Homework 1 solutions

1. Exercise 1.1. The figures show the signals $x^{(i)}$ and $v$.
The following Matlab code plots the vectors and computes the four angles:

```
>> [x1,x2,x3,x4,v] = ch1ex1;
>> figure(1)
>> subplot(411);
>> plot(x1);
>> subplot(412);
>> plot(x2);
>> subplot(413);
>> plot(x3);
>> subplot(414);
>> plot(x4);
>> figure(2)
>> plot(v);
>> angle1 = (180/pi)*acos(v'*x1/(norm(v)*norm(x1)));
>> angle2 = (180/pi)*acos(v'*x2/(norm(v)*norm(x2)));
>> angle3 = (180/pi)*acos(v'*x3/(norm(v)*norm(x3)));
>> angle4 = (180/pi)*acos(v'*x4/(norm(v)*norm(x4)));
```

The computed angles (for the signal \(v\) shown above) are

\[
\angle(v,x^{(1)}) = 70^\circ, \quad \angle(v,x^{(2)}) = 11^\circ, \quad \angle(v,x^{(3)}) = 90^\circ, \quad \angle(v,x^{(4)}) = 169^\circ.
\]

We see that the second input signal is most similar to the output signal. This is also clear from the figures. The signal \(v\) contains four periods of a signal that is roughly periodic. The first input is not periodic; it decays with time. The third input is periodic, but with a different period than \(y\). The fourth input is periodic with the same period, but the opposite sign.

Note that the Matlab script \texttt{ch1ex1} generates a slightly different \(v\) every time it is executed, so your \(v\) vector and the four angles may be different.

2. \textit{Exercise 1.6.}

(a) \(y = f(x)\) is a permutation of the elements of \(x\): the elements of \(y\) are the elements of \(x\) in a different order. For example,

\[
\begin{bmatrix}
  x_2 \\
  x_3 \\
  x_1 
\end{bmatrix} =
\begin{bmatrix}
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  1 & 0 & 0 
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 
\end{bmatrix}.
\]

(b) From the definition of permutation matrix, we see that \(A^T\) is also a permutation matrix and that \(A^T A = I\), so

\[g(f(x)) = A^T A x = x\]
for all $x$. The function $g(x) = A^T x$ is the inverse permutation of the permutation $f(x) = Ax$.

In the example,

\[
\begin{bmatrix}
y_3 \\
y_1 \\
y_2
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3
\end{bmatrix}.
\]

If we apply this permutation to $(y_1, y_2, y_3) = (x_2, x_3, x_1)$, we obtain $(x_1, x_2, x_3)$ again.

3. Exercise 1.7. $f$ is linear and we will show this by deriving a matrix $A$ such that $f(x) = Ax$ for all $x$.

We can express $f(x)$ as

\[
f(x) = \frac{1}{\|y\|^2} \begin{bmatrix}
(x_1y_1 + x_2y_2 + \cdots + x_ny_n)y_1 \\
(x_1y_1 + x_2y_2 + \cdots + x_ny_n)y_2 \\
\vdots \\
(x_1y_1 + x_2y_2 + \cdots + x_ny_n)y_n
\end{bmatrix} \\
= \frac{1}{\|y\|^2} \begin{bmatrix}
y_1^2 & y_1y_2 & \cdots & y_1y_n \\
y_2y_1 & y_2^2 & \cdots & y_2y_n \\
\vdots & \vdots & \ddots & \vdots \\
y_ny_1 & y_ny_2 & \cdots & y_n^2
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix},
\]

and this is of the form $f(x) = Ax$ with

\[
A = \frac{1}{\|y\|^2} \begin{bmatrix}
y_1^2 & y_1y_2 & \cdots & y_1y_n \\
y_2y_1 & y_2^2 & \cdots & y_2y_n \\
\vdots & \vdots & \ddots & \vdots \\
y_ny_1 & y_ny_2 & \cdots & y_n^2
\end{bmatrix} = \frac{1}{\|y\|^2} yy^T.
\]

In other words,

\[
x^T \frac{y}{\|y\|^2} y = \frac{1}{\|y\|^2} (y^T x) y = \frac{1}{\|y\|^2} y(y^T x) = \frac{1}{\|y\|^2} (yy^T) x.
\]

In the first step we use the fact that $x^T y = y^T x$. In the second step we write the multiplication of $y$ with the scalar $y^T x$ as a matrix-matrix multiplication of a matrix $y$ (with one column) and a $1 \times 1$ matrix $y^T x$. In the last step, we use the fact that matrix multiplication is associative.

4. Exercise 1.9.

(a) Step 1 can be written as $x = A^T y$. Step 2 can be written as $y = Ax$. 

3
(b) The Matlab code is as follows.

```matlab
>> ch1ex9;
>> x = ones(20,1); % or: x = rand(20,1);
>> y = ones(20,1); % or: y = rand(20,1);
>> for i=1:20
    x = A'*y;
    y = A*x;
    x = x/norm(x);
    y = y/norm(y);
end;
```

We note that if we start from different $x$ and $y$, the algorithm converges to the same final values:

$$x_1 = 0.2214, \quad x_2 = 0.1659, \quad x_3 = 0.1902, \quad \ldots$$

and

$$y_1 = 0.1812, \quad y_2 = 0.1020, \quad y_3 = 0.2620, \quad \ldots$$

5. Exercise 1.10.

(a) Reflecting about the $x_1$-axis is equivalent to changing the sign of the second coefficient:

$$f_{\text{ref}}(x) = \begin{bmatrix} x_1 \\ -x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$ 

The matrix-vector representation of a rotation is given on page 2-22 of the lecture notes (with $\gamma = 30^\circ$):

$$f_{\text{rot}}(x) = \begin{bmatrix} \sqrt{3}/2 & -1/2 \\ 1/2 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$ 

(b) A reflection followed by a rotation gives the function

$$f_{\text{rot}}(f_{\text{ref}}(x)) = \begin{bmatrix} \sqrt{3}/2 & -1/2 \\ 1/2 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 & 1/2 \\ 1/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$ 

(c) A rotation followed by a reflection gives the function

$$f_{\text{ref}}(f_{\text{rot}}(x)) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & -1/2 \\ 1/2 & \sqrt{3}/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 & -1/2 \\ -1/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$ 

6. Exercise 1.11. The $(i,j)$-element $b_{ij}$ in $B = A^k$ is the number of paths of length $k$ between nodes $i$ and $j$.

This is obviously correct for $k = 1$: $b_{ij} = 0$ if there is no branch between nodes $i$ and $j$ and $b_{ij} = 1$ if there is a branch (i.e., a path of length 1).
For $k = 2$, we have $B = A^2$, so the $(i, j)$ element is given by

$$b_{ij} = \sum_{l=1}^{n} a_{il}a_{lj}.$$  

The products in the sum are either one or zero, because the elements $a_{il}$ are one or zero. We have $a_{il}a_{lj} = 1$ if and only if there is a branch between nodes $l$ and $i$ (i.e., $a_{il} = 1$) and a branch between nodes $l$ and $j$ (i.e., $a_{jl} = 1$). In other words, $a_{il}a_{lj} = 1$ if and only if there is a path of length 2 from node $j$ to node $i$ via node $l$. By summing over all nodes $l$ we obtain the total number of paths of length 2 between nodes $i$ and $j$. For $k = 3$, $B = A^3 = AC$ if we define $C = A^2$. Therefore

$$b_{ij} = \sum_{l=1}^{n} a_{il}c_{lj},$$

and we just saw that $c_{lj}$ is the number of paths of length 2 between nodes $l$ and $j$. If $a_{il} = 1$, there is a branch from node $l$ to $i$, so each of these paths of length 2 from $l$ to $j$ can be extended to a path of length 3 from $i$ to $j$. By summing over all $l$ we obtain the number of paths of length 3 between nodes $i$ and $j$. The argument for higher $k$ is similar.

For the graph in the figure, we find

$$A^2 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 & 2 & 2 \\ 1 & 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 & 3 & 2 \\ 0 & 0 & 2 & 0 & 2 & 2 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 0 & 0 & 2 & 0 & 3 & 2 \\ 0 & 0 & 6 & 0 & 7 & 6 \\ 2 & 6 & 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 & 7 & 6 \\ 3 & 7 & 0 & 7 & 0 & 0 \\ 2 & 6 & 0 & 6 & 0 & 0 \end{bmatrix}, \quad \ldots$$

This tells us, for example, that there is no path of length 2 between nodes 1 and 3, that there are 3 paths of length 2 between nodes 2 and 4, that there are 6 paths of length 3 between nodes 2 and 3, etc.

7. Exercise 1.12.

(a) $A \in \mathbb{R}^{25 \times 25}$.

(b) $A = \begin{bmatrix} I_{10 \times 10} & B & 0_{10 \times 10} \\ B^T & 0_{5 \times 5} & 0_{5 \times 10} \\ 0_{10 \times 10} & 0_{10 \times 5} & BB^T \end{bmatrix}$.

8. Exercise 1.13. Multiplying an $m \times p$-matrix with an $p \times n$ matrix requires $2mnp$ flops (lecture 2, page 2-31).

(a) The matrix product $AB$ with $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times 10}$ requires $20n^2$ flops. The result $AB$ is an $n \times 10$ matrix, so multiplying with the $10 \times n$-matrix $C$ costs $20n^2$ flops. The total cost is $40n^2$ flops.
(b) The matrix product $BC$ requires $20n^2$ flops. The result is an $n \times n$ matrix, so multiplying with the $n \times n$-matrix $A$ costs $2n^3$ flops. The total is $2n^3 + 20n^2$.

Method (a) is faster for large $n$.

9. Exercise 1.14. The fastest method is to evaluate $C$ as $C = (Au)(v^TB)$. This requires

- $2n^2$ flops for $Au$ (matrix-vector product of an $n \times n$-matrix with an $n$-vector)
- $2n^2$ flops for $v^TB$ (transpose of a matrix-vector product of an $n \times n$-matrix with an $n$-vector)
- $n^2$ flops for $(Au)(v^TB)$. More generally, if $x$ and $y$ are two $n$-vectors, then the cost of calculating $xy^T$ is $n^2$. The result is an $n \times n$-matrix

$$xy^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_2 & \cdots & x_ny_n \end{bmatrix},$$

and each of the $n^2$ elements requires one multiplication.

The total is $5n^2$. All other methods include $n^3$ terms because we have at least one product of two $n \times n$ matrices.

10. Exercise 1.15. We first derive a flop count for the two methods.

- $y = (I + uv^T)x$. We first compute $uv^T$:

$$uv^T = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_nv_1 & u_nv_2 & \cdots & u_nv_n \end{bmatrix}.$$

This requires $n^2$ flops. Then we add $I$ ($n$ flops because we add one to each diagonal element). Finally we multiply the result with $x$ ($2n^2$ flops). The total is $3n^2 + n$ flops.

- $y = x + (v^Tx)u$. We first compute $v^Tx$ ($2n$ flops). The result is a scalar, so multiplying with $u$ costs $n$. Then we add $x$ ($n$ flops). The total is $4n$ flops.

On a 3.2GHz P4 we get the following CPU times

- For $n = 1000$, $t_1 = 0.04$ seconds, $t_2 = 0$ seconds.
- For $n = 2000$, $t_1 = 0.15$ seconds, $t_2 = 0$ seconds.
This confirms that method 2 is faster. For the first method the execution time increases roughly by a factor of four if we double $n$, which is also what we expect. The second method is so fast that the execution time is negligible for $n = 2000$. If we try the second method for $n = 10^6$, and $n = 10^7$ we get $t_2 = 0.04$ seconds and $t_2 = 0.31$ seconds, i.e., an increase by a factor of about 10, which is consistent with the $4n$ estimate.

Obviously, your results will be different depending on your computer (CPU and memory). Also note that the first method requires much more memory. We calculate an $n \times n$ matrix as an intermediate result, which requires $8n^2$ bytes of memory (8 bytes per floating-point number).