Torch for Matlab® Users

Ata Mahjoubfar
ata.m@ucla.edu

April 23, 2015

General Commands

Get help for a specific function:

Matlab	`help sqrt`
Torch	`help(torch.sqrt)`

Documentation of `sqrt` function is shown.

Scalar operations:

Matlab	`testVar1 = (4*5-1+10.1)/3^2;`
Torch	`testVar1 = (4*5-1+10.1)/3^2;`

The resultant value (3.2333) is assigned to variable `testVar1`.

Scalar assignment:

Matlab	`testVar2 = testVar1;` or `testVar2 = testVar1`
Torch	`testVar2 = testVar1;` or `testVar2 = testVar1`

The value of variable `testVar1` (3.2333) is assigned to variable `testVar2`.

Matrices and Tensors

Create a two-dimensional tensor or matrix:

Matlab	`m = [9, 6, 3, 4; 7, 2, 8, 1]`
Torch	`m = torch.Tensor({{9, 6, 3, 4}, {7, 2, 8, 1}})`

A 2×4 two-dimensional tensor (matrix) with specified elements is formed and assigned to `m`.

Create a row vector:

Matlab	`v = [9, 7, 6, 8]` or `v = [9 7 6 8]`
Torch	`v = torch.Tensor({{9, 7, 6, 8}})`

Unlike Matlab, this is not used as a vector in Torch because it is a two-dimensional Tensor.

Create a column vector:

Matlab	`v = [9; 7; 6; 8]`
Torch	`v = torch.Tensor({{9}, {7}, {6}, {8}})`

Again unlike Matlab, this is not used as a vector in Torch because it is still a two-dimensional Tensor.

---

*Matlab® is a registered trademark of The MathWorks, Inc.

Create a one-dimensional tensor:

Matlab	*Not available*
Torch	`t = torch.Tensor({9, 7, 6, 8})`

Matlab does not have *one-dimensional tensors*; this can be verified by running `ndims([9, 7, 6, 8])`. On the other hand, for many operations that Matlab uses row or column vectors, Torch uses one-dimensional tensors.

Access an element in a vector or one-dimensional tensor:

Matlab	`v(2)`
Torch	`t[2]`

Second element of the vector is accessed.

Access an element from the end of a vector or one-dimensional tensor:

Matlab	`v(end-1)`
Torch	`t[-2]`

Second element from the end of the vector is accessed.

Access a range of elements in a vector or one-dimensional tensor:

Matlab	`v(2:4)`
Torch	`t[{{2,4}}]`

Second to fourth elements of the vector are accessed.

Access an element in a matrix:

Matlab	`m(2, 3)`
Torch	`m[{2, 3}]` or `m[2][3]`

Second row, third column element is accessed.

Access a row in a matrix as a two-dimensional tensor:

Matlab	`m(2, :)`
Torch	`m[{{2}, {}}]`

The returned row is a two-dimensional tensor.

Access a row in a matrix as a one-dimensional tensor:

Matlab	*Not available*
Torch	`m[{2, {}}]` or `m[2]`

The returned row is a one-dimensional tensor.

Access a column in a matrix as a two-dimensional tensor:

Matlab	`m(:, 2)`
Torch	`m[{{}, {2}}]`

The returned column is a two-dimensional tensor.

Access a column in a matrix as a one-dimensional tensor:

Matlab	*Not available*
Torch	`m[{{}, 2}]`

The returned column is a one-dimensional tensor.

Access a range of elements in a matrix:

Matlab	m(2, 2:4)
Torch	m[{{2}, {2,4}}] or m[{2, {2,4}}]

The second to fourth columns of the second row are returned. In Torch, there is a slight difference between using `index` (e.g. 2) or {index} (e.g. {2}) for pointing to a singleton dimension. For {index}, the dimension of the returned tensor is same as the original tensor (e.g. tensor m). For `index`, the singleton dimension is removed, and the dimension of the returned tensor is one less than the original tensor (e.g. tensor m). Also, {} refers to all elements in that dimension. Finally, -index (e.g. -4) means index-th (e.g. fourth) element from the end.

Forming Basic Tensors

Create a vector over a range of values with unit step:

Matlab	3:8
Torch	torch.range(3, 8)

Torch's result is a one-dimensional tensor with elements spaced 1 strating at 3.

Create a vector over a range of values with an arbitrary step:

Matlab	3:-1.9:-4.2
Torch	torch.range(3, -4.2, -1.9)

Torch's result is a one-dimensional tensor with elements spaced -1.9 strating at 3.

Create a vector with linearly-located elements:

Matlab	linspace(3, 8, 50)
Torch	torch.linspace(3, 8, 50)

Torch's result is a one-dimensional tensor with 50 equally-spaced elements strating at 3 and ending at 8.

Create a vector with logarithmically-located elements:

Matlab	logspace(3, 8, 50)
Torch	torch.logspace(3, 8, 50)

Torch's result is a one-dimensional tensor with 50 exponentially-spaced elements strating at 10^3 and ending at 10^8.

Create an all zeros vector or one-dimensional tensor:

Matlab	zeros(1,4) or zeros(4,1)
Torch	torch.zeros(4)

Torch's result is a one-dimensional tensor with 4 zero elements.

Create an all zeros matrix:

Matlab	zeros(5,3)
Torch	torch.zeros(5,3)

5×3 matrix of zeros is generated.

Create an all ones vector or one-dimensional tensor:

Matlab	ones(1,4) or ones(4,1)
Torch	torch.ones(4)

Torch's result is a one-dimensional tensor with 4 one elements.

Create an all ones matrix:

Matlab	ones(5,3)
Torch	torch.ones(5,3)

5×3 matrix of ones is generated.

Create an identity matrix:

Matlab	eye(5,3)
Torch	torch.eye(5,3)

5×3 identity matrix is generated.

Create a square identity matrix:

Matlab	eye(4)
Torch	torch.eye(4)

4×4 identity matrix is generated.

Create a uniformly-distributed random vector or one-dimensional tensor:

Matlab	rand(1,4) or rand(4,1)
Torch	torch.rand(4)

Torch's result is a one-dimensional tensor with 4 random elements from uniform probability distribution.

Create a uniformly-distributed random matrix:

Matlab	rand(5,3)
Torch	torch.rand(5,3)

5×3 matrix of random elements from uniform probability distribution is generated.

Create a normally-distributed random vector or one-dimensional tensor:

Matlab	randn(1,4) or randn(4,1)
Torch	torch.randn(4)

Torch's result is a one-dimensional tensor with 4 random elements from normal probability distribution.

Create a normally-distributed random matrix:

Matlab	randn(5,3)
Torch	torch.randn(5,3)

5×3 matrix of random elements from normal probability distribution is generated.

Operations

Tensor assignment without memory copy:

Matlab	*Not directly available*
Torch	matOut = matIn or matOut = matIn;

Matlab internally handles memory assignment for a copied array. As soon, as the copied array is modified in Matlab, a copy of the initial array is generated. Whereas, Torch gives this option to have a copied tensor that points to the same location in the memory as the initial tensor. For example, if elements of the matOut tensor are modified here, the same changes happen to the elements of the matIn tensor.

Tensor assignment with memory copy:

Matlab	`matOut = matIn or matOut = matIn;`
Torch	`matOut = matIn:clone() or matOut = matIn:clone();`

Matlab's internal memory handling for array assignment is somewhat closer to this. Here, Torch generates a copy of the tensor content in memory. Any changes in `matOut` elements are independent of the changes in the elements of the `matIn` tensor.

Square root of elements returned in a new tensor:

Matlab	`matOut = sqrt(matIn)`
Torch	`matOut = torch.sqrt(matIn)`

Square root of each element is saved in an output tensor with the same size as the input tensor.

Square root of elements returned in the same tensor:

Matlab	`matIn = sqrt(matIn)`
Torch	`matIn = torch.sqrt(matIn) or matIn.sqrt(matIn) or matIn:sqrt()`

`matIn:sqrt()` is same as `matIn.sqrt(matIn)`. Generally in Torch, `object:function(p1, p2, ...)` is same as `object.function(object, p1, p2, ...)`, where the function's object (also known as `self`) is passed as the first input to the function.

Element-wise power to a scalar:

Matlab	`matOut = matIn.^5`
Torch	`matOut = torch.pow(matIn, 5)`

Each element to the power of 5 is saved in an output tensor with the same size as the input tensor.

Element-wise power to a tensor:

Matlab	`matOut = matIn.^matPow`
Torch	`matOut = torch.cpow(matIn, matPow)`

Each element of the `matIn` tensor to the power of the corresponding element of the `matPow` tensor is saved in an output tensor with the same size as the input tensor.