

Worksheet 4 (week5) MCC3, MATLAB HGFei, Edinburgh, Feb. 6th

EXPANDED, COMMENTED version of Feb. 8th

1. It is well known (from algebra courses) that it is enough to know the coefficients of a polynomial of order n (or degree $n - 1$) from the values of the corresponding polynomial $p(x) = p_{\mathbf{a}}(x) = a_1x^{n-1} + \dots + a_{n-1}x + a_n$ (following the MATLAB conventions). We can check this in two steps:
a) verify that (for some random polynomial of degree 5, say, and some random numbers, both of them ideally complex, but a “real-valued” experiment should make you confident as well) the Vandermonde matrix `vand(a)` is providing exactly such a matrix, and
b) verify that `V=vand(A)` is in fact invertible (by looking at `rref(V)` or determining `det(V)`). Recall that the VANDERMONDE matrix is the linear matrix which describes the mapping from the space of
2. It is clear that a linear system $\mathbf{A} * \mathbf{x} = \mathbf{b}$ has a unique solution for every $\mathbf{b}$ if and only if $\mathbf{A}$ is invertible. But not all invertible matrices are “equal” (think geometrically: acute angles produce imprecise information about the intersection point). In order to check this choose 4 points in the interval $[-1, 1]$ (not too close to each other) and build the corresponding Vandermonde matrix `V4`. The corresponding condition number `cond(V4)` should not be too big. Then add one more point (try to put it relatively close to one of the existing points or not so close, like halfway between the existing points), and check the influence on the corresponding 5–point Vandermonde matrix `V5`. Clearly enough it will be large (in fact `norm(inv(V5))` will be large, if the minimal distance is small, meaning that in this case small changes of the values of the polynomial may mean large changes in the coefficients!)
3. The unit roots of order n occur as entries in the FFT-matrix, which we can obtain via the call `F = fft(eye(n))`. Convince yourself that this is the case (by plotting the first few columns of `F`). Check that `F == F.'` (for some small n , e.g. $n = 5$), i.e. that the matrix `F` is invariant under ordinary transposition (consequently `F' == conj(F)!`). The command
`plot(F(:,n),'k*'); axis square; grid; shg;`
might be convincing (here $n = 12$ or $n = 36$ would look interesting) (for `F(:,2)` you get the same collection of points, but in a different order). Perhaps (if time permits) you also try
`for jj = 1 ; plot(F12(jj,:), 'k*'); axis square; shg; pause(1); end; CORRECTION:`
`n = 12; F12 = fft(eye(12)); for jj = 1:n; plot(F12(:,jj), 'k*'); axis square; shg; pause(1); end;`
4. Convince yourself (by plotting real and imaginary part of the columns) that they represent the “pure frequencies”. Now for say $n = 256$, `F = fft(eye(n));, plot(F(:,1:8));` should be revealing. Also try the last few columns, e.g. `plot(F(:,n-6 : n));`. What makes the difference between the second and the last column?
`>> figure; FL = fft(eye(256)); subplot(211); plot(real(FL(:,1:7))); axis tight;`
`>> subplot(212); plot(imag(FL(:,1:7))); axis tight; figure(gcf);`
5. Use this opportunity (figure) to learn how to store a figure (go to “file”, generate M-file etc.) or also to export it, e.g. as a JPG- or EPS or PS-picture (or try any other format offered, and

perhaps later on check the “quality” of what you have stored). Typically one would use/produce an EPS or PDF file to be included into a report (such as the course material, which is produced in this way).

ONE OPTION would be: `>>saveas(gcf, 'filename', 'format')` if you prefer command line actions over “clicking with the mouse”. Here format can be 'eps', 'fig' etc., for example I just did for myself for convenience the following M-file:

```
function saveps(name); saveas(gcf,name,'eps'); eval(['dir ' name '*']),
```

Note that here the blank after "dir" is important, that it really compiles a string that is then evaluated as command (within MATLAB), simply checking that the saving command was correct.

6. Check that F is an orthogonal system, with all vectors being of equal length $\sqrt{n}$. Hence $\text{inv}(F) == F'/n$; (verify this yourself). In order to understand the connection between our complex matrix and the classical view-point (where only $\cos(kx)$ and $\sin(kx)$ -functions are used, let us check the following (claim!): For any given $K \geq 0$ the linear span of all these functions (with $0 \leq k \leq K$) equals the linear span of the union of the first $K + 1$ together with the last K elements of the Fourier matrix (we may use FL for this test).

ANSWER: one possible realization is:

```
>> F = fft(eye(n)); C(:,1) = F(:,1);
>> C(:,2:n/2+1) = ( F(:,2:n/2+1) + conj(F(:,2:n/2+1)))/2;
% or for odd n one should in fact use:
% C(:,2:floor(n/2)+1) = ( F(:,2:floor(n/2)+1) + conj(F(:,2:floor(n/2)+1)))/2;
% (just a check for odd 'n')
>> C(:,n/2+2:n) = ( F(:,2:n/2) - conj(F(:,2:n/2)))/(2*i);
>> cond(C)
ans = 1.4142
```

In fact, for any pair of columns representing the same “frequency” one does two linear combinations, following EULER’s formula

$$\exp(ix) = \cos(x) + i * \sin(x), x \in R,$$

keeping in mind that the first column (constant 1) is by itself (and so is the one with column number $1 + n/2$, for the case of even n , it is just the highest oscillating frequency, with just alternating ± 1), while the second and the last, etc. have to be paired, up to the highest frequency. Since obviously we can get both $\exp(ix)$ and $\exp(-ix)$ back from $\cos$ and $\sin$ the linear span of pairs of columns, with either $\exp(ikx)$ and $\exp(-ikx)$ (more or less) resp. $\cos(kx)$ together with $\sin(kx)$ will be the same. Engineers like the $\cos/\sin$ -representations better, because they are REAL-valued (as opposed to the complex-valued Fourier terms, but they are slightly differently normalized, i.e. the FIRST column has a different norm now, just try `for jj=1:n; nc(jj)=norm(C(:,jj)); end; plot(nc); shg;`

7. If we want to see how the best approximation of a give “signal” (or vector of length n) by partial sums “looks like” we just of the project the signal onto a family of *pure frequencies*, we just choose

a subset of the index set $1 \dots n$. It should be symmetric to the zero-frequency, which occurs in column number 1 of F , hence we should choose the index set $ndk = [1 : k + 1, n - k + 1 : n]$ if we want to include precisely all the

8. Work out the matrix AT describing the mapping from the polynomials of order 4 (i.e. cubic) with the monomial basis in MATLAB order, i.e. $\{x^3, x^2, x, 1\}$ to the vector $T(p(x)) := [p(0), p(1), p'(0), p'(1)]$. The inverse of this matrix gives the coefficients of those 4 cubic polynomials (uniquely determined) which satisfy $T(p_k(x)) = e_k$, the k -th unit vector (their coefficients are described by $inv(AT)$, for $k = 1, 2, 3, 4$..

ANSWER: We are looking at the matrix AT which describes the mapping from R^4 to R^4 , from the coefficients of a matrix to the column! vector $T(c) = [p(0), p(1), p'(0), p'(1)]'$, where $p(x) = c_1x^3 + c_2x^2 + c_3x + c_4$ is the polynomial, that MATLAB outputs when you are doing `polyval(c, x)`. This matrix has full rank (call: `rank(AT)`), hence it is invertible (or check for `det(AT)`).

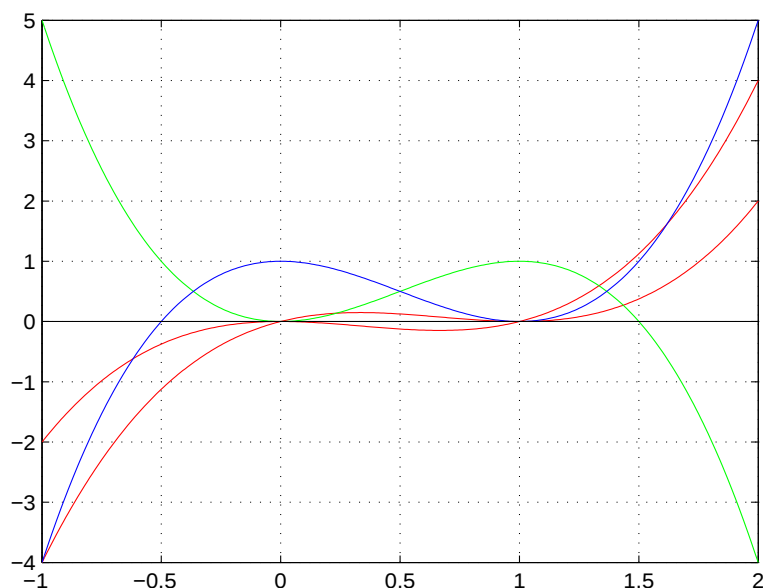
.	$T(x^3)$	$T(x^2)$	$T(x^1)$	$T(x^0) = 1$
$p(0)$	0	0	0	1
$p(1)$	1	1	1	1
$p'(0)$	0	0	1	0
$p'(1)$	3	2	1	0

Calculating the inverse of this matrix in MATLAB gives in fact:

```
>> ATI = inv(AT)
ATI =
    2.0000    -2.0000     1.0000     1.0000
   -3.0000     3.0000    -2.0000    -1.0000
         0         0     1.0000         0
    1.0000         0         0         0
```

Applying ATI to any vector $d \in R^4$, e.g. `d = 2 : 5` means to look for the coefficients c of polynomial with $T(c) = [2, 3, 4, 5]'$ resp. finding the polynomial with $p(0) = 2, p(1) = 3, p'(0) = 4, p'(1) = 5$, in MATLAB format. Hence `c = ATI * d` gives the coefficients, and $p(x)$ can be plotted by the command `bas = -1 : 1/299 : 2; plot(bas, polyval(c, bas));` to see how it looks like! (over $[-1, 2]$).

Choosing as d any of the unit vectors in R^4 of course means, something like `e2 = [0, 1, 0, 0]'`; to apply ATI to e_2 , or equivalently, use the second column of ATI, to make a specific polynomial from it. I have done this for all four unitvectors, and the resulting curves are given below. Obviously the blue curve corresponds to the first unit vector ($p(0) = 1$, the rest zero), while the second one is the green one, etc..



9. Obviously the shift-operator, let us take the shift by two to the right, i.e. $T_2 : p(x) \mapsto p(x - 2)$ (the new, shifted function $T_2(p(x)) = p(x - 2)$ takes the values two units to the left from where it is evaluated, this makes the graph really move 2 unit to the right, despite the disturbing -2 in the argument!), which is linear. You can of course also do T_{-2} which is obviously the inverse mapping (shifting all the cubic polynomials back by 2 units). Write up the corresponding matrices AT_2 and AT_{-2} (first on paper, then put it into MATLAB) and check then that they are indeed inverse to each other.
10. Similar task for the dilation: $D_a(p(x)) = p(x/a)$. If $a = 3$ then the graph appear to be stretched by a factor of 3 (again the "oneover" is a bit disturbing, but important in fact). Build the simple (diagonal) matrix corresponding to the dilation, and watch that again its inverse corresponds of course to $D_{1/a}$ or compression by the same factor (if $a > 1$, or vice versa if $a < 1$).
11. (Just a note). By combination of the previous example you can do any affine transformation $p(x) \mapsto p(ax+b)$, and this mapping is invertible and linear, and has an inverse (not just $p(x/a-b)$, but...??)

NEW MATERIAL:

Connection between the biorthogonal system, the inverse of the matrix (to be more precise: $\text{pinv}(A') = \text{pinv}(A)'$), and the Gram-Schmidt procedure! The last column of the Gram-Schmidt procedure is just the last column of $\text{pinv}(A)$ (up to normalization). In the Gram-Schmidt procedure the vector itself is normalized, i.e. the column vector has norm 1, while in the inverse matrix it is normalized so that its scalar product with the last column of the original matrix equals 1 (as part of the biorthogonality relation).

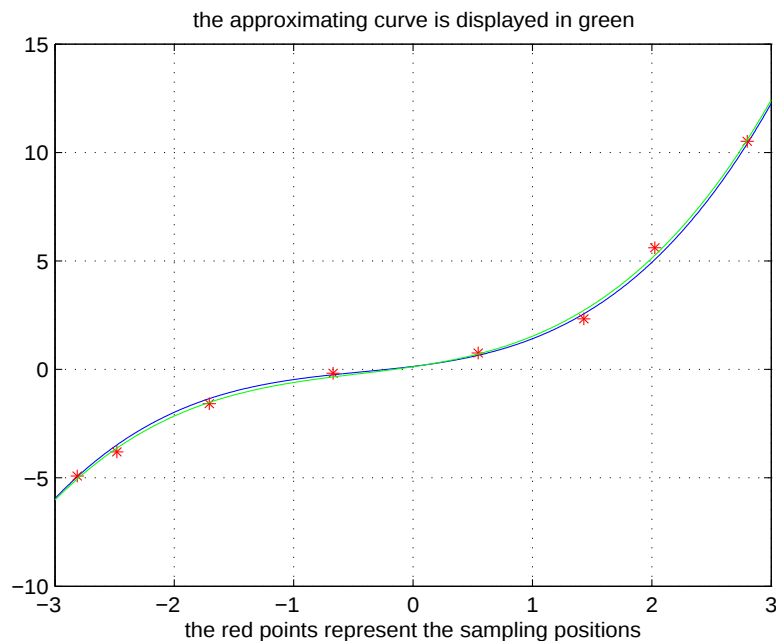
CODE to test this is:

```
A = rand(n); for jj = 1 : n; PA = A; PA(:,[jj,n]) = PA(:,[n,jj]);  
GPA=gramsfai(PA); % Gram Schmidt procedure  
BA(:,jj) = GPA(:,n); end;  
BA' * A % checking for biorthogonality
```

where the final result should be that those non-zero scalar products need some normalization (easy to find out and to compute) in order to be normalized to the value 1 (to get the INVERSE of the matrix).

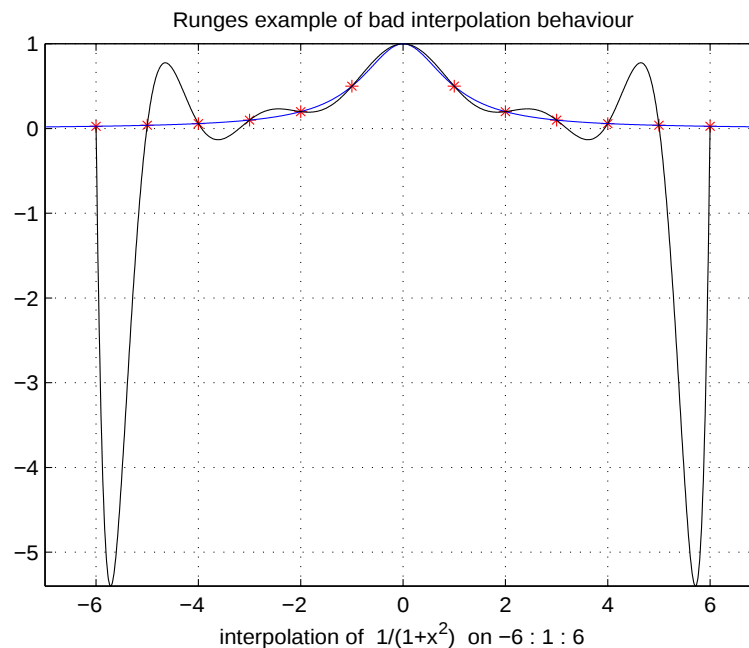
Fitting Polynomials:

```
>> p = rand(4,1); bas = -3 : 1/999 : 3;  
>> pvals = polyval(p,bas); plot(bas,pvals,'k'); grid, shg;  
>> [x,y] = ginput(7); % takes seven point by the mouse  
>> hold on; plot(x,y,'r*'); shg;  
>> pa = polyfit(x,y,3); plot(bas, polyval(pa,bas);  
>> title('showing the approximating polynomial); shg
```



Runge was able to show, that the quite harmless function $x \mapsto \frac{1}{1+x^2}$ does not give convergence, if one interpolates its values by polynomials of higher and higher degree:

```
bas13 = linspace(-6,6,13)
bas13 =
    -6    -5    -4    -3    -2    -1     0     1     2     3     4     5     6
v13 = ones(size(bas13))./(1+bas13.^2);
bas4 = linspace(-6,6,1000);
v4 = ones(size(bas4))./(1+bas4.^2);
plot(bas4,v4,'b'); hold on; plot(bas13,v13,'r*');
V13 = vander(bas13); cof13 = pinv(V13) * v13(:); % PINV to avoid error messages
p413 = polyval(cof13,bas4); plot(bas4,p413,'k');
title('Runges example of bad interpolation behaviour');
xlabel('interpolation of 1/(1+x^2) on -6 : 1 : 6');
```



The more points one takes, the more instable the construction becomes and despite the fact that the polynomials are interpolating at more and more points there is absolutely no control on what is happening “in between the points”.

End for now (10:30 p.m., Jan. 11th) . hgfei

- Another example: Do the list of data $T(p(x)) = [p(-1), p(1), p'(0), p'(2)]$ determine the polynomial in $\mathcal{P}_3(R)$ uniquely. If so, what is the polynomial that satisfies $p(-1) = 1, p(1) = 2; p'(0) = 1; p'(2) = -1$?

Again we have to write down the corresponding matrix (now called) 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	0	1	0
$p'(2)$	8	4	1	0

The result is then

```
>> cc = inv(BT) * [1,2,1,-1]
cc =
    -0.5000
     0.5000
     1.0000
     1.0000
```

- 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
```

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

- SOME useful MATLAB routine:

```
% plPOLS.M    hgfei    13.02.2008
%
% USAGE:      plPOLS(POLS, a, b);
```

```
function      plPOLS(POLS, a, b);
```

```
if nargin == 0; a = 0; b = 1; end;
```

```
bas = linspace(a,b,501);
```

```
[hh, ww] = size(POLS);
```

```
if min[hh,ww] == 1;  POLS = POLS(:);  ww=1; end;
```

```
%allows also input of a SINGLE row vector!
```

```
for kk = 1 : ww;
```

```
    % plot(bas, polyval(POLS(:,kk),bas));
```

```
    POLVALS(:,kk) = polyval(POLS(:,kk),bas);
```

```
end;
```

```
plot(bas, POLVALS);
```

- Making use of it to show (interpret) the content of an inverse Vandermonde matrix: see diary cours1202.m

- % GRAMSFEI.M - Gram-Schmidt orthogonalization.
 % Input : A = matrix
 % Output : Q = matrix with orthonormal columns
 % R = upper triangular matrix
 %
 % Usage : [Q,R] = gramsfei(A);

 % adapted from G.Strang's collection by H.G.Feichtinger
 % April 1994, Storrs hans.feichtinger@univie.ac.at

 function [Q,R] = gramsfei(A)

 [m,n] = size(A);
 Asave = A;
 tol = 1.e-6;
 Q(:,1) = A(:,1)/norm(A(:,1)); % start

 for j = 2:n
 Q(:,j) = A(:,j) - Q*(Q'*A(:,j)); %projection on linear span of previous vectors
 if norm(Q(:,j)) < tol;
 error('Columns are linearly dependent.');