

Worksheet 1, for the MATLAB course by Hans G. Feichtinger, Edinburgh, Jan. 9th

Start MATLAB via: Start > Programs > School applications > Sci.+Eng. > Eng.+Electronics > MATLAB > R2007 (preferably).

Generally speaking I suggest that you start your session by opening up a diary with the command:

`diary mydiary1.m` and concluding the session with the command `diary off`. If you want to save your workspace you may want to call `save today1.m` in order to save all the current variable (names + values).

Moreover, using the `HELP` command from MATLAB you can get help on more or less every MATLAB command. Try simply `help inv` or `help plot`.

1. Define an arbitrary (random) 3×3 matrix A , and check whether it is invertible. Calculate the inverse matrix. Then define an arbitrary right hand side vector b and determine the (unique) solution to the equation $A * x = b$. Find in two different ways the solution to this problem, by either using the inverse matrix, or alternatively by applying Gauss elimination (i.e. the `RREF` command) to the extended system matrix $[A, b]$. In addition look at the output of the command `rref([A,eye(3)])`. What does it tell you?
2. Produce an “arbitrary” 7×7 matrix of rank 5. There are at least two simple ways to do this. Either by factorization, i.e. by obtaining it as a product of some 7×5 matrix with another random 5×7 matrix, or by purposely making two of the rows or columns linear dependent from the remaining ones. Maybe the second method is interesting (if you have time also try the first one):
3. Plotting routines are quite simple in MATLAB. `plot(d)` simply plots the data vector d of length n over $1 \dots n$. If d is real-valued this is done in the usual way, if d is complex-valued, e.g. $x = \text{rand}(1,n) + i * \text{rand}(1,n)$ then the plot is performed in the complex plane. So try e.g.
`n = 35; x = rand(1,n) + i * rand(1,n); plot(x,'r*'); title('a plot in the complex plane');`

See what happens if you do in addition: `axis square; plot(x,'k'); hold off; grid; figure(gcf);` If you want to mark the first and the last point on this polygonal curve specifically add `hold on; plot(x(1),'b*'); plot(x(n),'g*'); hold off; figure(gcf);` This last command “figure(gcf)” brings the last figure to the surface. The command “figure” itself opens up a new figure.

If a matrix is inserted into the plotting command the entries (columns) are plotted column-wise. Just as an example try: `T = zeros(8,6); T(:) = 4 : 51; plot(T); xlabel('plotting a matrix');` From such a plot you can learn the sequence of colors automatically assigned to the first, second and so on column to be depicted as a graph.

4. Generate an arithmetic progression consisting of $12 + 1$ equidistant points on the interval $[0, 2\pi]$ by resizing the simple arithmetic progression $0 : 1/12 : 1$ by the factor $2 * \pi$. Recalling then *Euler's formula*

$$e^{ix} = \cos(x) + i * \sin(x)$$

or simple using the fact that the coordinates of a point on the unit circle are expressed as a pair $(\cos(x), \sin(x))$ for some $x \in [0, 2\pi)$ we can obtain the set of unit roots of order 12 by the simple command `r12 = exp(i * bas)`. To check that we have obtained the right object we can plot it in the complex plane, simply by the command `plot(r12); axis square;` You may want to mark the corners specifically with the additional command `hold on; plot(r12,'r*'); hold off; figure(gcf);` .

I can/will also make the LATEX code for THIS document and diaries available to you!

How can you realize such a task. Take the unit roots of order 12 above as example. The MATLAB input could be

```
>> format compact
>> diary r12plot.m
>> r = 0 : 1/12 : 1
r =
    0    0.0833    0.1667    0.2500    0.3333    0.4167    0.5000    0.5833
                                0.6667    0.7500    0.8333    0.9167    1.0000

>> bas = 2*pi*r;
>> r12 = exp(i*bas);
>> plot(r12); hold on; plot(r12,'r*'); hold off; title('unit roots of order 12');
>> grid; axis square; figure(gcf);
>> diary off;
```

Then you would edit this file, to allow such a plot for general n , not just $n = 12$. Hence we continue, by calling now the command

```
>> edit r12plot
```

in order to obtain this file:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PLUROOTS.M
%
% Plot unit-roots
%
% USAGE: rts = pluroots(n);

% this is only a comment which will NOT be displayed upon calling
% 'help pluroots'. Of course the file has to be in your working directory
% in order to be accessible
% 'which pluroots' displays that directory
% 'pwd' prints the current working directory

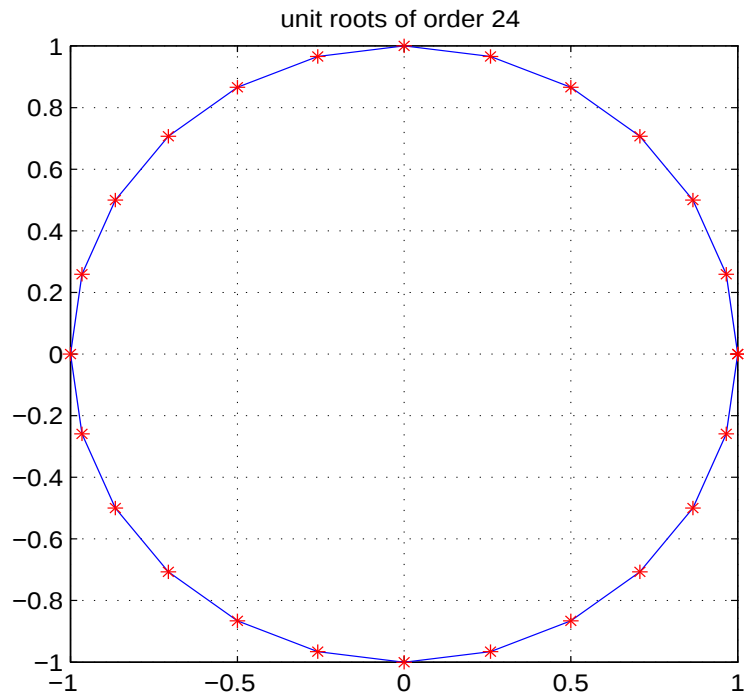
function r12 = pluroots(n);

if nargin == 0; n = 12; end;
r = 0: 1/n :1;
bas = 2*pi*r;
r12 = exp(i*bas);
plot(r12); hold on; plot(r12,'r*'); hold off;
title(['unit roots of order ' num2str(n) ' ']);
grid; axis square; figure(gcf);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

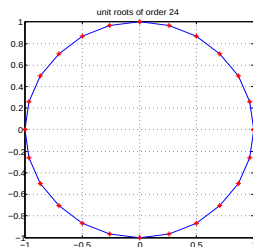
This file can now called as `pluroots(24)` in order to see a similar plot of the unit roots of order 24. You can save and store the figure (by clicking on the icon “file” on the left top the display, then go into export and

choose the mode, such as *.eps or *.jpg or *.pdf format. I usually export into EPS and run a converter called “epstopdf” on the resulting EPS-file, which takes away the unnecessary spaces.

Now we include the thus converted figure into the report:



If you do not go via EPS you get a lot of empty space around, but probably you can fix that using some commands at the time of PDF output from MATLAB. Concrete dimensions obtained experimentally!



Worksheet 2, for the MATLAB course by Hans G. Feichtinger, Edinburgh, Jan. 16th

1. Write a routine (an M-file which you may want to call `testmvmul.m`) which checks that the effect of matrix multiplication of some $m \times n$ -matrix A with some column vector x is just a linear combination of the columns of A (let us describe them mathematically as a^k , as a symbol for the k -th column, i.e. check that $A * x = \sum_{k=1}^n x_k a^k$ (recall that you get it via $A(:,k)$).
2. Write a routine defining a “random complex matrix” with entries (both real and imaginary part) equidistributed in the interval $[-1/2, 1/2]$, and call it `RANDC.M`. Input should be the same as that of `RAND` (call “help rand”), i.e. either `rand(n)` or `rand(m,n)`.
3. Take an arbitrary rectangular matrix and verify that for any pair of complex-valued vectors $x \in C^m$ and $y \in C^n$ one has:

$$\langle A * x, y \rangle = \langle x, A' * y \rangle.$$

Memorize this by thinking: moving the matrix to the other side of the scalar product one has to replace the matrix A by A' , i.e. one has to add the conjugation and transposition to the matrix (in this way one thinks more symmetrically of this fact).

4. Given a general vector $x \in C^n$ (or for simplicity just in R^n) and another vector $y \in R^n$, describe (experimentally) that the orthogonal projection of y onto the one-dimensional subspace generated by x is given by

$$P_x(y) := \frac{\langle y, x \rangle}{\langle x, x \rangle} x = \langle y, \frac{x}{\|x\|} \rangle \frac{x}{\|x\|} = \langle y, x_1 \rangle x_1,$$

were $x_1 = x/\|x\|$ is just the *normalized* version of the vector x . Hence the orthogonal projection can be written as $y \mapsto y * x_1' * x$, or in other words, the $n \times n$ -matrix `x1 * x1'` (or equivalently `x*x'/x'*x` (!!!)) describes the orthogonal projection (as a mapping from R^n into R^n).

5. We know that this is giving us the best approximation of y by scalar multiples of x and that the reminder, i.e. $y - P_x(y)$ should be orthogonal to y (hence to all the other vectors in the 1D-space of scalar multiples of x). We can test that also experimentally. For simplicity I suggest to work directly with x_1 , i.e. to do the computation with normalized vectors. Recall the Cauchy-Schwartz inequality, which says that the scalar product is never larger than the product of norms:

$$|\langle x, y \rangle| \leq \|x\| \cdot \|y\|.$$

6. Take two vectors in $u, v \in R^7$ of norm 1. Find out for which value $\lambda \in [-1, 1]$ the distance between v and λu is minimal (it should be for $\lambda = \langle v, u \rangle$).

```
>> LAM = -1 : 0.001 : 1; % possible range of values, in steps of h = 0.01;
>> for jj = 1 : length(LAM); dd(jj) = norm( v - u*LAM(jj)); end;
>> plot(LAM,dd);
>> [mindd, ndmin] = min(dd) %determining minimal value and index of minimal value
>> norm(v- u*u'*v)
>> norm(v- u*u'*v) - mindd
>> hold on; plot(LAM(ndmin),mindd,'r*'); hold off;
>> grid; title('distance of v to lam * v ');
>> gtext(['minimal distance at ' num2str(LAM(ndmin))]); % drop near red cross!
```

7. As time permits: check (by some random experiment) that for a random orthogonal matrix (e.g. $U = \text{orth}(\text{rand}(6,4))$) the projection onto the linear span by the 4 column vectors (which in fact equals the 6×6 -matrix $U * U$) is the same as the sum of the rank one projections on each of the columns of U .
8. Given some collection of vectors in C^m , say 4 column vectors in C^6 , or equivalently a complex 6×4 -matrix (call it V). Form another collection of 7 vectors, which are random linear combinations of those 4 column vectors. You can do this by multiplying V *from the right* with some random matrix U (4×7)

$$H = V * U.$$

Then (most likely) these new 7 vectors (the columns of H) span the same 4-dimensional space in C^6 as the original 4 columns of U . How can you verify this? ONE concrete way would be to calculate the projection onto the linear span of the corresponding vectors? (HINT: two families of vectors of equal length span the **same** linear subspace if and only if their projection operators onto their linear span are equal. Recall that one natural way to obtain the projection onto the column-space of H (for example) is best obtained by applying the Gram-Schmidt procedure ($OH = \text{orth}(H)$) to H and then build $PH = OH * OH'$. Recall that we have

$$PH * PH = OH * (OH' * OH) * OH = OH * Id * OH' = PH \quad \text{due to orthogonality, and} \quad PH = PH'.$$

Even if the orthonormal basis (here OH) is different, the projection is claimed to be the same, and of course the number of members in the orthonormal system is always the same, namely the dimension of the space generated by the ON system.

9. Verify on an arbitrary complex, rectangular matrix that the following two spaces are pairwise orthogonal: the null-space of A' and the range space or column space of A (and by the same reasoning the null-space of A and the column-space of A' , usually referred to as the *row-space* of A are orthogonal complements to each other!

EXTRA comment: Therefore in both cases their dimensions add up to n and m respectively. This explains the well-known dimension formula:

$$\text{defect}(A) + \text{rank}(A) = n$$

which is best seen via Gauss Elimination again: There are $r = \text{rank}(A)$ Pivot elements, and $n - r$ free variables, determining the dimension of the nullspace of A . The MATLAB command `null(A)`; provides an orthonormal basis for that null space = $\{x | A * x = 0\}$ directly.

10. Polynomials: Recall the "CONV" command of MATLAB, corresponding to forming the **Cauchy-product** of two polynomials (described by the sequence of coefficients, starting from the highest, non-zero, exponent), and generate **Pascal's triangle**, or equivalently the sequence of binomial coefficients!
11. In order to verify that the central limit theorem is valid try to following. Start with a general discrete probability measure over the integers. This is just an abstract way of saying: choose any finite sequence of non-negative numbers, adding up to one. This could be the sequence $[1, 1, 1, 1, 1, 1, 0]/6$, describing the probability of obtaining one of the numbers 6, 5, 4, 3, 2, 1 (in decreasing order!, in order to match the conventions of MATLAB about polynomials and powers), and do the following

```
p = rand(1,7); p = p/sum(p); sum(p)
q = p; for jj = 1 : 25 q = conv(q,p); stem(q); pause(2); end;
```

Table of content for Lesson Nr. 2 (Jan. 15th, 2008)

1. MATLAB and Polynomials (quote: `help polyval`), allows to evaluate a polynomial with possibly complex coefficients `a` at a sequence of points on the real line or also the complex plane (if matrices are inserted one should use `polyvalm`).

Recall that any polynomial over the complex can be split in linear factors, i.e. is a product of factors of the form $(z - z_i)$. When the polynomial is symmetric, then those complex numbers are either real or come in pairs of complex conjugate numbers. .

By the command `roots(a)` those linear factors, resp. the zeros (counted with multiplicity) are numerically evaluated resp. presented.

Also the usual convention of writing integers, say 123 for $1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$ is in accordance with this MATLAB rule. Hence one can multiple “arbitrary long integers” using the `CONV` command, if only the out put is recast in standard format (a nice little exercise to do so, assume you get the output [2 34 23 45] you have to start from the end: $45 = 5 + 4 \cdot 10$, but $23 + 4 = 27$ in turn 10 equals $70 + 10 + 200$ etc. etc. ...)

2. MATLAB does not have a specific call for scalar products, because the scalar product of two column vectors x, y is simply calculated by the command `y' * x`, while it is just `u * v'` if you have row vectors $u, v \in C^n$. Correspondingly, for each collection A of n column vectors in C^m the collection of all scalar product (the so-called **Gramian matrix**) is obtained as $A' \times A$ (it contains exactly n^2 entries).

Note that one can define a family of vectors to be an orthonormal system if and only if `A' * A == eye(m)`

3. Recall that a length of a vector can be calculated from its coordinates with respect to *any* orthonormal basis (resp. orthonormal system with the correct number of vectors = $\dim(A)$).
4. TO BE CONTINUED IN A FEW DAYS!
5. LATEX code is also available for this course
6. diary of the course meeting is provided (soon).