Linear Classification

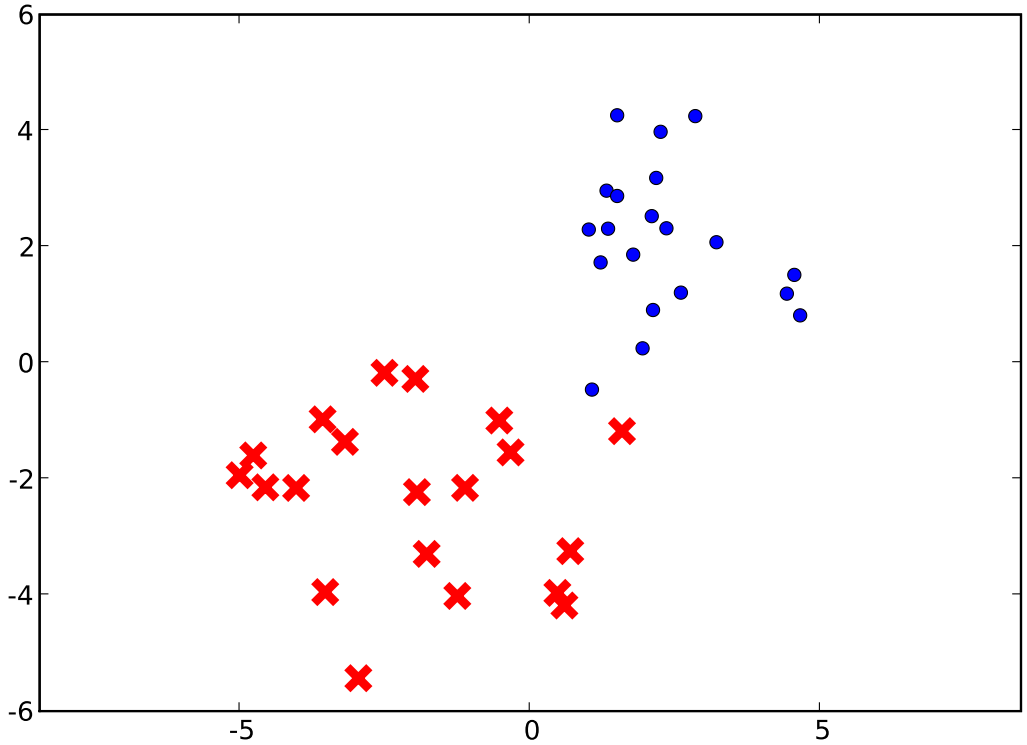
MLTA -2013

SAU, New Delhi

Dec. 15, 2013

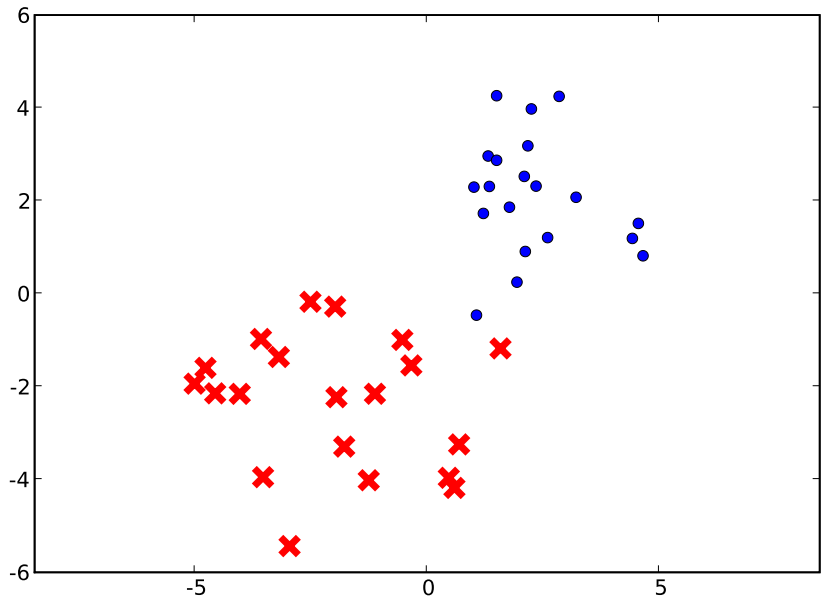
Basic Setup

- We want to separate the X's and the O's
- Today, we will see how to solve this (seemingly) simple task mathematically



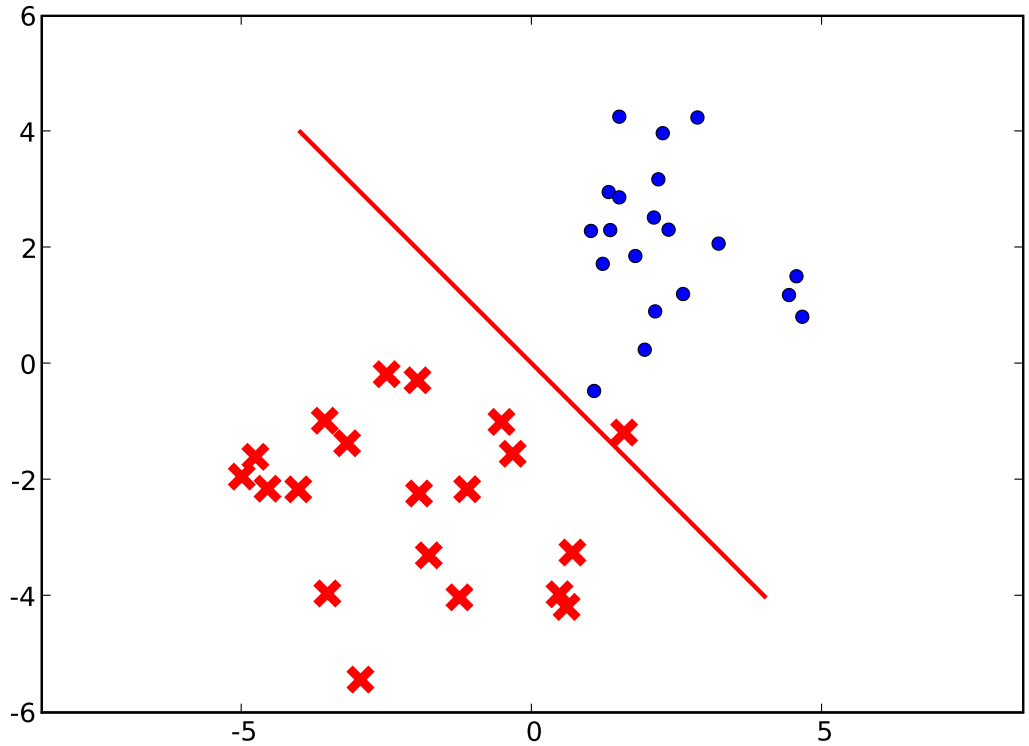
Basic Vocabulary

- We'll call each thing that we are classifying a *point*
- Each point is described by a set of numbers that we call *features*
  - It is your job to decide on the features. You need to pick features that help you do your job.
  - For instance, if you are looking for edges, you'll probably want to look at the response of different derivative filters.
- In the graph below, every point is described by two features.
- The point at index i in the dataset will be described as $\mathbf{x}_i$



Separating Points Linearly

- In building mathematical models for classifying, we are going to focus on dividing these points with a straight line.
- This is called linear classification



Expressing Linear Separation Mathematically

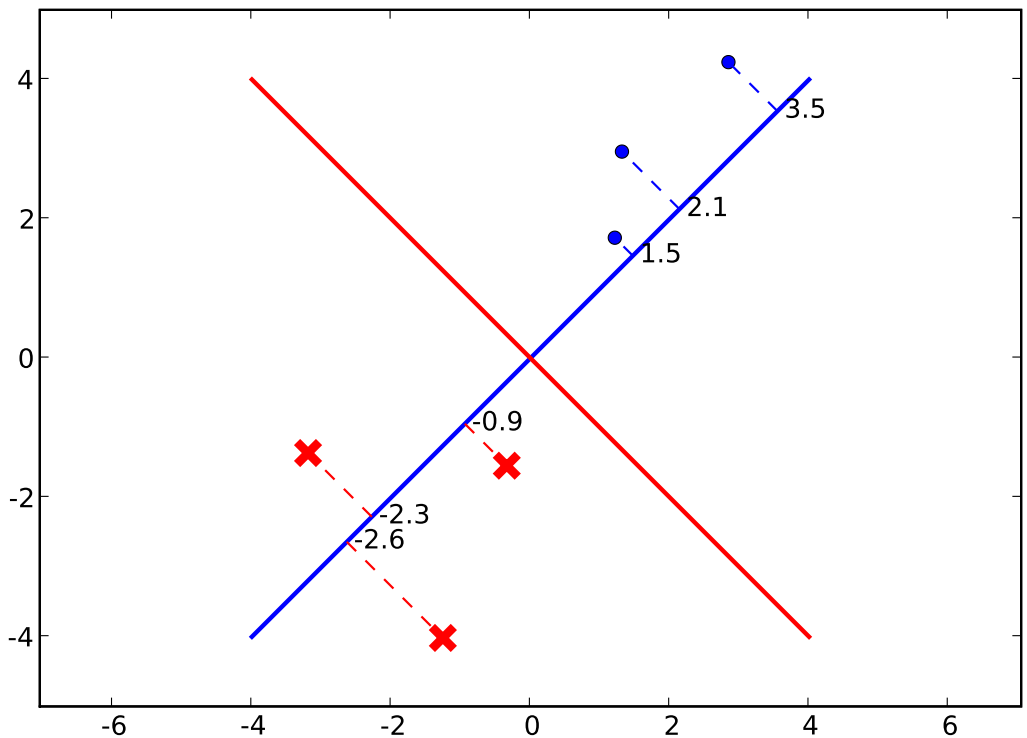
- Given a single point $\mathbf{x} = (x, y)$, we can express the classification of the point as

$$\text{sign}(ax + by + c)$$

  where a, b, and c are constants that define a line. We'll have to choose these somehow.
- This function will return a $+1$ if $ax + by + c$ is positive and -1 otherwise.

How this Translates Graphically

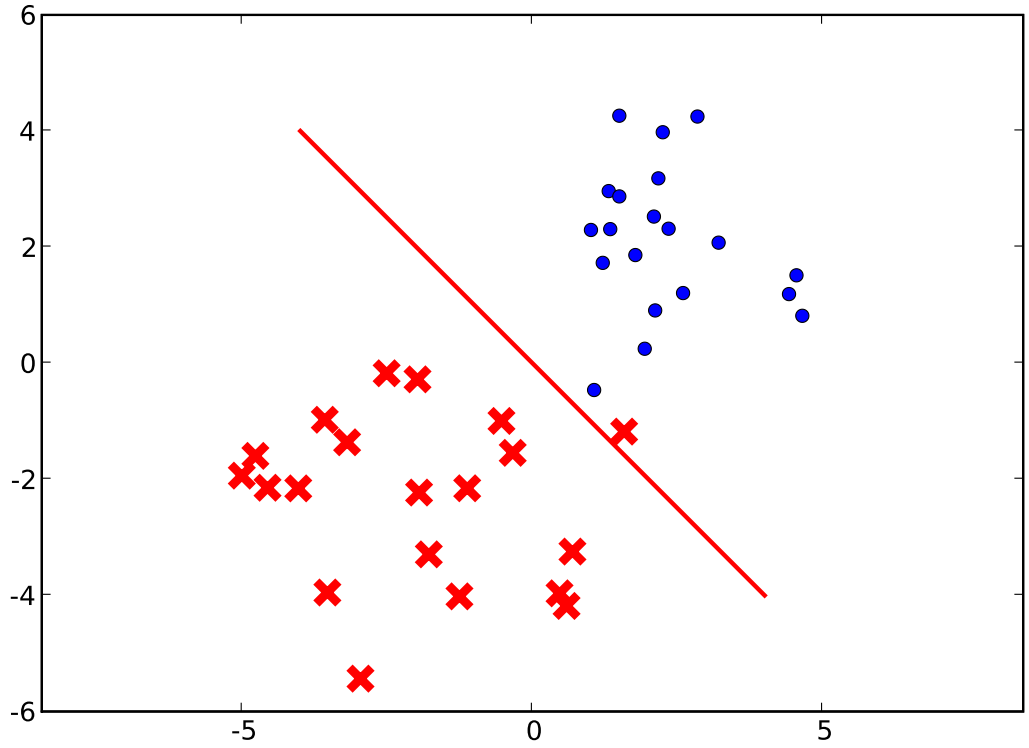
- Effectively, we are projecting every point onto a line.
- Every point projects to some point on the line. The sign of the location along the lines determines the classification of the point



How can we find the separating line?

- This separating line can be found by looking at the line

$$ax + by + c = 0$$

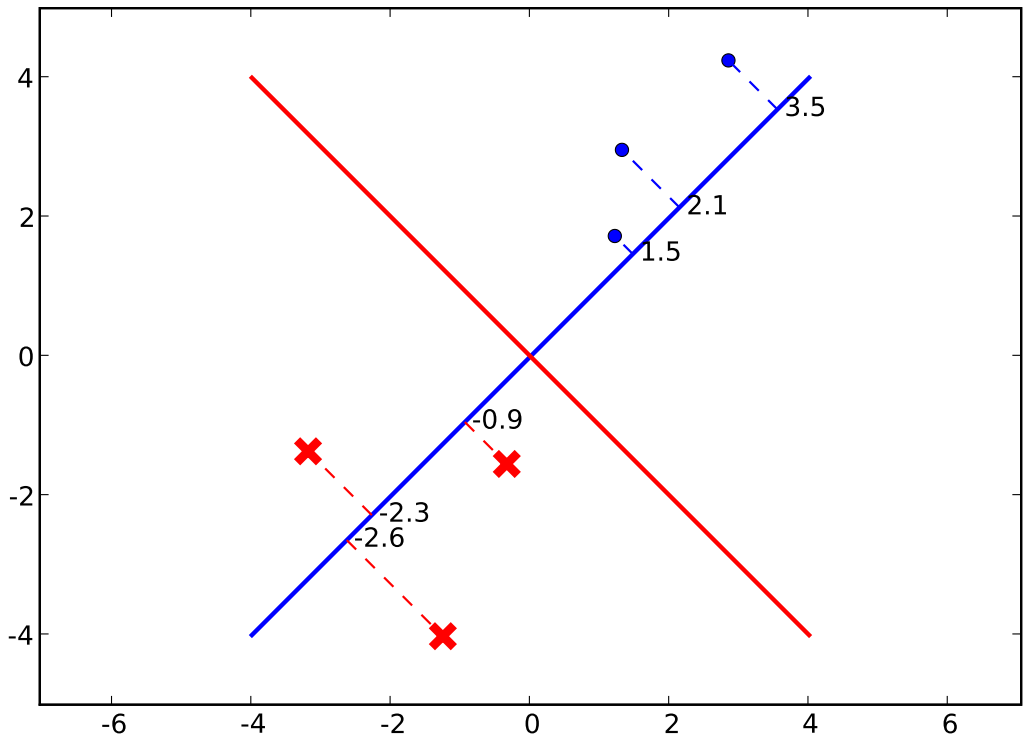


How do we get the parameters of this line?

- We'll find the parameters using a machine learning approach
- We'll form a *training set* of examples, along with the correct classification.
- We'll find the line that best separates the training examples.
- Now, we have to form a set of mathematical steps for finding the line that "best separates" the training set.

Optimizing the parameters of the line

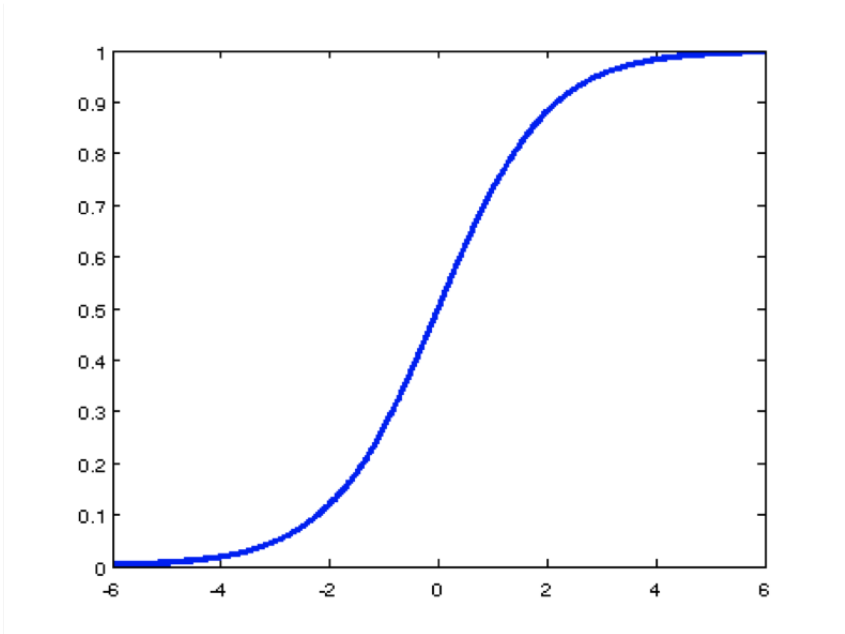
- Notice that the points that are the farthest from the red separating line have the largest response.
- In some sense, we can be more confident in a point's classification as the point gets farther from the separating line.
- So, the bigger the magnitude of a response, the more confident in the classification we can be.



Looking at the confidence in classification

- We can translate this response into a probability.
- We will use the *logistic function* to transform the response into a probability of the correct classification
- We will assume that each point will be labeled with a label l. l can take values $+1$ or -1.

$$P[l = +1|\mathbf{x}] = \frac{1}{1 + \exp(-(ax + by + c))}$$



Finding the line parameters

- Now, for any set of line constants, we can find out the probability assigned to the correct label of each item in the training set.

$$\prod_{i=1}^{N} P[l_i|\mathbf{x}_i] = \prod_{i=1}^{N} \frac{1}{1+\exp(-l_i(ax_i+by_i+c))}$$

- We've inserted l_i into the exponent because, if $l_i = -1$

$$P[l=-1|\mathbf{x}] = \frac{1}{1+\exp((ax+by+c))}$$

- We multiply the probabilities because we believe the points are drawn independently.
- Note that for each item, this number tells us how much the model defined by that particular line believes that the ground-truth right answer is the actual right answer.

Finding the line

$$\prod_{i=1}^{N} P[l_i|\mathbf{x}_i] = \prod_{i=1}^{N} \frac{1}{1+\exp(-l_i(ax_i+by_i+c))}$$

- This is also called the likelihood of the data
- Ideally, this likelihood should be as close to 1 as possible.
- We can find the parameters of the line by looking for the line parameters that maximize this likelihood.
- In other words, find the set of line parameters that make the right answers have as high a probability as possible.

Finding the line

$$\prod_{i=1}^{N} P[l_i|\mathbf{x}_i] = \prod_{i=1}^{N} \frac{1}{1+\exp(-l_i(ax_i+by_i+c))}$$

- Before going on, we are going to do a quick math trick.
- Because the $\log$ (natural logarithm or $\ln$) function is monotonically increasing (i.e. it is always increasing), the parameters that maximize the above equation will also maximize

$$\log\left(\prod_{i=1}^{N} P[l_i|\mathbf{x}_i]\right) = \sum_{i=1}^{N} -\log\left(1+\exp(-l_i(ax_i+by_i+c))\right)$$

Finding the line

$$\log\left(\prod_{i=1}^{N} P[l_i|\mathbf{x}_i]\right) = \sum_{i=1}^{N} -log\left(1+\exp(-l_i(ax_i+by_i+c))\right)$$

- Similarly, we can multiply this equation by -1 to change from a *maximization* problem to a *minimization* problem
- Leading to a function that we will call a "loss function" or "cost function" and can be denoted as L:

$$L = \sum_{i=1}^{N} \log\left(1+\exp(-l_i(ax_i+by_i+c))\right)$$

- Our goal is to find the line parameters that minimize L.

Minimizing L

- We can minimize L by using its gradient:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial a} \\ \frac{\partial L}{\partial b} \\ \frac{\partial L}{\partial c} \end{bmatrix}$$

- The gradient can be viewed as a vector that points in the direction of steepest ascent.
- So, the negative gradient points in the direction of steepest descent.

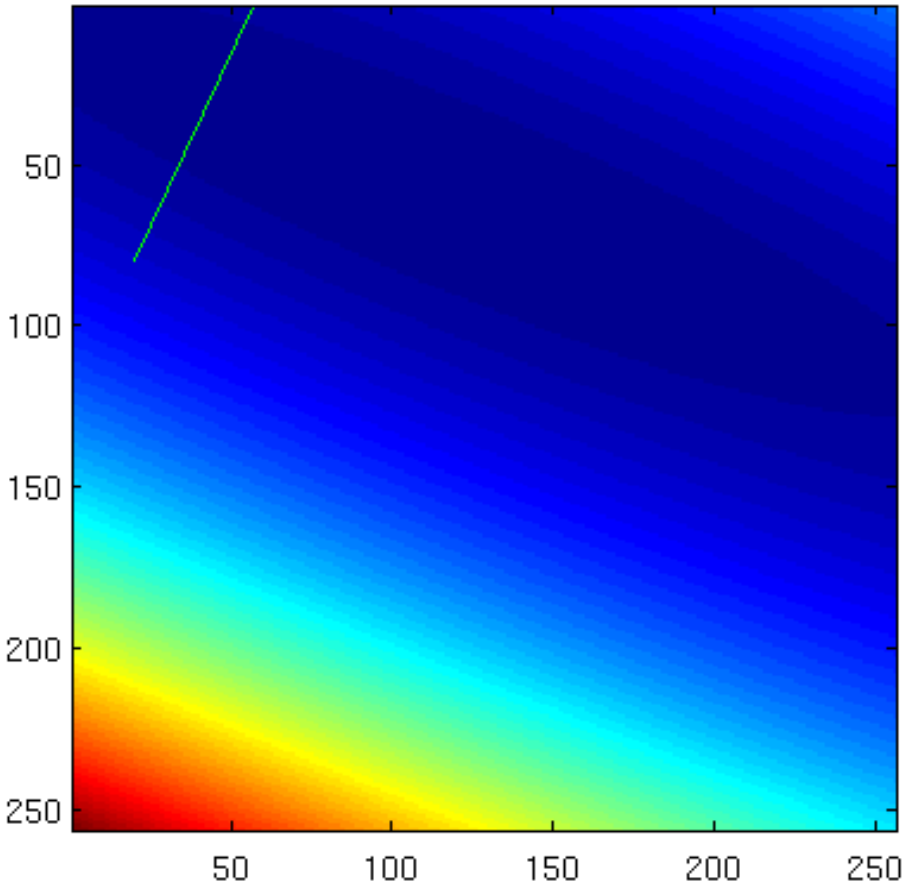
An algorithm for minimizing L

- This leads to a simple algorithm for optimizing the line parameters.
- We'll create a vector

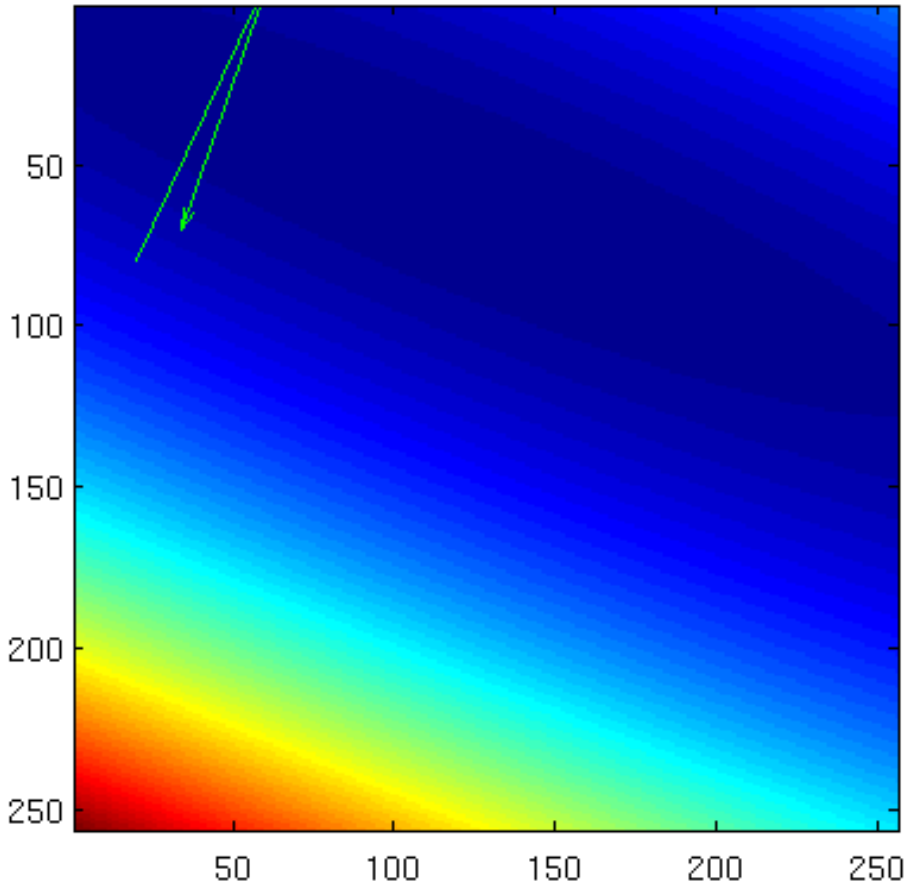
$$\mathbf{p} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- The steps are:
  1. Initialize $\mathbf{p}$ to some value $\mathbf{p}_0$
  2. Repeat these steps:
     - 2.1 Calculate ∇L using the current value of $\mathbf{p}$ (The value of the gradient depends on $\mathbf{p}$!)
     - 2.2 $\mathbf{p} \leftarrow \mathbf{p} + \eta(-\nabla L)$
- We go in the direction of the negative gadient because that this the direction of steepest descent.

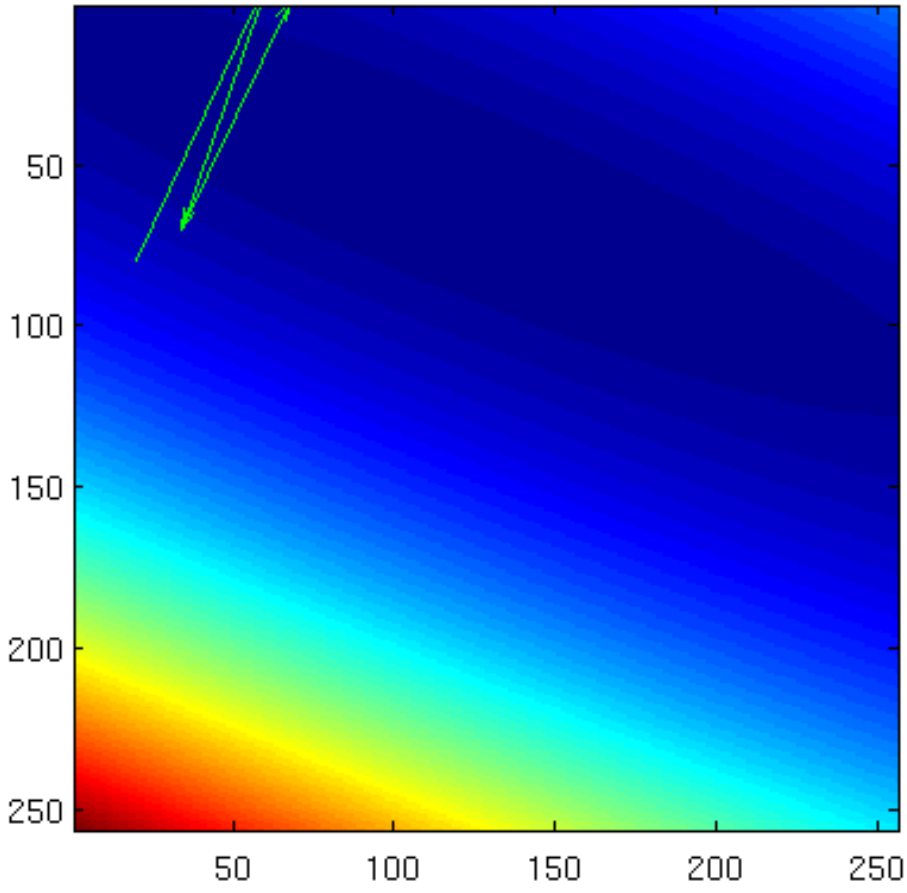
An Example



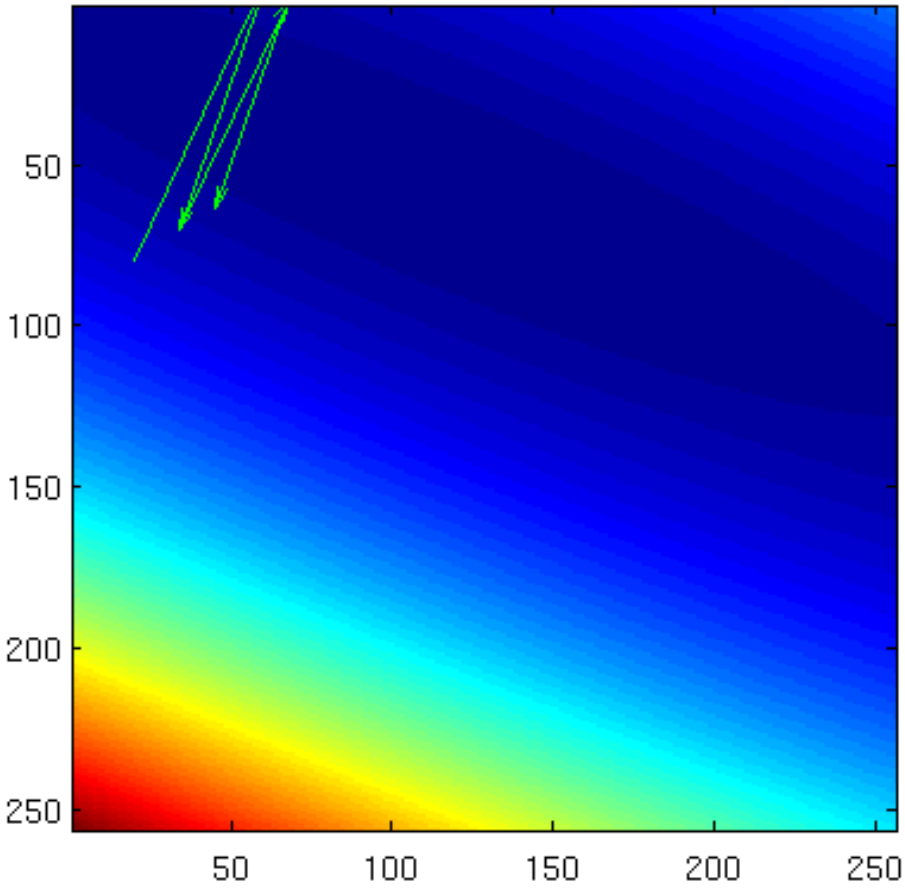
An Example



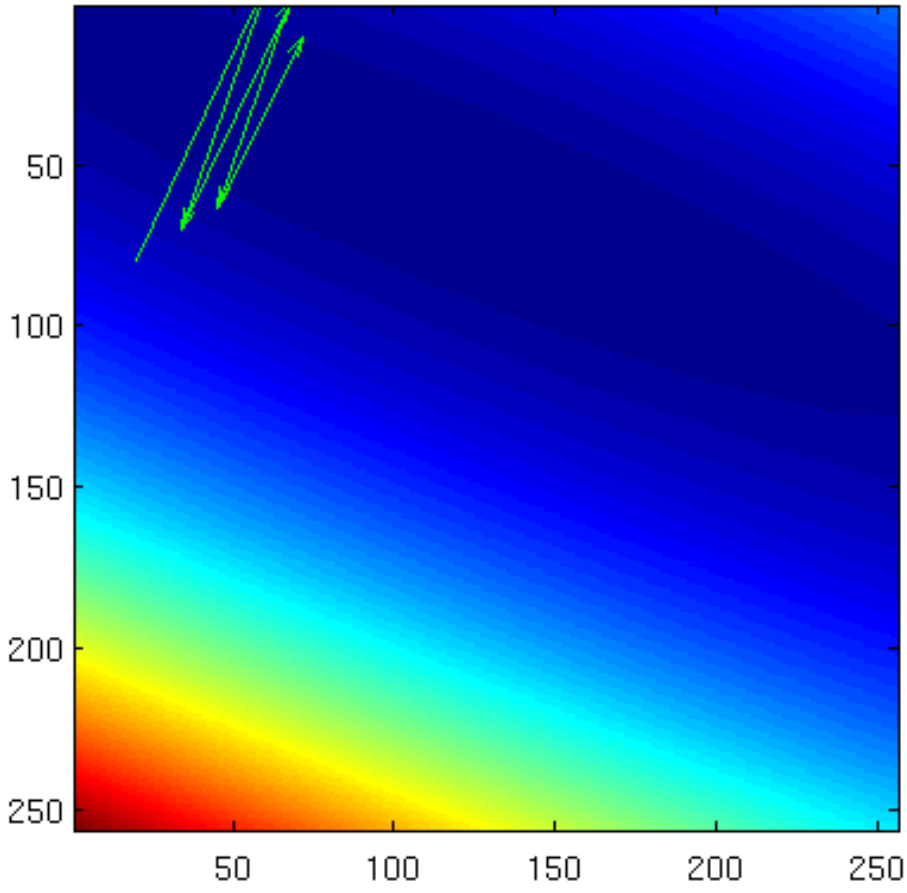
An Example



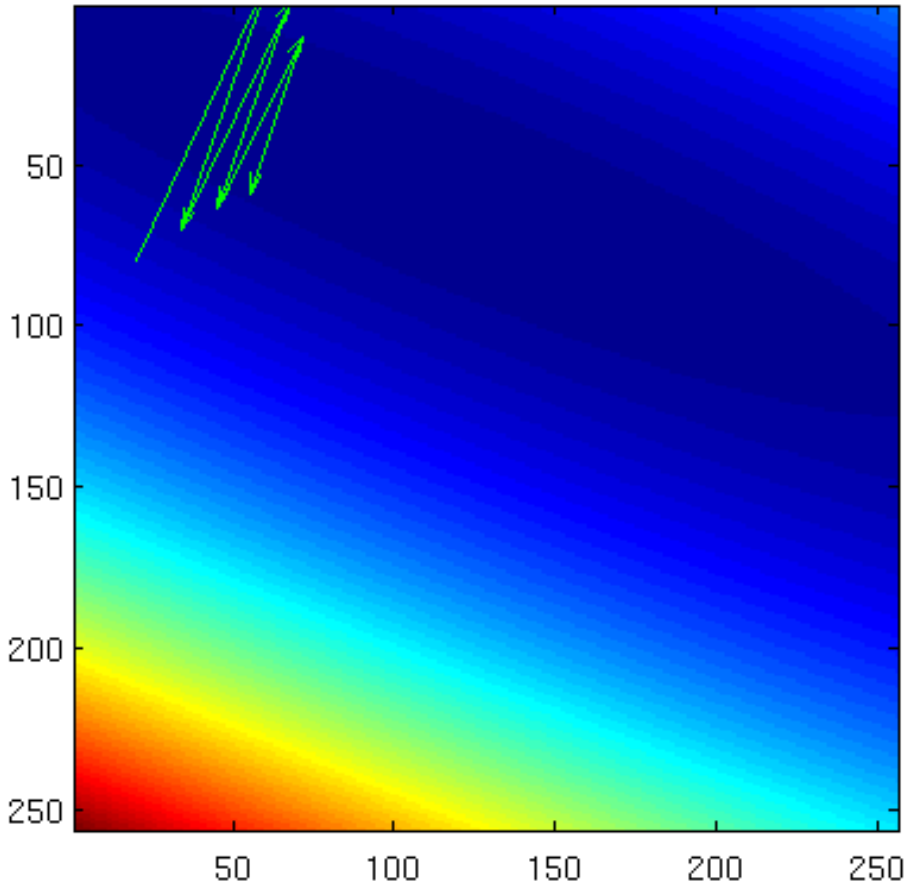
An Example



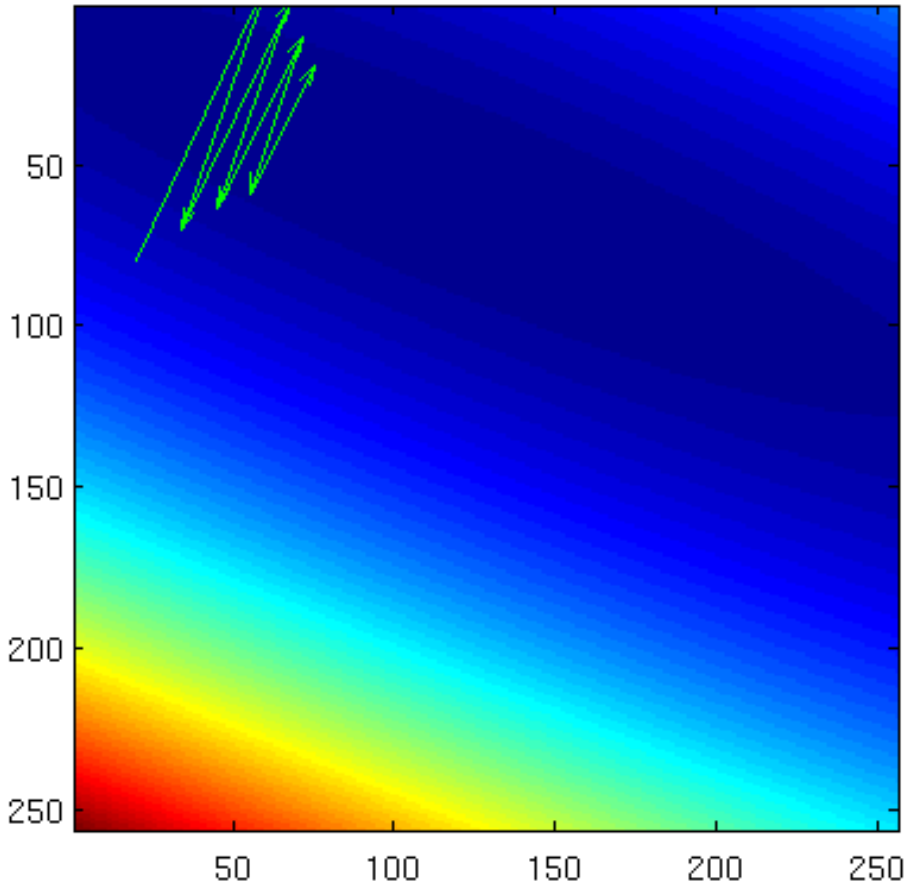
An Example



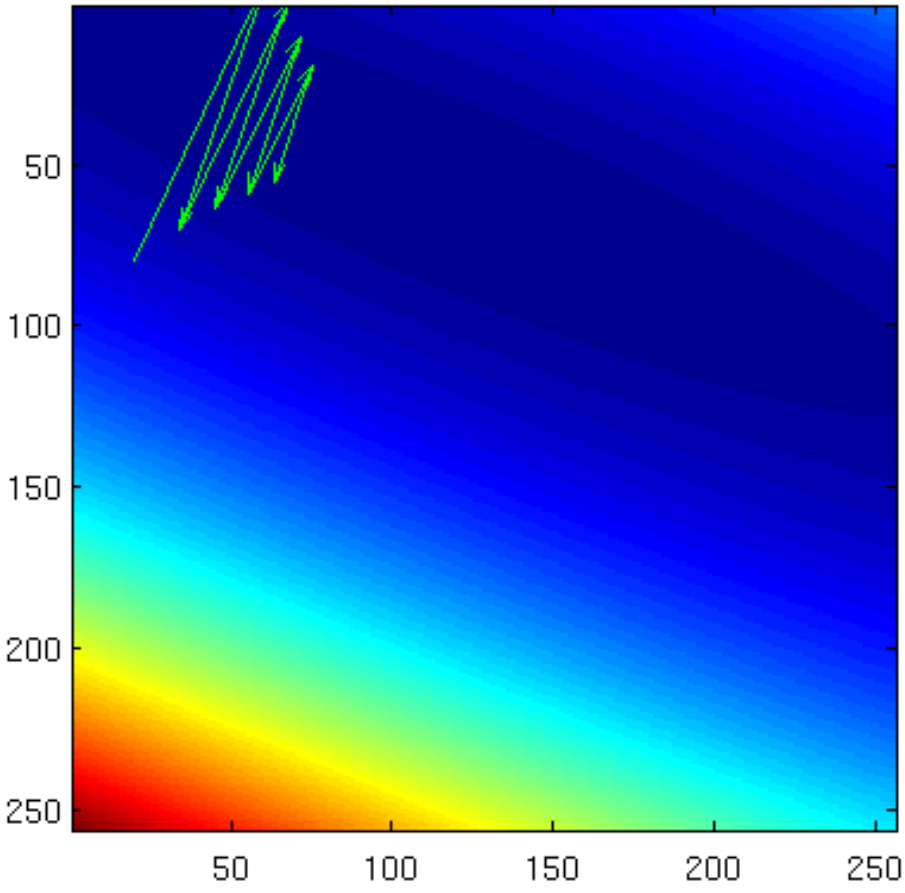
An Example



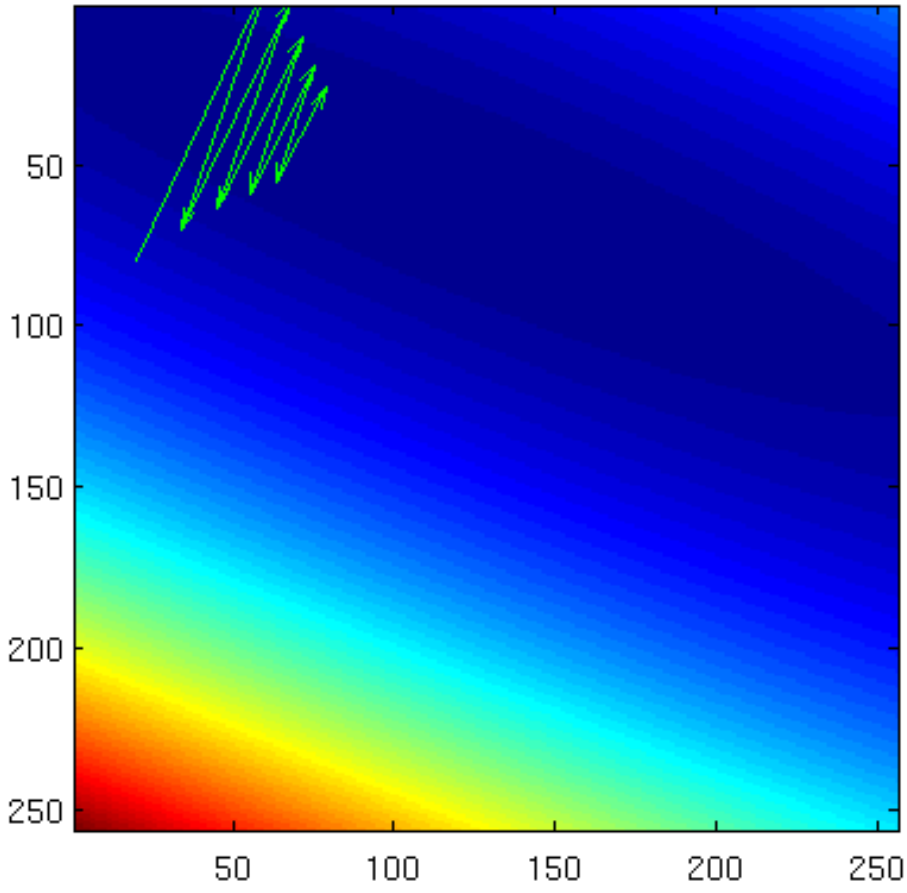
An Example



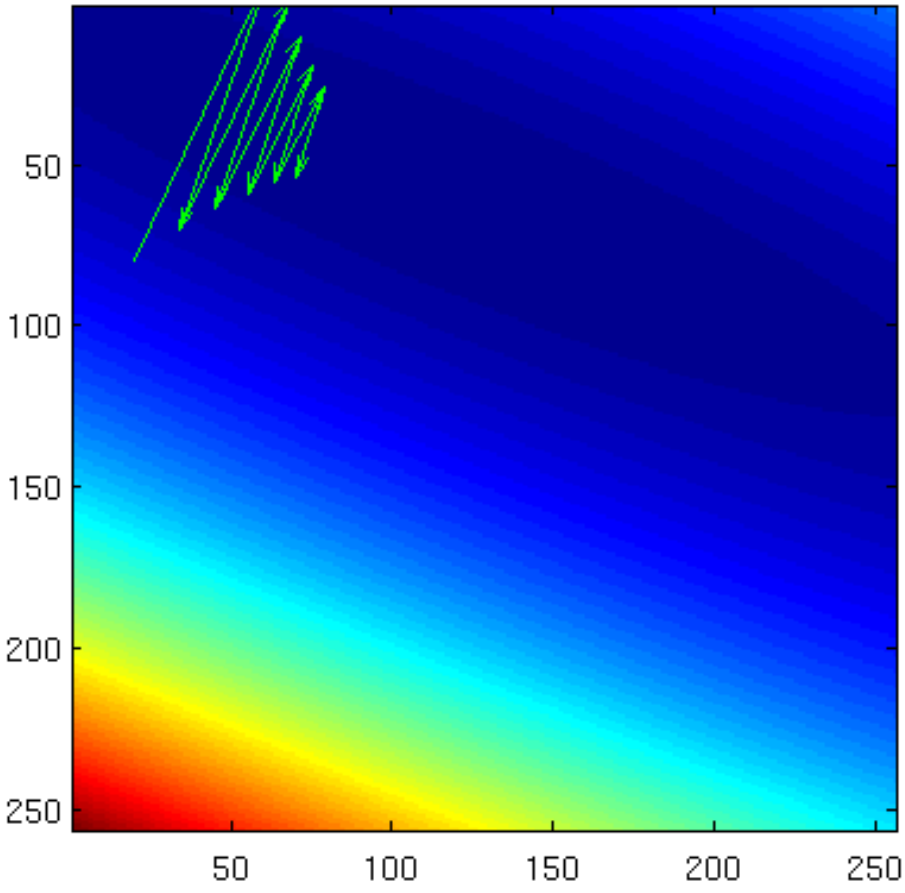
An Example



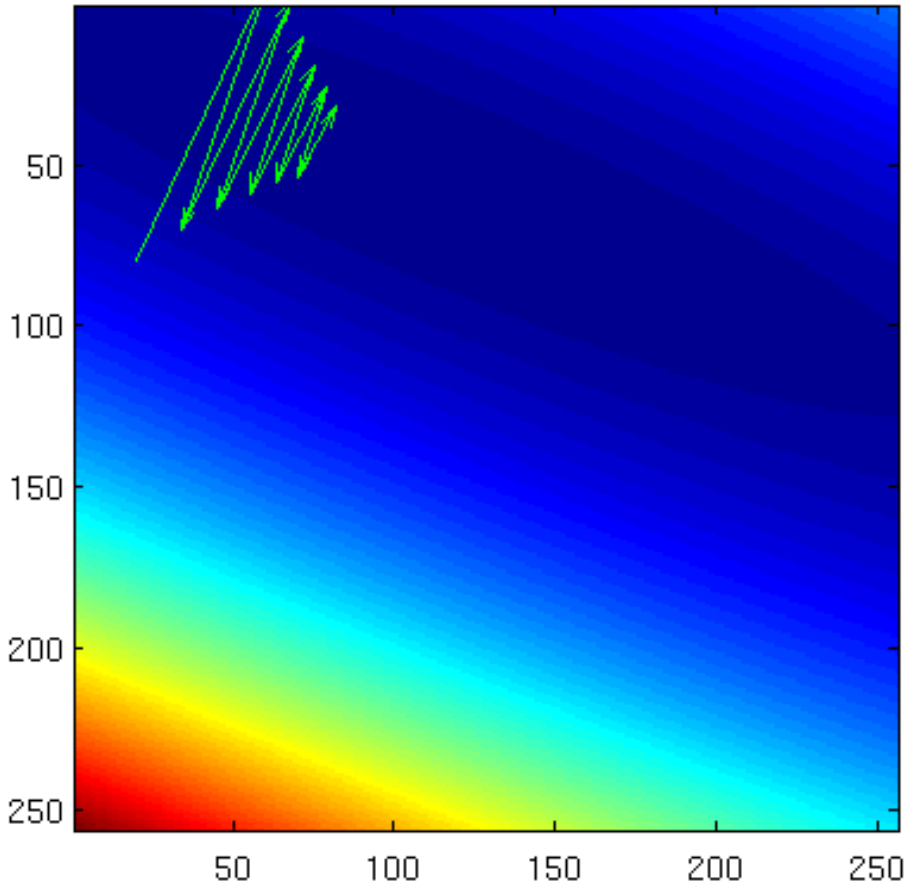
An Example



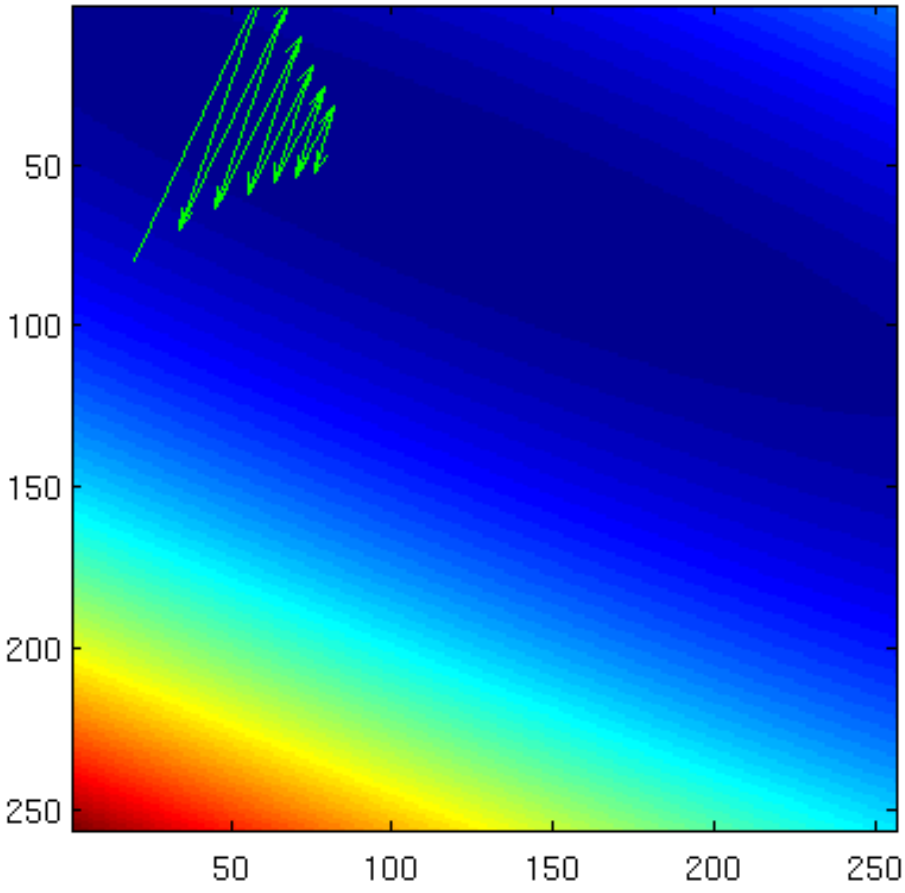
An Example



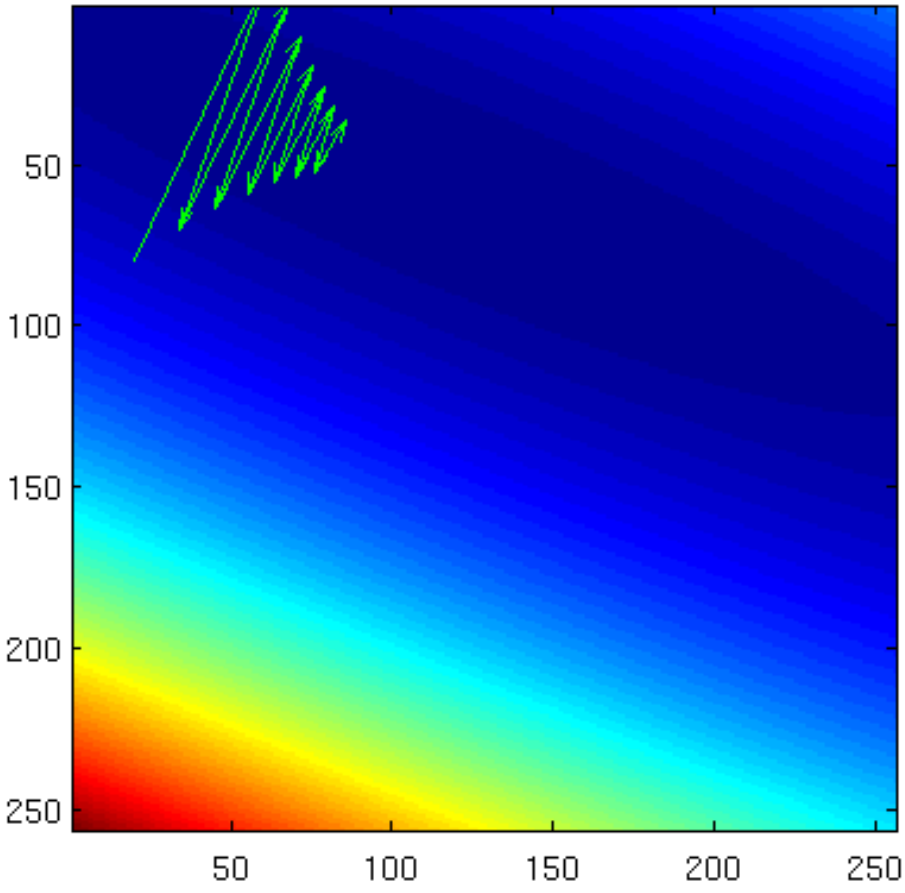
An Example



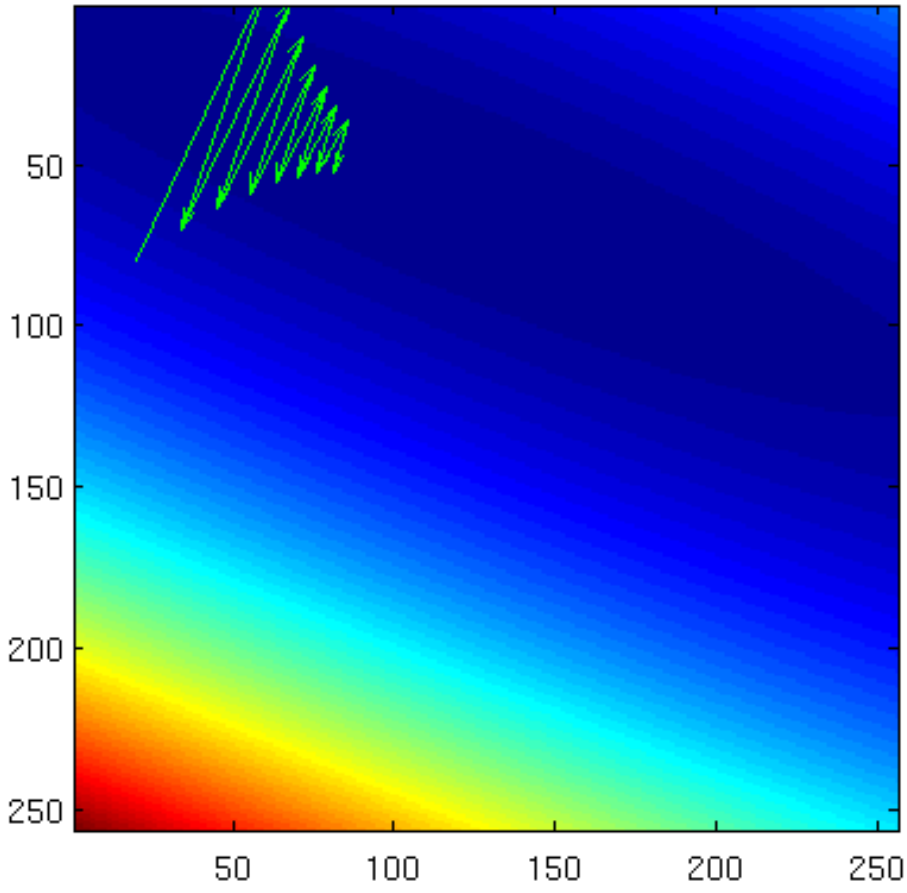
An Example



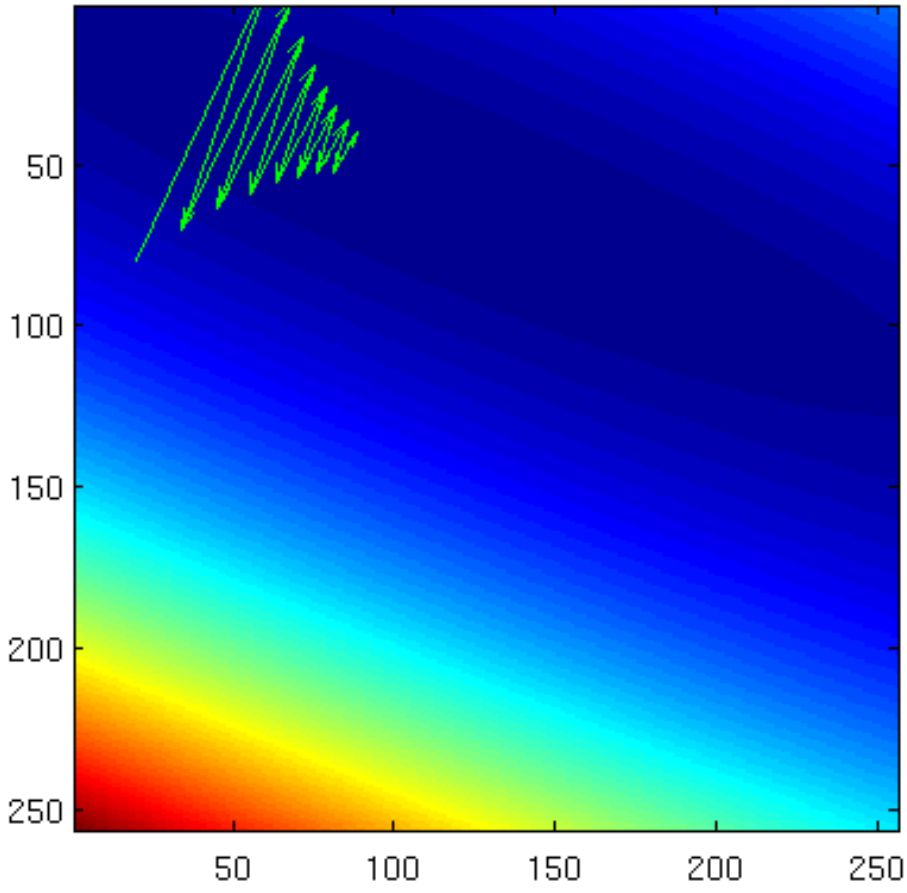
An Example



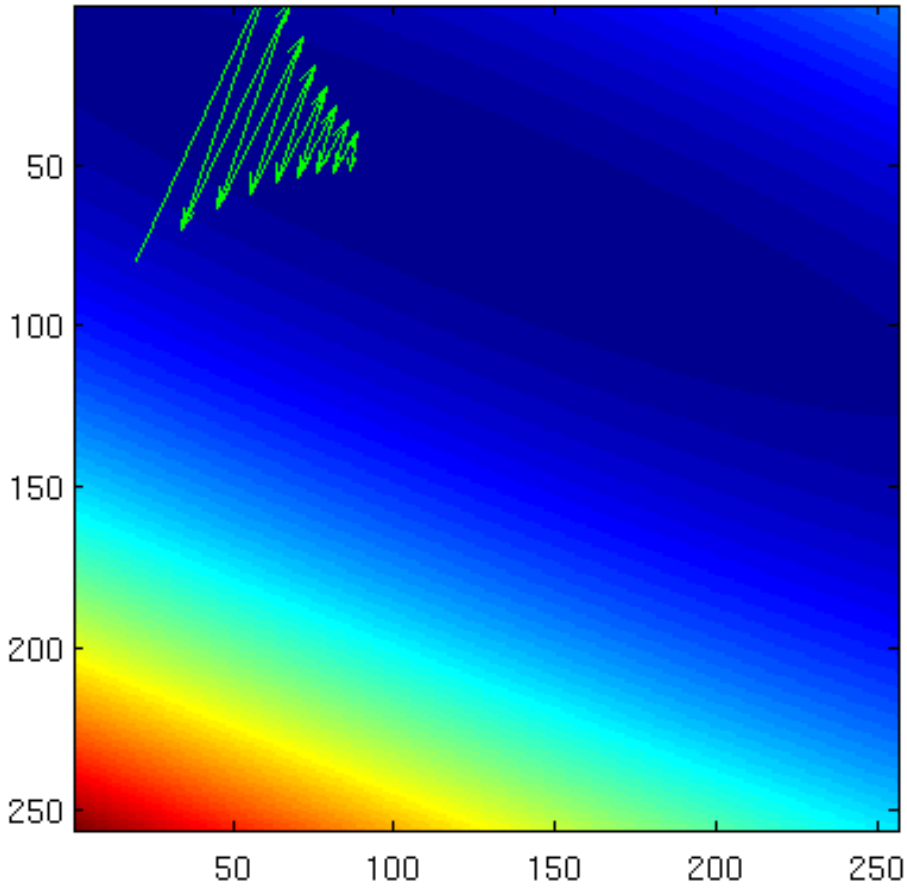
An Example



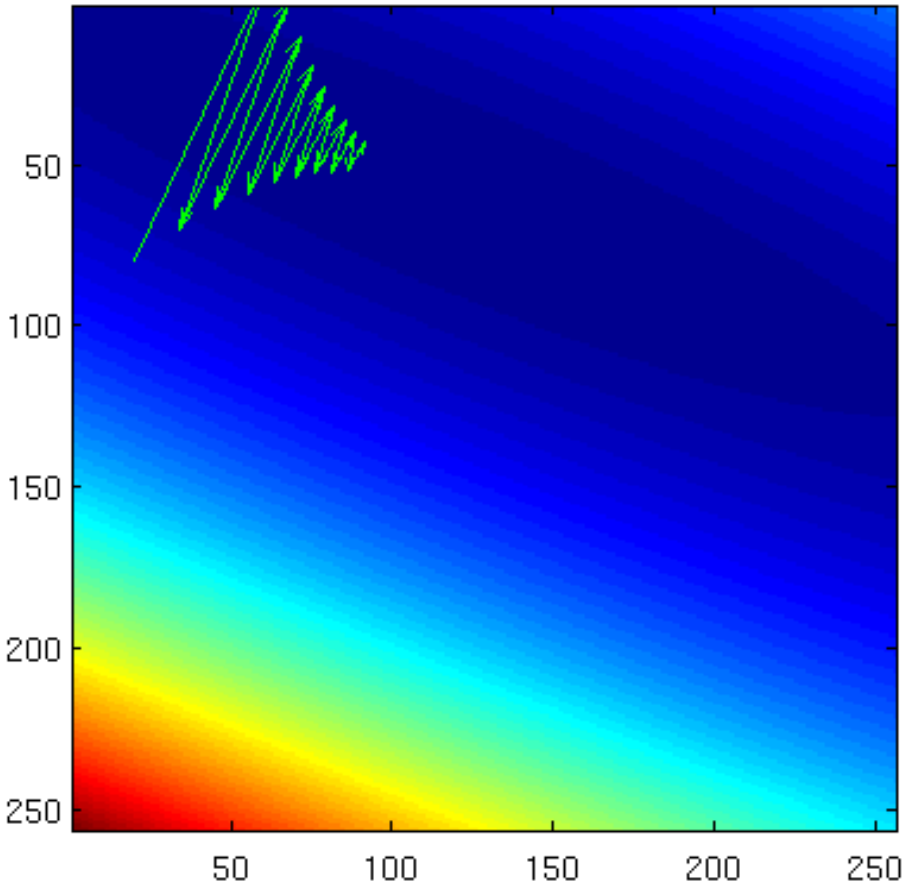
An Example



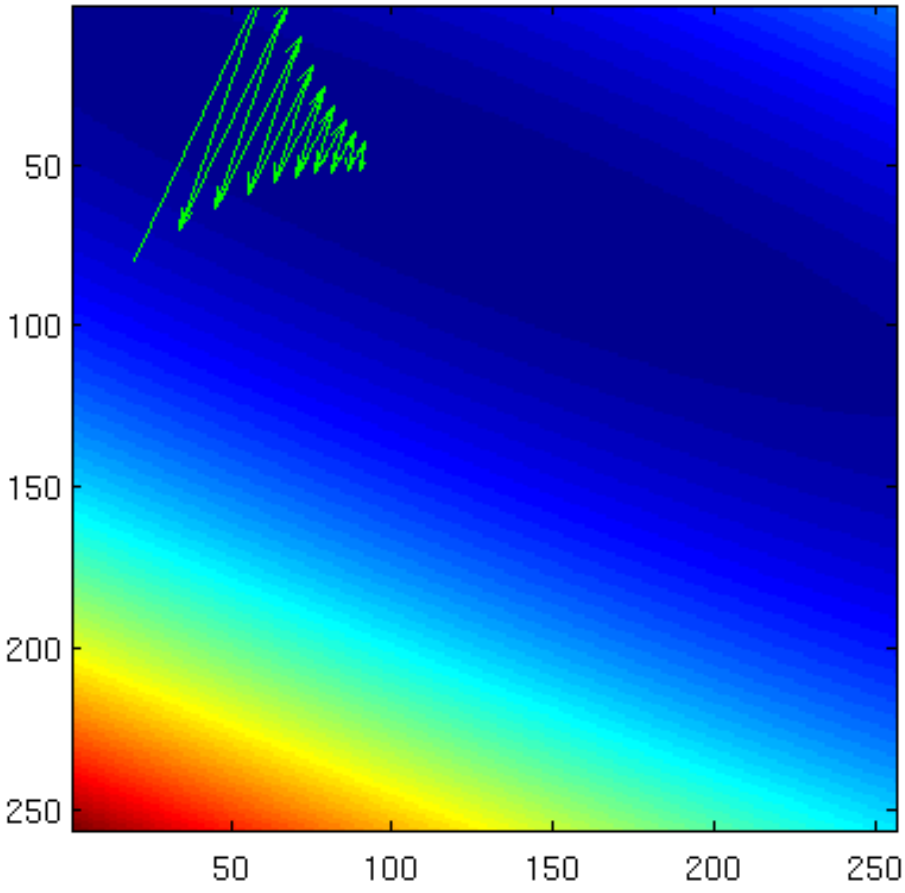
An Example



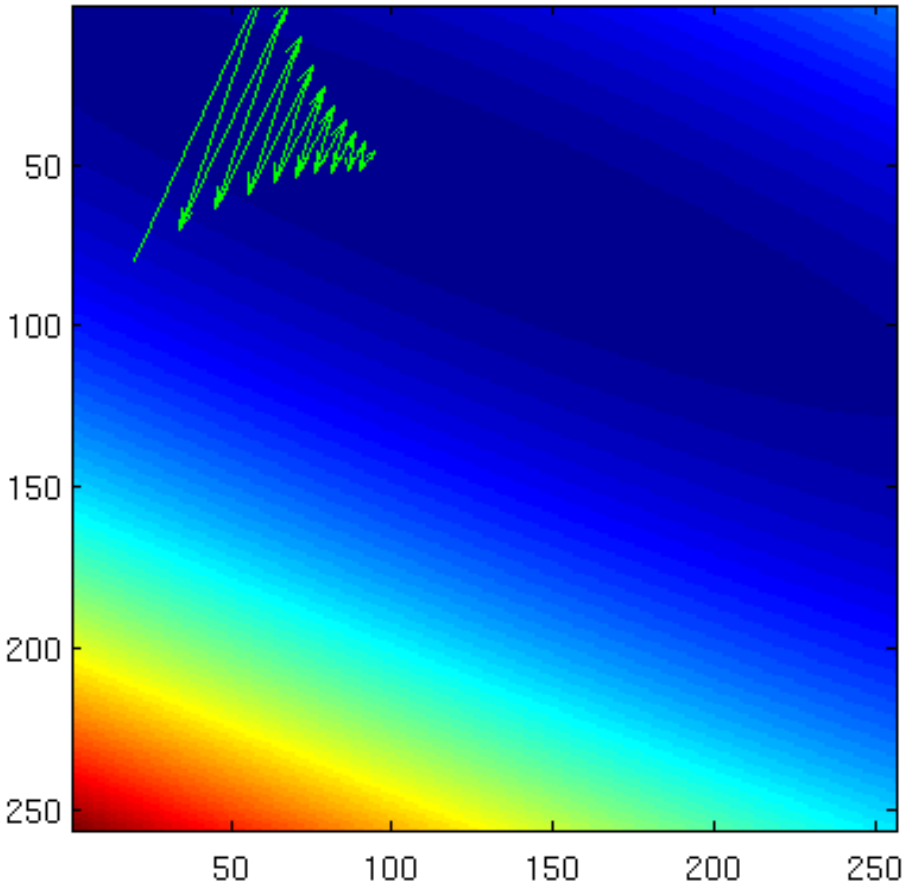
An Example



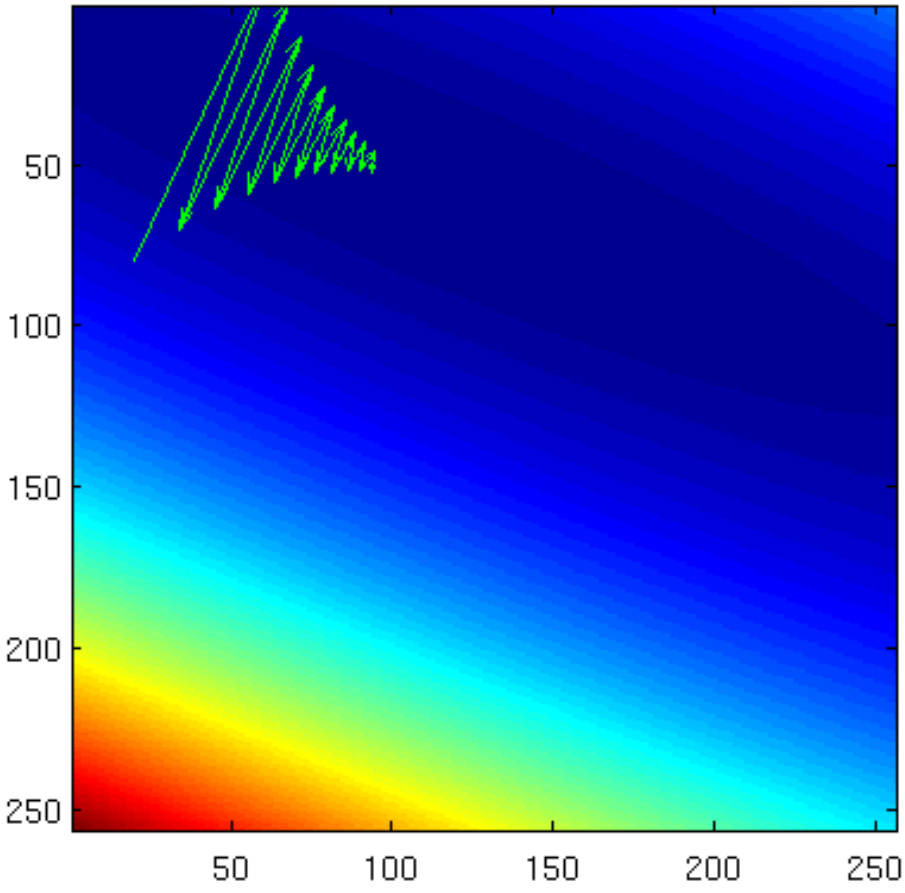
An Example



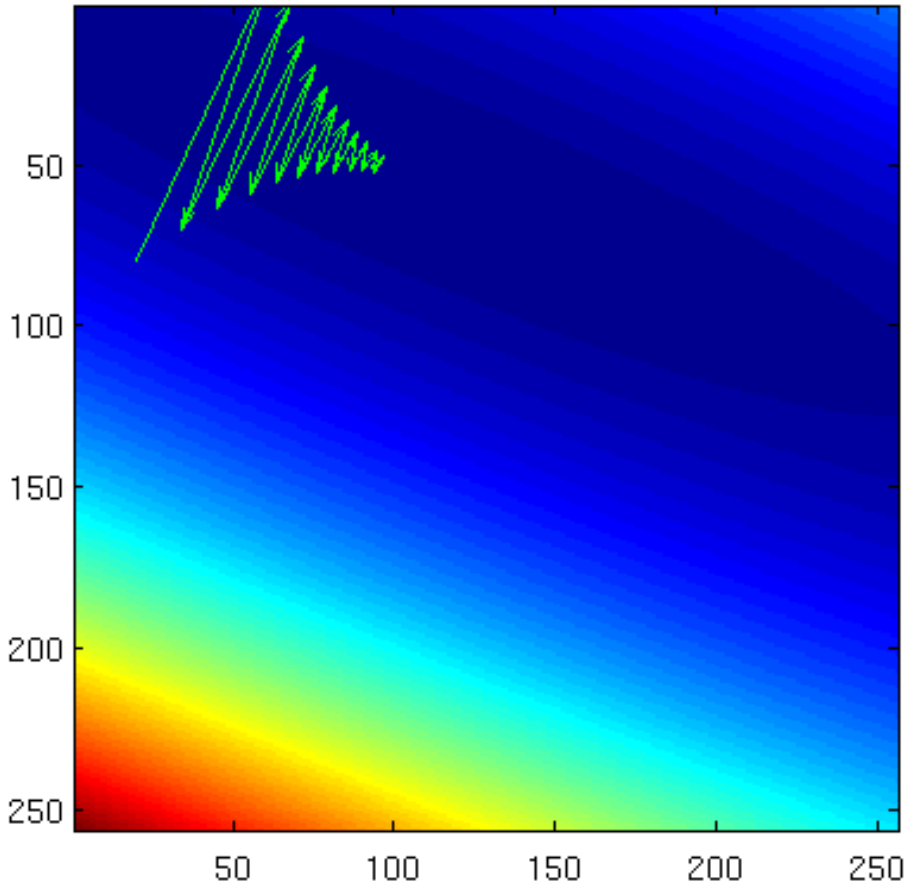
An Example



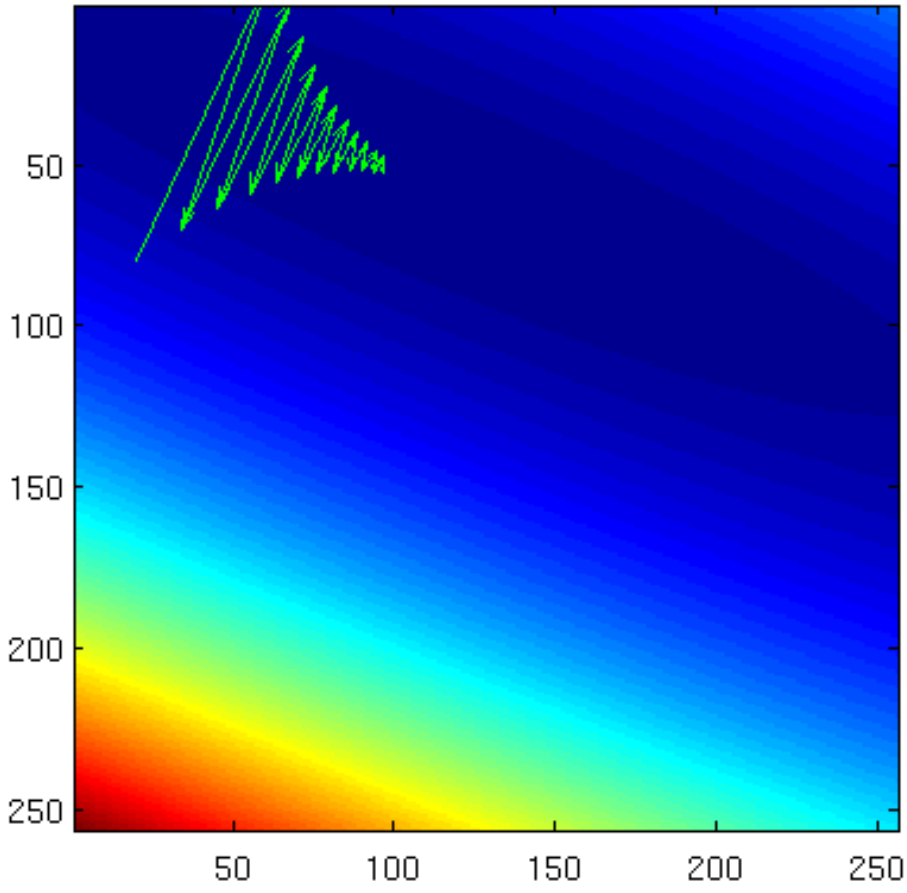
An Example



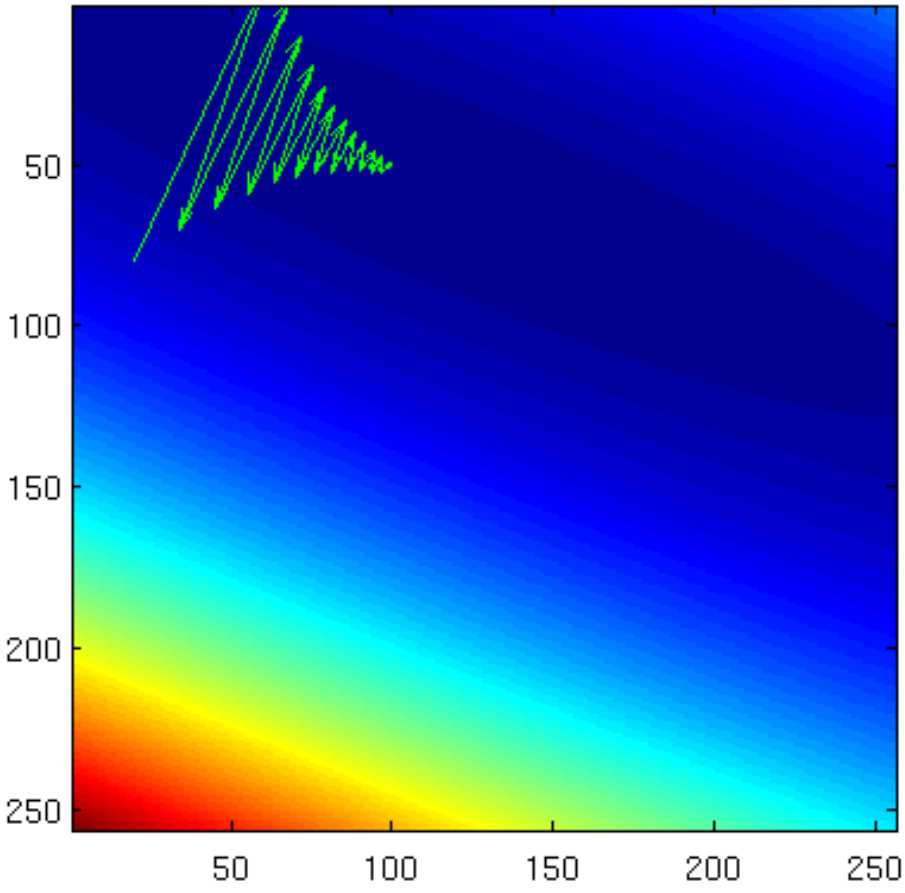
An Example



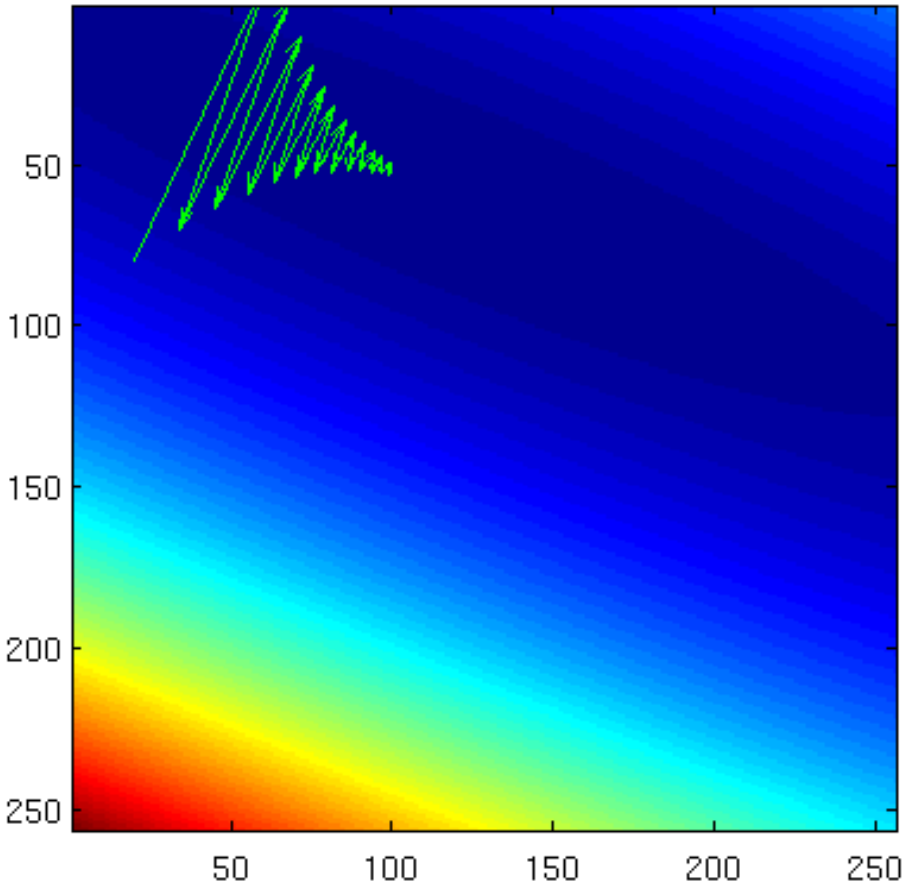
An Example



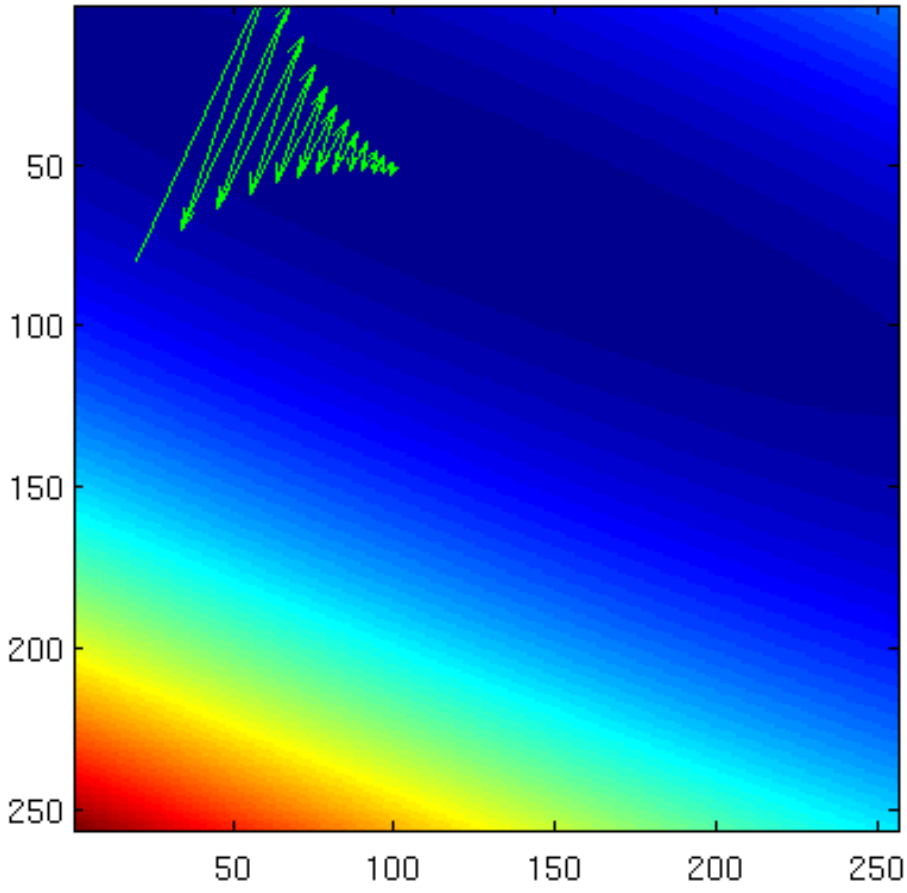
An Example



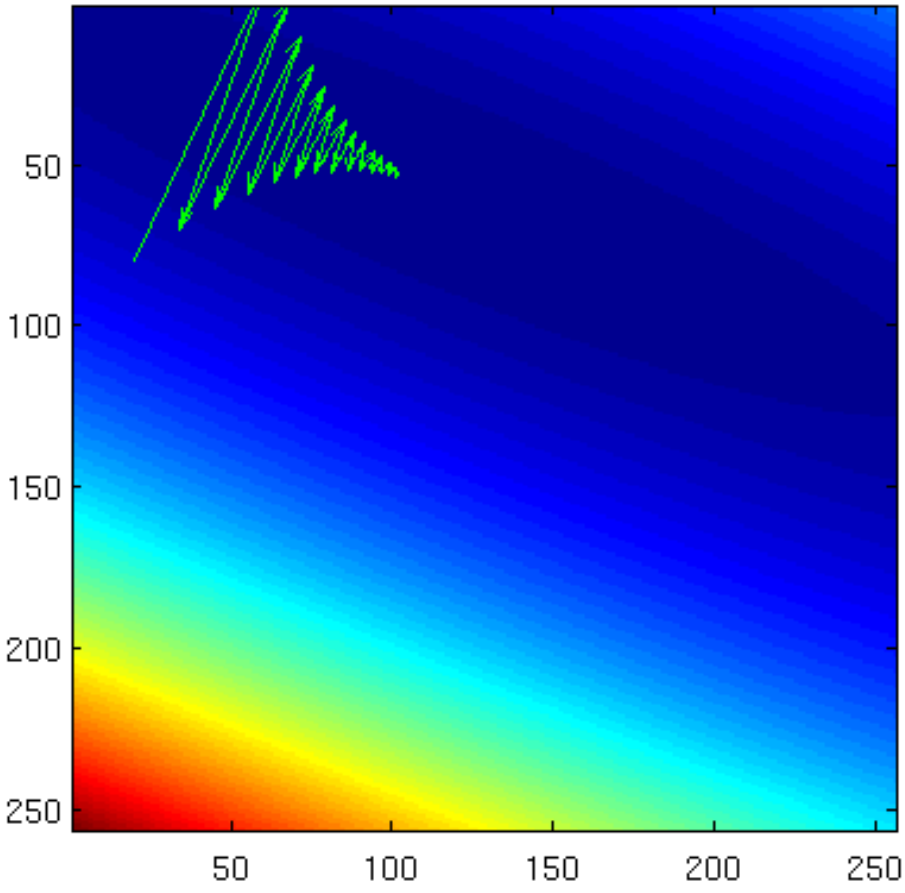
An Example



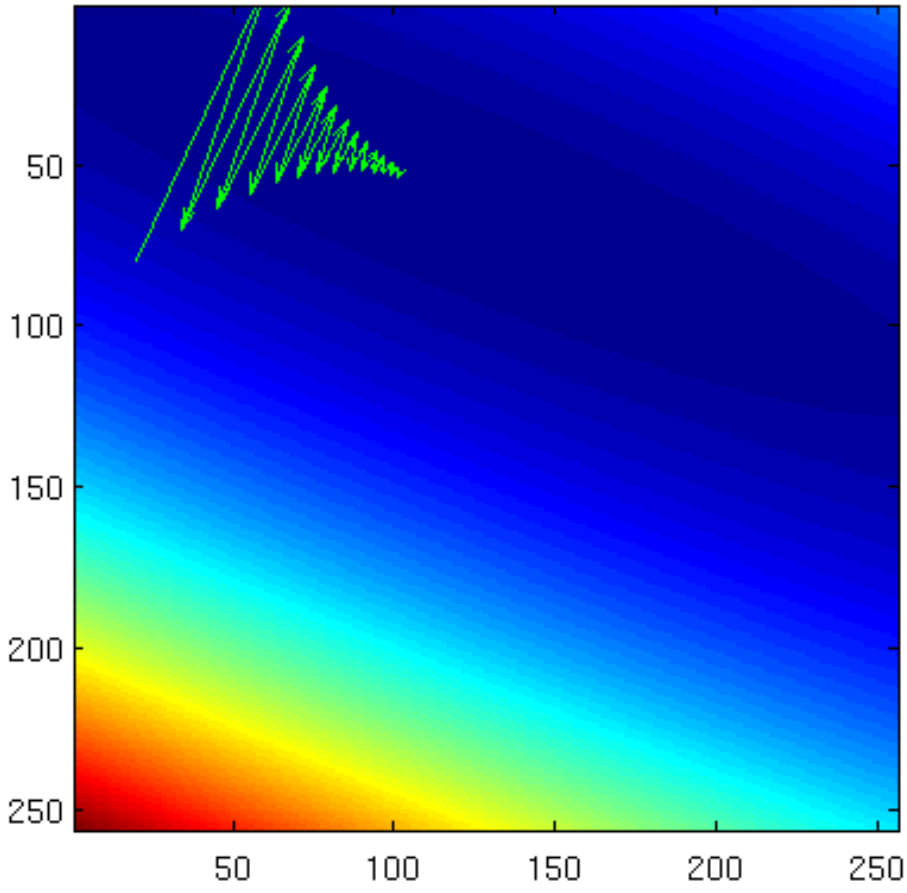
An Example



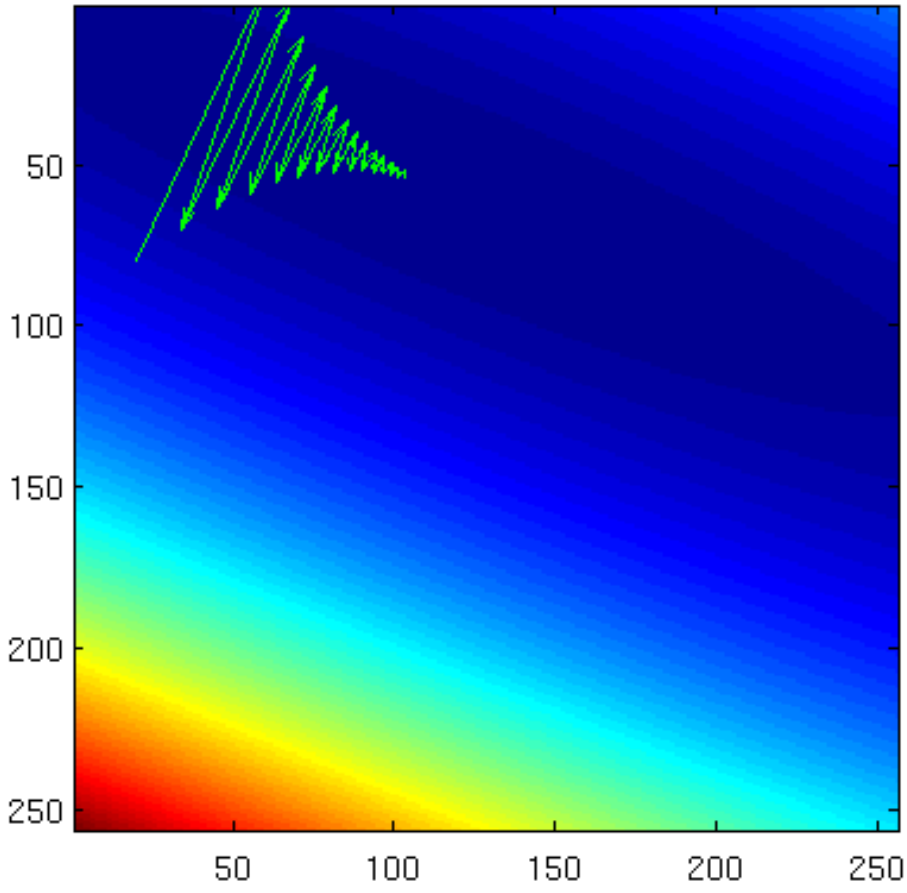
An Example



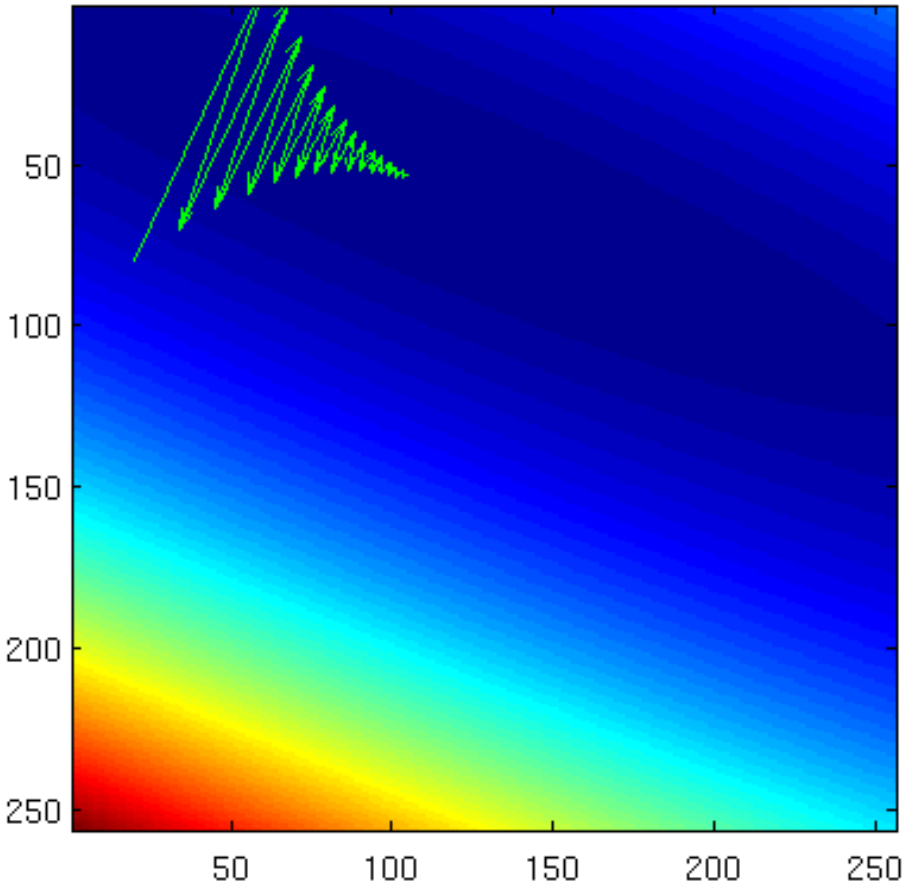
An Example



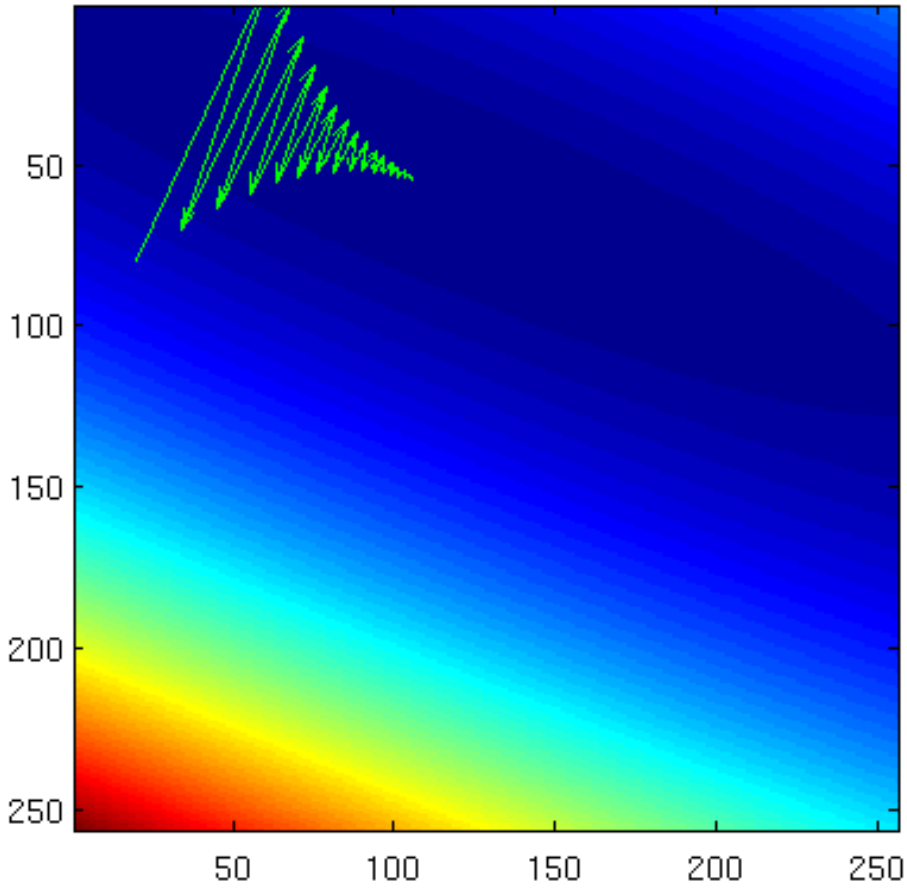
An Example



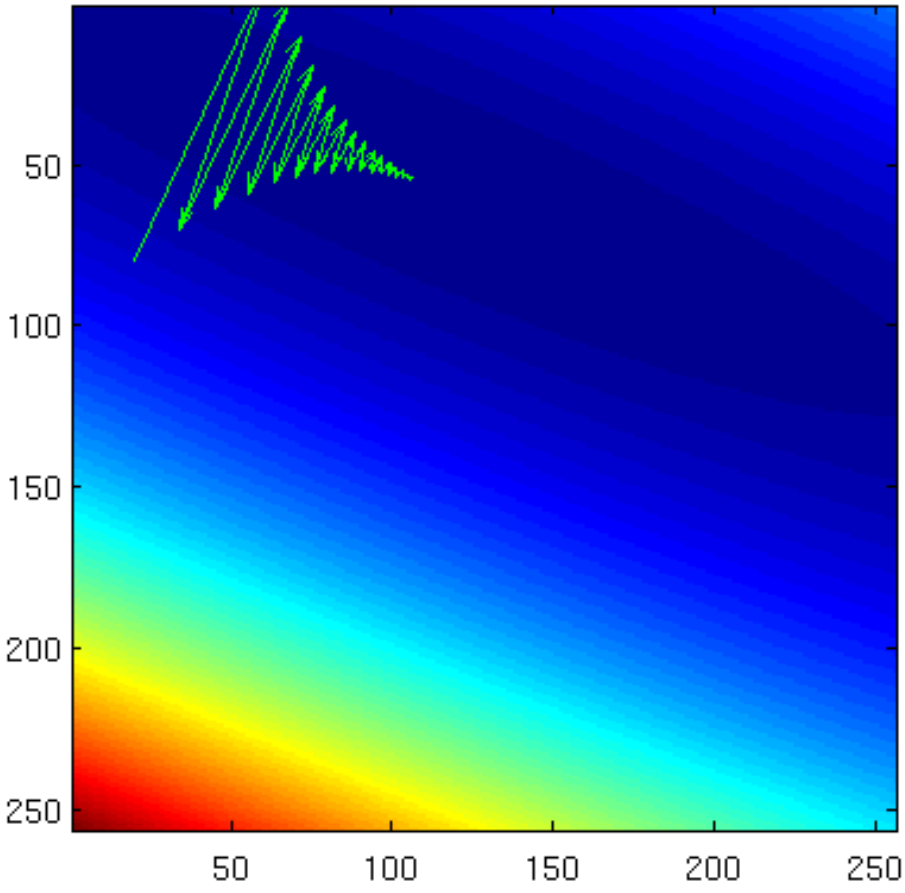
An Example



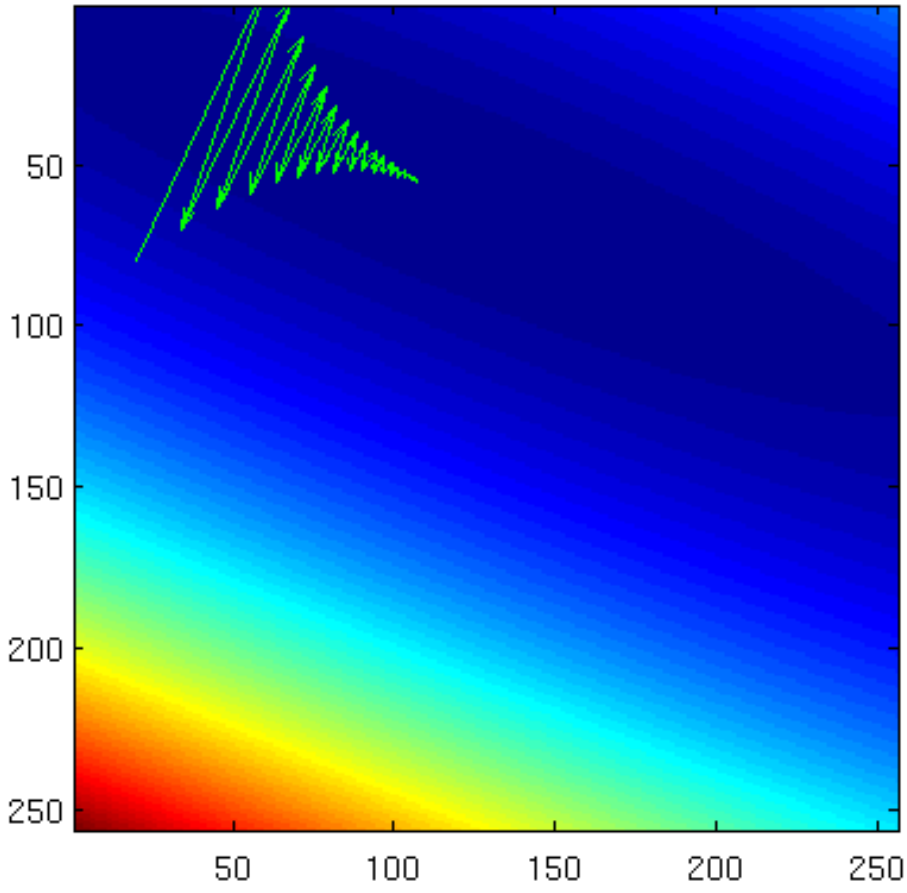
An Example



An Example



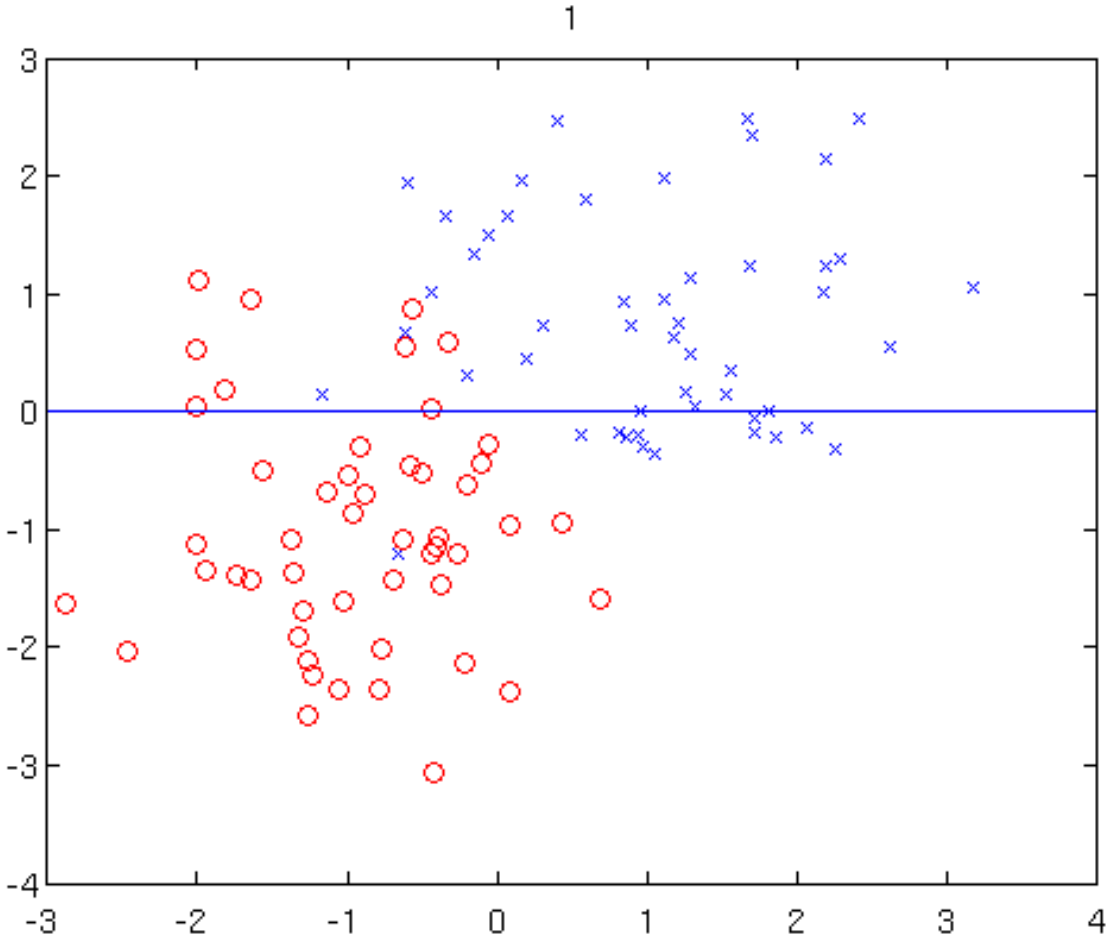
An Example



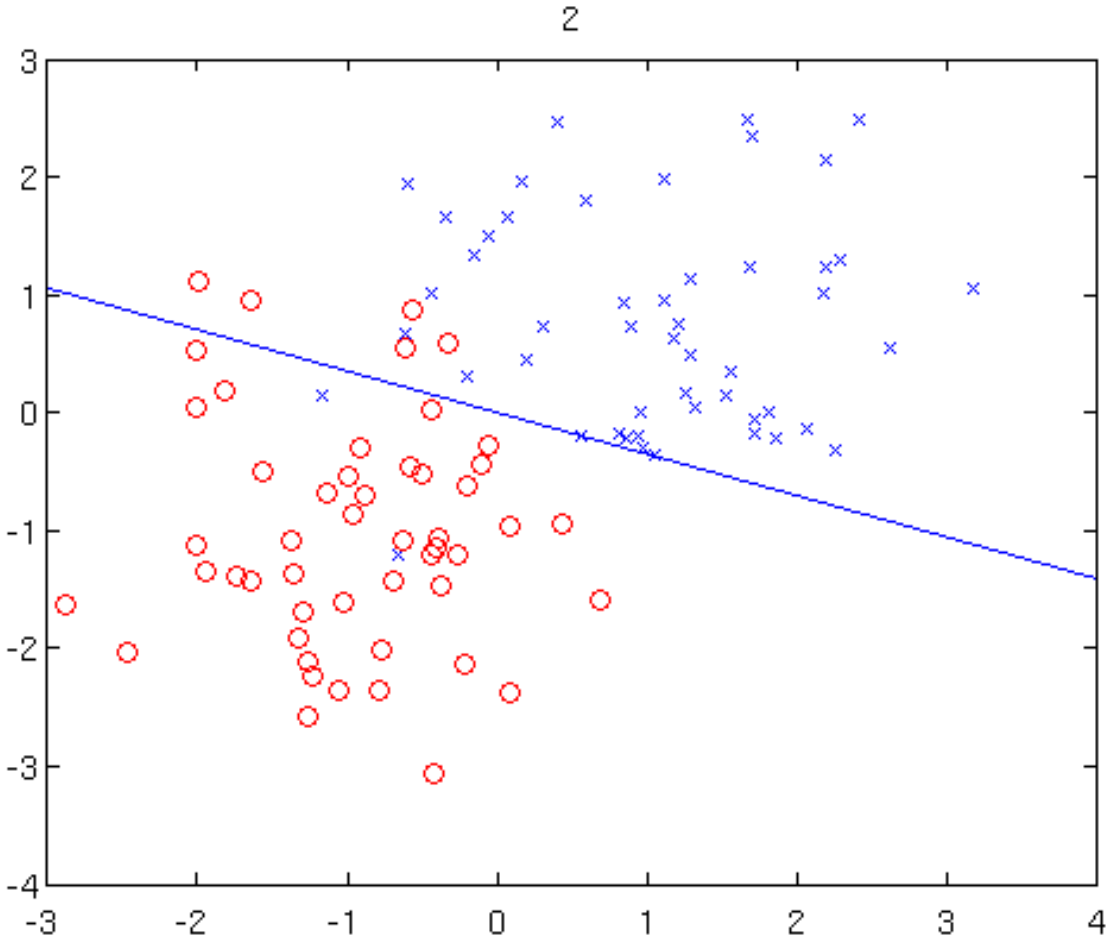
Learning Line parameters

- How does this relate to learning line parameters?
- We differentiate with respect L and optimize.
- The following few slides show how the line changes as the parameters change.

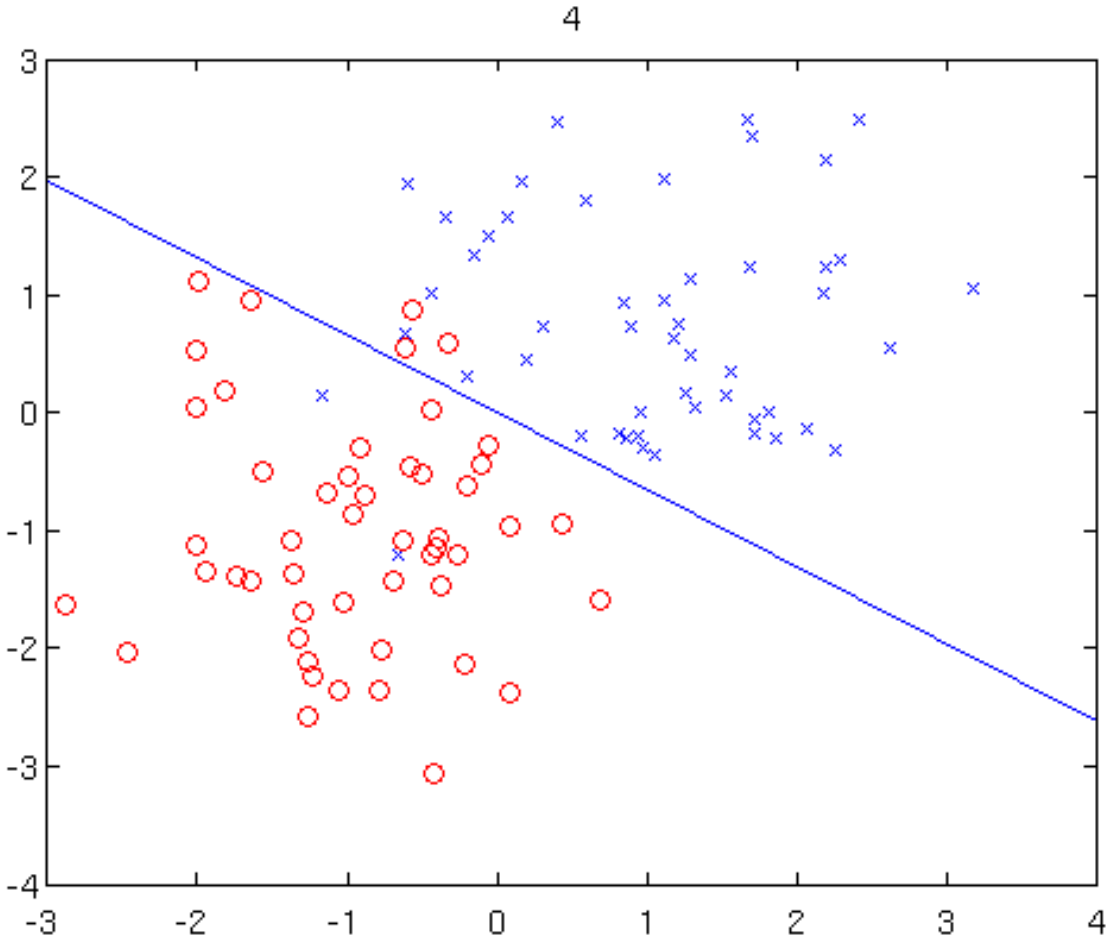
An Example



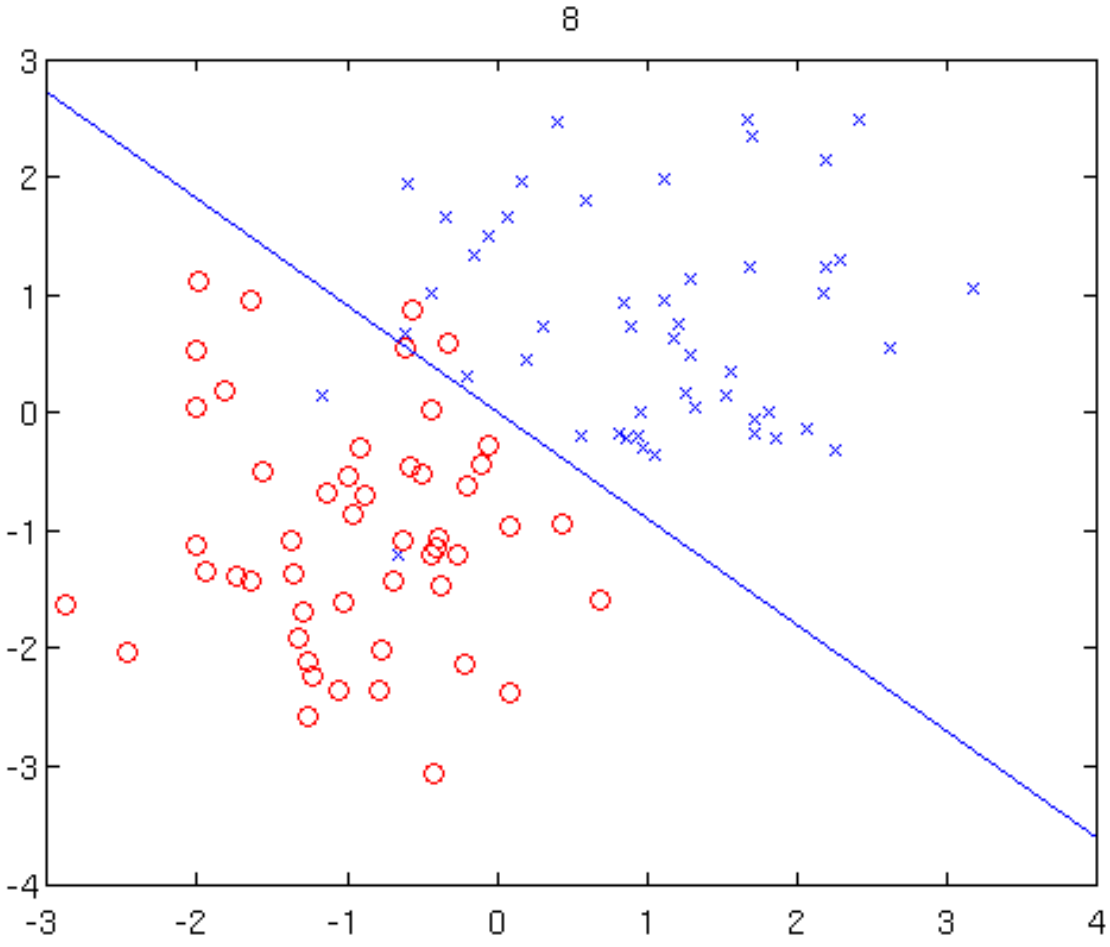
An Example



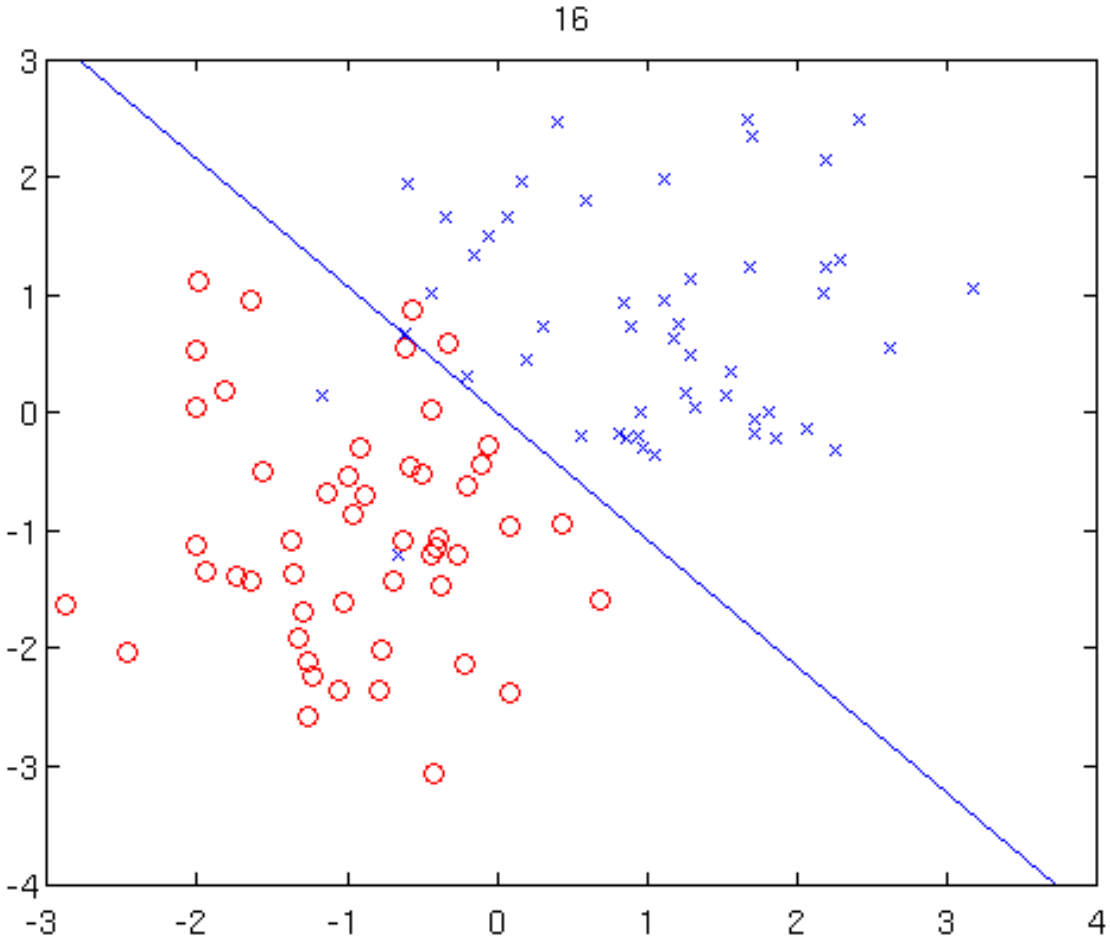
An Example



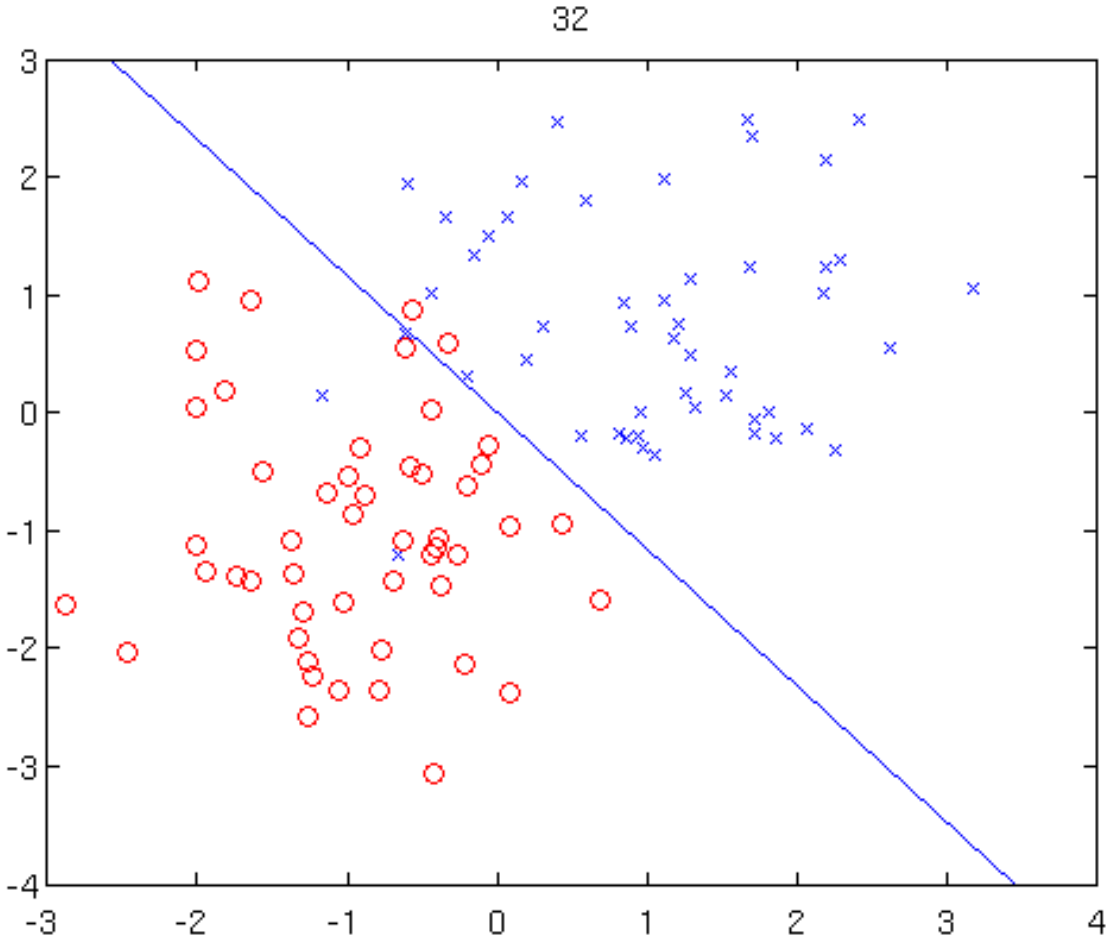
An Example



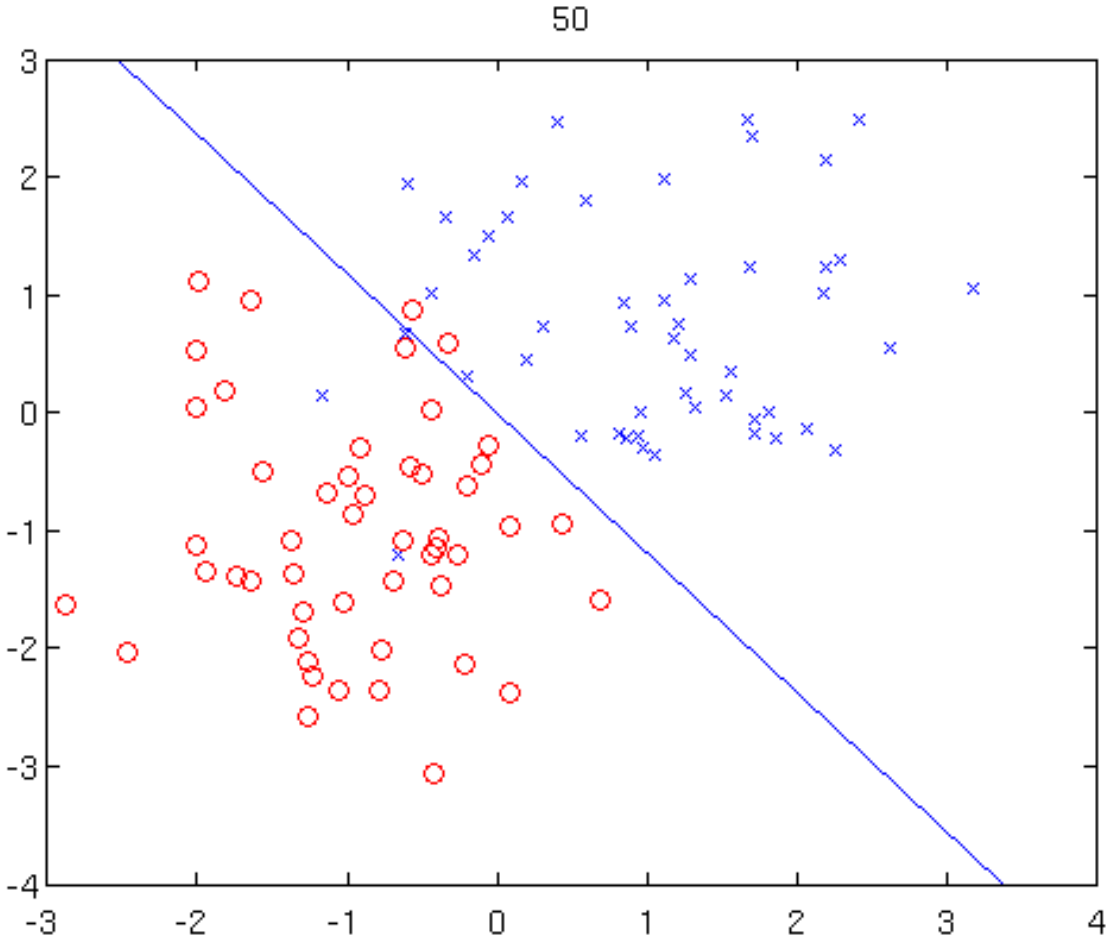
An Example



An Example



An Example



How does this get used in computer vision

- Step 1 - Define a labeling task
- Step 2 - Get a bunch of *training examples*, with *ground-truth labels*
- Step 3 - Train the classifier
- Step 4 - Get a bunch of *test examples*, also with *ground-truth labels*
- Step 5 - Test classifier on test examples

Let's talk more about step 3

- Each training example is numerically represented by a set of features.
- These features are stored in a vector. The vector for each training example will be denoted $\mathbf{x}_i$, where the i represents the ith training example.
- Each training example also has a label, l_i
- If we denote the weight vector as $\mathbf{w}$, then we can rewrite the log-loss criterion as

$$L = \sum_{i=1}^{N} \log(1 + \exp\left((-l_i(\mathbf{w}^T \mathbf{x}_i)\right)) \tag{1}$$

- (In the earlier example, $\mathbf{w} = [abc]$)

Is it really the same?

- Compare

$$L = \sum_{i=1}^{N} \log\left(1 + \exp(-l_i(ax_i + by_i + c))\right)$$

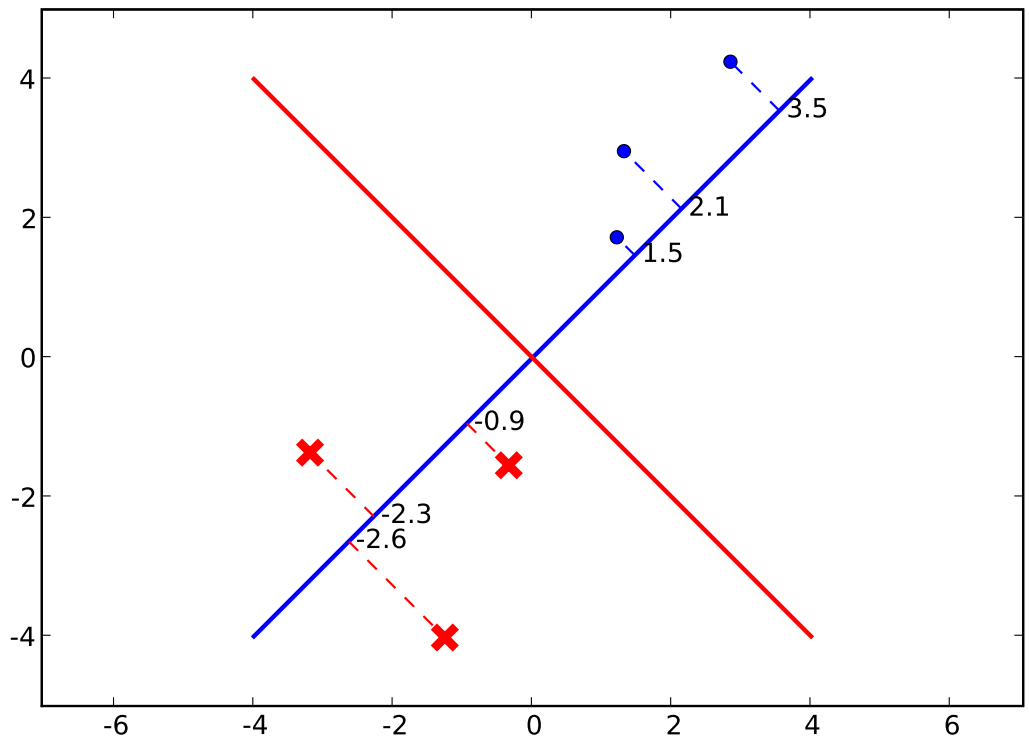
- With

$$L = \sum_{i=1}^{N} \log(1 + \exp\left((-l_i(\mathbf{w}^T\mathbf{x}_i)\right)) \tag{2}$$

- What's missing?
- In the upper equation c is a constant.
- This is called the bias term.

Function of the bias term

- Going back to this example:



How do we implement it?

- It's actually quite easy
- Make one feature be equal to all ones

How does this get used in computer vision

- Step 1 - Define a labeling task
- Step 2 - Get a bunch of *training examples*, with *ground-truth labels*
- Step 3 - Train the classifier
- Step 4 - Get a bunch of *test examples*, also with *ground-truth labels*
- Step 5 - Test classifier on test examples

Steps 4 and Steps 5 - Testing

- We want to want to know how well the system will perform on data that it has never seen
- *Overfitting* can be a problem, especially if there isn't much training data.
- In classification, we usually measure the percentage of points correctly labeled.