

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

We just want to WRAP up what has been done so far, try out unfinished work or ask questions that might be relevant to the realization of the project (including questions about how to write the TEX-code!?)

TOPICS for (re)consideration:

A matrix can be diagonalized (by changing the basis from the given or the standard one to another, suitably chosen orthonormal basis, such that with respect to this matrix the linear mapping has diagonal form $A = U' * D * U$, or equivalently $A * U = D * U$, with U being a matrix whose columns (or equivalently its rows) form an orthonormal system. In fact, each of those column vectors turns out to be (necessarily) an eigenvector, and the corresponding eigenvalue of A is just the diagonal entry with the same index!

A complex matrix can be diagonalized if and only if it is *normal*, which by definition means that $A' * A = A * A$.

Check out that the eigenvalue routine `[V,D] = eig(A);`, when applied to some real symmetric or to some hermitean matrix (with $A == A'$;) really gives an orthonormal basis (with $norm(V'*V - eye(n))$ practically zero) of eigenvectors, i.e. $A * v = d * v$, where v is any of the column vectors, and d is the corresponding number in the main diagonal of D .

check the HELP-file of `EIG`. Also, check that the diagonal elements could be alternatively obtained via the “standard procedure”: Do the characteristic polynomial of the matrix A , i.e. $det(A - \lambda I)$, as a function of λ . It is a polynomial of degree n (for any $n \times n$ matrix A). Its zeros can be numerically determined using MATLAB with the `roots` command! Check that `roots(poly(A));` gives exactly the eigenvalues of the matrix A (which are all real zeros in the case of an hermitean matrix!).

There is a nice identity known in linear algebra: If you insert the matrix (formally) into the polynomial, i.e. replace at each position x^k by A^k , then the result is zeros, in other words, check that

`polyvalm(poly(A), A)` is zero (numerically). Here we use (instead of `polyval` the `polyvalm` routine, which allows this replacement of complex numbers by matrices.

Another little test to understand the 2D Fourier transform would be the check for its building blocks, the so-called **plane waves**

Take $n = 256$, and

`H = zeros(n); H(12,21) = 1; PWH = ifft2(H); imagesc(real(PWH)); axis square; shg` should show a nice “plane wave”. Try to replace (12,21) by some other figures to see how the position of that 1 within H influences the orientation and wave number of the plane wave.

Note that a plane wave is also the same as the pointwise product of two “pure frequencies”. So you could equally take to different columns from the FFT-matrix `F = fft(eye(n))` and then determine a plane wave via `PW = F(:,16) * F(:,5)'; imagesc(real(PW)); axis square; shg;`

If you download some JPG-image from the internet you should be able to load it into MATLAB and “filter it” (just a hint for those who will work on FFT2).

TOPICS for the PRESENTATIONS

NOT YET READY FOR GENERAL PRINT OUT! 10 A.M.

1. **Polynomials from the point of view of linear algebra:** Polynomials of a fixed degree are nice finite dimensional vector spaces. The usual (standard) basis is the set of all monomials, i.e. the functions (x^k) , with $k = 0, 1, \dots, r$, the maximal degree (for MATLAB they should be taken in inverse order). There are a lot of linear mappings on the polynomials, e.g. the shifting-operators, or the dilation operators. Differentiation is a non-invertible operation on any of these spaces. Hence also $(x - x_0)^k$ is a basis for the polynomials of degree r .

I suggest to do most of the demonstrations for cubic polynomials ($r = 3$), but higher degrees might be suitable in some cases!

- Could one take also take $(x - x_k)^k$, for $0 \leq k \leq r$?
- for any four points determine a (linear independent) set of Lagrange interpolators, which form another set of linear independent cubic polynomials, hence another basis for the cubic polynomials;
- the *Hermite interpolation problem*, given sufficiently many function values and derivatives find the exact solution among the polynomials which satisfy the corresponding conditions (for example, giving the first three derivatives at four different points should determine a polynomial of degree 11, as it has 12 coefficients); but how can one obtain the polynomial from the data?
- can one find a quadratic polynomial by determining the value of the derivative at two values? if the answer is no, under which conditions can it be done (and when not)?

2. **Spline functions** are typically defined to be piecewise polynomial functions with “knots” at the integers.

Since polynomials of higher order are not such a good idea for the purpose of (uniformly) approximating functions, even if they are smooth (cf. Runge’s example) the so-called spline-functions have been invented. The most important case is the case that of cubic splines, which are in fact functions which have a continuous first and second derivative (but possible jumps in their third derivative at the node points, let us consider only integer nodes). Hence over the interval $[0, 3]$ the space of splines consists a priori of the space of cubic polynomials over each of the subintervals $[0, 1]$, $[1, 2]$ and $[2, 3]$, but with the ‘gluing conditions’ that the left hand limit (of the function and its first and second derivative) is matching the right hand limit of the corresponding derivative on the right hand side. This defines a 12-dimensional space (since we have 4 coefficients over each of the 3 intervals), but within that we have $2 \times 3 = 6$ coupling conditions, so overall one may expect $12 - 6 = 6$ dimensions of the spline functions over $[0, 3]$. So we ask questions like this one:

- Given this space, how can we find a basis for this space, or in other words, what is the dimension of the space? Can we find for a typical choice of data, e.g. by requiring 6 specified values, 2 per interval at arbitrary chosen points, to be exactly interpolated by some element of the spline type space?

- If we choose in each interval the basis such that the ‘Hermite interpolation conditions are satisfied’ one can in fact show that the coupling conditions mean that one can remove some of the variables (it is like taking the plane in R^3 described by the condition $y = z$, which obviously has a basis of the form $[1, 0, 0]$ and $[0, 1, 1]$).
- An important property of the spline functions over the integers is that one can find a so-called basis of B-splines, which is one function (a standard cubic polynomial) which is a cubic spline, supported on $[-2, 2]$ and symmetric around 0.

3. The **FFT, the Fast Fourier Transform**, i.e. the realization of the DFT via some fast algorithm, is one of the most interesting and most important algorithms for signal processing applications.

It can be (first of all) interpreted as an orthonormal change of bases, to a basis which consists of functions which are eigenvectors of the cyclic shift matrix.

At the same time it describes the (linear) mapping from the coefficients in C^n of a complex polynomial to its values over the unit-roots of order n . From this property we can gain a number of consequences:

- (a) The so-called convolution theorem: doing either the Cauchy-product of two polynomials or the pointwise product of their Fourier transform. Of course one has to consider the degree of the product polynomial, because in order to properly gain all the coefficients correct of the product polynomial one needs at least $r + 1$ values in the complex plane, so at least some FFT of length $r + 1$. One does so by adding zeros to the product polynomial.
- (b) The same applies to powers of polynomials, which can be realized not by an inductive FOR-loop using CONV but instead much faster by taking a pointwise power on the Fourier transform side. For example, in order to determine the coefficients of $(1 + x)^{53}$ one observes that one needs at least 54 coefficients, hence one should extend the sequence $[1, 1]$ describing $1 \cdots x + 1$ by 62 more zeros, in order to obtain a sequence of length $64 = 2^6$, and work with that sequence;
- (c) The interpretation of polynomials allows to also demonstrate the interpolation property that one can get by zero-padding on the IFT-side. Given for example 8 data points, interpreted as values of some polynomial of order 7 over the unit-roots of order 8 we can interpolate and obtain the same polynomial over the unit roots of order 32
- (d) x

4. Linear Algebra from a geometric/numerical point of view

Given a matrix, how can one determine the relevant sub-objects:

- a basis for the row or the column space (using Gauss elimination)
- obtain orthonormal bases by applying
- built-in MATLAB commands (such as ORTH)
- do it using the PINV-command as described in the course

- show that the result are equal, either for a particular random matrix (which should be allowed to be *not of full rank*, i.e. with $\text{rank}(A) < \min(m, n)$. (recall that $\text{rand}(8, 3) * \text{rand}(3, 5)$ is likely to give some random 8×5 matrix of rank 3!

5. **Orthogonalization methods:** Given a basis of a vector space, how can one obtain an orthogonal basis for the same space in a way which is maximally fair. Standard *Gram-Schmidt* is *not* fair, in the sense that it gives the first vector the chance to stay unchanged, while the last one has to adapt maximally to all the other ones. Hence the result of Gram-Schmidt (and also the stability of the process) depends heavily on the order in which the basis vectors are chosen. The symmetric or Loewdin orthogonalization is the most important one (NuHAG: Orth-Best command). It can be best explained using the *SVD*, the *singular value decomposition* of a general matrix. One can say, that one “takes out the rescaling part” and leaves just a change of basis of vectors, from the vectors originally taken making up the column space, to the orthonormal system U which occurs in the factorization $A = U * S * V'$ (obtained within MATLAB by calling `[U,S,V] = svd(A)`; Then the best approximation to A (in the sense of the so-called Frobenius norm, `norm(A,'fro')`), i.e. considering A as an element of C^{n^2} (with the Euclidean norm) we get the best approximation by a unitary matrix by choosind $U * V'$.

There is an interesting connection between the so-called biorthogonal system and the (rows) of the inverse of a matrix. Write $B = \text{inv}(A)$; then $B * A = Id$ (the unit-matrix), resp. `norm(B*A - eye(n))`; Hence (at least for real matrices) the row-vector number 3 has scalar product 1 with the third column of A and zero scalar product with respect to all the other column vectors. In this sense the rows of B are biorthogonal to the column vectors of A .

Such a biorthogonality condition also occurs if we look at the LAST vector in the Gram Schmidt system. So let us take the vectors of A in some random order, just making sure that the third (to give an example) vector is put in last position. Apply then the Gram-Schmidt procedure. Then obviously the last vector will be just the third column of A stripped of all its components in the direction of all the other ones. This means, that we can obtain the rows of the inverse matrix (in fact their normalized versions only) by applying Gram-Schmidt repeatedly to the matrix, for example by the command, applied to some random 6×6 -matrix A :

```
for jj = 0:5;AA = A(:, 1 + mod( -jj+(0:5),6)), pause, end.
```

6. **Image Compression Algorithms** (lossy versus lossless compression)

The goal of this project is to give a short summary over various compression schemes, applicable to digital images. Many of these algorithms are based on the following idea: Take a suitable transform (such as an orthonormal change of basis) and then determine in some way the “most important coefficients” that have to be stored.

It would be relatively stupid and simple-minded to reduce the storage space for a large image by a factor of 9 by just taking every third pixel in each direction. The result would depend very much on the offset. and not be very representative for the full image.

I hope that there will be a way to download/obtain the basic routines for image processing in MATLAB without having the license for the image analysis (and signal analysis) toolbox. See `dct2`, `fft2`, `idct2`, `ifft2`.

7. The Spectrogram or Short-time Fourier Transform

It is available as a built-in function of MATLAB (please test)

`spectrogram`, `periodogram`, `pwelch`, `spectrum`, `goertzel` but these routines might be part of the signal processing toolbox, so I recommend to use the NuHAG routines instead, to be found/downloaded from www.nuhag.eu (go to DB+tools > MATLAB).

There will be much more material on this for those who get this part assigned.

8. Diagonalization of matrices and eigenvectors of matrices

Recall the definition of eigenvalues and eigenvectors (from your favorite book in linear algebra), and try to see how the program described in that book works out in order to obtain first the eigenvalues and then the eigenvectors.

We start by building the characteristic polynomial, which is described by the MATLAB coefficients `poly(A)`. Recall that this characteristic polynomial has roots (you get all of them by the command `roots(poly(A))`) and check that these roots are indeed eigenvalues, because for any λ which is among the roots of $p_A(z)$

There is also a nice numerical methods whose action can be demonstrated easily (without going into the proof, by the following simple experiment:

```
for kk = 1 : 100; [Q,R] = gramsfei(A); A= R*Q; imagesc(abs(A)); shg; end;
```