $\vec{e} = \vec{b} - \mathbf{A}\vec{\tilde{x}} = \vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$, and since $\mathbf{A}^T\vec{b} = \sum_{i=1}^{N} \beta_i\vec{v_i}$, so we can further simplify the error vector (along with the properties of eigenvalues and eigenvectors) as:

$$\vec{e} = \vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\sum_{i=1}^{N}\beta_i\vec{v_i} = \vec{b} - \mathbf{A}\sum_{i=1}^{N}\frac{\beta_i}{\lambda_i}\vec{v_i}$$

Therefore,

$$\|\vec{e}\| = \|\vec{b} - \mathbf{A}\sum_{i=1}^{N}\frac{\beta_i}{\lambda_i}\vec{v_i}\| = \left\|\vec{b} - \mathbf{A}\Big(\frac{\beta_1}{\lambda_1}\vec{v_1} + \frac{\beta_2}{\lambda_2}\vec{v_2} + \cdots + \frac{\beta_N}{\lambda_N}\vec{v_N}\Big)\right\|$$

as desired. Q.E.D.

(d) Yes, we do.

The maximum eigenvalue is about 526, and all other eigenvalues (except one at about 377) is less than 10, which means that we're encountering the issue described in part (c).

(e) $\tilde{\mathbf{A}} = \mathbb{R}^{100\times 10}$, $\vec{\tilde{b}} = \mathbb{R}^{100\times 1}$

By the nature of concatenation, since the identity matrix $\mathbf{I}$ would have dimensions 10×10, so $\tilde{\mathbf{A}}$ has dimensions 100×10, and $\vec{\tilde{b}}$ has dimensions 100×1. (Implemented with IPython.)

(f) Direct Proof

First, define some variables for notation: let

$$\tilde{\mathbf{A}} = \left[\frac{\mathbf{A}}{\sqrt{\gamma}\mathbf{I}}\right], \qquad \tilde{\vec{b}} = \left[\frac{\vec{b}}{\vec{0}}\right]$$

So, the solution $\tilde{\vec{x}}$ to the least squares problem $\tilde{\mathbf{A}}\vec{x} = \tilde{\vec{b}}$ is, by our formula:

$$\tilde{\vec{x}} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\vec{b}}$$

Thus, using the hint regarding matrix-matrix multiplication handled in blocks, we have that:

$$\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} = \left[\mathbf{A}^T \middle| \sqrt{\gamma}\mathbf{I}\right]\left[\frac{\mathbf{A}}{\sqrt{\gamma}\mathbf{I}}\right] = \mathbf{A}^T\mathbf{A} + \gamma\mathbf{I}$$

$$\tilde{\mathbf{A}}^T\tilde{\vec{b}} = \left[\mathbf{A}^T \middle| \sqrt{\gamma}\mathbf{I}\right]\left[\frac{\vec{b}}{\vec{0}}\right] = \mathbf{A}^T\vec{b} + \vec{0} = \mathbf{A}^T\vec{b}$$

Therefore, we can conclude that solution to the modified linear least squares problem is:

$$\tilde{\vec{x}} = (\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I})^{-1}\mathbf{A}^T\vec{b}$$

as desired. Q.E.D.

(g) Direct Proof

First, since the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are denoted as λ_i, so for any arbitrary eigenvector $\vec{u_k}$ of $\mathbf{A^TA}$, we have that

$$(\mathbf{A^TA} + \gamma\mathbf{I})\ \vec{u_k} = \mathbf{A^TA}\vec{u_k} + \gamma\mathbf{I}\vec{u_k} = \lambda_k\vec{u_k} + \gamma\vec{u_k} = (\lambda_k + \gamma)\ \vec{u_k}$$

Thus, by definition, all the eigenvectors $\vec{u_i}$ of $\mathbf{A^TA}$ are also eigenvectors $\vec{v_i}$ of $\mathbf{A^TA}+\gamma\mathbf{I}$ (this result could also be achieved via the diagonalization of $\mathbf{A}^T\mathbf{A}$, which I've used as a sanity check for myself), and with our calculation, the corresponding eigenvalues are $(\lambda_i + \gamma)$. Then, similarly, as we proved in HW earlier, we have that the eigenvalues of $(\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I})^{-1}$ are $\frac{1}{\lambda_i+\gamma}$.

Then, the solution $\tilde{\vec{x}}$ to the least squares problem $\mathbf{A}\vec{x} = \vec{b}$ is, by our formula:

$$\tilde{\vec{x}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$$

Thus, $\vec{e} = \vec{b} - \mathbf{A}\tilde{\vec{x}} = \vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}$, and given that $\mathbf{A}^T\vec{b} = \sum_{i=1}^{N}\beta_i\vec{v_i}$, so we can further simplify the error vector (along with the properties of eigenvalues and eigenvectors) as:

$$\vec{e} = \vec{b} - \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\sum_{i=1}^{N}\beta_i\vec{v_i} = \vec{b} - \mathbf{A}\sum_{i=1}^{N}\frac{\beta_i}{\lambda_i + \gamma}\vec{v_i}$$

Therefore,

$$\|\vec{e}\| = \|\vec{b} - \mathbf{A}\sum_{i=1}^{N}\frac{\beta_i}{\lambda_i}\vec{v_i}\| = \left\|\vec{b} - \mathbf{A}\Big(\frac{\beta_1}{\lambda_1 + \gamma}\vec{v_1} + \frac{\beta_2}{\lambda_2 + \gamma}\vec{v_2} + \cdots + \frac{\beta_N}{\lambda_N + \gamma}\vec{v_N}\Big)\right\|$$

as desired. Q.E.D.

(h)

As γ increases, the eigenvalues of $\mathbf{A}^T\mathbf{A}$ increases as well, and they shift directly up the y-axis.

(i) $\gamma = 9.326$; Error slightly decreased.

The best choice, optimal γ is $\gamma = 9.326$. Here, using the modified least squares with the optimal γ, we achieved Total Prediction Error on testing data as 0.3065, which is just a bit smaller than the total prediction error on testing data calculated in part (b), 0.3203.

6. OMP Exercise

(a) $x_1 = \frac{19}{3},\ x_2 = 0,\ x_3 = 0,\ x_4 = \frac{8}{3}$

Given that $\mathbf{M} = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

Iteration 1:

Calculating the four inner products ($\vec{b}$ with every column of $\mathbf{M}$) gives us $[10, 7, -3, -1]$, and the largest value is at column 1. Thus, we have $\mathbf{A_1} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ and $\vec{b_1} = \vec{b} = \begin{bmatrix} 4 \\ 6 \\ 3 \end{bmatrix}$. So, we have the least squares solution $\vec{x_1} = (\mathbf{A_1}^T \mathbf{A_1})^{-1} \mathbf{A_1}^T \vec{b_1} = \frac{1}{2} \cdot 10 = 5$, which gives us that $x_1 = 5$, and so we have our estimate currently at $\vec{r_1} = 5x_1$ with the residue after first iteration being $\vec{y_1} = \vec{b_1} - \mathbf{A_1}\vec{x_1} = \begin{bmatrix} -1 \\ 1 \\ 3 \end{bmatrix} \neq \vec{0}$.

Iteration 2:

Again, calculate the four inner products ($\vec{y_1}$ with every column of $\mathbf{M}$) gives us $[0, 2, 2, 4]$, and the largest value is at column 4. Thus, we have $\mathbf{A_2} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\vec{b_2} = \vec{b} = \begin{bmatrix} 4 \\ 6 \\ 3 \end{bmatrix}$. So, we can calculate the least squares solution $\vec{x_2} = \begin{bmatrix} 19/3 \\ 8/3 \end{bmatrix}$, which gives us updated value that $x_1 = \frac{19}{3}$ and $x_4 = \frac{8}{3}$, and so the second residue is $\vec{y_2} = \vec{b_2} - \mathbf{A_2}\vec{x_2} = \begin{bmatrix} 1/3 \\ -1/3 \\ 1/3 \end{bmatrix}$

Since we are given that $\vec{x}$ has only 2 non-zero entries, so we're done, with

$$x_1 = \frac{19}{3},\ x_2 = 0,\ x_3 = 0,\ x_4 = \frac{8}{3}$$

Therefore, we have that:

$$\vec{x} = \begin{bmatrix} 19/3 \\ 0 \\ 0 \\ 8/3 \end{bmatrix}$$

(b) $\vec{x} = \mathbf{A} \cdot \text{OMP}(\mathbf{MA}, \vec{b})$

Given matrix $\mathbf{A}$ and that $\vec{x} = \mathbf{A}\vec{x_a}$, we first calculate the inverse of $\mathbf{A}$, $\mathbf{A}^{-1}$. Then, by multiplying both sides of the equation by $\mathbf{A}^{-1}$, we have that $\vec{x_a} = \mathbf{A}^{-1}\mathbf{A}\vec{x_a} = \mathbf{A}^{-1}\vec{x}$.

Since we have that $\mathbf{M}\vec{x} = \vec{b}$, and since by definition, $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, so we have:

$$(\mathbf{MA})\vec{x_a} = \mathbf{MAA}^{-1}\vec{x} = \mathbf{M}\vec{x} = \vec{b}$$

Now, since we are given that $\vec{x_a}$ is sparse, so our OMP function would successfully compute $\vec{x_a}$ by plugging in parameters $\mathbf{MA}$ and $\vec{b}$, i.e. we can successfully determine $\vec{x_a} = \text{OMP}(\mathbf{MA}, \vec{b})$.

Finally, we can compute $\vec{x}$ back from $\vec{x_a}$ with the equation $\vec{x} = \mathbf{A}\vec{x_a}$. Therefore, in other words, we could compute $\vec{x}$ in this method:

$$\vec{x} = \mathbf{A} \cdot \text{OMP}(\mathbf{MA}, \vec{b})$$

7. Sparse Imaging

(a) Implemented OMP function.

(b) The image is the Cal logo.

8. Homework Process and Study Group

I worked with Helen Peng (SID: 3033831506) on several problems, including #5 etc. I worked alone without any help (except reading Piazza) on all the other problems.

EE16A Homework 13

Question 1: Recipe Reconnaissance

In [199]:

```
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
import sys
```

```
Populating the interactive namespace from numpy and matplotlib

/anaconda3/lib/python3.6/site-packages/IPython/core/magics/pylab.py:16
0: UserWarning: pylab import has clobbered these variables: ['gamma',
'rec', 'display']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

Note that for the following sections, we use the following labels for the unknown variables:

Decadent Dwight

- D_e = eggs/cookie
- D_b = (grams butter)/cookie
- D_s = (grams sugar)/cookie

Heavenly Hearst

- H_e = eggs/cookie
- H_b = (grams butter)/cookie
- H_s = (grams sugar)/cookie

Note: in the following sections we only solve for 4 of the unknowns with least squares, as we are given exact values for 2 unknowns.

Part (b)

In [268]:

```
print('4 unknowns (set H_b and D_b = 10 g)')
# x_ls = [D_e, H_e, D_s, H_s]

#### Your Task: fill in the correct values for A and b.
A = np.array([
    [40, 50, 0, 0],
    [0, 0, 280, 350],
    [1/6, 0, 1/200, 0],
    [0, 1/6, 0, 1/200],
    [0, 0, 0, 1]
])
b = np.array([12, 5000, 0.1, 0.1, 10])

x_ls, res, _, _ = numpy.linalg.lstsq(A,b, rcond=None)
np.set_printoptions(precision=4, suppress=True)
print(A)
print(b)
print('D_e, H_e, D_s, H_s\t=\t%s' % x_ls)
print('Sum Squared Residuals\t=\t%f' % res)

average_error = res / 5
print('Average error for data points\t=\t%f' % average_error)
```

```
4 unknowns (set H_b and D_b = 10 g)
[[ 40.      50.       0.       0.    ]
 [  0.       0.     280.     350.    ]
 [  0.1667   0.       0.005    0.    ]
 [  0.       0.1667   0.       0.005 ]
 [  0.       0.       0.       1.    ]]
[  12.  5000.     0.1    0.1   10. ]
D_e, H_e, D_s, H_s      =       [ 0.2386  0.0491  5.3571 10.    ]
Sum Squared Residuals   =       0.002867
Average error for data points   =       0.000573
```

Part (c)

In [269]:

```
print('4 unknowns (set H_b and D_b = 10 g)')
# x_ls = [D_e, H_e, D_s, H_s]

#### Your Task: fill in the correct values for A and b.
A = np.array([
    [40, 50, 0, 0],
    [0, 0, 280, 350],
    [1/6, 0, 1/200, 0],
    [0, 1/6, 0, 1/200],
    [0, 0, 0, 1],
    [50, 0, 1, 0],
    [0, 50, 0, 1]
])
b = np.array([12, 5000, 0.1, 0.1, 10, 15, 14])

x_ls, res, _, _ = numpy.linalg.lstsq(A,b, rcond=None)
np.set_printoptions(precision=4, suppress=True)
print(A)
print(b)
print('D_e, H_e, D_s, H_s\t=\t%s' % x_ls)
print('Sum Squared Residuals\t=\t%f' % res)

average_error = res / 7
print('Average error for data points\t=\t%f' % average_error)
```

```
4 unknowns (set H_b and D_b = 10 g)
[[ 40.      50.       0.       0.    ]
 [  0.       0.     280.     350.    ]
 [  0.1667   0.       0.005    0.    ]
 [  0.       0.1667   0.       0.005 ]
 [  0.       0.       0.       1.    ]
 [ 50.       0.       1.       0.    ]
 [  0.      50.       0.       1.    ]]
[  12.  5000.     0.1    0.1   10.    15.    14. ]
D_e, H_e, D_s, H_s      =       [ 0.1946  0.0822  5.3572 10.     ]
Sum Squared Residuals   =       0.033903
Average error for data points   =       0.004843
```

Question 3: Noise Cancelling Headphones

In [202]:

```
%matplotlib inline

import numpy as np
from matplotlib.pyplot import plot
from scipy.io import wavfile

from audio_support import wavPlayer
from audio_support import loadSounds
from audio_support import recordAmbientNoise
```

Part c)

In the following cell, implement the least squares solution to

$$min_{\vec{x}}||A\vec{x} - \vec{b}||$$

In [203]:

```
def doLeastSquares(A,b):
    # BEGIN
    At = np.transpose(A)
    AtA = np.dot(At, A)
    inv = np.linalg.inv(AtA)
    last_step = np.dot(inv, At)
    x = np.dot(last_step, b)
    # END
    return x;
```

Part d)

Use your least squares solution to find the gamma that minimizes the effect of noise given:

$$\vec{n} = \begin{bmatrix} 0.10 \\ 0.37 \\ -0.45 \\ 0.068 \\ 0.036 \end{bmatrix}; \vec{r}_A = \begin{bmatrix} 0 \\ 0.11 \\ -0.31 \\ -0.012 \\ -0.018 \end{bmatrix}; \vec{r}_B = \begin{bmatrix} 0 \\ 0.22 \\ -0.20 \\ 0.080 \\ 0.056 \end{bmatrix}; \vec{r}_C = \begin{bmatrix} 0 \\ 0.37 \\ -0.44 \\ 0.065 \\ 0.038 \end{bmatrix}$$

In [204]:

```
n1 = 0.10;
n2 = 0.37;
n3 = -0.45;
n4 = 0.068;
n5 = 0.036;

a1 = 0;
a2 = 0.11;
a3 = -0.31;
a4 = -0.012;
a5 = -0.018;

b1 = 0;
b2 = 0.22;
b3 = -0.20;
b4 = 0.080;
b5 = 0.056;

c1 = 0;
c2 = 0.37;
c3 = -0.44;
c4 = 0.065;
c5 = 0.038;

# BEGIN

A = np.array([
    [a1, b1, c1],
    [a2, b2, c2],
    [a3, b3, c3],
    [a4, b4, c4],
    [a5, b5, c5]
])
b = np.array([
    [n1],
    [n2],
    [n3],
    [n4],
    [n5]
]).dot(-1)

gamma = doLeastSquares(A, b)

# END
print(gamma)
```

```
[[-0.0883]
 [-0.093 ]
 [-0.9184]]
```

Report the results for your gamma-vector.

Part e)

First, we'll load the sounds from the included .wav files.

In [205]:

```
[music_Fs, music_y, noise1_y, noise1_Fs, noise2_y, noise2_Fs] = loadSounds();
```

In [206]:

```
noise1_y
```

Out[206]:

```
array([ 0.1075,  0.3667, -0.4518, ...,  0.0679,  0.0358,  0.0766])
```

We can use the following function to listen to our signals throughout this notebook.

Listen to each of the loaded sounds (`music_y` , `noise1_y` , and `noise2_y`). What do you hear?

In [207]:

```
wavPlayer(music_y, music_Fs)
wavPlayer(noise1_y, music_Fs)
wavPlayer(noise2_y, music_Fs)
```

0:00 / 0:17

0:00 / 0:17

0:00 / 0:17

Add the first noise to the signal and listen to the result.

In [208]:

```
noisyMusic = music_y + noise1_y;
wavPlayer(noisyMusic, music_Fs)
```

0:00 / 0:17

Add the second noise to the signal and listen to the result.

In [209]:

```
# BEGIN
noisyMusic = music_y + noise2_y;
wavPlayer(noisyMusic, music_Fs)
# END
```

0:00 / 0:17

Part (f)

Next, we will simulate the recording of `noise1` using a simulated microphone array.

In [210]:

```
numberOfMicrophones = 3;
R = recordAmbientNoise(noise1_y,noise1_Fs,numberOfMicrophones);
```

In the cell below, calculate the gamma-vector using the least squares approach (you should calculate `gamma` from `R` and `noise1_y`).

In [211]:

```
# BEGIN
gamma =
# END
```

```
  File "<ipython-input-211-1234c23eda79>", line 2
    gamma =
           ^
SyntaxError: invalid syntax
```

In the cell below, create the noise cancellation signal by multiplying `R` and `gamma` . Add the result to `music_y` (with the right sign) to get `signalFromSpeaker` .

In [ ]:

```
# BEGIN
'...'
signalFromSpeaker =
# END
```

Part (g)

Generate the signal at the listener's ear by adding the speaker signal (`signalFromSpeaker`) to the original noise signal (`noise1_y`).

In [ ]:

```
# BEGIN
signalAtEar =
# END
```

Listen to the noisy and noise-cancelled signal.

In [ ]:

```
wavPlayer(noisyMusic, music_Fs)
wavPlayer(signalAtEar, music_Fs)
```

What difference can you hear between these signals?

Part (h)

Now, we'll see how well this gamma works for other noise.

We will run through the simulation again, but this time, we will just use the gamma from before instead of going through a training step.

In [ ]:

```
noisyMusic_2 = music_y + noise2_y;
R_2 = recordAmbientNoise(noise2_y,noise2_Fs,numberOfMicrophones);
# BEGIN
'...'
signalFromSpeaker_2 = '...'
signalAtEar_2 = '...'
# END

wavPlayer(noisyMusic_2, music_Fs)
wavPlayer(signalAtEar_2, music_Fs)
```

What do you hear in the noise-cancelled signal?

Question 4: Image Analysis

In [251]:

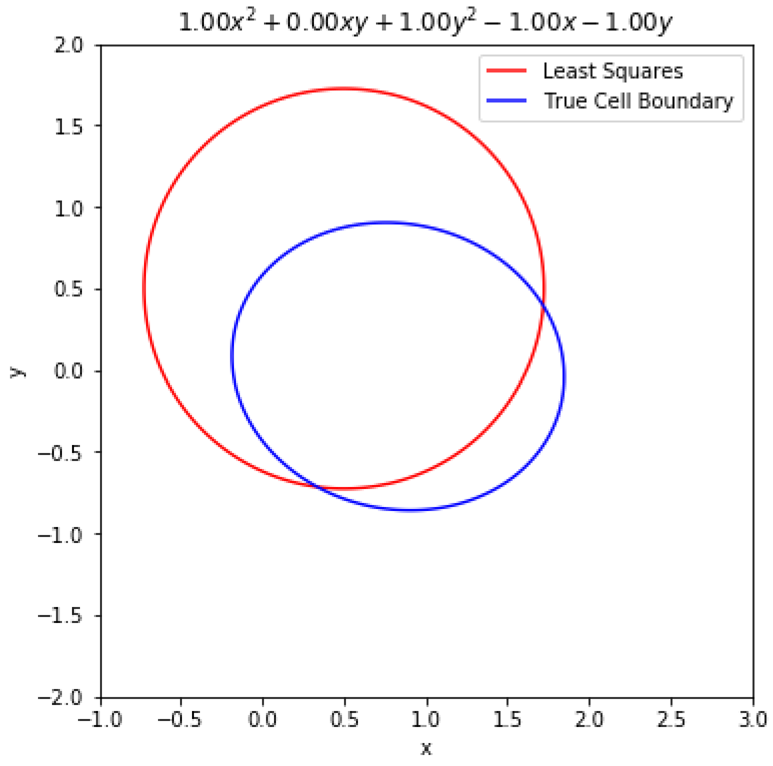
```
def plot_circle2(a, d, e):
    """
    This plots the true ellipse along with the circle
    that you plot.

    You can comment out the line that starts with `plt.title`
    because this makes assumptions regarding the title of your plot.
    """
    is_circle = d**2 + e**2 - 4*a > 0
    assert is_circle, "Not a circle"

    XLIM_LO = -1
    XLIM_HI = 3
    YLIM_LO = -2
    YLIM_HI = 2
    X_COUNT = 400
    Y_COUNT = 400

    x = np.linspace(XLIM_LO, XLIM_HI, X_COUNT)
    y = np.linspace(YLIM_LO, YLIM_HI, Y_COUNT)
    x, y = np.meshgrid(x, y)
    f = lambda x,y: a*(x**2 + y**2) + d*x + e*y
    f2 = lambda x,y: 3*x*x + .5*x*y + 4*y*y -5*x -.6*y

    c1 = plt.contour(x, y, f(x,y), [1], colors='r')
    c2 = plt.contour(x, y, f2(x,y), [1], colors='b')
    plt.axis('scaled')
    plt.xlabel('x')
    plt.ylabel('y')
#     plt.title(r'${:.2f}(x^2 + y^2) {:+.2f}x {:+.2f}y$'.format(a,d,e))

    lines = (c1.collections[0], c2.collections[0])
    labels = ('Least Squares', 'True Cell Boundary')
    plt.legend(lines, labels, loc='upper right')
```

In [252]:

```
def plot_ellipse2(a, b, c, d, e):
    """
    This plots the true ellipse along with the ellipse
    that you plot.

    You can comment out the line that starts with `plt.title`
    because this makes assumptions regarding the title of your plot.
    """
    is_ellipse = b**2 - 4*a*c < 0
    assert is_ellipse, "Not an ellipse"

    XLIM_LO = -1
    XLIM_HI = 3
    YLIM_LO = -2
    YLIM_HI = 2
    X_COUNT = 400
    Y_COUNT = 400

    x = np.linspace(XLIM_LO, XLIM_HI, X_COUNT)
    y = np.linspace(YLIM_LO, YLIM_HI, Y_COUNT)
    x, y = np.meshgrid(x, y)
    f = lambda x,y: a*x**2 + b*x*y + c*y**2 + d*x + e*y
    f2 = lambda x,y: 3*x*x + .5*x*y + 4*y*y -5*x -.6*y

    c1 = plt.contour(x, y, f(x,y), [1], colors='r')
    c2 = plt.contour(x, y, f2(x,y), [1], colors='b')
    plt.axis('scaled')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(r'${:.2f}x^2 {:+.2f}xy {:+.2f}y^2 {:+.2f}x {:+.2f}y$'.format(a,b,c,d,e

    lines = (c1.collections[0], c2.collections[0])
    labels = ('Least Squares', 'True Cell Boundary')
    plt.legend(lines, labels, loc='upper right')
```

In [253]:

```
# Here is an example of plot_ellipse.
# This plots (x-1)**2 + (y-1)**2 = 1,
# which is a circle centered at (1,1).

plt.figure(figsize=(6,6))
plot_ellipse2(1, 0, 1, -1, -1)
```



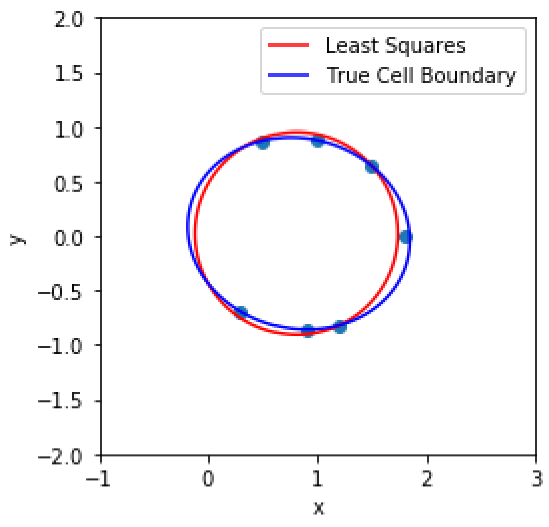
Part (c)

You may find plt.scatter (http://matplotlib.org/api/pyplot_api.html) useful for plotting the points.

In [262]:

```
xy = np.array([[0.3, -0.69],
               [0.5, 0.87],
               [0.9, -0.86],
               [1, 0.88],
               [1.2, -0.82],
               [1.5, .64],
               [1.8, 0]])

# YOUR CODE HERE
N = np.size(xy, axis=0)

A_c = []
for i in range(N):
    x, y = xy[i]
    A_c.append([x**2 + y**2, x, y])

b_c = np.array([1] * N)

x_c = doLeastSquares(A_c, b_c)
print('x_circle =', x_c, '\n')

e_c = np.subtract(b_c, np.dot(A_c, x_c))
print('Average error:', np.linalg.norm(e_c) / N)

# Graphing
plt.scatter(xy[:, :1], xy[:, 1:])
plot_circle2(x_c[0], x_c[1], x_c[2])
```

```
x_circle = [ 4.8731 -7.8929 -0.2265]

Average error: 0.13749056224314807
```

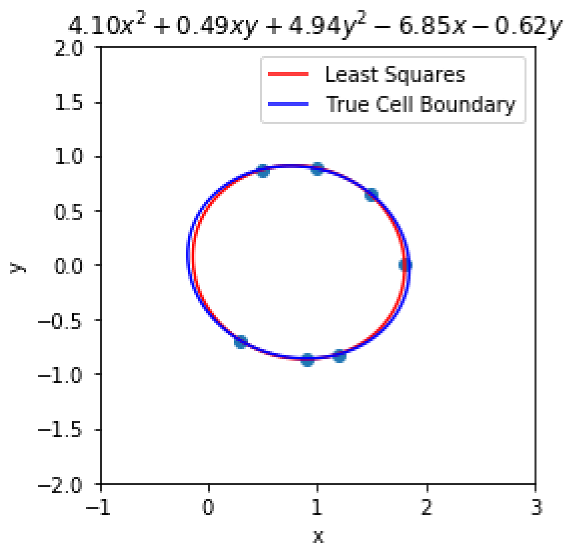


Part (d)

In [265]:

```
# YOUR CODE HERE
N = np.size(xy, axis=0)

A_e = []
for i in range(N):
    x, y = xy[i]
    A_e.append([x**2, x*y, y**2, x, y])

b_e = np.array([1] * N)

x_e = doLeastSquares(A_e, b_e)
print('x_ellipse =', x_e, '\n')

e_e = np.subtract(b_e, np.dot(A_e, x_e))
print('Average error:', np.linalg.norm(e_e) / N)

# Graphing
plt.scatter(xy[:, :1], xy[:, 1:])
plot_ellipse2(x_e[0], x_e[1], x_e[2], x_e[3], x_e[4])
```

```
x_ellipse = [ 4.1038  0.4871  4.9394 -6.8503 -0.6226]

Average error: 0.012853829087236082
```



Question 5: Pollster

In [212]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio
%matplotlib inline

data = sio.loadmat('data.mat')

# All data examples/labels (training & testing)
examples = data['data']
labels = data['labels'][0,:]

N_examples = examples.shape[0]
N_features = examples.shape[1]

# Least Squares
doLeastSquares = lambda A,b : np.linalg.lstsq(A,b,rcond=0.)[0]
```

part (b)

The total prediction error is the length of the error vector, $||\vec{b} - \mathbf{A}\hat{\vec{x}}||$.

In [213]:

```
# Setup A,b for only the training examples
K = 90
A = examples[:K, :]
b = labels[:K]

# Solve Linear Least Squares (doLeastSquares)
x_hat = doLeastSquares(A, b)
print('x =', x_hat, '\n')

# Evaluate training prediction error
pred_training_error = np.linalg.norm(np.subtract(b, np.dot(A, x_hat)))
print('Training Error:', pred_training_error)

# Evaluate testing prediction error
A_test = examples[K:,:]
b_test = labels[K:]

pred_testing_error = np.linalg.norm(np.subtract(b_test, np.dot(A_test, x_hat)))
print('Testing Error:', pred_testing_error)
```

```
x = [-0.1805 -0.0688  0.0761 -0.0986  0.0907  0.0558  0.2183  0.1526 -
0.0483
 -0.2307]

Training Error: 0.863316443561086
Testing Error: 0.3203415321831521
```

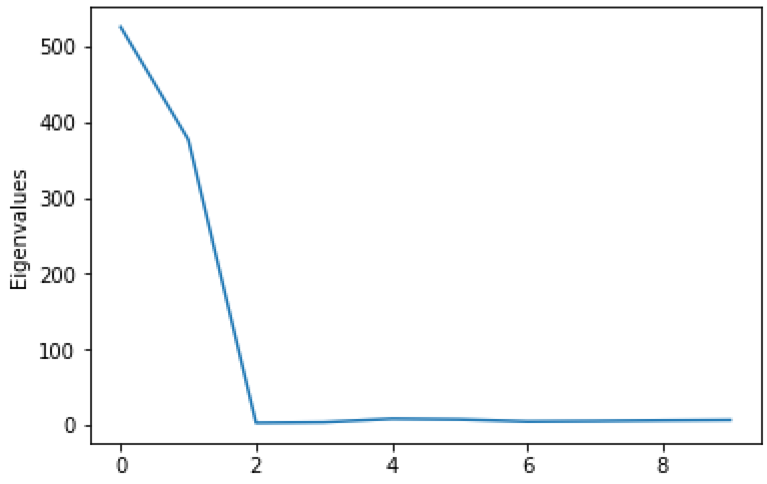
part (d)

Compute the eigenvalues of $\mathbf{A}^T\mathbf{A}$.

In [214]:

```
# YOUR CODE HERE
AtA = np.dot(A.transpose(), A)
val, vec = np.linalg.eig(AtA)
print('Eigenvalues:', val)

plt.ylabel("Eigenvalues");
plt.plot(val);
```

```
Eigenvalues: [526.1331 377.138    3.075    3.8993   8.4823   7.657
4.8908   5.4385
   6.2888   6.8492]
```



part (e)

Create the augmented matrix $\tilde{\mathbf{A}}$ and the augmented vector $\tilde{\vec{b}}$. Fill in the ... to complete the {\it np.concatenate} statements.

In [215]:

```
gamma = 1
gammaI = np.sqrt(gamma) * np.eye(N_features)
zeroVec = np.zeros(N_features)

# augmented A matrix (complete statement)
augA = np.concatenate((A, gammaI),axis=0)
print('Old A dimensions:',A.shape)
print('New A dimensions:',augA.shape)

# augmented b vector (complete statement)
augb = np.concatenate((b, zeroVec),axis=0)
print('Old b dimensions:',b.shape)
print('New b dimensions:',augb.shape)
```

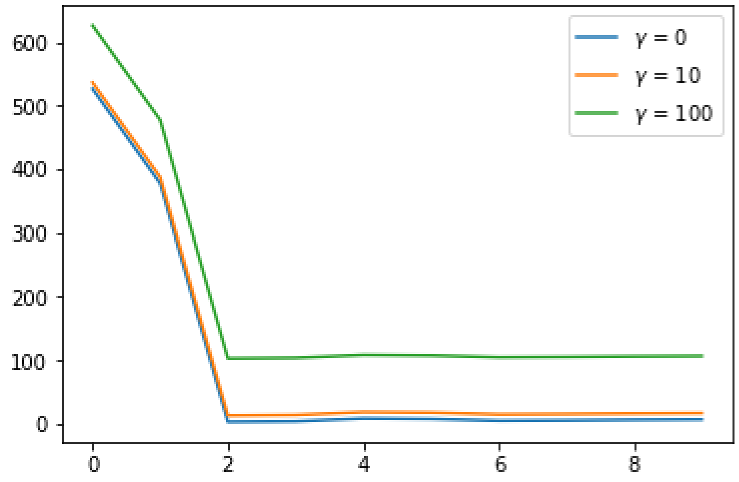
```
Old A dimensions: (90, 10)
New A dimensions: (100, 10)
Old b dimensions: (90,)
New b dimensions: (100,)
```

part (h)

Compute the eigenvalues of $\mathbf{A}^T\mathbf{A} + \gamma\mathbf{I}$ for each $\gamma$ in the list. Plot them on the same axis.

In [216]:

```
gammas = [0, 10, 100]

AtA = np.dot(A.transpose(), A)

for g in gammas:
    gI = np.dot(g, np.eye(10))
    val, _ = np.linalg.eig(AtA + gI)
    plt.plot(val, label='$\gamma$ = ' + str(g))

plt.legend();
```



part (i)

For each $\gamma$ in the list, recreate the augmented A matrix, compute the modified least squares solution by calling the function doLeastSquares, and for each solution compute its corresponding total testing prediction error.

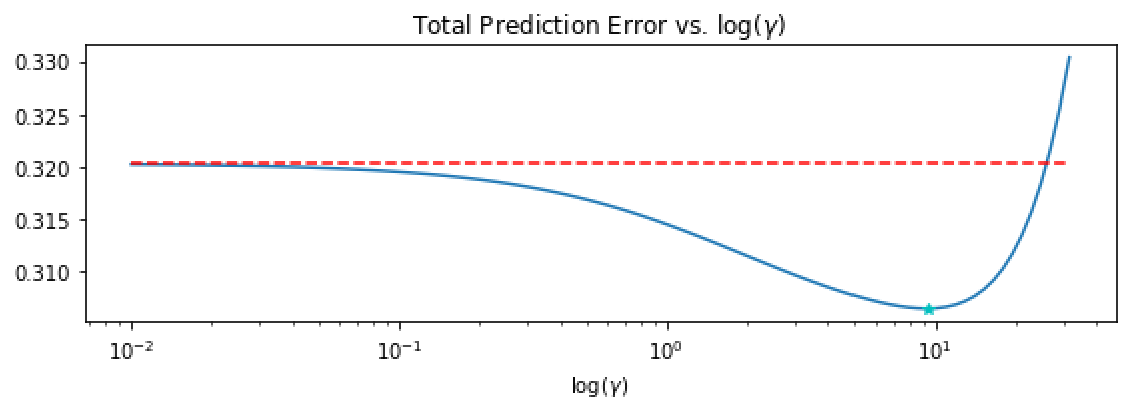
In [217]:

```
N_gamma = 100
gammaList = np.logspace(-2,1.5,N_gamma)
reg_total_testing_pred_error = np.zeros(N_gamma)

K = 90
A_testing = examples[K:,:]
b_testing = labels[K:]
print(b_testing.shape)

for ii in range(N_gamma):
    # create the augmented A matrix (similar to in part e)
    sqrtgammaI = np.sqrt(gammaList[ii])*np.eye(N_features)
    augA = np.concatenate((A, sqrtgammaI),axis=0)

    # Call lls to find linear least squares solution
    x = doLeastSquares(augA, augb)

    # Evaluate testing error and store in list reg_total_testing_pred_error
    error = np.linalg.norm(np.subtract(b_testing, np.dot(A_testing, x)))
    reg_total_testing_pred_error[ii] = error


# Do not edits this
plt.figure(figsize=(8,3))
plt.semilogx(gammaList,reg_total_testing_pred_error)
plt.semilogx(gammaList,pred_testing_error*np.ones(N_gamma),'r--')
plt.xlabel('log($\gamma$)')
plt.title('Total Prediction Error vs. log($\gamma$)')
plt.tight_layout()

gamma_reglls_opt = gammaList[np.argmin(reg_total_testing_pred_error)]
testing_error_reglls_opt = np.min(reg_total_testing_pred_error)
print('Optimal gamma: ', gamma_reglls_opt)
print('Achieved Total Prediction Error: ', testing_error_reglls_opt)

plt.semilogx(gamma_reglls_opt, testing_error_reglls_opt, 'c*')
```

```
(10,)
Optimal gamma:  9.326033468832199
Achieved Total Prediction Error:  0.30650228118716716
```

Out[217]:

```
[<matplotlib.lines.Line2D at 0xb18727390>]
```



Question 7: Sparse Imaging

This example generates a sparse signal and tries to recover it using the Orthogonal Matching Pursuit algorithm.

In [171]:

```
# imports
import matplotlib.pyplot as plt
import numpy as np
from scipy import misc
from IPython import display
import sys
%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
```

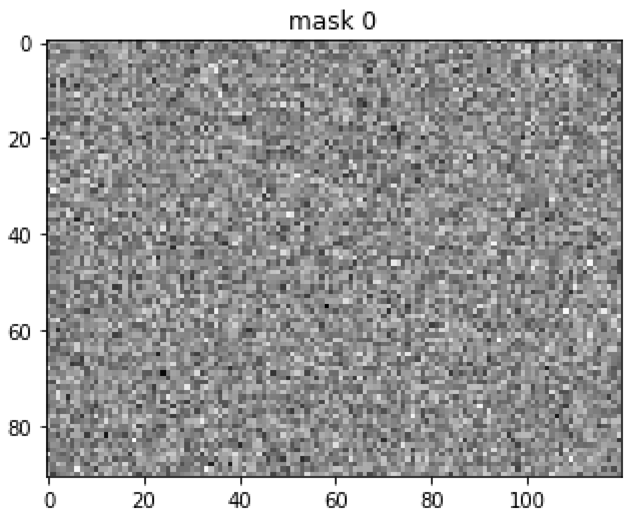
Part (a)

In [172]:

```
measurements, A = simulate()

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

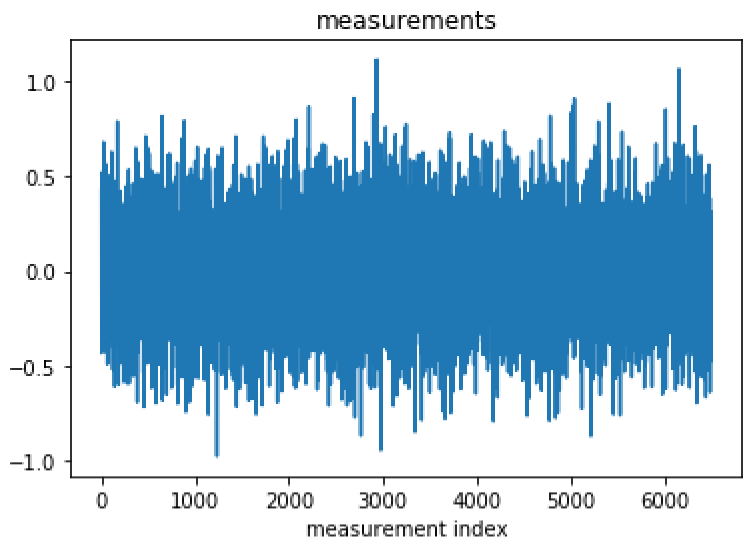
In [173]:

```
# 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 [174]:

```
# measurements
plt.title('measurements')
plt.plot(measurements)
plt.xlabel('measurement index')
plt.show()
```



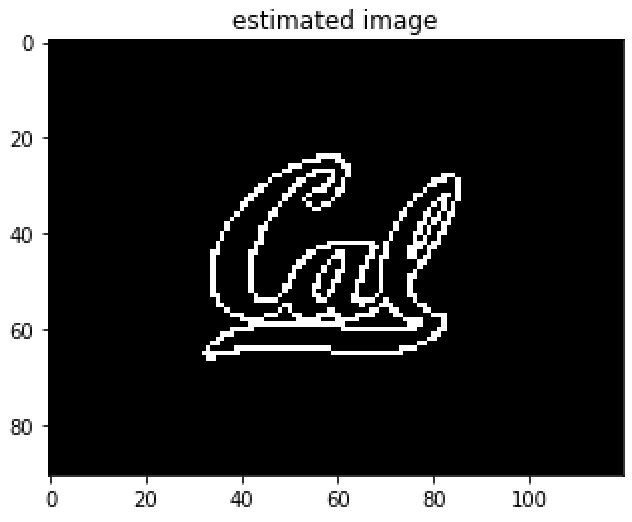
In [177]:

```
# 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 collection
        # 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 [178]:

```
rec = OMP((height,width), sparsity, measurements, A)
```



PRACTICE: Part (c)

In [ ]:

```
# the setting

# file name for the sparse image
fname = 'figures/smiley.png'
# number of measurements to be taken from the sparse image
numMeasurements = 6500
# the sparsity of the image
sparsity = 400

# read the image in black and white
I = misc.imread(fname, flatten=1)
# 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 [ ]:

```
# 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 [ ]:

```
# measurements
plt.title('measurements')
plt.plot(measurements)
plt.xlabel('measurement index')
plt.show()
```

In [ ]:

```
rec = OMP(imageShape, sparsity, measurements, A)
```

In [ ]:

```
```