CS513 Spring 21
Prof. Ron

HW #6, Factor=.5, special rules apply, see announcement on Canvas

Prepared by Narfi Stefansson

Due March 6, 2021

Comment:

The actual assignment is found at the end of this document.

Further `Matlab` introduction

After this second intorduction, you should be familiar with the following commands:

`path`	`addpath`
`startup`	`load`
`save`	`sort`
`min`	`max`
`sum`	`abs`
`==`	`<=`
`>=`	`~=`
`&`	`|`
`~`	`find`
`all`	`conv`
`norm(.,2), norm(.,1), norm(.,inf)`	`clear`
`^`	`.^` (pointwise exponentiation)
`/`	`./` (pointwise division)
`*` (matrix mult)	`.*` (pointwise multiplication)
`diag`	`orient tall/landscape`
`subplot`	`suptitle`
`hold on/off`	`plot` and `*, --, r, g, ...`
`axis on/off`	`axis(a)`
`axis tight`	`clf`
`fft`	`ifft`

and comfortable with using:
logical indices
submatrices: A(ind1, ind2) = ...

If you are familiar with the above `Matlab` commands, skip the preface and go directly to the last page, where the actual assignment is to be found. Otherwise, keep on reading.

Preface

If you want to be able to conveniently organize your projects, homework etc. you should consider putting all the files for each problem set or project into a separate directory and then use:

- `path`
- `addpath`
- `startup`

By giving the command `path` at the `Matlab` prompt, you will see `Matlab`'s current search path. You can add to `Matlab`'s path by either issuing the command:
`path('~/matlab/cs513/HWK1/', path);`
or
`addpath('~/matlab/cs513/HWK1/');`
The file `~/matlab/startup.m` is read (and run) every time you start `Matlab`, so you may want to add some commands to that file to e.g. add to `Matlab`'s search path at startup.

- `load`
- `save`

You can save your variables to disk by using the `save` command. `save` saves all variables to the file `matlab.mat`. `save mysave.mat` saves to the file `mysave.mat`. And you use `load` or `load mysave.mat` to load the variables back into `Matlab`.

In the 1st assignment, we accessed individual elements of matrices as well as single rows and columns. But it is also possible to access other submatrices of a matrix `A`. Look at the following example:
`A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];`
`B = A([1,2], [3, 4])` gives:

```
B =
     3     4
     7     8
```

We can look at this as:
the first row of `B` is taken from row 1 of `A`. And from that row of `A`, we take entries 3 and 4. And the second row of `B` is taken from row 2 of `A`, and from that row we take entries 3 and 4.

Or, we can look at this as:
The first column of `B` is taken from column 3 of `A`, the first element of that column is taken from row 1 of `A` and the second element of that column is taken from row 2 of `A`. Etc.

We can also repeat indices as often as we wish:
`A(3, [2 2 2])`

```
    10    10    10
```

In general, if `A` is a matrix and `b` and `c` are vectors of integers, then
`B = A(b,c);`
creates a matrix `B` of the size `length(b)`-by-`length(c)`, whose entries `B(i,j)` are `A(b(i),c(j))`. Of course this is provided the entries in `b` are all between 1 and the number of rows in `A`, and the entries in `c` are all between 1 and the number of columns in `A`. You should also recall that `b` and `c` can be replaced by `:`.

When just one argument is used for indexing a matrix, the matrix is treated as a vector, namely the vector `A(:)`, that is obtained by concatenating the columns of the matrix `A`. An example is given below.

The commands

- `sort`
- `min`
- `max`

all accept both vectors and matrices as arguments and can return either 1 or 2 arguments.
`s = sort(A);`
If `A` is a vector, `s` is a vector of the same size as `A`, with the elements of `A` sorted in ascending

order. If A is a matrix, sort(A) treats the columns of A as vectors, returning sorted columns. The command:
[s, ind] = sort(A);
returns s as before, as well as an array of indices, ind. The size of ind is size(A), and each column of ind is a permutation vector of the corresponding column of A. In particular, if A is a vector, then s = A(ind).

If A is a vector, then m = max(A); returns the maximum element of A. And if A is a matrix then m = max(A); treats the columns of A as vectors, returning a row rector containing the maximum element from each column.
max can also return 2 output arguments: [m, ind] = max(A); returns m as before as well as the indices of the maximum values of A. If A is a vector, then m = A(ind), but if A is a matrix, then m(i) = A(ind(i), i) for i = 1, ..., size(A,2).

Note that if you want to find the maximum element of a matrix A, and not just the maximum element in each column of A, you have to use either max(A(:)) or max(max(A)).

It should now come as now surprise that

- sum

works on each column of its input argument, if it is a matrix. If the input argument is a vector, sum simply returns the sum of its elements. There is also prod, cumsum and cumprod.

- abs

abs(A) has the same size as A, with each element of A replaced by its absolute value or modulus.

The relational operators

- == (equal to)
- <= (less than or equal to)
- >= (greater than or equal to)
- ~= (not equal to)

obey the same rule as all (actually, most) other binary operators in Matlab, which is as follows: Binary operators accept two matrices as arguments and they have to be of the same size. Except if one of them is a scalar, in which case it is regarded as a matrix of the same size as the other argument, with all elements equal to the scalar value.
As an example, we take the == operator. The command
[1, 2; 3, 4] == [1, 1; 4, 4] returns

```
     1     0
     0     1
```

And now we try to let one argument be a matrix and the other be a scalar:
[1, 2; 3, 4] >= 3 returns

```
     0     0
     1     1
```

An equivalent way of doing this, albeit more cumbersome, would have been to issue the command: [1, 2; 3, 4] >= 3*ones(2, 2).

The relational operators listed above return matrices containing true and false, and they can, in turn, be used as indices as the following example shows:

```
A = [1, 2; 3, 4];
A(A >= 3)
```

```
     3
     4
```

We get a powerful tool when we combine the relational operators with the logical operators

- `&` (and)
- `|` (or)
- `~` (not)

Again, the same rule as before applies: `A & B` is defined for matrices `A` and `B` that are of the same size, or if one of them is a scalar. Now `A` and `B` are matrices containing true and false, with the convention if `A` or `B` contain numerical values, then non-zero numerical values are interpreted as true, and zeros are interpreted as false.

- `find`

In its simplest form, when `A` is a vector, `k = find(A)` returns the indices of the elements of `A` that are true (or non-zero, if `A` contains numerical values). If `A` is a matrix, `k = find(A)` returns the indices of the true elements of `A(:)`. Look at the following:

```
A = [1, 3, 6; -3, 10, 50]
```

```
A =
     1     3     6
    -3    10    50
```

```
k = find(A < 0 | A >= 15)
```

```
k =
     2
     6
```

I.e. the 2nd and 6th elements of `A(:)` are `< 0` or `>= 15`:
`A(k)` gives:

```
    -3
    50
```

Make sure you understand how the vector `k` was used as an index into the matrix `A`, and compare the above to the command:

```
A(A < 0 | A >= 15)
```

The function

- `all`

tests whether all elements are true. If `A` is a vector then `all(A)` returns logical true if all elements of `A` are true (or non-zero) and logical false otherwise. If `A` is a matrix, `all(A)` treats the columns of `A` as vectors, returning a row vector of true and false. With `A` as above:

```
A = [1, 3, 6; -3, 10, 50];
```

we can easily see which columns have only non-negative elements:

```
all(A >= 0)
```

```
     0     1     1
```

There are many ways to look at the convolution operator, and one of these ways is to look at its relationship with trigonometric polynomials. Let a and b be trigonometric polynomials with positive exponents:

$$a(\omega) = \sum_{k=0}^{m} a(k) e^{ik\omega}$$

$$b(\omega) = \sum_{k=0}^{n} b(k) e^{ik\omega}$$

then the product ab is a trigonometric polynomial with (no more than) $m+n+1$ nonzero coefficients. And we let $c(k)$ be the coefficients of that polynomial:

$$ab(\omega) = \sum_{k=0}^{m+n} c(k)e^{ik\omega}.$$

- `conv`

This is exactly what the `Matlab` command `conv` does: It accepts 2 vectors as input (which would in our notation be `[`$a(0)$`, ..., `$a(m)$`]` and `[`$b(0)$`, ..., `$b(n)$`]`) and returns the coefficients of the product of the corresponding trigonometric polynomials, `[`$c(0)$`, ..., `$c(m+n)$`]`:

`conv([`$a(0)$`, ...,`$a(m)$`], [`$b(0)$`, ...,`$b(n)$`]) = [`$c(0)$`, ...,`$c(m+n)$`]`.

It is very instructive to do a few calculations by hand and compare with the `conv` command in `Matlab`, and you are encouraged to do so. As an example, if we want to convolve the vectors `[1, 3, 5]` and `[0, 2, 4]` by hand, we multiply together the trigonometric polynomials

$$1 + 3e^{i\omega} + 5e^{2i\omega} \quad \text{and} \quad 2e^{i\omega} + 4e^{2i\omega}.$$

Once we have expanded the product and collected all terms, we get:

$$\left(1 + 3e^{i\omega} + 5e^{2i\omega}\right)\left(2e^{i\omega} + 4e^{2i\omega}\right) = 2e^{i\omega} + 10e^{2i\omega} + 22e^{3i\omega} + 20e^{4i\omega}$$

which is in perfect agreement with `Matlab`:

```
conv([1, 3, 5], [0, 2, 4])

     0     2    10    22    20
```

The command

- `norm`

calculates matrix or vector norm. If `A` is a matrix then `norm(A)` is the 2-norm of `A`.
If `A` is a vector, `norm(A)` is the 2-norm of `A`. You may also need to use `norm(A, 1)` and `norm(A, inf)`.

- `clear`

The `clear` command clears variables from the memory. By itself, `clear` removes all variables from the memory, `clear x` removes only the variable `x` from the memory.

Common arithmetic operations:

- `^`
- `/`
- `*`
- `.^`
- `./`
- `.*`

The first three operations deviate a little bit from the Matlab convention that binary operators accept as input either two matrices of the same size, or a matrix and a scalar. But a moment's reflection should be enough to convince you that this is natural since these three operate on the entire matrices, and not just element-by-element. Some examples are given below.

^ If X is a matrix and p is an integer, X ^ p = X*X*...*X, p times.

/ slash or matrix right division. B/A is roughly the same as B*inv(A).

* matrix multiplication. If both A and B are matrices, A*B is the matrix product of A and B. If one of the two is a scalar, A*B is the scalar-matrix product of A and B.

However, the remaining three are pointwise operations, and in all cases A and B must have the same size, unless one of them is a scalar.

.^ pointwise exponentiation. A.^ B is the matrix with elements A(i,j) to the B(i,j) power.

./ pointwise division. A./B is the matrix with elements A(i,j)/B(i,j).

.* pointwise multiplication. A.*B is the matrix with elements A(i,j)*B(i,j).

Consider these examples:

```
A = [1, 2; 3, 4];
B = [1, 2; 3, 4];
```

A.^ 3 (raising each element to the third power)

```
     1     8
    27    64
```

A.*B (element-by-element multiplication)

```
     1     4
     9    16
```

A./B (element-by-element division)

```
     1     1
     1     1
```

- diag

The command diag creates diagonal matrices and extracts the diagonals of a matrix. If v is a vector, then A = diag(v) creates a matrix with v on the main diagonal and zeros elsewhere. A = diag(v, k) creates a matrix with v on the k-th diagonal of A. See diag in the helpdesk.

If A is a matrix, then diag(A) returns the diagonal of A, and diag(A, k) returns the elements on the k-th diagonal of A.

You can put several functions on one picture by using:

- plot
- hold

Try the following commands:

```
x = linspace(0, pi, 10); plot(x, sin(x), x, cos(x));
x = linspace(0, pi, 10); plot(x, sin(x), 'r--', x, cos(x), 'g*');
```

and notice how you can depict several functions in one picture as well as add attributes to each of the graphs, such as colors, (r or g), dashed lines (--) or disconnected stars (*). You can also tell Matlab to hold the current figure and add to it:

```
x = linspace(0, pi, 10); plot(x, sin(x));
hold on; plot(x, cos(x), 'r--'); hold off;
```

If you omit the hold commands, Matlab will create the first plot, clear that graph and then draw the second figure. If you want to clear the current figure, use:

- `clf`

You can further improve on the appearance of the plot by using the `axis` command.

- `axis on`
- `axis off`
- `axis tight`
- `axis equal`

See `help axis`.

You can put several plots on one figure by using

- `subplot`

The command `subplot(r, c, i)` divides the current figure into `r*c` subfigures, which are laid out in `r` rows and `c` columns. They are numbered row-wise and the `i`-th subfigure is set to be the current subfigure . You should try the following:

```
x = linspace(0, pi, 10);
subplot(2, 2, 1); plot(x, sin(x)); title('sin');
subplot(2, 2, 2); plot(x, cos(x)); title('cos');
subplot(2, 2, 3); plot(x, tan(x)); title('tan');
subplot(2, 2, 4); plot(x, atan(x)); title('atan');
```

When figures contain many fine details, as is often the case after heavy use of the `subplot` command, it becomes important to make sure that the details are all visible. By default `Matlab` leaves wide margins around its plots, but this can be changed by using `orient`.

- `orient`

The command `orient tall` tells `Matlab` to print the current figure in portrait mode and to reduce the borders around it, and `orient landscape` tells `Matlab` to print in landscape mode.

- `suptitle`

One thing is missing from `Matlab` and that is the ability to automatically add a title to the top of the figure, above all subplots. The function `suptitle` does just that, and it is available on the course homepage. You should download the file and install in your `Matlab` path and experiment with it. Try to issue the `subplot` commands given above, followed by, say:

```
suptitle('3 trig functions and one inverse trig function')
```

Last, but not least, are the functions:

- `fft`
- `ifft`

The functions `fft` and `ifft` are, as the names suggest, the Fast Fourier Transform and its inverse.

Assignment

1. Write a `Matlab` function that accepts 3 arguments, `m` and `n` which are assumed to be integer, and `s` which is assumed to be a 2-by-2 matrix of integers.

   Your function should return a matrix with 2 rows, such that its columns list the points $\left[\begin{smallmatrix} j \\ k \end{smallmatrix}\right]$ that satisfy:

   $$1 \le j \le \mathtt{m}, \quad 1 \le k \le \mathtt{n}, \quad \text{and} \quad \left[\begin{smallmatrix} j \\ k \end{smallmatrix}\right] \in s\mathbb{Z}^2$$

   where the last condition means that there should exist $\left[\begin{smallmatrix} p \\ o \end{smallmatrix}\right] \in \mathbb{Z}^2$ such that $\left[\begin{smallmatrix} j \\ k \end{smallmatrix}\right] = s\left[\begin{smallmatrix} p \\ o \end{smallmatrix}\right]$. Furthermore, your function should not print out anything. Note: try to avoid any use, in this question, `for` or `while` loops (it is possible to do without any loop).

   Turn in a printout of your function as well as the results from calling your function with the following parameters:

   (a) $\mathtt{m} = 4$, $\mathtt{n} = 6$, $\mathtt{s} = \left[\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right]$
   (b) $\mathtt{m} = 5$, $\mathtt{n} = 6$, $\mathtt{s} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$

   Hints: (1) There is only one vector $\left[\begin{smallmatrix} p \\ o \end{smallmatrix}\right]$ such that

   $$s\begin{bmatrix} p \\ o \end{bmatrix} = \begin{bmatrix} j \\ k \end{bmatrix},$$

   so the real issue is whether the entries of this unique solution are integers. (2) You may find it useful to look at the `Matlab` function `meshgrid` and to use `round`.

2. Let f be the function

   $$f(x) := x^2 e^{-3x^2} + (x/40)^2$$

   Return **on one page** plots of

   (a) $y = f(x), -10 \le x \le 10$
   (b) $y = f'(x), -10 \le x \le 10$
   (c) $y = f''(x), -10 \le x \le 10$

   as well as numerical estimates of where the function attains its global maximum/maxima on the interval $[-10, 10]$.

   Note: instead of finding expressions for f' and f'' you may find the following approximations useful: Let $h > 0$ be small, then

   $$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$
   $$f''(x) \approx \frac{f(x+h) + f(x-h) - 2f(x)}{h^2}$$

   If you are courageous, consider using the `Matlab` commands `diff` and `inline`, as in:

```
fs = 'x ^ 2 - x ^ 3';
dfs = diff(fs)
df = inline(dfs)
```