Digital Systems & Linear Algebra

Marius Gavrilescu

1. (a) $A = 0.1, B = 0.1, C = 0.1$

   (b) A pair (λ, v) such that $Av = \lambda v$.

   (c) $$\begin{bmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.2 \\ 0.3 & 0.1 & 0.7 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.3 \end{bmatrix}$$
   $$\begin{bmatrix} 0.6 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.2 \\ 0.3 & 0.1 & 0.7 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0.4 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.41 \\ 0.34 \end{bmatrix}$$

   (d) $$p(\lambda) = \det\left(\begin{bmatrix} 0.6-\lambda & 0.1 & 0.1 \\ 0.1 & 0.8-\lambda & 0.2 \\ 0.3 & 0.1 & 0.7-\lambda \end{bmatrix}\right)$$

   (e) As $p(1) = \begin{vmatrix} -0.4 & 0.1 & 0.1 \\ 0.1 & -0.2 & 0.2 \\ 0.3 & 0.1 & -0.3 \end{vmatrix} = 0$ (since all rows sum to 0), 1 is an eigenvalue.
   The corresponding eigenvector is $[0.2, 0.45, 0.35]$.

   (f) Since this is a Leslie matrix the other eigenvalues will be less than 1 so the distribution vector will converge to the eigenvector of the eigenvalue 1.

2. (a) The rank of a matrix is the dimension of its column space. The row space is the vector space formed by the matrix's rows. The column space is the vector space formed by the matrix's columns. The null space is the set of solutions x to $Ax = 0$ if the matrix is A.

   (b) $$\begin{bmatrix} 1 & 2 & -1 & 4 \\ 0 & 0 & 5 & -3 \\ 0 & 0 & 5 & -3 \end{bmatrix}$$
   Basis for row space is $[1, 2, -1, 4], [0, 0, 5, 3]$. Rank is 2.

   (c) Basis for column space is $[1, 1, -1], [-1, 4, 6]$.

   (d) $Ax = 0$. We get a basis for the null space to be $[-1, -8, 3, 5]$, $[-2, 1, 0, 0]$.

   (e) $x = x_p + x_n$ where x_p is a particular solution and x_n is any element of the null space. We know $x_n = [-a - 2b, -8a + b, 3a, 5a]$ for any $a, b \in \mathbb{R}$, and a particular solution is $x_p = [1, 1, 1, 0]$.
   Thus $x = [1 - a - 2b, 1 - 8a + b, 1 + 3a, 5a]$ for any $a, b \in \mathbb{R}$.

3. (a) $C_{i,j} = \sum A_{i,k}B_{k,j}$ and as $A_{i,j} = B_{i,j} = 0 \forall i < j$, any $C_{i,j}$ where $i < j$ will be a sum of zeroes.

   (b) Suppose there are two different factorizations $L_1U_1 = L_2U_2$ Then $L_2^{-1}L_1 = U_2U_1^{-1}$. Now the LHS is unit lower triangular and the RHS is upper triangular. Therefore they are both the identity matrix. Thus $U_2U_1^{-1} = I \implies U_2 = U_1$ and similarly $L_2 = L_1$, contradicting the assumption. So the factorization is unique.

   (c) $\begin{bmatrix} 1 & 2 & 4 \\ 0 & 2 & 2 \\ 0 & 2 & 5 \end{bmatrix}$

   $\begin{bmatrix} 1 & 2 & 4 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{bmatrix}$

   $A = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 2 & 2 \\ 0 & 0 & 3 \end{bmatrix}$.

   (d) Determinant is 6.

4. (a) Make a three-bit full adder as follows:
   $S_i = XOR(X_i, Y_i, C_{i-1})$, $C_i = maj(X_i, Y_i, C_i) =$
   $OR(AND(X_i, Y_i), AND(XOR(X_i, Y_i), C_i))$ for $i \in \{1, 2, 3\}$ and $C_0 = 0$.
   We will use three filp-flops D_1, D_2, D_3 to store the state. Connect the outputs of the flip-flops to B0, B1, B2. Then we will feed the adder's outputs S_1, S_2, S_3 into the flip flops and set the adder's inputs as follows: $X_i = D_i$, $Y_1 = 1$, $Y_2 = Y_3 = \neg A$.

   (b) A k-input gate will have propagation delay of $k + 1$ ns.
   [delay to be calculated]

5.

6. (a) As register addresses cannot be part of the instructions, a register architecture would be unwieldy and inefficient as register addresses would have to be retrieved from memory. A stack achitecture would work better.

   `add` would pop the top two element off the stack, add them, and push the result onto the stack. The `li` instruction would push its argument onto the stack.

   (b) A `ld` instruction could pop the top element of the stack, access the memory at the location represented by this value, and push the contents of the memory onto the stack. Then a `st` (store) instruction could use the top element as a memory address and the second to top element as a value to store at that address.

   Hence the assignment could be implemented as (assuming x and y are at immediate addresses X and Y):

```
li X
ld
li Y
st
```

   (c) We will add the instructions `dup` (that pops the top element and pushes it back into the stack twice) and `exc` which exchanges the top and second elements of the stack.

   Assume variables a (a pointer) and i (an integer) are stored at addresses A and I.

```
li A
ld      ; load a
li I
ld      ; load i
add     ; obtain &a[i]
dup     ; duplicate &a[i]
ld      ; load a[i]
li 1
add     ; obtain a[i]+1
exc     ; swap top two elements for st
st      ; store a[i]+1 into &a[i]
```

   (d) Due to the commutativity of addition we have two ways of computing e + f: either compute e, then compute f, then add the results or first compute f, then compute e, then add the results.

   Assume WLOG $depth(e) \geq depth(f)$. If $depth(e) > depth(f)$ we can compute e (using $depth(e)$ space), then f (using $1 + depth(f) \leq depth(e)$ space), then add them together. Thus we use $depth(e)$ space. If $depth(e) = depth(f)$ whichever order we compute them in we will use $depth(e) + 1$ space.

   (e) One way was used above: introduce an `exc` instruction that swaps the top two elements of the stack. Another way is to introduce reverse instructions, e.g. `rsub` that treat their arguments in reverse order (e.g. `rsub` will do the same as `exc` followed by `sub`).

   The first solution produces a simpler yet less efficient instruction set (as several operations will require two instructions); the second produces a more complicated but more efficient instruction set.