

# Classification

## Numerical Methods for Deep Learning

# Logistic Regression

Assume our data falls into two classes. Denote by  $\mathbf{c}_{\text{obs}}(\mathbf{y})$  the probability that example  $\mathbf{y} \in \mathbb{R}^{n_f}$  belongs to first category.

Since output of our classifier  $f(\mathbf{y}, \boldsymbol{\theta})$  is supposed to be probability, use logistic sigmoid

$$\mathbf{c}(\mathbf{y}, \boldsymbol{\theta}) = \frac{1}{1 + \exp(-f(\mathbf{y}, \boldsymbol{\theta}))}.$$

Example (Linear Classification): If  $f(\mathbf{y}, \boldsymbol{\theta})$  is a linear function (adding bias is easy),  $\boldsymbol{\theta} = \mathbf{w} \in \mathbb{R}^{n_f}$  and

$$\mathbf{c}(\mathbf{y}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{y})}.$$

from now on consider linear models for simplicity

# Multinomial Logistic Regression

Suppose data falls into  $n_c \geq 2$  categories and the components of  $\mathbf{c}_{\text{obs}}(\mathbf{y}) \in [0, 1]^{n_c}$  contain probabilities for each class.

Applying the logistic sigmoid to each component of  $f(\mathbf{y}, \mathbf{W})$  not enough (probabilities must sum to one). Use

$$\mathbf{c}(\mathbf{y}, \mathbf{W}) = \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{W}\mathbf{y})} \right) \exp(\mathbf{W}\mathbf{y}).$$

Note: Division and exp are applied element-wise!

# Logistic Regression - Loss Function

How similar are  $\mathbf{c}(\cdot, \mathbf{W})$  and  $\mathbf{c}_{\text{obs}}(\cdot)$ ?

Naive idea: Let  $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$  be examples with class probabilities  $\mathbf{C}_{\text{obs}} \in [0, 1]^{n_c \times n}$ , use

$$\phi(\mathbf{W}) = \frac{1}{2n} \sum_{j=1}^n \|\mathbf{c}(\mathbf{y}_j, \mathbf{W}) - \mathbf{c}_{j,\text{obs}}\|_F^2$$

Problems

- ▶ ignores that  $\mathbf{c}(\cdot, \mathbf{W})$  and  $\mathbf{c}_{\text{obs}}(\cdot)$  are distributions.
- ▶ leads to non-convex objective function

Need to be careful to treat  $\mathbf{c}$  appropriately.

## Example: Designing a Code

Goal: Communicate using minimal number of bits.

Example: Bob talks  $\mathbf{c} = [1/2, 1/4, 1/8, 1/8]$  of the time about dogs, cats, fish, and birds, respectively.

How many bits need to be transferred on average?

word	naive code	better code
dog	00	0
cat	01	10
fish	10	110
bird	11	111

Idea: Quantify information content in probability distribution using average length.

# Entropy

Note: Length of word depends on its probability being used.  
How long should a word be?

Optimal choice for information for any category

$$I = \log_2(\mathbf{c}_j^{-1}) = -\log_2(\mathbf{c}_j)$$

The larger  $\mathbf{c}_j$ , the more common we use it, the shorter the word should be.

The entropy is the average (expectation) of information over all classes.

$$E(\mathbf{c}) = -\sum \mathbf{c}_j \log_2(\mathbf{c}_j) = -\mathbf{c}^\top \log_2(\mathbf{c})$$

## Example: Designing a Code - 2

Entropy for Bob's code is

$$\frac{1}{2} \log(2) + \frac{1}{4} \log(4) + 2\frac{1}{8} \log(8) = 1.75$$

average length of word is 1.75 bits < 2 bits for naive code!

For the complete tutorial on entropy, read

<http://colah.github.io/posts/2015-09-Visual-Information/>

# Properties of Entropy

- ▶ recall  $\lim_{x \rightarrow 0} x \log x = 0$
- ▶ prefer sparse distributions (why?)
- ▶ has been used in compressed sensing type methods

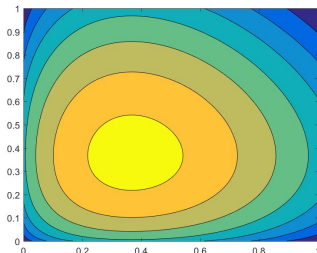


Figure: The entropy of a vector  $\mathbf{c} = (c_1, c_2)^T$



# Cross Entropy

Measure the average word length when using code designed for  $\mathbf{c}$  for sending information with probability  $\hat{\mathbf{c}}$

$$E(\hat{\mathbf{c}}, \mathbf{c}) = -\hat{\mathbf{c}}^T \log(\mathbf{c}).$$

Clearly

$$E(\hat{\mathbf{c}}, \mathbf{c}) \geq E(\mathbf{c}, \mathbf{c})$$

Example: Alice talks  $\mathbf{c} = [1/8, 1/2, 1/4, 1/8]$  of the time about dogs, cats, fish, and birds, respectively. If she used Bob's code, the average word length would be

$$\frac{1}{8} \log(2) + \frac{1}{2} \log(4) + \frac{1}{4} \log(8) + \frac{1}{8} \log(8) = 2.25 > 1.75$$

$E$  measures how similar the distributions  $\mathbf{c}$  and  $\hat{\mathbf{c}}$  are.

One flaw:  $E(\mathbf{c}, \hat{\mathbf{c}}) \neq E(\hat{\mathbf{c}}, \mathbf{c})$  (verify for our example!)

# Cross Entropy for Logistic Regression - 1

Recall: For a single example and two classes we have

$$\mathbf{c}(\mathbf{y}, \mathbf{w}) = \begin{pmatrix} \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{y})} \\ 1 - \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{y})} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} h(\mathbf{w}^\top \mathbf{y}) \\ 1 - h(\mathbf{w}^\top \mathbf{y}) \end{pmatrix}$$

Assume we have the observation  $\mathbf{c}_{\text{obs}} = \begin{pmatrix} \mathbf{c}_{\text{obs}} \\ 1 - \mathbf{c}_{\text{obs}} \end{pmatrix}$  then

$$\begin{aligned} E(\mathbf{c}_{\text{obs}}, \mathbf{c}) &= -\mathbf{c}_{\text{obs}}^\top \log(\mathbf{c}(\mathbf{y}, \mathbf{w})) \\ &= -\mathbf{c}_{\text{obs}} \log(h(\mathbf{w}^\top \mathbf{y})) - (1 - \mathbf{c}_{\text{obs}}) \log(1 - h(\mathbf{w}^\top \mathbf{y})). \end{aligned}$$

where

$$h(z) = \frac{1}{1 + \exp(-z)}$$

## Cross Entropy for Logistic Regression - 2

In the case we have many examples need to sum over the data

$$\mathbf{C}(\mathbf{Y}, \mathbf{w}) = \begin{pmatrix} \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{Y})} \\ 1 - \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{Y})} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} h(\mathbf{w}^\top \mathbf{Y}) \\ 1 - h(\mathbf{w}^\top \mathbf{Y}) \end{pmatrix} \in \mathbb{R}^{2 \times n}$$

Assume we have the observation  $\mathbf{c}_{\text{obs}} \in \mathbb{R}^n$ . Define

$$\mathbf{C}_{\text{obs}} = \begin{pmatrix} \mathbf{c}_{\text{obs}}^\top \\ 1 - \mathbf{c}_{\text{obs}}^\top \end{pmatrix} \in \mathbb{R}^{2 \times n}.$$

Then the cross entropy is

$$\begin{aligned} E(\mathbf{C}_{\text{obs}}, \mathbf{C}) &= -\frac{1}{n} \text{tr}(\mathbf{C}_{\text{obs}}^\top \mathbf{C}) \\ &= -\frac{1}{n} \mathbf{c}_{\text{obs}}^\top \log(h(\mathbf{w}^\top \mathbf{Y})) \\ &\quad - \frac{1}{n} (1 - \mathbf{c}_{\text{obs}})^\top \log(1 - h(\mathbf{w}^\top \mathbf{Y})) \end{aligned}$$

# Cross Entropy for Multinomial Logistic Regression

Similarly, for general case ( $n_c \geq 2$  classes,  $n$  examples).

Recall:

$$\mathbf{C}(\mathbf{Y}, \mathbf{W}) = \exp(\mathbf{WY}) \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{WY})} \right)$$

Get cross entropy by summing over all examples

$$E(\mathbf{C}_{\text{obs}}, \mathbf{C}(\mathbf{Y}, \mathbf{W})) = -\frac{1}{n} \text{tr}(\mathbf{C}_{\text{obs}}^\top \log(\mathbf{C}(\mathbf{Y}, \mathbf{W}))).$$

We will also call this the *softmax* (cross-entropy) function.

# Simplifying the Softmax Function

Let  $\mathbf{S} = \mathbf{W}\mathbf{Y}$ , then

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n} \text{tr} \left( \mathbf{C}_{\text{obs}}^{\top} \log \left( \exp(\mathbf{S}) \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^{\top} \exp(\mathbf{S})} \right) \right) \right).$$

Verify that this is equal to

$$\begin{aligned} E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = & -\frac{1}{n} \mathbf{e}_{n_c}^{\top} (\mathbf{C}_{\text{obs}} \odot \mathbf{S}) \mathbf{e}_n \\ & + \frac{1}{n} \mathbf{e}_{n_c}^{\top} \mathbf{C}_{\text{obs}} \log (\mathbf{e}_{n_c}^{\top} \exp(\mathbf{S}))^{\top} \end{aligned}$$

( $\odot$  is Hadamard product, exp and log component-wise)

If  $\mathbf{C}_{\text{obs}}$  has a unit row sum (why?) then  $\mathbf{e}_{n_c}^{\top} \mathbf{C}_{\text{obs}}^{\top} = \mathbf{e}_n^{\top}$  and

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n} \mathbf{e}_{n_c}^{\top} (\mathbf{C}_{\text{obs}} \odot \mathbf{S}) \mathbf{e}_n + \frac{1}{n} \log(\mathbf{e}_{n_c}^{\top} \exp(\mathbf{S})) \mathbf{e}_n$$

# Numerical Considerations

Scale to prevent overflow. Note that for an arbitrary  $s \in \mathbb{R}$  we have

$$E(\mathbf{C}_{\text{obs}}, \mathbf{WY} - s) = E(\mathbf{C}_{\text{obs}}, \mathbf{WY})$$

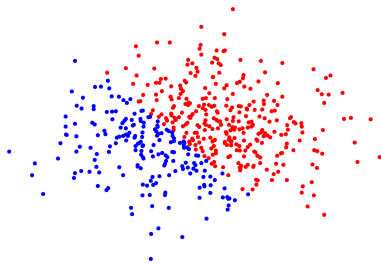
This prevents overflow, but may lead to underflow (and divisions by zero).

Note that  $s$  does not need to be the same in each row (example). Hence, we can choose  $\mathbf{s} = \max(\mathbf{WY}, [], 1) \in \mathbb{R}^{1 \times n}$  to avoid underflow and overflow.

For stability use  $E(\mathbf{C}_{\text{obs}}, \mathbf{S})$  where  $\mathbf{S} = \mathbf{WY} - \mathbf{e}_{n_c} \mathbf{s}$ .

# Test Problem: Linear Classification

Generate data that is linearly separable:



```
a = 3; b = 2;
```

```
Y = randn(2,500);
```

```
C = a*Y(1,:) + b*Y(2,:) + 1;
```

```
C(C>0) = 1; C(C<0) = 0;
```

```
C = [C; 1-C]
```

# Coding: Softmax Regression Objective Function

Write a function that computes the softmax function given a data matrix  $\mathbf{Y}$ , its class  $\mathbf{C}$ , and a matrix  $\mathbf{W}$ .

```
function[E] = softmaxFun(W,Y,C)
```

```
% Your code here
```

```
end
```

Test your code using `testSoftmax.m`



# Linear Classification

If  $\mathbf{W}$  can separate the classes then the goal is to minimize the cross entropy (with some potential regularization)

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} \quad -\frac{1}{n} \mathbf{e}_{n_c}^\top (\mathbf{C}_{\text{obs}} \odot \mathbf{S}) \mathbf{e}_n + \frac{1}{n} \log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}) \mathbf{e}_n)$$

subject to  $\mathbf{S} = \mathbf{W}\mathbf{Y} - \mathbf{e}_{n_c} \mathbf{s}$

This is a smooth convex optimization problem  
 $\Rightarrow$  many existing optimization techniques will work

For large-scale problems, use derivative-based optimization algorithm. (Examples: Steepest Descent, Newton-like methods, Stochastic Gradient Descent, ...)

Excellent references: [? ? ? ]

# Differentiating the Softmax Function - 1

We need to compute the derivative of the cross entropy function with respect to  $\mathbf{W}$ . Three hints:

- ▶  $\sum \mathbf{w} \odot \mathbf{y} = \mathbf{w}^\top \mathbf{y}$
- ▶  $\nabla_{\mathbf{w}}(\mathbf{w}^\top \mathbf{y}) = \mathbf{y}$
- ▶  $\text{vec}(\mathbf{W}\mathbf{Y}) = (\mathbf{Y}^\top \otimes \mathbf{I})\text{vec}(\mathbf{W}) = (\mathbf{I} \otimes \mathbf{W})\text{vec}(\mathbf{Y})$

where  $\otimes$  is the Kronecker product.

We use the common notation:

- ▶  $\text{vec}(\mathbf{A})$  reshapes matrix  $\mathbf{A}$  (column-wise) into vector
- ▶  $\text{mat}(\mathbf{a})$  reshapes vector  $\mathbf{a}$  into matrix,  $\text{mat}(\text{vec}(\mathbf{A})) = \mathbf{A}$ .
- ▶  $\text{diag}(\mathbf{A}) = \text{diag}(\text{vec}(\mathbf{A}))$  builds diagonal matrix with entries given by  $\mathbf{A}$ .

# Differentiating the Softmax Function - 2

Do it in three steps:

1.  $\nabla_{\mathbf{S}} E(\mathbf{S})$  with  $\mathbf{S} = \mathbf{Y}\mathbf{W}$  (assume w.l.o.g. no shift)
2.  $\nabla_{\mathbf{W}} \mathbf{S} = \nabla_{\mathbf{W}} (\mathbf{W}\mathbf{Y})$
3. use chain rule to get  $\nabla_{\mathbf{W}} E(\mathbf{W}\mathbf{Y})$ .

Break the first step down into two terms

$$E(\mathbf{S}) = \frac{1}{n} \overbrace{-\text{tr}(\mathbf{C}_{\text{obs}}^{\top} \mathbf{S})}^{E1} + \frac{1}{n} \overbrace{\log(\mathbf{e}_{n_c} \exp(\mathbf{S})) \mathbf{e}_n}^{E2},$$

First term is linear

$$\nabla_{\mathbf{S}} E_1 = \nabla_{\mathbf{S}} \text{tr}(\mathbf{C}_{\text{obs}}^{\top} \mathbf{S}) = \mathbf{C}_{\text{obs}}.$$

## Differentiating the Softmax Function - 3

$$E(\mathbf{S}) = \frac{1}{n} \overbrace{-\text{tr}(\mathbf{C}_{\text{obs}}^\top \mathbf{S})}^{E1} + \frac{1}{n} \overbrace{\log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})) \mathbf{e}_n}^{E2}$$

Second term requires a bit more care

$$\mathbf{J}_S E_2 = \mathbf{J}_S \mathbf{e}_n^\top \log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})) = \mathbf{e}_n^\top \mathbf{J}_S \log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))$$

and

$$\mathbf{J}_S \log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})) = \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) \mathbf{J}_S (\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))$$

# Differentiating the Softmax Function - 4

Recall:

$$\mathbf{J}_S E_2 = \mathbf{e}_n^\top \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) \mathbf{J}_S (\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))$$

Focus on last term:

$$\begin{aligned} \mathbf{J}_S (\mathbf{e}_{n_c}^\top \exp(\mathbf{S})) &= \mathbf{J}_S ((\mathbf{I} \otimes \mathbf{e}_{n_c}^\top) \text{vec}(\exp(\mathbf{S}))) \\ &= (\mathbf{I} \otimes \mathbf{e}_{n_c}^\top) \text{diag}(\text{vec}(\exp(\mathbf{S}))) \end{aligned}$$

Putting it together

$$\mathbf{J}_S E_2 = \mathbf{e}_n^\top \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) (\mathbf{I} \otimes \mathbf{e}_{n_c}^\top) \text{diag}(\text{vec}(\exp(\mathbf{S})))$$

Left to do: Take transpose

## Differentiating the Softmax Function - 5

$$\nabla_{\mathbf{S}} E_2 = \text{diag}(\text{vec}(\exp(\mathbf{S}))) (\mathbf{I} \otimes \mathbf{e}_{n_c}) \text{diag} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) \mathbf{e}_n$$

simplifying

$$\nabla_{\mathbf{S}} E_2 = \text{diag}(\text{vec}(\exp(\mathbf{S}))) (\mathbf{I} \otimes \mathbf{e}_{n_c}) \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right)$$

Avoid diag and simplify using matrix representation

$$\nabla_{\mathbf{S}} E_2 = \exp(\mathbf{S}) \odot \left( \mathbf{e}_{n_c} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) \right)$$

Finally combine gradients of  $E_1$  and  $E_2$

$$\nabla_{\mathbf{S}} E = -\frac{1}{n} \mathbf{C}_{\text{obs}} + \frac{1}{n} \exp(\mathbf{S}) \odot \left( \mathbf{e}_{n_c} \left( \frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})} \right) \right).$$

# Differentiating the Softmax Function - 6

$$E(\mathbf{W}) = \underbrace{-\frac{1}{n} \text{tr}(\mathbf{C}_{\text{obs}}^{\top}(\mathbf{W}\mathbf{Y}))}_{E1} + \underbrace{\frac{1}{n} \mathbf{e}_n^{\top} \log(\mathbf{e}_{n_c}^{\top} \exp(\mathbf{W}\mathbf{Y}))}_{E2}$$

Note that

$$\nabla_{\mathbf{W}} \mathbf{S} = \nabla_{\mathbf{W}}(\mathbf{W}\mathbf{Y}) = \nabla_{\mathbf{W}}((\mathbf{Y}^{\top} \otimes \mathbf{I}) \text{vec}(\mathbf{W})) = \mathbf{Y} \otimes \mathbf{I}.$$

Hence, applying the chain rule gives

$$\nabla_{\mathbf{W}} E = \frac{1}{n} \left( -\mathbf{C}_{\text{obs}} + \exp(\mathbf{S}) \odot \left( \mathbf{e}_{n_c} \left( \frac{1}{\mathbf{e}_{n_c}^{\top} \exp(\mathbf{S})} \right) \right) \right) \mathbf{Y}^{\top}.$$

## Coding: Differentiating the Softmax Function

Extend your softmax function, so that it returns the gradient if needed.

```
function[E,dE] = softmaxFun(W,Y,C)
```

```
% Your code from before
```

```
if nargin > 1
```

```
% Your code for gradient here
```

```
end
```

```
end
```



# Testing your Derivatives

Your derivatives are assumed to be wrong unless you prove otherwise.

Test based on Taylor theorem. Let  $\mathbf{W}$  be fixed and  $\mathbf{D}$  be a random direction (same size as  $\mathbf{W}$ ):

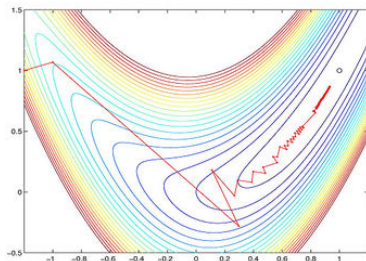
$h$	$E(\mathbf{W} + h\mathbf{D}) - E(\mathbf{W})$	$E(\mathbf{W} + h\mathbf{D}) - E(\mathbf{W}) - h\text{tr}(\mathbf{D}^\top \nabla E(\mathbf{W}))$
1		
$2^{-1}$		
$2^{-2}$		
$2^{-3}$		
$2^{-4}$		
$2^{-5}$		

First column should decay as  $\mathcal{O}(h)$

Second column should decay as  $\mathcal{O}(h^2)$

# Derivative-Based Optimization: Steepest Descent

To minimize the energy go “down-hill”



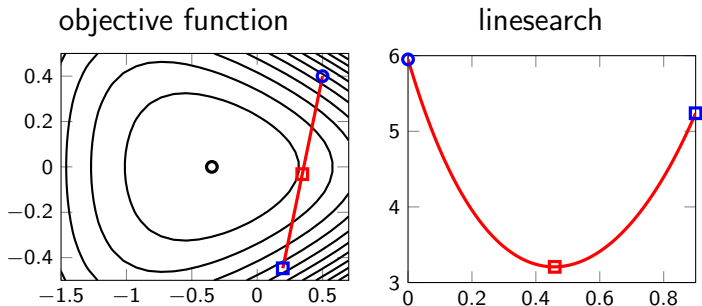
Iterate:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \mathbf{D}, \quad \mathbf{D} = -\nabla E(\mathbf{W}_k).$$

Guaranteed to be a descent direction but need to make sure that

$$E(\mathbf{W}_k + \alpha \mathbf{D}) < E(\mathbf{W}_k)$$

# Line Search Problem



Let  $E$  be the cross entropy,  $\mathbf{W}_k$  the current weights, and  $\mathbf{D}$  the search direction. The line search problem is:

$$\min_{\alpha > 0} \phi(\alpha) \quad \text{where} \quad \phi(\alpha) = E(\mathbf{W}_k + \alpha \mathbf{D}).$$

# Armijo Line Search

A method for inexact line search

- ▶ Start with  $\alpha = \alpha_0$
- ▶ Test  $E(\mathbf{W} + \alpha \mathbf{D}) < E(\mathbf{W})$
- ▶ If fail  $\alpha \leftarrow \frac{1}{2}\alpha$

A few (small) but helpful tricks

- ▶ Choose your  $\alpha_0$  based on the problem
- ▶ If line search is needed ( $\alpha \neq \alpha_0$ ) at iteration  $k$  set  $\alpha_0 = \alpha_k$  in next iteration.
- ▶ If no line search is needed ( $\alpha = \alpha_0$ ) at iteration  $k$  set  $\alpha_0 = \gamma \alpha_k$ ,  $\gamma > 1$  in next iteration.

# Coding: Steepest Descent

Write a code for steepest descent with Armijo linesearch.

```
function W = steepestDescent(E,W,param)
% Inputs:
%      E - function that provides value and gradient
%      W - starting guess
%  param - struct with parameters

alpha      = param.maxStep; % max step size
maxIter    = param.maxIter; % max number of iterations

for i=1:maxIter

% Your code here

end
```

# References