# Setting Up Your Programming Assignment Environment

The Machine Learning course includes several programming assignments which you’ll need to finish to complete the course. The assignments require the Octave or MATLAB scientific computing languages.

* Octave is a free, open-source application available for many platforms. It has a text interface and an experimental graphical one.
* MATLAB is proprietary software, but a free trial license to MATLAB Online is being offered for the completion of this course.

## FAQ

### Does it cost money?

While you’re taking the course, both software packages are available free of charge. Octave is distributed under the GNU Public License, which means that it is always free to download and distribute. MATLAB Online licenses are available for completing the programming assignments in the course only. For any other purposes (like your own work after you complete the course), MATLAB can be licensed to [individuals](https://www.mathworks.com/products/matlab-home.html) or companies from Mathworks directly.

### Is there a difference in quality?

There are several subtle differences between the two software packages. MATLAB may offer a smoother experience (especially for Mac users), contains a larger number of functions, and can be more robust to failure. However, the functions used in this course are available in both packages, and many students have successfully completed the course using either.

### How do I install one of them?

See installation instructions for [Windows](https://www.coursera.org/learn/machine-learning/supplement/p9ckf/installing-octave-matlab-on-windows), [Mac OS X (10.10 Yosemite and 10.9 Mavericks)](https://www.coursera.org/learn/machine-learning/supplement/ykU6M/installing-octave-matlab-on-mac-os-x-10-10-yosemite-and-10-9-mavericks), [other Mac OS X](https://www.coursera.org/learn/machine-learning/supplement/M16Qr/installing-octave-matlab-on-mac-os-x-10-8-mountain-lion-and-earlier), or [GNU/Linux](https://www.coursera.org/learn/machine-learning/supplement/NSG5j/installing-octave-matlab-on-gnu-linux).

## Access MATLAB Online and Upload the Programming Exercise Files

Access to MATLAB Online is being provided by MathWorks to Machine Learning students for the duration of the course. MATLAB Online is the online version of the MATLAB desktop program. It provides most features of the original program in a web-based interface. No download or installation is required, and the program can be accessed from any computer running a common web browser.

### Follow the steps below to access MATLAB Online

1. If you do not already have one, [create a MathWorks account](https://www.mathworks.com/mwaccount/register).
2. Click on the [MATLAB Online license link](https://www.mathworks.com/licensecenter/classroom/machine_learning/) and provide your MathWorks account credentials (if requested).
3. Click on the blue 'Access MATLAB Online' button, and log-in to MATLAB Online with your MathWorks account credentials.
4. Confirm that you have access to MATLAB Online. For help with MATLAB Online access or technical issues, see the MATLAB Help discussion forum.

Bookmark <https://matlab.mathworks.com/> for quicker access to MATLAB Online in the future.

### Follow the steps below to add the programming exercise files to MATLAB Online

The files below have been modified for use in MATLAB Online only. MATLAB Desktop and Octave users should access the exercise files provided later in the course.

Your MathWorks account comes with free cloud storage via [MATLAB Drive](https://drive.matlab.com/) which you can access in MATLAB Online. To add the programming exercise files to MATLAB Online, download them files as a zip file to your desktop using the link below, then rename the file to: machine-learning-ex (Coursera adds extra characters to the filename upon download that must be removed)

[machine-learning-ex.zip](https://d3c33hcgiwev3.cloudfront.net/Rj4ZVxtxEemPNBKo-MLnmA_46741d601b7111e9b9927b6cf63392a9_machine-learning-ex.zip?Expires=1568419200&Signature=KoKlulFZ0lLVY2n1Nmtzc6exwrLVquVNoUwvI8GSuA51ffwZOF8kGl6z45OcZivO4tVkTSJown0pHtoJt5hPbYRZvuiFEn2HZtsdOJx5ydwSPcxkpPxVeh9TcthgVMMFJdpsVhnh67UPp1WfRRbI0BR04lZB72Iss8tbUvABeZk_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

To add the exercise files to MATLAB Online,

1. Open MATLAB Online and click the 'Upload' button in the 'Home' tab.
2. Use the folder window that opens to find and select the 'machine-learning-ex.zip' file, then click 'Open'. You should see the file in MATLAB Online after upload is complete.
3. Unzip the file by entering the command: unzip machine-learning-ex.zip at the command line.

Note that you can access your files even after your MATLAB Online license expires through MATLAB Drive.

## Completing the programming exercises in MATLAB Online

There are eight programming exercises in the course, the first is posted at the end of Week 2. When you reach a programming exercise page in the course, you do not need to download the exercise zip file, as the necessary files for all exercises are included in the zip folder above. Make note of your assignment token, however, you will need it to submit your solutions.

IMPORTANT: The exercise files above been updated specifically for MATLAB Online users. Instructions for completing the exercises in MATLAB Online and tips for troubleshooting problems are found in the included README.mlx file. Before attempting your first programming exercise, read all of the instructions in README.mlx. Note that the instructions for MATLAB Online users differ from the instructions and video demonstration provided later in the course which only apply to Octave and MATLAB Desktop users.

# Installing Octave on Windows

## Installing Octave on Windows

Use this link to install Octave for windows: <http://wiki.octave.org/Octave_for_Microsoft_Windows>

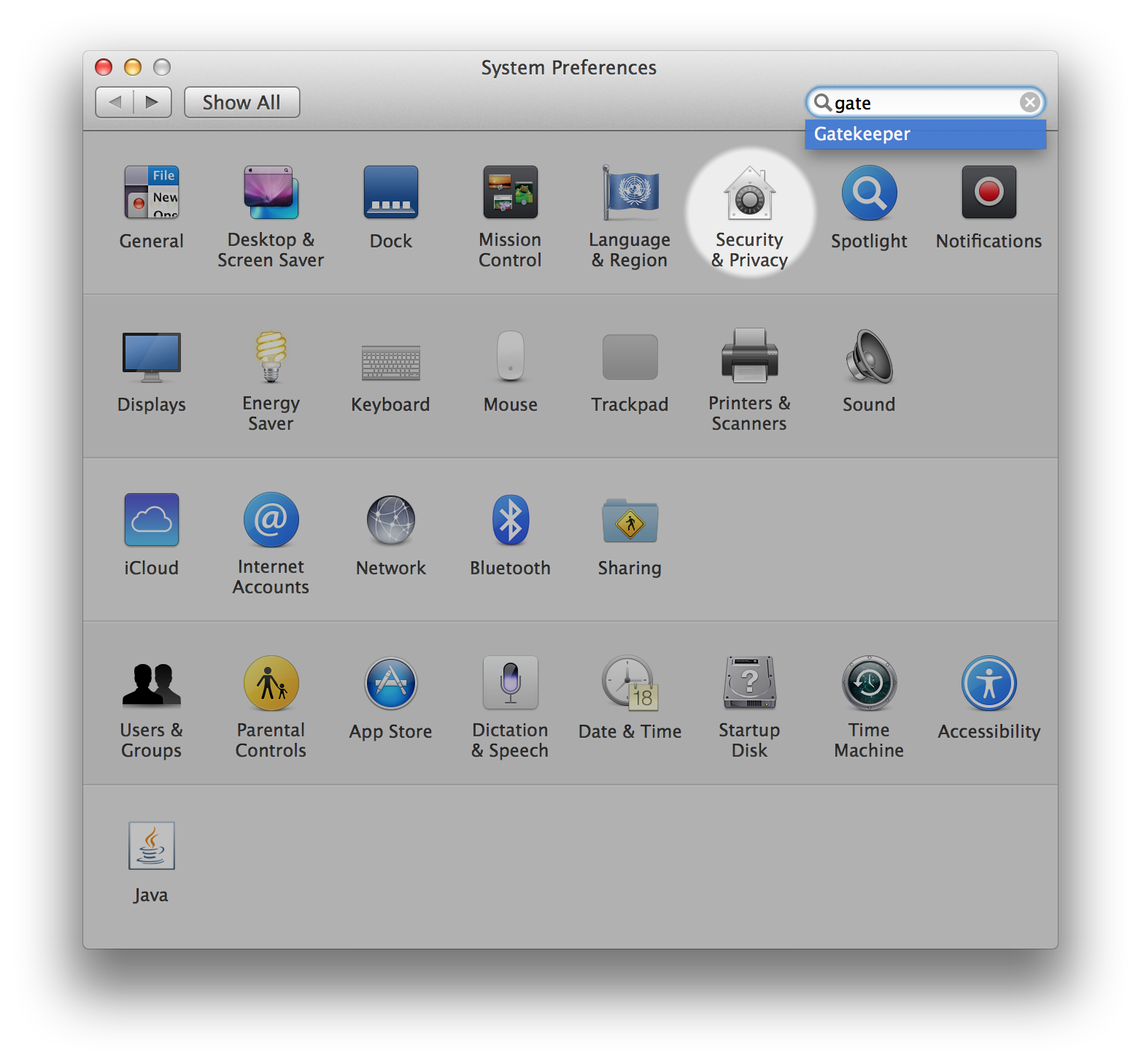
Octave on Windows can be used to submit programming assignments in this course but will likely need a patch provided in the discussion forum. Refer to [https://www.coursera.org/learn/machine-learning/discussions/vgCyrQoMEeWv5yIAC00Eog?](https://www.coursera.org/learn/machine-learning/discussions/vgCyrQoMEeWv5yIAC00Eog?page=2) for more information about the patch for your version.

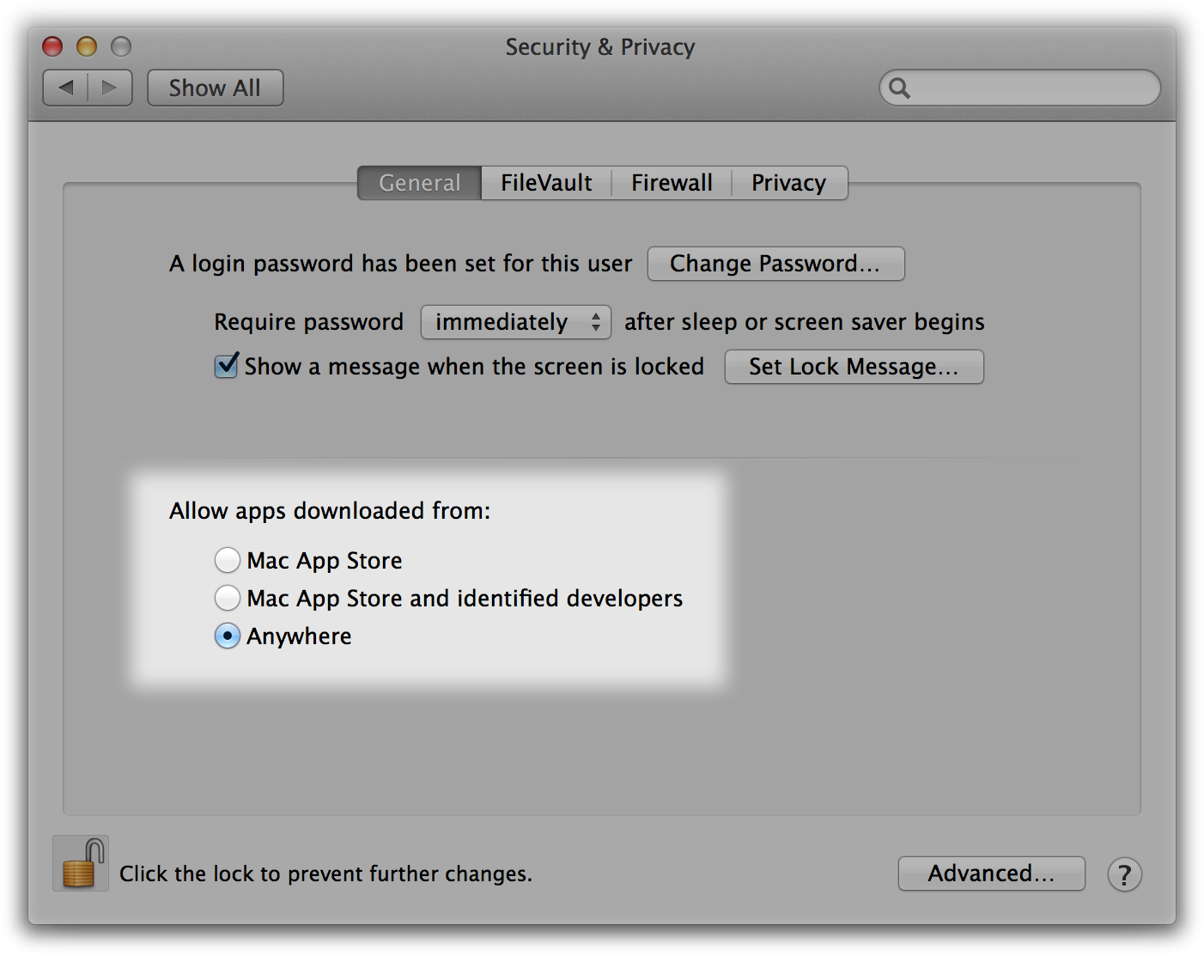
"Warning: Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

# Installing Octave on Mac OS X (10.10 Yosemite and 10.9 Mavericks)

## Installing Octave on Mac OS X (10.10 Yosemite and 10.9 Mavericks)

Mac OS X [has a feature called Gatekeeper](http://support.apple.com/en-us/HT202491) that may only let you install applications from the Mac App Store. You may need to configure it to allow the Octave installer. Visit your System Preferences, click Security & Privacy, and check the setting to allow apps downloaded from Anywhere. You may need to enter your password to unlock the settings page.





2. Download [the Octave 3.8.0 installer](http://sourceforge.net/projects/octave/files/Octave MacOSX Binary/2013-12-30 binary installer of Octave 3.8.0 for OSX 10.9.1 (beta)/GNU_Octave_3.8.0-6.dmg/download) or the latest version that isn't 4.0.0. The file is large so this may take some time.

3. Open the downloaded image, probably named GNU\_Octave\_3.8.0-6.dmg on your computer, and then open Octave-3.8.0-6.mpkg inside.

4. Follow the installer’s instructions. You may need to enter the administrator password for your computer.

5. After the installer completes, Octave should be installed on your computer. You can find Octave-cli in your Mac’s Applications, which is a text interface for Octave that you can use to complete Machine Learning’s programming assignments.

Octave also includes an experimental graphical interface which is called Octave-gui, also in your Mac’s Applications, but we recommend using Octave-cli because it’s more stable.

Note: If you use a package manager (like MacPorts or Homebrew), we recommend you follow [the package manager installation instructions](_blank).

"Warning: Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

# Installing Octave on Mac OS X (10.8 Mountain Lion and Earlier)

## Installing Octave on Mac OS X (10.8 Mountain Lion and Earlier)

If you use Mac OS X 10.9, we recommend following the [instructions above](https://www.coursera.org/learn/machine-learning/supplement/ykU6M/installing-octave-matlab-on-mac-os-x-10-10-yosemite-and-10-9-mavericks). For other Mac OS X versions, the Octave project doesn’t distribute installers. We recommend installing Homebrew, a package manager, using [their instructions](_blank).

"Warning: Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

# Installing Octave on GNU/Linux

## Installing Octave on GNU/Linux

We recommend [using your system package manager to install Octave](http://wiki.octave.org/Octave_for_GNU/Linux).

On Ubuntu, you can use:

* sudo apt-get update && sudo apt-get install octave

On Fedora, you can use:

* sudo yum install octave-forge

Please consult [the Octave maintainer’s instructions](http://wiki.octave.org/Octave_for_GNU/Linux) for other GNU/Linux systems.

"Warning: Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

# More Octave/MATLAB resources

## Octave Resources

At the Octave command line, typing help followed by a function name displays documentation for a built-in function. For example, help plot will bring up help information for plotting. Further documentation can be found at the Octave [documentation pages](http://www.gnu.org/software/octave/doc/interpreter/).

## MATLAB Resources

At the MATLAB command line, typing help followed by a function name displays documentation for a built-in function. For example, help plot will bring up help information for plotting. Further documentation can be found at the MATLAB [documentation pages](http://www.mathworks.com/help/matlab/).

## Introduction to MATLAB with Onramp

Made for MATLAB beginners or those looking for a quick refresh, the MATLAB Onramp is a 1-2 hour interactive introduction to the basics of MATLAB programming. Octave users are also welcome to use Onramp (requires creation of a free MathWorks account). To access Onramp:

1. If you don’t already have one, create a MathWorks account at: <https://www.mathworks.com/mwaccount/register>

2. Go to: <https://matlabacademy.mathworks.com/> and click on the MATLAB Onramp button to start learning MATLAB!

## MATLAB Programming Tutorials

These short tutorial videos introduce MATLAB and cover various programming topics used in the assignments. Feel free to watch some now and return to reference them as you work through the programming assignments. Many of the topics below are also covered in MATLAB Onramp. \*Indicates content covered in Onramp.

### Get Started with MATLAB and MATLAB Online

* [What is MATLAB?](https://youtu.be/WYG2ZZjgp5M)\*
* [MATLAB Variables](https://youtu.be/0w9NKt6Fixk)\*
* [MATLAB as a Calculator](https://youtu.be/aRSkNpCSgWY)\*
* [MATLAB Functions](https://youtu.be/RJp46UVQBic)\*
* [Getting Started with MATLAB Online](https://youtu.be/XjzxCVWKz58)
* [Managing Files in MATLAB Online](https://youtu.be/B3lWLIrYjC0)

### Vectors

* [Creating Vectors](https://youtu.be/R5Mnkrk9Mos)\*
* [Creating Uniformly Spaced Vectors](https://youtu.be/_zqTOV5yl8Y)\*
* [Accessing Elements of a Vector Using Conditions](https://youtu.be/8D04GW_foQ0)\*
* [Calculations with Vectors](https://youtu.be/VQaZ0TvjF0c)\*
* [Vector Transpose](https://youtu.be/vgRLwjHBmsg)

### Visualization

* [Line Plots](https://youtu.be/-hhJoveE4sY)\*
* [Annotating Graphs](https://youtu.be/JyovEGPSdoI)\*
* [Multiple Plots](https://youtu.be/fBx8EFuXFLM)\*

### Matrices

* [Creating Matrices](https://youtu.be/qdTdwTh6jMo)\*
* [Calculations with Matrices](https://youtu.be/mzzJ9gnMrYE)\*
* [Accessing Elements of a Matrix](https://youtu.be/uWPHxpTuZRA)\*
* [Matrix Creation Functions](https://youtu.be/VPcbpVd_mPA)\*
* [Combining Matrices](https://youtu.be/ejTr3ekTTyA)
* [Determining Array Size and Length](https://youtu.be/IF9-ffmxuy8)
* [Matrix Multiplication](https://youtu.be/4hsx3bdNjGk)
* [Reshaping Arrays](https://youtu.be/UQpDIHlFo8A)
* [Statistical Functions with Matrices](https://youtu.be/Y97W3_u7cM4)

### MATLAB Programming

* [Logical Variables](https://youtu.be/bRMg4GsFDQ8)\*
* [If-Else Statements](https://youtu.be/JZSuU-Laigo)\*
* [Writing a FOR loop](https://youtu.be/lg65bzgvI5c)\*
* [Writing a WHILE Loop](https://youtu.be/PKH5lCMJXbk)
* [Writing Functions](https://youtu.be/GrcNN04eqXU)
* [Passing Functions as Inputs](https://youtu.be/aNCwR9dRjHs)

### Troubleshooting

* [Using Online Documentation](https://youtu.be/54n5zJwR8aM)\*
* [Which File or Variable Am I Using?](https://youtu.be/Z09BvGeYNdE)
* [Troubleshooting Code with the Debugger](https://youtu.be/DB4aJMnZtNQ)

Eg :

Octave Help

>> eye(5)

ans =

Diagonal Matrix

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

>>

>> help eye

'eye' is a built-in function from the file libinterp/corefcn/data.cc

-- eye (N)

-- eye (M, N)

-- eye ([M N])

-- eye (..., CLASS)

Return an identity matrix.

If invoked with a single scalar argument N, return a square NxN

identity matrix.

If supplied two scalar arguments (M, N), 'eye' takes them to be the

number of rows and columns. If given a vector with two elements,

'eye' uses the values of the elements as the number of rows and

columns, respectively. For example:

eye (3)

=> 1 0 0

0 1 0

0 0 1

The following expressions all produce the same result:

eye (2)

==

eye (2, 2)

==

eye (size ([1, 2; 3, 4]))

The optional argument CLASS, allows 'eye' to return an array of the

specified type, like

val = zeros (n,m, "uint8")

Calling 'eye' with no arguments is equivalent to calling it with an

argument of 1. Any negative dimensions are treated as zero. These

odd definitions are for compatibility with MATLAB.

See also: speye, ones, zeros.

Additional help for built-in functions and operators is

available in the online version of the manual. Use the command

'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW

at https://www.octave.org and via the help@octave.org

mailing list.

#### Multiple Features

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Size (feet)2 | Number of bedrooms | Number of floors | Age of home (years) | Price ($1000) |
| 2104 | 5 | 1 | 45 | 460 |
| 1416 | 3 | 2 | 40 | 232 |
| 1534 | 3 | 2 | 30 | 315 |
| 852 | 2 | 1 | 36 | 178 |
| ... | ... | ... | ... | ... |

In the training set above, what is x1(4)​?

The size (in feet2) of the 1st home in the training set

The age (in years) of the 1st home in the training set

The size (in feet2) of the 4th home in the training set (A)

The age (in years) of the 4th home in the training set

# Multiple Features

Note: [7:25 - θT is a 1 by (n+1) matrix and not an (n+1) by 1 matrix]

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables.

|  |
| --- |
| x(i)jx(i)mn=value of feature j in the ith training example=the input (features) of the ith training example=the number of training examples=the number of features |

The multivariable form of the hypothesis function accommodating these multiple features is as follows:

hθ(x)=θ0+θ1x1+θ2x2+θ3x3+⋯+θnxn

In order to develop intuition about this function, we can think about θ0​ as the basic price of a house, θ1​ as the price per square meter, θ2​ as the price per floor, etc. x1​ will be the number of square meters in the house, x2​ the number of floors, etc.

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

|  |
| --- |
| hθ(x)=[θ0θ1...θn]⎡⎣⎢⎢⎢x0x1⋮xn⎤⎦⎥⎥⎥=θTx |

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

Remark: Note that for convenience reasons in this course we assume x(i)0=1 for (i∈1,…,m). This allows us to do matrix operations with theta and x. Hence making the two vectors 'θ' and x(i) match each other element-wise (that is, have the same number of elements: n+1).]

When there are n features, we define the cost function as

J(θ)=2m1​i=1∑m​(hθ​(x(i))−y(i))2.

For linear regression, which of the following are also equivalent and correct definitions of J(θ)?

J(θ)=2m1​∑i=1m​(θTx(i)−y(i))2 **(A)**

J(θ)=2m1​∑i=1m​((∑j=0n​θj​xj(i)​)−y(i))2 (Inner sum starts at 0) **(A)**

J(θ)=2m1​∑i=1m​((∑j=1n​θj​xj(i)​)−y(i))2 (Inner sum starts at 1) **(A)**

J(θ)=2m1​∑i=1m​((∑j=0n​θj​xj(i)​)−(∑j=0n​yj(i)​))2 **(A)**

# Gradient Descent For Multiple Variables

## Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

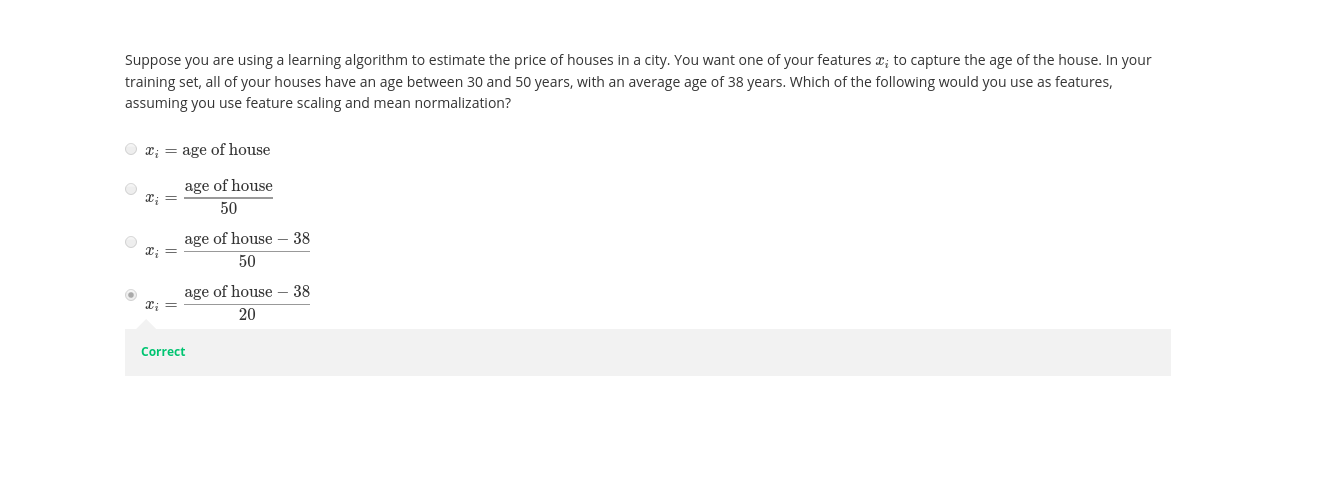
|  |
| --- |
| repeat until convergence:  {  θ0:=θ0−α1m∑i=1m(hθ(x(i))−y(i))⋅x(i)0  θ1:=θ1−α1m∑i=1m(hθ(x(i))−y(i))⋅x(i)1  θ2:=θ2−α1m∑i=1m(hθ(x(i))−y(i))⋅x(i)2⋯  } |

In other words:

|  |
| --- |
| repeat until convergence:  {  θj:=θj−α1m∑i=1m(hθ(x(i))−y(i))⋅x(i)jfor j := 0...n  } |

The following image compares gradient descent with one variable to gradient descent with multiple variables:





# Gradient Descent in Practice I - Feature Scaling

Note: [6:20 - The average size of a house is 1000 but 100 is accidentally written instead]

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

−1 ≤ x(i)​ ≤ 1

or

−0.5 ≤ x(i)​ ≤ 0.5

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

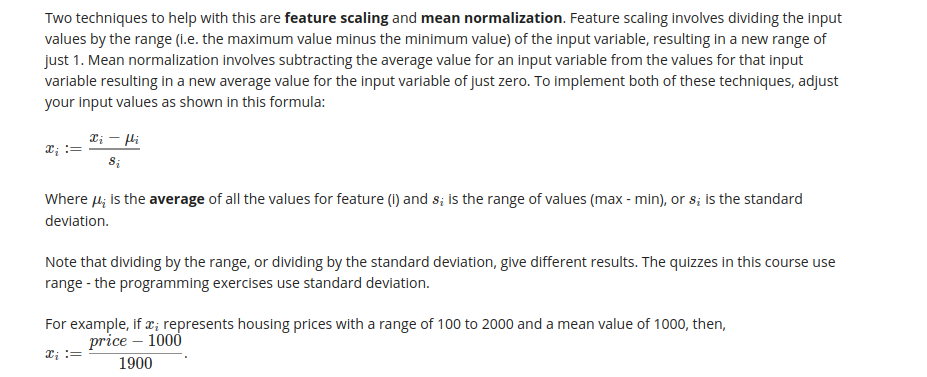
Two techniques to help with this are feature scaling and mean normalization. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

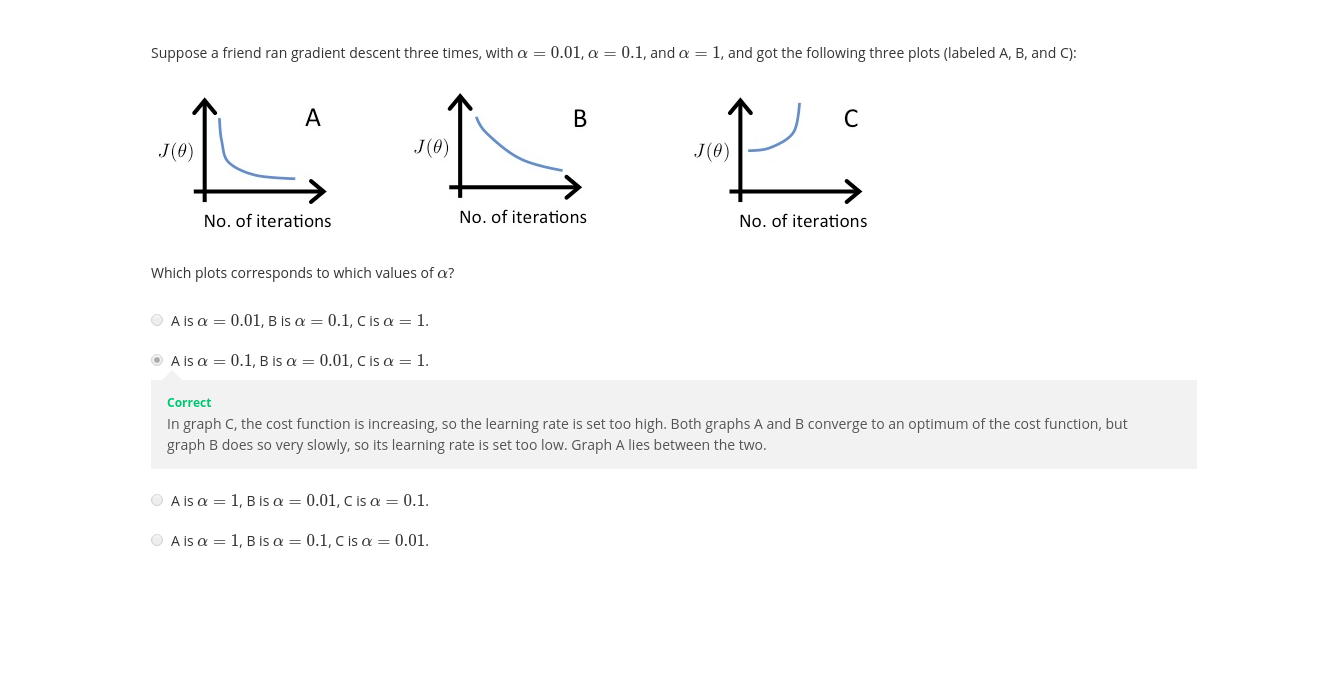
xi​:=si​xi​−μi​​

Where μi is the average of all the values for feature (i) and si​ is the range of values (max - min), or si​ is the standard deviation.

Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.

For example, if xi​ represents housing prices with a range of 100 to 2000 and a mean value of 1000, then, xi​:=1900price−1000​.



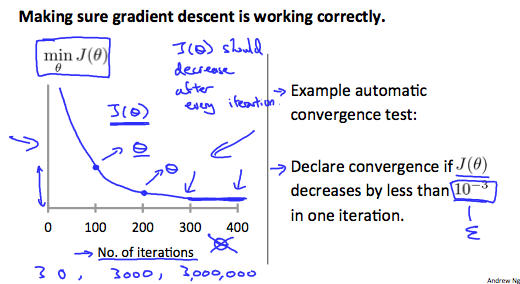


# Gradient Descent in Practice II - Learning Rate

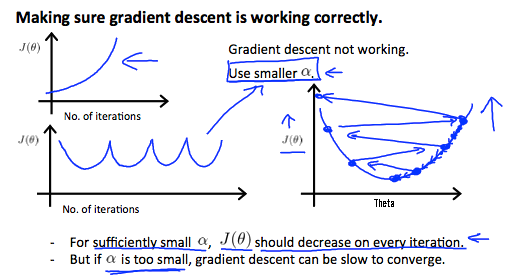
Note: [5:20 - the x -axis label in the right graph should be θ rather than No. of iterations ]

Debugging gradient descent. Make a plot with number of iterations on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α.

Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10−3. However in practice it's difficult to choose this threshold value.



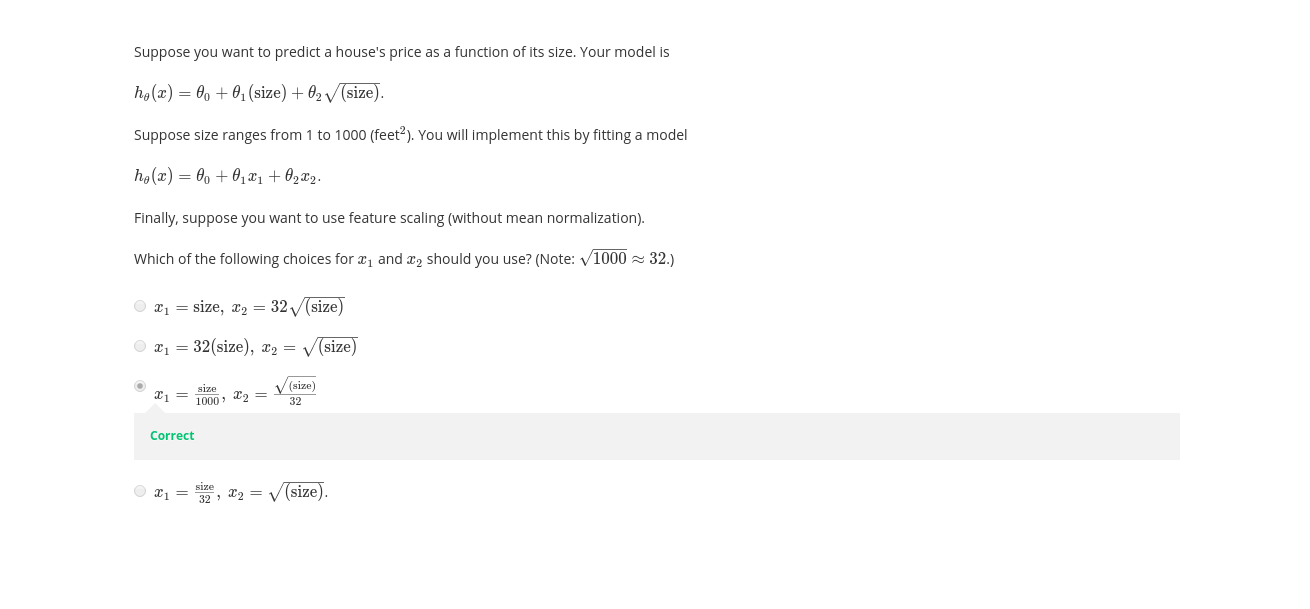
It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.



To summarize:

If α is too small: slow convergence.

If α is too large: ￼may not decrease on every iteration and thus may not converge.



# Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways.

We can combine multiple features into one. For example, we can combine x1​ and x2​ into a new feature x3​ by taking x1​⋅x2​.

### Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can change the behavior or curve of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is hθ​(x)=θ0​+θ1​x1​ then we can create additional features based on x1​, to get the quadratic function hθ​(x)=θ0​+θ1​x1​+θ2​x12​ or the cubic function hθ​(x)=θ0​+θ1​x1​+θ2​x12​+θ3​x13

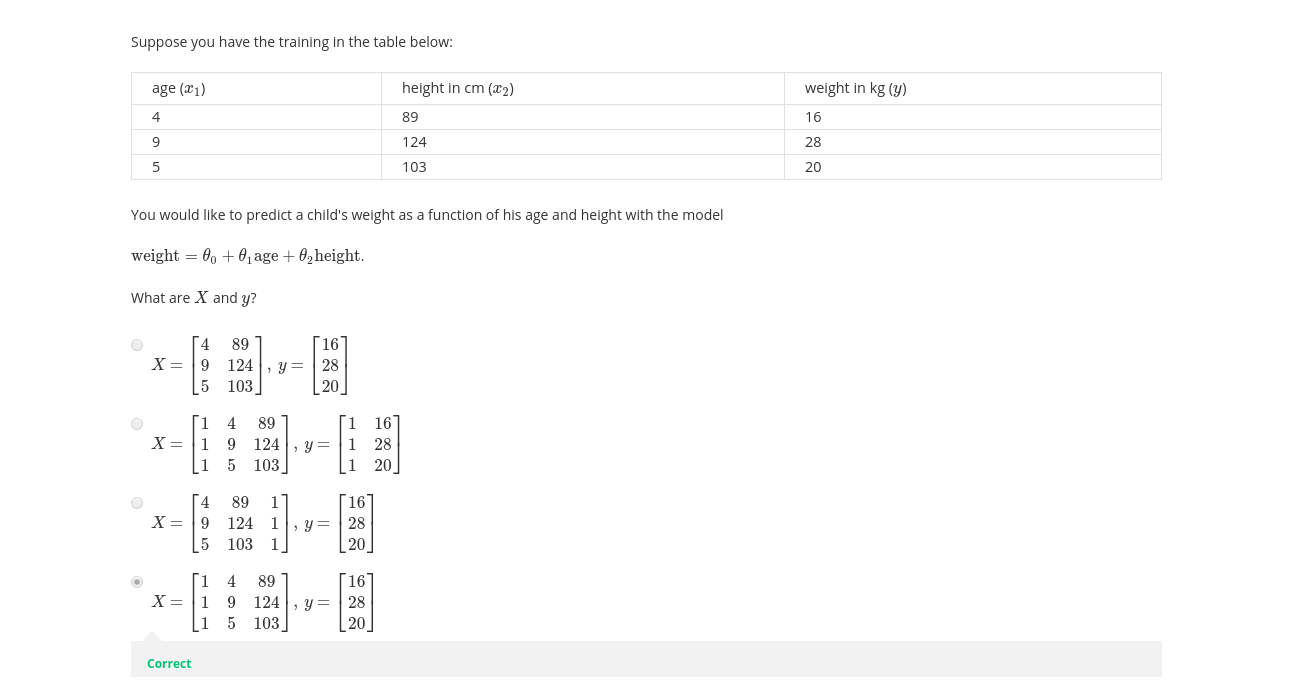
​

In the cubic version, we have created new features x2​ and x3​ where x2​=x12​ and x3​=x13​.

To make it a square root function, we could do: hθ​(x)=θ0​+θ1​x1​+θ2​x1​​

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

eg. if x1​ has range 1 - 1000 then range of x12​ becomes 1 - 1000000 and that of x13​ becomes 1 - 1000000000

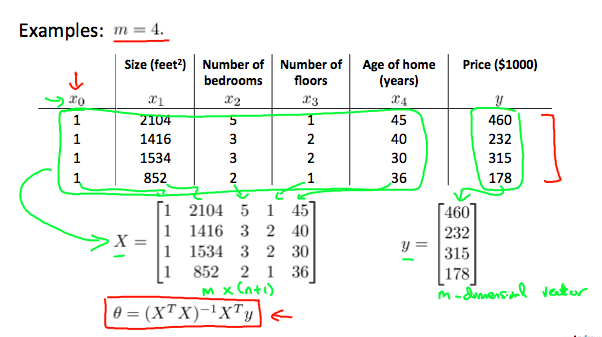


# Normal Equation

Note: [8:00 to 8:44 - The design matrix X (in the bottom right side of the slide) given in the example should have elements x with subscript 1 and superscripts varying from 1 to m because for all m training sets there are only 2 features x0​ and x1​. 12:56 - The X matrix is m by (n+1) and NOT n by n. ]

Gradient descent gives one way of minimizing J. Let’s discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj ’s, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

θ=(XTX)−1XTy

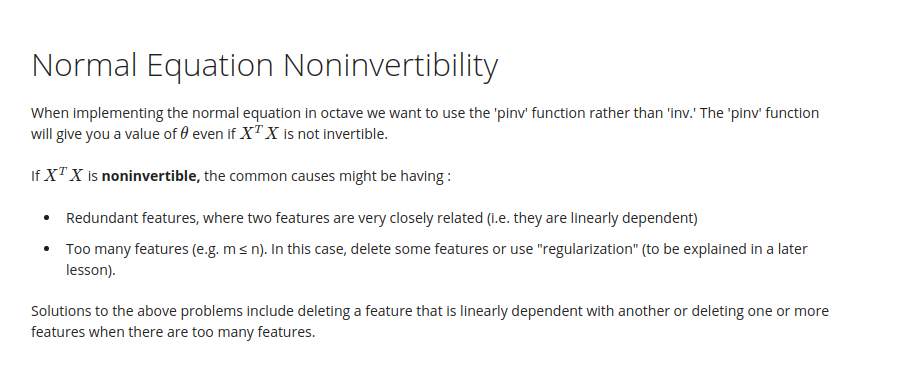


There is no need to do feature scaling with the normal equation.

The following is a comparison of gradient descent and the normal equation:

|  |  |
| --- | --- |
| Gradient Descent | Normal Equation |
| Need to choose alpha | No need to choose alpha |
| Needs many iterations | No need to iterate |
| O (kn2) | O (n3), need to calculate inverse of XTX |
| Works well when n is large | Slow if n is very large |

With the normal equation, computing the inversion has complexity O(n3). So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.



Thank you to Machine Learning Mentor, Tom Mosher, for compiling this list

## Subject: Confused about "h(x) = theta' \* x" vs. "h(x) = X \* theta?"

### Text:

The lectures and exercise PDF files are based on Prof. Ng's feeling that novice programmers will adapt to for-loop techniques more readily than vectorized methods. So the videos (and PDF files) are organized toward processing one training example at a time. The course uses column vectors (in most cases), so h (a scalar for one training example) is theta' \* x.

Lower-case x typically indicates a single training example.

The more efficient vectorized techniques always use X as a matrix of all training examples, with each example as a row, and the features as columns. That makes X have dimensions of (m x n). where m is the number of training examples. This leaves us with h (a vector of all the hypothesis values for the entire training set) as X \* theta, with dimensions of (m x 1).

X (as a matrix of all training examples) is denoted as upper-case X.

Throughout this course, dimensional analysis is your friend.

## Subject: Tips from the Mentors: submit problems and fixing program errors

### Text:

This post contains some frequently-used tips about the course, and to help get your programs working correctly.

### The Most Important Tip:

Search the forum before posting a new question. If you've got a question, the chances are that someone else has already posted it, and received an answer. Save time for yourself and the Forum users by searching for topics before posting a new one.

### Running your scripts:

At the Octave/Matlab command line, you do not need to include the ".m" portion of the script file name. If you include the ".m", you'll get an error message about an invalid indexing operation. So, run the Exercise 1 script by typing just "ex1" at the command line.

You also do not need to include parenthesis () when using the submit script. Just type "submit".

You cannot execute your functions by simply typing the name. All of the functions you will work on require a set of parameter values, enter between a set of parenthesis. Your three methods of testing your code are:

1 - use an exercise script, such as "ex1"

2 - use a Unit Test (see below) where you type-in the entire command line including the parameters.

3 - use the submit script.

### Making the grader happy:

The submit grader uses a different test case than what is in the PDF file. These test cases use a different size of data set and are more sensitive to small errors than the ex test cases. Your code must work correctly with any size of data set.

Your functions must handle the general case. This means:

- You should avoid using hard-coded array indexes.

- You should avoid having fixed-length arrays and matrices.

It is very common for students to think that getting the same answer as listed in the PDF file means they should get full credit from the grader. This is a false hope. The PDF file is just one test case. The grader uses a different test case.

Also, the grader does not like your code to send any additional outputs to the workspace. So, every line of code should end with a semicolon.

### Getting Help:

When you want help from the Forum community, please use this two-step procedure:

1 - Search the Forum for keywords that relate to your problem. Searching by the function name is a good start.

2 - If you don't find a suitable thread, then do this:

2a - Find the unit tests for that exercise (see below), and run the appropriate test. Attempt to debug your code.

2b - Take a screen capture of your whole console workspace (including the command line), and post it to the forum, along with any other useful information (computer type, Octave/Matlab version, other tests you've tried, etc).

### Debugging:

If your code runs but gives the wrong answers, you can insert a "keyboard" command in your script, just before the function ends. This will cause the program to exit to the debugger, so you can inspect all your variables from the command line. This often is very helpful in analysing math errors, or trying out what commands to use to implement your function.

There are additional test cases and tutorials listed in pinned threads under "All Course Discussions". The test cases are especially helpful in debugging in situations where you get the expected output in ex but get no points or an error when submitting.

### Unit Tests:

Each programming assignment has a "Discussions" area in the Forum. In this section you can often find "unit tests". These are additional test cases, which give you a command to type, and provides the expected results. It is always a good idea to test your functions using the unit tests before submitting to the grader.

If you run a unit test and do not get the correct results, you can most easily get help on the forums by posting a screen capture of your workspace - including the command line you entered, and the results.

### Having trouble submitting your work to the grader?:

- This section will need to be supplemented with info appropriate to the new submission system. If you run the submit script and get a message that your identity can't be verified, be sure that you have logged-in using your Coursera account email and your Programming Assignment submission password.

- If you get the message "submit undefined", first check that you are in the working directory where you extracted the files from the ZIP archive. Use "cd" to get there if necessary.

- If the "submit undefined" error persists, or any other "function undefined" messages appear, try using the "addpath(pwd)" command to add your present working directory (pwd) to the Octave execution path.

-If the submit script crashes with an error message, please see the thread "Mentor tips for submitting your work" under "All Course Discussions".

-The submit script does not ask for what part of the exercise you want to submit. It automatically grades any function you have modified.

### Found some errata in the course materials?

This course material has been used for many previous sessions. Most likely all of the errata has been discovered, and it's all documented in the 'Errata' section under 'Supplementary Materials'. Please check there before posting errata to the Forum.

Error messages with fmincg()

The "short-circuit" warnings are due to use a change in the syntax for conditional expressions (| and & vs || and &&) in the newer versions of Matlab. You can edit the fmincg.m file and the warnings may be resolved.

### Warning messages about "automatic broadcasting"?

See [this](https://www.gnu.org/software/octave/doc/interpreter/Broadcasting.html) link for info.

### Warnings about "divide by zero"

These are normal in some of the exercises, and do not represent a problem in your function. You can ignore them - Octave senses the issue and substitutes a +Inf or -Inf value so your program continues to execute.