

Torch for Matlab[®] Users

Ata Mahjoubfar
ata.m@ucla.edu

April 28, 2015

General Commands

Get help for a specific function:

Matlab	<code>help sqrt</code>
Torch	<code>help(torch.sqrt)</code>

Documentation of sqrt function is shown.

Scalar operations:

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

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

Scalar assignment:

Matlab	<code>testVar2 = testVar1; or testVar2 = testVar1</code>
Torch	<code>testVar2 = testVar1; or testVar2 = testVar1</code>

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

Display strings and numbers:

Matlab	<code>disp(['The distance is ', num2str(100), ' meters.'])</code>
Torch	<code>print('The distance is ' .. 100 .. ' meters.')</code>

This prints The distance is 100 meters. in the output.

Determine type:

Matlab	<code>class('Hi')</code>
Torch	<code>torch.type('Hi')</code>

For this example, Matlab returns char, and Torch returns string.

Matrices and Tensors

Create a two-dimensional tensor or matrix:

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

A 2x4 two-dimensional tensor (matrix) with specified elements is formed and assigned to m.

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

Number of dimensions of a tensor:

Matlab	<code>ndims(m)</code>
Torch	<code>m:dim()</code>

For example here, for the previously defined two-dimensional tensor (matrix) `m`, the returned number is 2.

Size of a tensor in all dimensions:

Matlab	<code>size(m)</code>
Torch	<code>m:size()</code> or <code>#m</code>

For example here, for the previously defined two-dimensional tensor (matrix) `m`, the returned numbers are 2 and 4, which are the numbers of rows and columns, respectively.

Size of a tensor in a specific dimension:

Matlab	<code>size(m,2)</code>
Torch	<code>m:size(2)</code> or <code>(#m) [2]</code>

For example here, for the previously defined two-dimensional tensor (matrix) `m`, the returned number is 4, which is the size of the second dimension (number of columns).

Number of elements in a tensor:

Matlab	<code>numel(m)</code>
Torch	<code>m:numel()</code>

For example here, for the previously defined two-dimensional tensor (matrix) `m`, the returned number is 8.

Create a row vector:

Matlab	<code>v = [9, 7, 6, 8]</code> or <code>v = [9 7 6 8]</code>
Torch	<code>v = torch.Tensor({{9, 7, 6, 8}})</code>

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

Create a column vector:

Matlab	<code>v = [9; 7; 6; 8]</code>
Torch	<code>v = torch.Tensor({{9}}, {7}, {6}, {8})</code>

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

Create a one-dimensional tensor:

Matlab	<i>Not available</i>
Torch	<code>t = torch.Tensor({9, 7, 6, 8})</code>

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	<code>v(2)</code>
Torch	<code>t[2]</code>

Second element of the vector is accessed.

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

Matlab	<code>v(end-1)</code>
Torch	<code>t[-2]</code>

Second element from the end of the vector is accessed.

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

Matlab	<code>v(2:4)</code>
Torch	<code>t[{2,4}]</code>

Second to fourth elements of the vector are accessed.

Access an element in a matrix:

Matlab	<code>m(2, 3)</code>
Torch	<code>m[{2, 3}]</code> or <code>m[2][3]</code>

Second row, third column element is accessed.

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

Matlab	<code>m(2, :)</code>
Torch	<code>m[{2}, {}]</code>

The returned row is a two-dimensional tensor.

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

Matlab	<i>Not available</i>
Torch	<code>m[{2, {}}]</code> or <code>m[2]</code>

The returned row is a one-dimensional tensor.

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

Matlab	<code>m(:, 2)</code>
Torch	<code>m[{}, {2}]</code>

The returned column is a two-dimensional tensor.

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

Matlab	<i>Not available</i>
Torch	<code>m[{}, 2]</code>

The returned column is a one-dimensional tensor.

Access a range of elements in a matrix:

Matlab	<code>m(2, 2:4)</code>
Torch	<code>m[{2}, {2,4}]</code> or <code>m[2, {2,4}]</code>

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	<code>3:8</code>
Torch	<code>torch.range(3, 8)</code>

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	<code>3:-1.9:-4.2</code>
Torch	<code>torch.range(3, -4.2, -1.9)</code>

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

Create a vector with linearly-located elements:

Matlab	<code>linspace(3, 8, 50)</code>
Torch	<code>torch.linspace(3, 8, 50)</code>

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	<code>logspace(3, 8, 50)</code>
Torch	<code>torch.logspace(3, 8, 50)</code>

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	<code>zeros(1,4)</code> or <code>zeros(4,1)</code>
Torch	<code>torch.zeros(4)</code>

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

Create an all zeros matrix:

Matlab	<code>zeros(5,3)</code>
Torch	<code>torch.zeros(5,3)</code>

5×3 matrix of zeros is generated.

Create an all ones vector or one-dimensional tensor:

Matlab	<code>ones(1,4)</code> or <code>ones(4,1)</code>
Torch	<code>torch.ones(4)</code>

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

Create an all ones matrix:

Matlab	<code>ones(5,3)</code>
Torch	<code>torch.ones(5,3)</code>

5×3 matrix of ones is generated.

Create an identity matrix:

Matlab	<code>eye(5,3)</code>
Torch	<code>torch.eye(5,3)</code>

5×3 identity matrix is generated.

Create a square identity matrix:

Matlab	<code>eye(4)</code>
Torch	<code>torch.eye(4)</code>

4×4 identity matrix is generated.

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

Matlab	<code>rand(1,4)</code> or <code>rand(4,1)</code>
Torch	<code>torch.rand(4)</code>

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

Create a uniformly-distributed random matrix:

Matlab	<code>rand(5,3)</code>
Torch	<code>torch.rand(5,3)</code>

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

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

Matlab	<code>randn(1,4)</code> or <code>randn(4,1)</code>
Torch	<code>torch.randn(4)</code>

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

Create a normally-distributed random matrix:

Matlab	<code>randn(5,3)</code>
Torch	<code>torch.randn(5,3)</code>

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

Operations

Tensor assignment without memory copy:

Matlab	<i>Not directly available</i>
Torch	<code>matOut = matIn</code> or <code>matOut = matIn;</code>

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	<code>matOut = matIn</code> or <code>matOut = matIn;</code>
Torch	<code>matOut = matIn:clone()</code> or <code>matOut = matIn:clone();</code>

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.

Change type of a tensor:

Matlab	<code>uint8(matIn), int8(matIn), int16(matIn), int32(matIn), int64(matIn), single(matIn), or double(matIn)</code>
Torch	<code>matIn:byte(), matIn:char(), matIn:short(), matIn:int(), matIn:long(), matIn:float(), or matIn:double()</code>

By default, double precision is used.

Multiplication of a tensor by a scalar:

Matlab	<code>matIn * 17</code> or <code>17 * matIn</code>
Torch	<code>matIn * 17</code>

Pay attention that in Torch, the scalar cannot be the first argument of the multiplication.

Matrix multiplication of a tensor by another tensor:

Matlab	<code>matA * matB</code>
Torch	<code>matA * matB</code>

Tensor sizes must be appropriate for the multiplication.

Element-wise multiplication of a tensor by another tensor:

Matlab	<code>matA .* matB</code>
Torch	<code>torch.cmul(matA, matB)</code>

Tensor sizes must be identical.

Transpose of a two-dimensional tensor (matrix):

Matlab	<code>matIn.'</code>
Torch	<code>matIn:t()</code>

A new tensor, which is the transpose of the input matrix (two-dimensional tensor) is returned.

Vertical tensor concatenation:

Matlab	<code>[matTop; matBottom]</code>
Torch	<code>torch.cat(matTop, matBottom, 1)</code>

The number of columns in input tensors must be equal.

Horizontal tensor concatenation:

Matlab	<code>[matLeft, matRight]</code>
Torch	<code>torch.cat(matLeft, matRight, 2)</code>

The number of rows in input tensors must be equal.

Square root of elements returned in a new tensor:

Matlab	<code>matOut = sqrt(matIn)</code>
Torch	<code>matOut = torch.sqrt(matIn)</code>

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	<code>matIn = sqrt(matIn)</code>
Torch	<code>matIn = torch.sqrt(matIn)</code> or <code>matIn.sqrt(matIn)</code> or <code>matIn:sqrt()</code>

`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	<code>matIn.^5</code>
Torch	<code>torch.pow(matIn, 5)</code>

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	<code>matIn.^matPow</code>
Torch	<code>torch.cpow(matIn, matPow)</code>

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.