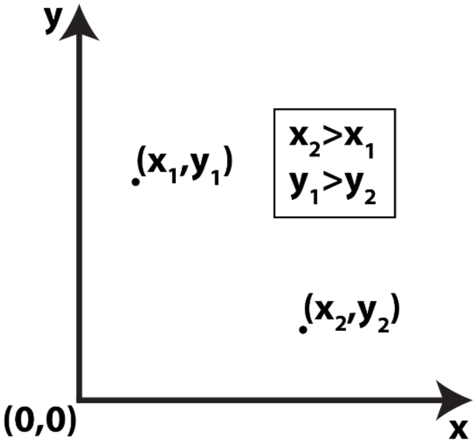
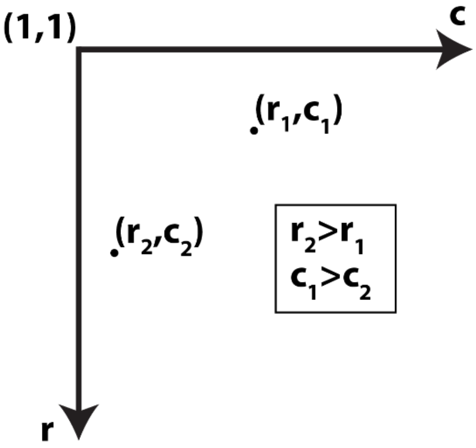
Matlab has eschewed an analogy with the “normal” convention used in a Cartesian coordinate system where we locate a point (x,y) taking x to be the horizontal coordinate and y the vertical coordinate, where y increases from 0 going upward and x increases from 0 going to the right, as in:



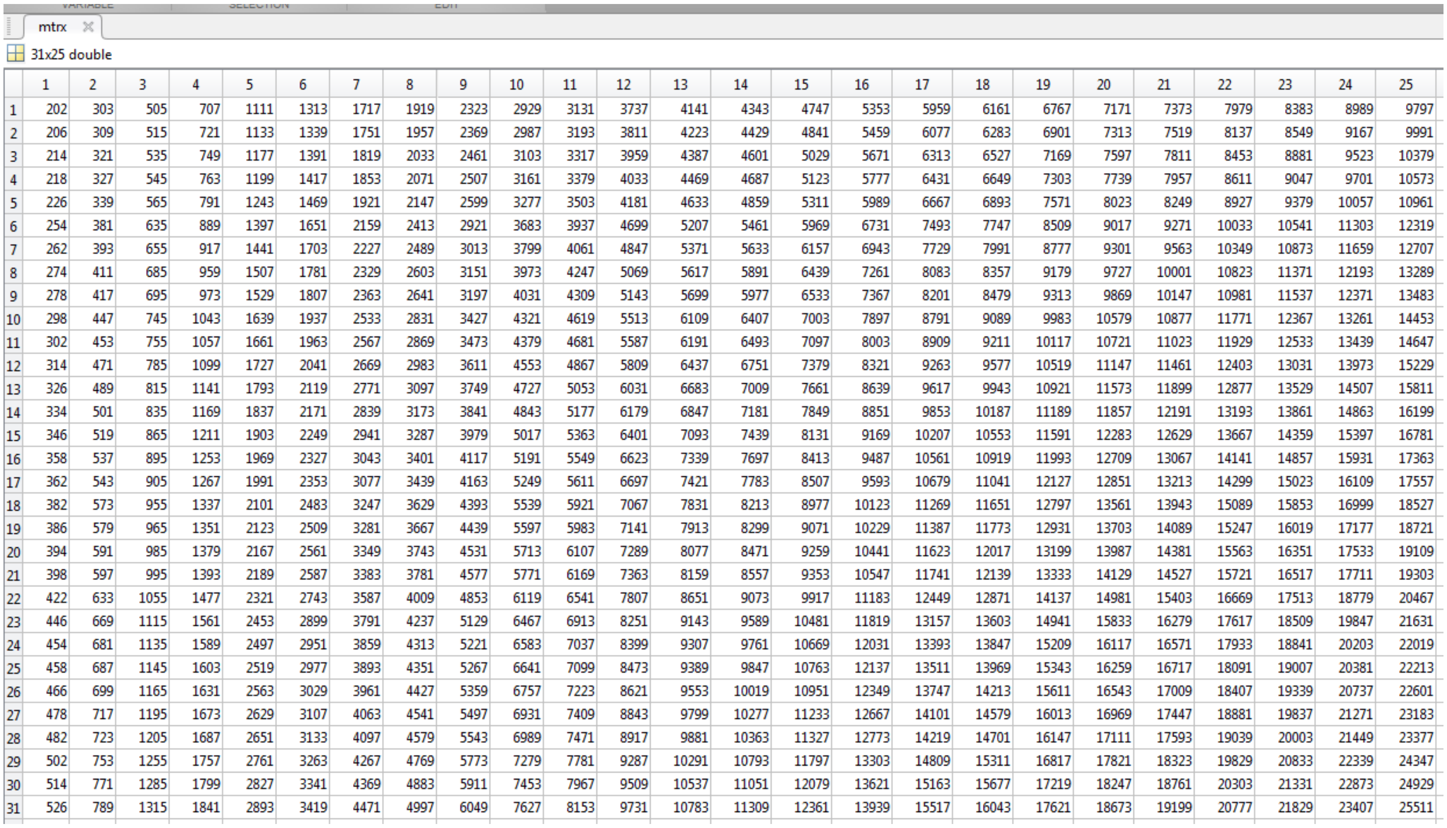
In Matlab, we’re not discussing coordinates exactly, but rather an element in an array. As such, instead of having a coordinate (x,y), we have an element located at (r,c) where r is the row number and c is the column number. The coordinate system then is just an analogy for integer array element locations. The coordinate system is changed in three important ways. Firstly, the origin is now (1,1) instead of (0,0). Secondly, the vertical dimension now corresponds to the first entry in the index, r in (r,c), and the horizontal dimension now corresponds to the second entry in the index, c in (r,c). Finally, the vertical dimension now increases going downwards. In Matlab the coordinate system is:



To illustrate some concepts about Matlab coordinates let’s create a matrix called mtrx, which has all unique elements. To do this I’ve used products of primes, you can create mtrx in the workspace by entering the following commands:

```
>> pr=primes(1e4);
>> n=28;a=pr(1:n-3);b=pr((n-2):(2*n));
>> mtrx=repmat(a,[n+3,1]).*repmat(b',[1,n-3]);
```

This creates a matrix shown below:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	202	303	505	707	1111	1313	1717	1919	2323	2929	3131	3737	4141	4343	4747	5353	5959	6161	6767	7171	7373	7979	8383	8989	9797
2	206	309	515	721	1133	1339	1751	1957	2369	2987	3193	3811	4223	4429	4841	5459	6077	6283	6901	7313	7519	8137	8549	9167	9991
3	214	321	535	749	1177	1391	1819	2033	2461	3103	3317	3959	4387	4601	5029	5671	6313	6527	7169	7597	7811	8453	8881	9523	10379
4	218	327	545	763	1199	1417	1853	2071	2507	3161	3379	4033	4469	4687	5123	5777	6431	6649	7303	7739	7957	8611	9047	9701	10573
5	226	339	565	791	1243	1469	1921	2147	2599	3277	3503	4181	4633	4859	5311	5989	6667	6893	7571	8023	8249	8927	9379	10057	10961
6	254	381	635	889	1397	1651	2159	2413	2921	3683	3937	4699	5207	5461	5969	6731	7493	7747	8509	9017	9271	10033	10541	11303	12319
7	262	393	655	917	1441	1703	2227	2489	3013	3799	4061	4847	5371	5633	6157	6943	7729	7991	8777	9301	9563	10349	10873	11659	12707
8	274	411	685	959	1507	1781	2329	2603	3151	3973	4247	5069	5617	5891	6439	7261	8083	8357	9179	9727	10001	10823	11371	12193	13289
9	278	417	695	973	1529	1807	2363	2641	3197	4031	4309	5143	5699	5977	6533	7367	8201	8479	9313	9869	10147	10981	11537	12371	13483
10	298	447	745	1043	1639	1937	2533	2831	3427	4321	4619	5513	6109	6407	7003	7897	8791	9089	9983	10579	10877	11771	12367	13261	14453
11	302	453	755	1057	1661	1963	2567	2869	3473	4379	4681	5587	6191	6493	7097	8003	8909	9211	10117	10721	11023	11929	12533	13439	14647
12	314	471	785	1099	1727	2041	2669	2983	3611	4553	4867	5809	6437	6751	7379	8321	9263	9577	10519	11147	11461	12403	13031	13973	15229
13	326	489	815	1141	1793	2119	2771	3097	3749	4727	5053	6031	6683	7009	7661	8639	9617	9943	10921	11573	11899	12877	13529	14507	15811
14	334	501	835	1169	1837	2171	2839	3173	3841	4843	5177	6179	6847	7181	7849	8851	9853	10187	11189	11857	12191	13193	13861	14863	16199
15	346	519	865	1211	1903	2249	2941	3287	3979	5017	5363	6401	7093	7439	8131	9169	10207	10553	11591	12283	12629	13667	14359	15397	16781
16	358	537	895	1253	1969	2327	3043	3401	4117	5191	5549	6623	7339	7697	8413	9487	10561	10919	11993	12709	13067	14141	14857	15931	17363
17	362	543	905	1267	1991	2353	3077	3439	4163	5249	5611	6697	7421	7783	8507	9593	10679	11041	12127	12851	13213	14299	15023	16109	17557
18	382	573	955	1337	2101	2483	3247	3629	4393	5539	5921	7067	7831	8213	8977	10123	11269	11651	12797	13561	13943	15089	15853	16999	18527
19	386	579	965	1351	2123	2509	3281	3667	4439	5597	5983	7141	7913	8299	9071	10229	11387	11773	12931	13703	14089	15247	16019	17177	18721
20	394	591	985	1379	2167	2561	3349	3743	4531	5713	6107	7289	8077	8471	9259	10441	11623	12017	13199	13987	14381	15563	16351	17533	19109
21	398	597	995	1393	2189	2587	3383	3781	4577	5771	6169	7363	8159	8557	9353	10547	11741	12139	13333	14129	14527	15721	16517	17711	19303
22	422	633	1055	1477	2321	2743	3587	4009	4853	6119	6541	7807	8651	9073	9917	11183	12449	12871	14137	14981	15403	16669	17513	18779	20467
23	446	669	1115	1561	2453	2899	3791	4237	5129	6467	6913	8251	9143	9589	10481	11819	13157	13603	14941	15833	16279	17617	18509	19847	21631
24	454	681	1135	1589	2497	2951	3859	4313	5221	6583	7037	8399	9307	9761	10669	12031	13393	13847	15209	16117	16571	17933	18841	20203	22019
25	458	687	1145	1603	2519	2977	3893	4351	5267	6641	7099	8473	9389	9847	10763	12137	13511	13969	15343	16259	16717	18091	19007	20381	22213
26	466	699	1165	1631	2563	3029	3961	4427	5359	6757	7223	8621	9553	10019	10951	12349	13747	14213	15611	16543	17009	18407	19339	20737	22601
27	478	717	1195	1673	2629	3107	4063	4541	5497	6931	7409	8843	9799	10277	11233	12667	14101	14579	16013	16969	17447	18881	19837	21271	23183
28	482	723	1205	1687	2651	3133	4097	4579	5543	6989	7471	8917	9881	10363	11327	12773	14219	14701	16147	17111	17593	19039	20003	21449	23377
29	502	753	1255	1757	2761	3263	4267	4769	5773	7279	7781	9287	10291	10793	11797	13303	14809	15311	16817	17821	18323	19829	20833	22339	24347
30	514	771	1285	1799	2827	3341	4369	4883	5911	7453	7967	9509	10537	11051	12079	13621	15163	15677	17219	18247	18761	20303	21331	22873	24929
31	526	789	1315	1841	2893	3419	4471	4997	6049	7627	8153	9731	10783	11309	12361	13939	15517	16043	17621	18673	19199	20777	21829	23407	25511

which we can verify has all unique entries by comparing the number of elements in the array (numel) with the number of unique elements

```
>> numel(mtrx)
ans =   775
>> size(unique(mtrx))
ans =   775     1
```

In Matlab the convention for array size is m x n, where m is the number of rows and n is the number of columns, thus:

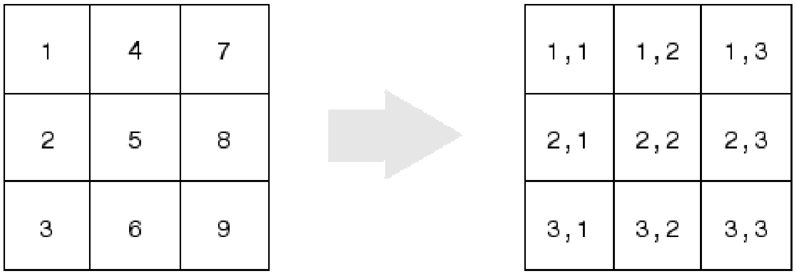
```
>> size(mtrx)
ans =     31    25
```

When you index an entry with subscripts you follow this convention, (row(s) , column(s)). Some examples:

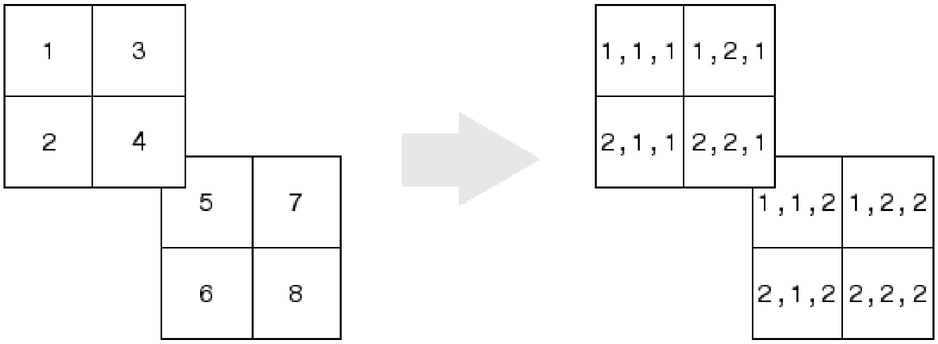
```
>> mtrx(2,3)
ans =   515
>> mtrx(3,2)
ans =   321
>> mtrx(1,end)
ans =    9797
>> mtrx(2:5,3)
ans =
   515
   535
   545
   565
>> mtrx(3,2:5)
```

```
ans =         321         535         749        1177
>> mtrx(7:9,2:5)
ans =
        393         655         917        1441
        411         685         959        1507
        417         695         973        1529
>> mtrx(3,[1,3,5])
ans =         214         535        1177
```

Array elements can also be indexed using linear indices, the convention in Matlab for linear indices (figure from Matlab's website) is, for a 3x3 array:



and for a 2x2x2 array:

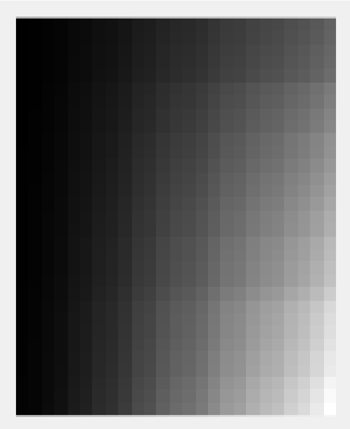


Thus in our unique matrix

```
>> mtrx(5)
ans =   226
>> mtrx(75)
ans =   815
>> mtrx(30:34)
ans =   514   526   303   309   321
```

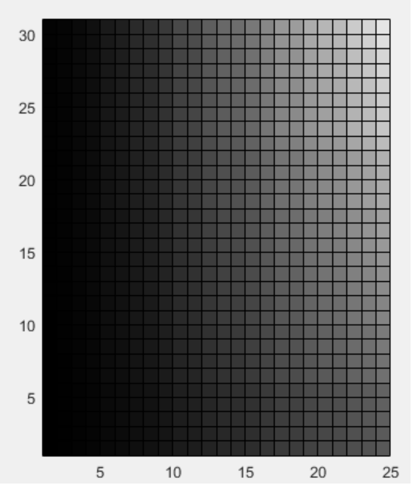
Now turning to images, if we use Matlab's most basic image display function imshow, we see that this coordinate system is preserved. In this figure white is the max value and black is the min value, with the greyscale linearly increasing between the two. So in mtrx where the values increase going down and to the right, and do so faster going to the right has the expected appearance in imshow.

```
>> imshow(mtrx,[])
```



Unfortunately, other plotting commands can produce different results, for example pcolor flips the vertical axis:

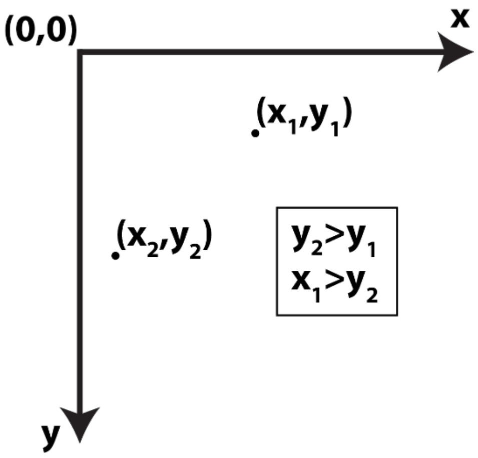
```
>> pcolor(mtrx)
>> axis image
```

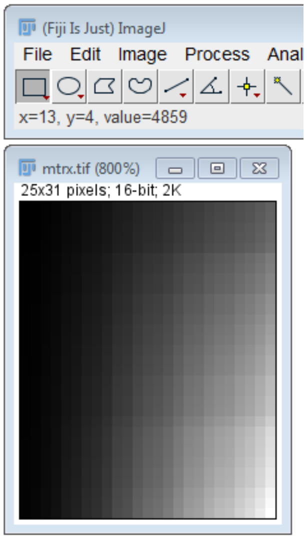


Let’s take a look at reading and writing images into Matlab. We can write mtrx to a .tif image using imwrite, where we first have to convert the array elements from double to uint16, as

```
>> imwrite(uint16(mtrx),'mtrx.tif')
```

Opening the image in ImageJ preserves the display orientation and but changes the coordinate system convention. In ImageJ the coordinate system and our image are:





x=13, y=4, value=4859

25x31 pixels; 16-bit; 2K

Note that the origin is shifted and (r,c)=(y,x), so whereas ImageJ reports that (x=13,y=4) = 4859, in Matlab we get

```
>> mtrx(13,4)
ans =       1141
>> mtrx(5,14)
ans =       4859
```

Reading the image in with imread also preserves the orientation and values, we can check this by importing and looking for any values which aren’t equal like:

```
>> mtrx_read=double(imread('mtrx.tif'));
>> find(mtrx_read~=mtrx)
ans =   Empty matrix: 0-by-1
```

We can verify that my preferred .tif reading and writing functions also work in the same fashion, using saveastiff and TIFFStack gives:

```
>> saveastiff(uint16(mtrx),'mtrx1.tif')
>> tfstk=TIFFStack('mtrx1.tif');
>> mtrx_tfstk=double(tfstk(:,:));
>> find(mtrx_tfstk~=mtrx)
ans =   Empty matrix: 0-by-1
```