# Analysis of Steady State Flow in Systems

*by*
*Ryan Mitchell*

*for*
*MECH2700*
*Numerical Analysis I*

*19 Oct 2015.*

# Contents



# List of Figures

# 1 Introduction

Most systems display attributes of both a transient and steady state response. A transient response, also known as the warm up period, is the initial flow that moves through the network and charges / fills storage devices. The network has to be able to contain these surges and influxes without breaking. As the system reaches an equilibrium condition, also known as the steady state, it becomes easier and more predictable to model.

Once the system has reached steady state the system can be analysed analytically in a number of ways. Simple systems can be analysed by hand but as the complexity of the system grows the time requirement for hand calculations increases dramatically. If a steady state system can be described in the form of a set of constraints on all the unknown flow quantities, and the vector of values satisfies all the equations simultaneously, a computer can be used to decrease computational time. If the simultaneous equations can be modelled as a matrix, the processing time is further reduced.

This assignment looks at two network systems and the methods behind solving them:

1. A Chebyshev Filter
2. A Water-Supply Network

A Chebyshev Filter is a high-order filter excellent performance. It is built with passive components and filters electrical signals. The filter contains a system of linear constraint equations but the coefficients as well as the unknown values can be complex numbers. The Water-Supply Network is a network of pipes with the flow of water being described / given. The network contains a system of non-linear constraint equations but the coefficients are real numbers.

This assignment makes use of the Gauss-Jordan matrix elimination method. This method uses row-reduction to modify the values of the lower left-hand corner of the matrix until it is filled with zeros. Once complete the matrix is said to be an upper triangular matrix in row echelon form. Once all the leading non-zero entries on the diagonal are reduced to 1 the matrix is a reduced row echelon matrix. There are three types of elementary row operations:

1. Swapping two rows,
2. Multiplying a row by a non-zero number
3. Adding a multiple of one row to another row

This solver has been designed to handle floats and complex numbers and the code can be seen in Section 2.

## 2 General Gauss-Jordan Code

```
# file: gjSolver.py
"""
Python script for MECH2700, 2015, Semester 2.
Contains a generic Gauss-Jordan matrix solver that reduces a matrix
until a Gauss-Jordan Matrix is achieved. This solver is equipped to
handle floats and complex numbers.
The functions in this program have been abstracted in order to
make use of them in other programs.

R. C. Mitchell
19 Oct 2015
"""

#-----------------------------------------------------------------------
# Import appropriate modules
from math import *
import numpy as np

#-----------------------------------------------------------------------
# Global definitions of the variables for ease of editing


#-----------------------------------------------------------------------
# Start function definition

def GaussJordan(A, b):
    """
    Gauss-Jordan Matrix elimination solver with partial pivoting.

    Params:
        A: square (nxn) matrix of coefficients
        B: column vector of RHS values (nx1 matrix)

    Returns:
        x, which is the solution of Ax = b
    """
    rows, cols = A.shape
    c = np.hstack([A, b])
    z = np.complex(1j)
    # print 'z = ', j, ' & Type(z)', type(j)        # Check

    for j in range(0, rows):
        # Find pivot
        p = j
        for i in range(j + 1, rows):
            if abs(c[i, j]) > abs(c[p, j]): p = i

        # Test for singularity by comparison to a small number
        if abs(c[p, j]) < 1.0e-15:
            raise RuntimeError('Matrix may be singular')

        # Swap rows using the pivot & record the change
        c[p,:], c[j,:] = c[j,:].copy(), c[p,:].copy()
```

```
            # Perform the elimination
            c[j,:] = c[j,:] / c[j, j]
            for i in range(0,rows):
                if i != j:
                    c[i,:] = c[i,:] - c[i, j]*c[j,:]
        I, x = c[:, 0:rows], c[:, -1]

        return x

    def ConditionNumber(A):
        """
        Assuming that the matrix is non-singular, its condition number can be
        calculated as the scalar of
        ||A||.||A^(-1)||
        The actual value depends on which norm is used, but for this function
        the row-sum norm is used.

        Params:
            A: square (nxn) matrix of coefficients

        Returns:
            norm, as a float
        """
        # Test for singularity by comparison to a small number
        if abs(np.linalg.det(A)) < 1.0e-15:
            raise RuntimeError('Matrix may be singular')
        return float(np.linalg.norm(A, 1))

    print 'Ready for use'

    if __name__ == "__main__":

        # Check code is working with 2 smaller matrices
        '''A = np.array([[4.0, -2.0, 1.0],
                         [-3.0, -1.0, 4.0],
                         [1.0, -1.0, 3.0]])
        b = np.array([[15.0, 8.0, 13]]).transpose()
        print 'A = ', A
        print 'B = ', b
        x = GaussJordan(A, b)
        print 'x = ', x
        print 'Check RHS = ', np.dot(A, x)

        A = np.array([[0.0, 2.0, 0.0, 1.0],
                      [2.0, 2.0, 3.0, 2.0],
                      [4.0, -3.0, 0.0, 1.0],
                      [6.0, 1.0, -6.0, -5.0]])
        b = np.array([[0.0, -2.0, -7.0, 6.0]]).transpose()
        print 'A = ', A
        print 'B = ', b
        x = GaussJordan(A, b)
        print 'x = ', x
        print 'Check RHS = ', np.dot(A, x)'''
```

# 3 A Chebyshev Filter For Electrical Signals

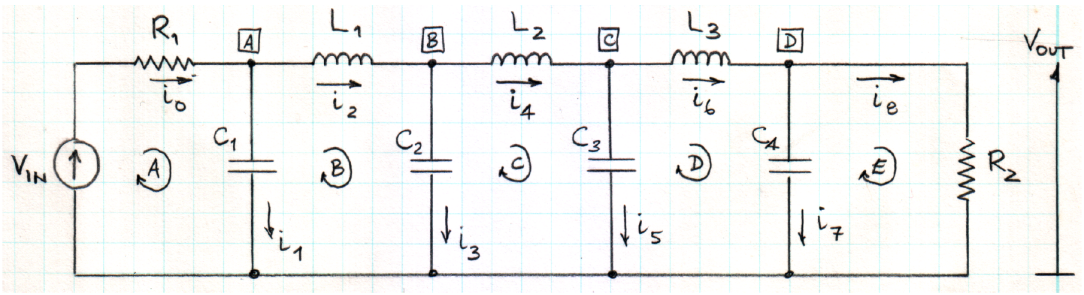



Figure 1: Chebyshev Filter Circuit

To analyse the circuit provided in Figure 1, Mesh and Nodal Analysis techniques are used. Mesh Analysis is derived from Kirchoff's Voltage Law which states that 'the directed sum of the electrical potential differences (voltage) around any closed network is zero'. Nodal Analysis is derived form Kirchoff's Current Law which states that 'at any node (junction) in an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node'. Both tools are required to derive the nine equations in order to solve for the unknowns, $\omega$ and $V_{IN}$, in the system. The five Mesh equations (equations 1 to 5) and four Node equations (equations 6 to 9) were found to be

$$\text{(A)} \quad V_{IN} - Z_{R1}i_0 - Z_{C1}i_1 = 0 \tag{1}$$
$$V_{IN} = Z_{R1}i_0 + Z_{C1}i_1$$
$$\text{(B)} \quad Z_{C1}i_1 - Z_{L1}i_2 - Z_{C2}i_3 = 0 \tag{2}$$
$$\text{(C)} \quad Z_{C2}i_3 - Z_{L2}i_4 - Z_{C3}i_5 = 0 \tag{3}$$
$$\text{(D)} \quad Z_{C3}i_5 - Z_{L3}i_6 - Z_{C4}i_7 = 0 \tag{4}$$
$$\text{(E)} \quad Z_{C4}i_7 - Z_{R2}i_8 = 0 \tag{5}$$

$$\boxed{A} \quad i_0 - i_1 - i_2 = 0 \tag{6}$$
$$\boxed{B} \quad i_2 - i_3 - i_4 = 0 \tag{7}$$
$$\boxed{C} \quad i_4 - i_5 - i_6 = 0 \tag{8}$$
$$\boxed{D} \quad i_6 - i_7 - i_8 = 0 \tag{9}$$

These equations can then be compiled into matrix form in order to make use of software tools. The software processes these matrices much faster and to a much higher level of accuracy than a human. Due to this faster iteration times are applicable e.g. if a small error was made in the initial calculations re-doing the computation is faster or if a passive component needed to be changed the matrix could quickly be updated and the calculations redone. It can also bee seen that the matrices contain mostly zeros and as such can be coded to take advantage of zeros and increase computational power / efficiency, however, this was not made use of for this question. The matrix was first created using the impedance equations 1 to 9,

$$\begin{bmatrix} Z_{R1} & Z_{C1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Z_{C1} & -Z_{L1} & -Z_{C2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Z_{C2} & -Z_{L2} & -Z_{C3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & Z_{C3} & -Z_{L3} & -Z_{C4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_{C4} & -Z_{R2} \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \\ i_7 \\ i_8 \end{bmatrix} = \begin{bmatrix} V_{IN} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

which, when changed to resistive formulae, becomes,

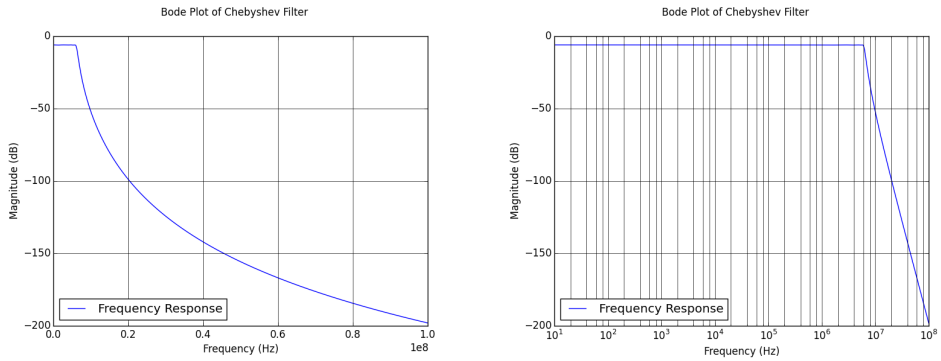
$$\begin{bmatrix} R_1 & \frac{-j}{\omega C_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{-j}{\omega C_1} & -j\omega L_1 & \frac{j}{\omega C_2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-j}{\omega C_2} & -j\omega L_2 & \frac{j}{\omega C_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-j}{\omega C_3} & -j\omega L_3 & \frac{j}{\omega C_4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-j}{\omega C_4} & -R_2 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 \end{bmatrix} * \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \\ i_7 \\ i_8 \end{bmatrix} = \begin{bmatrix} V_{IN} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

As with any matrix operations the first step is to check for singularity. Numpy has a built in check that was used:

```
if np.linalg.det(A) == 0
```

A tolerance was initially trialled but kept failing at higher frequencies as the current became negligible due to the nature of the low-pass filter.

A low-pass filter is a filter that passes signals with a frequency lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The amount of attenuation for each frequency depends on the filter design. A Chebyshev filter is named after Pafnuty Chebyshev because the filter has been designed using Chebyshev polynomials. The cut off frequency is 5 735 210 Hertz (5.735MHz) while the initial gain is -6 dB.




(a) Linear $x$ axis (b) Log $x$ axis

Figure 2: Chebyshev Filter Bode Plots

### 3.1 Conclusion

The Chebyshev filter being used is an extremely good low-pass filter. It has a steady gain of -6dB (flat region of Figure 2b) for a wide range of lower frequencies i.e. 10Hz to 5MHz, and drops off extremely quickly. As can be seen from Figure 2a, the Frequency response of the higher frequencies drops off significantly.

The filter is a passive 7th order filter with both inductors, capacitors and resistors making it acutely difficult to analyse by hand and very time consuming to plot, and therefore visualise the system response. Numpy's linear algebra tools have made relatively short work (approx. 1min) of analysing the matrix, solving it and then running through with $10e^6$ iterations in order to plot the graphs in Figures 2.

# 4 A Water-Supply Network

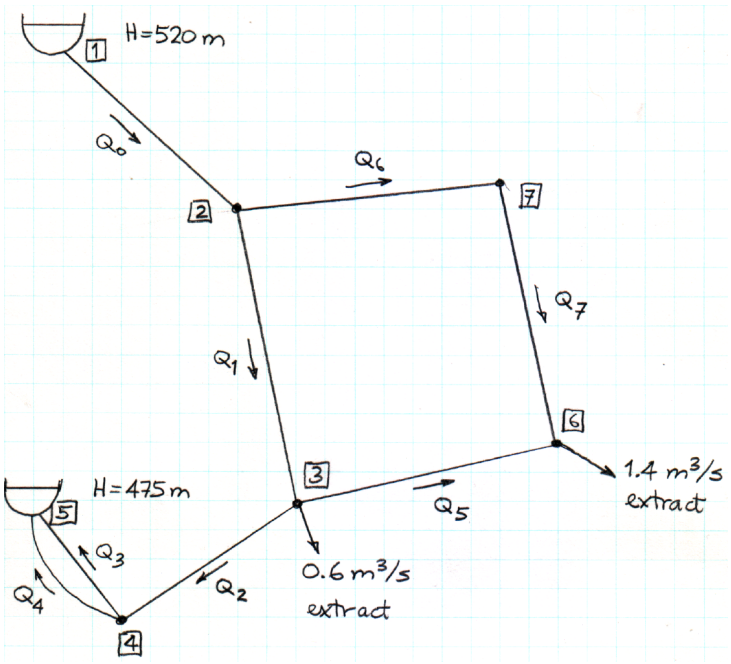



Figure 3: A Water-Supply Network

2 $0 = Q_0 - Q_1 - Q_6$

3 $0 = Q_1 - Q_2 - 0.6m^3.s^{-1} - Q_5$

4 $0 = Q_2 - Q_3 - Q_4$

6 $0 = Q_5 + Q_7 - 1.4m^3.s^{-1}$

7 $0 = Q_6 - Q_7$

In all honesty, I have no idea how to analyse this system - have not taken any fluids course - and did not have enough time to fully understand the initial equations and how to solve it.