Applications of matrix computations Exercise 5, October 10–14, 2016

Lecturer: Samuli Siltanen.
Teaching assistants: Santeri Kaupinmäki and Jonatan Lehtonen.

Please send your solutions to application.matrixcomputation@gmail.com by Monday, October 17, at 10 AM.

1. Study the file *Convolution.pdf* available at the course website. Here the idea is to use Matlab for computing $p * s \in \mathbb{R}^n$ defined by the formula

$$(p * s)_j = \sum_{\ell=1}^{n} p_\ell s_{j-\ell}, \tag{1}$$

where $s_{j-\ell}$ is defined by periodic extension for the cases $j-\ell < 1$ and $j-\ell > n$. For example, $s_0 = s_n$ and $s_{-1} = s_{n-1}$ and $s_{n+1} = s_1$.

   (a) Give the precise definition of the vectors $s \in \mathbb{R}^{10}$ and $p \in \mathbb{R}^{10}$ used in *Convolution.pdf*. Make sure that the indices in formula (1) match perfectly and give the correct result.

   (b) Construct in Matlab the same matrix A that appears in *Convolution.pdf*. Use the Matlab command `convmtx.m` to do so. (Note that you will need to modify the output of `convmtx.m` to achieve the desired result because the output has too many columns and gets the priodic boundary conditions wrong). Compute $As \in \mathbb{R}^n$ by matrix-vector multiplication.

   When you're working with Matlab, you will want to only use the nonzero part of p, keeping in mind the periodic nature of convolution. Thus, if you have, say, $p = [1, 2, 0, 0, 0, 3, 4, 5]$, then the vector you want to use is $\tilde{p} = [3, 4, 5, 1, 2]$ (the reason for this is that Matlab uses a non-periodic definition for the convolution). Furthermore, when calling `convmtx.m`, you will need to provide $\tilde{p}$ in reverse order as a horizontal vector (reversing a vector can be done using the `flip`-function). Here is an example of what this might look like in Matlab:

```
p = [3,4,5,1,2]; % Make sure that this is horizontal
A = convmtx(flip(p), 10);
```

   Then you will need to modify the output matrix A as described above.

   (c) Write `help conv` in Matlab to find out how the command `conv.m` works. Use it to implement the formula (1), and verify that the result matches the vector As from part b. (Note that using the option `'same'` with `conv.m` plays a crucial role. Also, `conv.m` does not use periodic boundary conditions but just inserts zeros instead. However, in this example it does not matter (why?).)

2. The Fast Fourier Transform can be used for computing convolutions. This is based on a theorem implying the formula $\mathrm{FFT}(p * s) = \mathrm{FFT}(p) .* \mathrm{FFT}(s)$, where ".*" denotes element-wise multiplication of vectors (Matlab notation).

   (a) Create your own test signal $s \in \mathbb{R}^{4096}$ with the following properties. Ensure that $s_j = 0$ for indices $j = 1, 2, \ldots, 1024$ and $j = 3073, 3074, \ldots, 4096$. Furthermore, set most of the elements $s_{1025}, s_{1026}, \ldots, s_{3072}$ to be nonzero. Try to design your signal so that it has interesting features, such as both jumps and smoothly varying parts. Plot your signal.

   (b) Construct two PSF vectors with the following properties. The first vector $p \in \mathbb{R}^{31}$ has strictly positive elements with the largest element being the middle one (p_{16}). The second vector $p' \in \mathbb{R}^{2047}$ also has strictly positive elements with the largest element being the middle one (p'_{1024}). Use `help conv2` to compute the convolutions $(p * s) \in \mathbb{R}^{4096}$ and $(p' * s) \in \mathbb{R}^{4096}$. (Note: feed the vectors in their originl lengths specified above to `help conv2` in the correct order and use the option `'same'`.) Plot the results of both convolutions in the same plot with different colors for comparison. Use the commands `tic` and `toc` to measure the time needed for each of the two computations.

   (c) First practice the use of FFT for convolution by convolving your signal $s \in \mathbb{R}^{4096}$ with the *unit impulse vector* $\delta \in \mathbb{R}^{4096}$. The vector δ has all elements zero except one that has value $\delta_j = 1$. But what is the correct j? We should have

   $$\delta * s = s,$$

   so you can experiment with different locations of the unit element in vector δ so that you get s back exactly from the computation

   $$\mathrm{IFFT}(\mathrm{FFT}(\delta) .* \mathrm{FFT}(s)).$$

   What happens when the unit element is in the wrong location? Plot the real part of $\mathrm{FFT}(\delta)$ for both the corret and incorrect locations for the unit element. What do you see?

   (d) Use FFT for computing $(p * s) \in \mathbb{R}^{4096}$ and $(p' * s) \in \mathbb{R}^{4096}$. But how is that possible since $p \in \mathbb{R}^{31}$ and $p' \in \mathbb{R}^{2047}$? Well, now you have to learn the art of *zero-padding*. You need to define two vectors $\widetilde{p} \in \mathbb{R}^{4096}$ and $\widetilde{p'} \in \mathbb{R}^{4096}$ that are otherwise zero but contain $p \in \mathbb{R}^{31}$ and $p' \in \mathbb{R}^{2047}$ as "subsets." Where to place them among all the zeros? The trick is that the center element, namely the largest one, needs to be located at the place you found in (c) to be the correct location for the unit element. Remember also the periodic boundary conditions! Now compute

   $$\mathrm{IFFT}(\mathrm{FFT}(\widetilde{p}) .* \mathrm{FFT}(s)),$$
   $$\mathrm{IFFT}(\mathrm{FFT}(\widetilde{p'}) .* \mathrm{FFT}(s)),$$

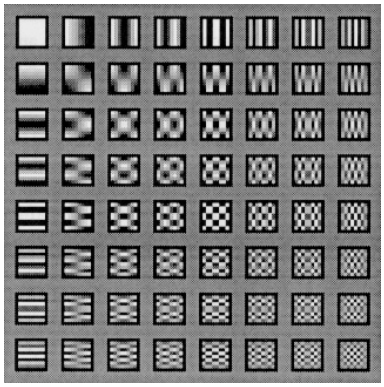
   and make sure you get the same results as in (b). Use the commands `tic` and `toc` to measure the time needed for each of the two computations. How do the times compare to those of (b)?

3. **Your own JPEG compression algorithm.** Here you will implement a simplified version of the discrete cosine transform for two-dimensional pixel images of size 8×8. That is the basis of the jpeg image format.

   (a) First, let's study the building blocks. See the definition of the DCT-II transform. Also, study the Matlab file *DCT_testing.m* available at the course website. Define two-dimensional coordinates using the commands

   ```
   nvec = 0:(N-1);
   mvec = 0:(N-1);
   [nmat,mmat] = meshgrid(nvec,mvec);
   ```

   Then construct 2D building blocks as 8×8 matrices having independently oscillations horizontally and vertically, by changing `cos(pi/N*(nvec+1/2)*kkk)` in the 1D example code to `cos(pi/N*(mmat+1/2)*kkk).*cos(pi/N*(nmat+1/2)*lll)`, where `kkk` and `lll` both go from 0 to 7; thus, you will have a total of 64 blocks. Construct an image showing all the blocks. Your image should look something like this:

   (b) Make sure that your 2D building blocks are orthonormal. Namely, take any two *different* 8×8 matrices `M1,M2` from the set of building blocks and make sure that `sum(sum(M1.*M2))` is close to zero. Also, test that `sum(sum(M.*M))` is approximately one for each building block `M`.

   Now, let's denote the 2D building blocks by $M^{(1)}, M^{(2)}, \ldots, M^{(64)}$. Take any real-valued 8×8 matrix B and represent it as

   $$B \approx \sum_{j=1}^{64} \langle B, M^{(j)} \rangle M^{(j)}.$$

   Here the inner product $\langle B, M^{(j)} \rangle$ is computed practically as `sum(sum(B.*Mj))`. (The notation is not perfect, but I hope you get the point.) Do you get the same 8×8 matrix back from the formula?

   (c) Take a grayscale image of size $8K \times 8K$ with some power-of-two number $K \geq 32$. Loop over the 8×8 sub-images and compute the inner products $\langle B, M^{(j)} \rangle$. Store all the inner products in a vector `IP` of length $64K^2$. Determine a compression threshold value `th` by writing `tmp = sort(abs(PI))` and `th=tmp(round(0.9*length(tmp)))`. Then reconstruct each 8×8 sub-image from its DCT coefficients but so that each coefficient less in absolute value than `th` is set to zero. Compare the resulting compressed $8K \times 8K$ image to the original both visually and in relative square norm error.

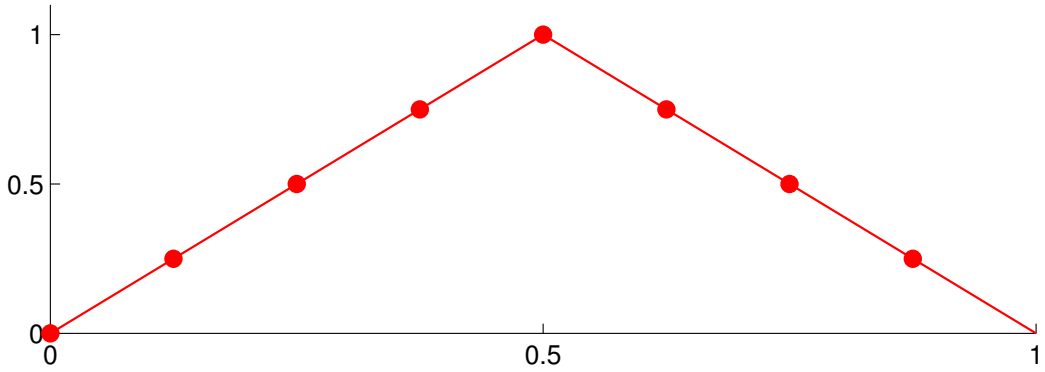
4. **Connection between inner products and discrete Haar transform.** Define a function $f : [0, 1] \to \mathbb{R}$ as follows:

$$f(x) = \begin{cases} 2x & \text{for } 0 \le x < 1/2, \\ 2 - 2x & \text{for } 1/2 \le x \le 1. \end{cases}$$

Also, define a set of 8 evaluation points: $x_\nu = (\nu - 1)/8$ for $\nu = 1, 2, \ldots, 8$. Our discrete signal $\mathbf{f} \in \mathbb{R}^8$ to be analysed is constructed as point values of the function f:

$$\mathbf{f} = [f(x_1), f(x_2) \ldots f(x_8)]^T = [0,\ \frac{1}{4},\ \frac{1}{2},\ \frac{3}{4},\ 1,\ \frac{3}{4},\ \frac{1}{2},\ \frac{1}{4}]^T.$$

Here is a picture of $f(x)$ (solid red line) and $\mathbf{f}$ (red dots):



(a) Compute the following inner products analytically (it is a simple geometric exercise since f is piecewise linear and the other functions are piecewise constant):

$$\langle f, \varphi \rangle, \qquad \langle f, \psi_{10} \rangle, \qquad \langle f, \psi_{23} \rangle,$$

where the inner product $\langle u, v \rangle$ is defined as the integral of $u(t)v(t)$ from 0 to 1.

Recall that the Haar wavelet's mother wavelet is defined through

$$\varphi(t) = \begin{cases} 1, & 0 \le t < \frac{1}{2}, \\ -1, & \frac{1}{2} \le t < 1, \\ 0, & \text{otherwise.} \end{cases}$$

Thus we have $[\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_8)] = [1, 1, 1, 1, -1, -1, -1, -1]$. The Haar wavelet ψ_{jk} is then defined through

$$\psi_{jk} = 2^{j/2} \varphi(2^j t - k).$$

(b) Use the Matlab files given at the course website to compute the wavelet coefficients of the signal $\mathbf{f}$. (Compute them at all the three possible scales, not only one division into low-pass part and details.) Use the inverse transform to check that you get the same signal back.

(c) How do the numbers from (a) and (b) correspond to each other?

5. **Wavelet-based noise removal.** Use the routine `TokyoRow.m` to get a signal $\mathbf{f} \in \mathbb{R}^{256}$ (one row of the Tokyo photo).

   (a) Compute the discrete Haar wavelet transform of $\mathbf{f}$ with depth four. Apply the inverse transform and check that you get the original signal back. Then set the coefficients of three finest detail levels to zero, inverse transform, and call the result $\tilde{\mathbf{f}}$. Plot $\mathbf{f}$ and $\tilde{\mathbf{f}}$ with different colors to the same plot. Can you explain the difference?

   (b) Take the synthetic signal created by routine `signals_comp.m` and saved into file *s1.mat*. Add noise to the signal using the command

   ```
   sn = s + 0.1*randn(size(s));
   ```

   Measure the relative difference between the noisy and clean signal by `norm(sn-s)/norm(s)`. As in (a), put the wavelet coefficients to zero at a couple of the finest scales and inverse transform. How close is the result to the clean signal?

   (c) Take the signal used in (a) and add noise to it as in (b). You may choose the noise amplitude to be something more suitable than 0.1 if you want. Compute the wavelet transform as deeply as possible. Then remove 5%, 10%, 15%, ..., 95% of the smallest wavelet coefficients (order the coefficients from smallest to biggest, replace the desired percentage of the coefficients by zero, and reverse order the coefficients. You might want to use the index vector `I` returned by the command `[Y,I] = sort(X)`.) Inverse transform. Plot the difference between the result and the clean signal as function of the removal percentage. What do you observe?