Worksheet 5 (week6) MCC3, MATLAB HGFei, Edinburgh, Feb. 13th

1. RECALLl that the mapping from coefficients to the values of polynomial of order N (or degree $N-1$) at a sequence of points $x = [x_1 \dots x_N]$ is given by the Vandermonde matrix $vander(x)$; Keeping the conventions of the MATLAB `polyval`-routine in mind the powers $x^{N-1} \dots, x^1, 1 = x^0$ are coming in decreasing order. Unit roots of order N (starting from ω^0 (instead of ending with ω^N) are easily obtained by `u = exp( 2 * pi * i * (0:1/N:(N-1)/N));` .

2. We will use this to check that the Fourier transform is realizing the mapping from $a = [a_1, \dots a_N]$ to the values of the polynomial $p_a(z) = a_1 z^{N-1} + a_2 z^{N-2} + \dots + a_{N-1} z + a_N$. by checking that the matrix, which realizes **the Fourier transform**, i.e. `FN = fft(eye(N));` **conincides (up to a left-right reshuffle of the columns) with the Vandermonde matrix of the unit roots of order N, in clockwise order!** , by running the command
   `norm(vander(exp(-2*pi*i*(0:1/N:(N-1)/N))) - fliplr(fft(eye(N))))`

   If I don't forget to upload PLOTNUM.M you can do also: `plotnum(exp(-2 * pi * i * (0: 1/N : (N-1)/N)));`

3. One more general fact: Given a family of (column) vectors (arranged in the usual way in a matrix $\mathbf{A}$, the mapping $\mathbf{x} \mapsto \mathbf{A} * \mathbf{x}$ does *linear combitaions*: $\sum_k x_k \mathbf{a}_k$. On other hand the mapping $\mathbf{y} \mapsto \mathbf{A}' * \mathbf{y}$ produces a sequence of scalar products $(\langle \mathbf{a}_k, \mathbf{y} \rangle_k$. If we have an N-dimensional Euclidean space (R^N or C^n with standard scalar product) the ONBs (the orthonormal bases are characterized by the fact that these two operations are inverse to each other:

$$\mathbf{x} = \sum_k \langle \mathbf{x}, \mathbf{a}_k \rangle \mathbf{a}, \quad \text{for all} \quad \mathbf{x} \in C^N.$$

   In other words the mapping from vectors to coefficients with respect to some ONB and the synthesis mapping (doing linear combinations) are inverse to each other. This applies for example to the FFT. Hence the command $IFN = ifft(eye(n))$; shows us the elements from which a vector is synthesized (complex linear combinations of COS and SIN according to Euler's formula, nothing new in a way). Let us put this to work in the next example:

4. Engineers like to expand a real signal into an orthonormal basis consisting of REAL-valued functions. This has led to a *large variety* of transforms building on suitable COS-functions with increasing numbers of oscillations have been devised (whole books exists on the subjects). The Hartley-transform is a relatively recent version of such a system. MATLAB has built in a version of the DCT, the DISCRETE COSINE TRANSFORM. Since `DCT(x)` determines the *coefficients* in such an ONB of COSINE functions one as to apply the inverse transform on the collection of unit-vectors in order to SEE the building blocks!

   TASK: Do the command `D64 = idct(eye(64));` in order to obtain the building blocks of the DCT of size 64. Check that this is really an orthonormal basis, but checking on `check = norm( D64' * D64 - eye(64)),` (it should be "practically zeros".

   Then go on to plot at east the first few functions in this system, e.g. by the command
   `plot(D64(:,1:10));` Add a (black) zero-line by the extra command

```
hold on; plot(zeros(1,64),'k'); hold off; shg;
```

Extra question: what is the level of the constant (blue) line? (and why?, perhaps you try similar things on DCT of size 10 or 100.

NOTE: The DCT of size 64 is in the background of the JPEG compression scheme used for image compression on most of todays digital cameras and in the internet. In fact, one simply splits a arge image into blocks of size 8×8, sorts those 64 pixel values in a specific way (reminiscent of Peano's method to enumerate the rational numbers!, in the order $a_{1,1}, a_{2,1}, a1, 2, a_{1,3}, a_{2,2}, a_{3,1}, a_{4,1}$ etc. and then applying the DCT. Depending on the compression ratio required only a certain number of DCT coefficients is preserved (a 1 : 4 compression is achieved by storing only 16 out of the 64 coefficients. Of course those coefficients which correspond to highly oscillating terms are omitted.

Use this opportunity to store the result of the figure in different ways resp. export the plot (as EPS, or PDF, or TIFF or JPEG image, as time permits). Try to look at the output obtained in those different formats.

5. MATERIAL already earlier put into FeiMATLAB57.pdf: e.g.

6. Just a small variation. Assume only the values of a cubic polynomial $p(x)$ at -1 and 1 are given, and the condition that $p''(0) = 0$. Is that already determining the polynomial. Well, the condition is linear, but obviously three equations will only determine $p(x)$ up to some subspace of cubic polynomials. We establish again the matrix, to be called CT, and similar to BT:

.	$T(x^3)$	$T(x^2)$	$T(x^1)$	$T(x^0) = 1$
$p(-1)$	-1	1	-1	1
$p(1)$	1	1	1	1
$p''(0)$	0	2	0	0

In MATLAB we have:

```
>> CT
    -1     1    -1     1
     1     1     1     1
     0     2     0     0
CT(3,:) = [0,2,0,0];
null(CT);  ans(:)'
ans =      0.7071      0.0000     -0.7071      0.0000
nCT = ans/ans(1)
nCT =         1.0000        0.0000       -1.0000        0.0000
```

shoing that the solution is determined up to multiples of the polynomial $x^3 - x$.

7. Any of the other examples described in FeiMATLAB57.pdf, as time permits.