Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature: _____

Date: _____

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Moroni Jenks find it satisfactory and recommend that it be accepted.

Dr. Ken W. Bosworth, Ph.D.
Major Advisor

Dr. Anish Sebastian, Ph.D.
Committee Member

Dr. James Mahar, Ph.D.
Graduate Faculty Representative

Identification of Linear Time Invariant Nonminimum Phase Systems

by

Moroni Ardell Jenks

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

Master of Science in Measurement and Control Engineering

Idaho State University

Pocatello, Idaho

Summer 2022

DEDICATION

I would like to dedicate this thesis to my wife Sarah, my daughter Tanya, and my son Evan. They are the light of my life and always give me a reason to work diligently.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Ken W. Bosworth for his guidance in the early stages of writing, and his proofreading in the later stages. His help in grasping the underlying principles related to this thesis enabled me to make it through to the end. I would also like to thank my committee members for their efforts and contributions to this work: Dr. James Mahar and Dr. Anish Sebastian. I would additionally like to thank Dr. Schoen for showing me what it means to care deeply about your students and to help them succeed. Special thanks go to Professor Holfe, Dr. Bosworth, and Dr. Westover for finding funding for this research.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

ABSTRACT

Identification of nonminimum phase noisy systems is a difficult problem to solve using ordinary linear regression because these systems have unstable inverses. This work explores the novel use of open-source software in identifying nonminimum phase systems. The software simplifies empirical transfer function estimates from a high order approximation to a lower order approximation even when output noise is present. A carefully crafted input signal is used to overcome some of the difficulties of these systems. An approach for system order selection based on the mean square error is used to simplify the decision-making process. Some remaining issues and possible extensions are discussed.

Key Words: System Identification, Nonminimum Phase, LTI

CHAPTER 1. OVERVIEW

1.1 Introduction

System identification is the process of creating models based on observational data. Solutions to the system identification problem usually come by using some form of least squares regression. These models often take the form of polynomial approximations. There are many methods available to identify systems but they are often obscured in the form of proprietary software (e.g., the System Identification Toolbox in Matlab®) This makes it difficult to carefully track the details of what happens in these algorithms and make further improvements to those of us outside the organization. In his master's thesis Anene V. Omeje presents a method of system identification through rational approximation using the open source *ratdisk* function [1, 2]. This thesis presents an extension of Omeje's work and addresses several of the issues identified as areas for further research in his work. The purpose of this present work is to identify nonminimum phase systems from noisy measurements using *ratdisk* and other methods.

Nonminimum phase systems are more prevalent than one would initially assume. Some examples include: boats turning left or right initially move in the wrong direction, aircraft trying to gain elevation lose some elevation before moving in the right direction, digestion requires significant energy input before energy can be extracted from the food, and balancing an inverted pendulum on a cart requires the cart to move in the wrong direction to get the pendulum to move in the right direction. Each of these systems can be difficult to identify and control. This work gives some insight into the identification of these types of systems.

What follows is an abbreviated explanation of the theory and definitions needed to understand the research presented in the remaining chapters of this thesis.

1.1.1 Linear Time-Invariant Systems

Linear time-invariant systems (LTI's) are often described using a set of differential equations because the math is powerful, and the descriptions are intuitive. For example, a simple differential equation with a forcing function can be written as $\dddot{y} + a_1 \ddot{y} + a_2 \dot{y} + a_3 y = b_0 \ddot{u} + b_1 \dot{u} + b_2 u$. Solutions to simple differential equations are relatively easy find based mostly on the fact that the exponential function is its own derivative. Higher order systems are more difficult to solve. The differential equation form is included here only to provide a baseline for comparison to the other models shown. Differential equations can also be represented as a system of first order differential equations including input, output, and pass-through. This is known as the state space representation of the system. In the above example, let $\boldsymbol{q}_{3x1} = [y, \dot{y}, \ddot{y}]'$ then in system form the above 3rd order scalar equation becomes:

$$\dot{\boldsymbol{q}} = \boldsymbol{A}\boldsymbol{q} + \boldsymbol{B}u = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{bmatrix} \boldsymbol{q} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

$$y = \boldsymbol{C}\boldsymbol{q} + Du = [b_2 \;\; b_1 \;\; b_0]\boldsymbol{q} + 0 \cdot u$$

Notice that most of the rows of the A matrix are simply definitions, while the last row contains all of the information required to solve the homogeneous part of the differential equation. The B matrix show how each input influences the state of the system, and the C matrix shows the parameters of the non-homogeneous part of the solution. State space descriptions are not unique and can be rearranged in order to make different aspects of

observability or controllability easier to distinguish. State space representations are readily solvable using a variety of different numerical and analytical methods. The layout of the state space description is helpful in identifying what portions of a system influence other portions of the system. All of this is helpful in understanding the details of the system, but there are other descriptions that allow for a simpler more straightforward analysis of the stability of the system and the stability of the system inverse.

1.1.2 LTI's in the Frequency Domain

Describing a system in the frequency domain is a useful tool that will be discussed later in this work. For now, we will explore several types of transformations and a little about their applications. The first and most robust of the transformations is the Laplace Transform. This transformation takes a function from the continuous time domain and moves it into the continuous frequency domain. The full transform includes all of the system information and any initial conditions. The transform is accomplished by taking a very specific integral in this form:

$$\mathscr{L}(f) := F(s) = \int_0^\infty f(t)e^{-st}\,dt$$

where s is a complex number frequency parameter $s = \sigma + i\omega$ with real numbers σ and ω. The symbol σ is generally considered to be the transient part and $i\omega$ is the frequency part of the response. Notice that the lower case f is transformed into the upper case F. This notation is typical for any transformation that moves data or a function from the time domain into the frequency domain. Extensive tables have been produced that allow the transform and inverse transform to be calculated without significant computation. This makes some problems simple to solve by transforming them, combining like terms,

separating terms, and transforming back. There are also several other transforms that are useful.

The Fourier Transform is a special case of the Laplace Transform where the transient part is ignored and the integral is evaluated for all time instead of only positive time:

$$\mathscr{F} := F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt$$

with ω a real number, usually called the "frequency". This transform allows us to evaluate the system based only on its frequency response and ignore any short-term transients. This is one of the most used transforms because it allows for a simplified evaluation in steady-state conditions.

Several types of engineers, including mechanical, electrical and controls, find it useful to represent LTI's in the frequency domain using a ratio of the Fourier Transforms of the system output to the system input. This ratio is known as a transfer function. These linear systems could have multiple inputs and multiple outputs (MIMO), but for the sake of simplicity while extending the state of the art, we will focus on the single input single output, or scalar case. Most often the transfer function is a ratio of polynomials. These transfer functions are essential when designing a controller because they provide a model of the system. They also take the difficult problem of convolution of a given input signal with the system's Impulse Response Function in the time domain and turn it into multiplication of the corresponding transforms in the frequency domain. Combining multiple transfer functions together is also as easy as multiplying them together in the frequency domain. Transfer functions assume zero as the initial condition of all state variables which simplifies things further. Unfortunately, this assumption can cause a loss of some system information if it is not properly handled. The identified models in this work will take the form of discrete time transfer functions.

Using the same coefficients as the examples given above, and using the Laplace Transform, we get this transfer function:

$$H(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^2 + b_1 s + b_2}{s^3 + a_1 s^2 + a_2 s + a_3}$$

Notice that all the coefficients are the same, but we now have a quotient of two polynomials in the s variable. So far, all the models we have described have been continuous time models. These models are great to get an understanding of the system and to build models, but there is another type of model that lends itself better to use in computers due to their discrete nature. The Z-Transform takes discrete time functions, or observations, and transforms them into complex frequency functions. If $x = (\ldots, x(-1), x(0), x(1), x(2), \ldots)$ is a discrete time sequence of values at times $t = (\ldots, -1, 0, 1, 2, \ldots)$ then the Z-Transform is defined by taking the following sum:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

where $z = Ae^{j\phi} = A \cdot (\cos\phi + j \sin\phi)$, A is the magnitude of z, j is the imaginary unit and ϕ is the complex argument. The output of the Z-transform is a continuous function in the z variable, similar to the output of the Laplace transform. This function is what we will be approximating using the discrete Fourier Transform.

The discrete Fourier Transform, or DFT is a method of converting one set of discrete points (i.e., observations at equally spaced time instants) into another set following the equation:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n / N} \quad for\ 0 \le k \le N-1$$

where X_k represents each individual transformed value, N is the number of data points, and x_n represents each data point before the transformation. Notice that the lower-case x is transformed into the capital X through the DFT process. The DFT transforms points described by the discrete time function into points described by the Z-transform of the system. The Z-transform gives the frequency content of a signal at all frequencies, while the DFT gives values of that function only at a finite set of frequencies. The most popular method to calculate the DFT relies on a symmetry that arises from using a data set that has a number of data points equal to a power of two. This method is much faster and more accurate and is called the fast Fourier Transform, or FFT.

Frequency domain functions can be approximately converted between discrete time and continuous time using a bilinear transformation, or Tustin's method. This transformation locally preserves angles between sets of points but does not necessarily preserve distance. This means that all the properties and relationships found in one type of system (stability, minimum phase, relative location of poles and zeros) will also be present in the other type, but possibly at slightly different frequencies. Converting between discrete and continuous systems can also be done using a zero-order hold with uniform sampling time: this is where an observation is treated as a single point with no attempt at conserving any sort of smooth transition. When going from continuous to discrete a set of points is produced. When going from discrete to continuous a series of steps is produced.

1.1.3 Least Squares Estimation

There are two general classes of LTI systems that we are interested in identifying: minimum phase systems which are addressed by Omeje, and nonminimum phase systems which are addressed here. The roots of the numerator of a transfer function are known as zeros, and the roots of the denominator are known as poles. Stable minimum phase systems have all their poles and zeros inside the complex unit circle for discrete time

systems, or on the left half complex plane for continuous time systems. Stable nonminimum phase systems have all their poles inside the unit circle and at least one zero outside the unit circle. Stable minimum phase systems have stable inverses, while stable nonminimum phase systems possess unstable inverses. If a system has a stable inverse then its input can be directly calculated using its output. For a detailed explanation of the implications of nonminimum phase systems see [3]. Linear least squares regression is a useful tool in identifying linear systems. It allows us to take noisy data and find the best fit within the linear span of a chosen class of functions. These could be any of a large set of basis functions including Chebyshev, radial, Bessel, polynomial, and many others. Each of these classes of basis functions is attractive for different situations because of their numerical stability, ease of understanding, or direct applicability in different situations. We define a basis function as an element in a function space. Any function that can be described in this space can be represented by a linear combination of basis functions. For example, in the polynomial space we can describe a function using the linear combination of $a_0 + a_1 x + a_2 x^2$ and so on, and solving for the coefficients of a using linear regression. Many of the other basis functions come from solutions to the Sturm–Liouville eigenvalue problem of partial differential equations. The regression problem can be set up as follows: let $i \in \{1, \ldots, n\}$ denote a sample index. let x_{ij} denote basis function j at sample index i, and $\beta = \left[\beta_i, \ldots, \beta_k\right]'$ be the vector of unknown linear combination coefficients. Then let:

$$r_i = y_i - (\beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 ... + x_{ik}\beta_k)$$

where r_i is the residual or error, y_i is the dependent variable observation, x_i is the independent variable observation, and β represents the adjustable parameters. The least squares method is attractive as an identification method because it guarantees one optimal

set of parameters. This can be described using the residual sum of squares RSS which takes the form:

$$RSS = \sum_{i=1}^{n} (y_i - (\beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 ... + x_{ik}\beta_k))^2$$

Solving the least squares problem for the parameter estimate while looking for the minimum RSS leads to the “normal equations”:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

where $\hat{\beta}$ is the parameter estimate, $X_{nxk} = [x_{ij}]_{i-1:n, j=1:k}$ is the independent variable observation matrix, and $Y_{nx1} = [y_i]_{i=1:n}$ is the dependent variable observation matrix. The normal equation as described here only strictly applies to systems with stable inverses. The “normal equations” are very clear in how they work, but they come with a few problems. The “normal equations” are rarely used as written because they are numerically unstable with long observations, or too few observations, and do not work at all when systems are close to singular. Typically, the QR factorization is used when one has a large number of observations unless there is a specific need to use a more complicated method. The singular value decomposition (SVD) is used when a system is singular or nearly singular; it also gives a clearer indication of how each input is connected to the output and allows us to exclude data that has little effect on the results. As nonminimum phase systems have unstable inverses, a more creative solution is needed.

1.1.4 Estimation of Nonminimum Phase Transfer Functions

A high order approximation of function values from the transfer function can be obtained by dividing the FFT of the output, Y, by the FFT of the input, U, on an element-by-element basis. This is known as the empirical transfer function estimate or

ETFE. The ETFE has one entry for every measurement so with long observation times it gets quite large, unless the data set is broken up into shorter segments. Often, a few entries in the ETFE are infinite when dealing with nonminimum phase systems because the input is missing a specific frequency that is present in the output. This makes it difficult to describe the system based on the ETFE alone. One way to solve this problem is by using only key entries from the ETFE known as the roots of unity for the approximation. In Omeje's work the 2048 entries of the ETFE are separated into 64 equally spaced chunks of 32 entries. The first entry in each chunk is a root of unity. Each section corresponds to a repeated input signal. The values at each root of unity is an approximation of the transfer function at that point. Using a periodic signal and running it several times averages out the noise in the system which typically leads to better results. Using equally spaced points from the ETFE works as an approximation even without using a periodic input signal, but a lot of information is lost in that approximation for non-periodic signals. Instead of picking evenly spaced points and hoping that none of those values are missing from the input signal the entire ETFE can be fed into *ratdisk* by filling in the missing values with the average of its neighbors. The more missing frequencies there are the worse the approximation is, but with a decently well designed input signal there are few missing frequencies. All of the entries in the ETFE are approximations of the values of the transfer function at that point, so approximating missing values has little effect on the final result. Other smoothing methods also work but this method is simple and sufficient.

CHAPTER 2. Working principles of ratdisk

The *ratdisk* function interpolates/fits function values, the roots of unity (all of the point in the ETFE can be considered roots of unity as long as they are evenly spaced), on the complex unit disk. Only a few key features of this function are examined here. For a detailed explanation see [1]. The function *ratdisk* solves an equation that has the same basic form as the normal equation above: $\hat{a} = \hat{Z}b$ where a is the vector of the unknown

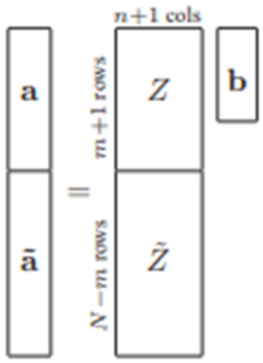


Figure 2.1 A schematic of the least squares problem taken from [1]

coefficients of the numerator of the transfer function, $\tilde{a}$ is the vector of the unused values, $\hat{a}$ is the combination of the two stacked vertically, Z is the first m+1 rows in the Toeplitz matrix (a matrix with constant diagonals, or a matrix where the initial vector is repeated and shifted down a row for each subsequent column), $\tilde{Z}$ is the rest of the Toeplitz matrix, $\hat{Z}$ is the combination of the two stacked vertically, and b is the vector of the unknown coefficients of the denominator of the transfer function. This matrix equation is arranged as in the figure above. "The least-squares problem is to minimize $\|\tilde{a}\|$ subject to the constraint $\|b\| = 1$" [1].

First, the approximate values of the transfer function at the roots of unity are extracted from the ETFE. Next the values of the transfer function at the roots of unity are convolved with radial basis functions in order to see how much of each basis function is contained at each of the roots of unity. This convolution is done by arranging the FFT of the roots of unity into a Toeplitz matrix of whichever size is specified. The entries of $\hat{Z}$ can be solved for explicitly with the sum

$$z_{jk} = \frac{1}{N+1} \sum_{l=0}^{N} z_l^{k-j} f_l$$

Where N is one less than the number of entries, Z_l^{k-j} is the basis function, and f_l is the value of the function fed in at that point (the roots of unity). The first column of the Toeplitz matrix is the DFT of the data divided by the number of data points.

Taking the singular value decomposition of $\hat{Z}$ gives the coefficients of b in the form of the right transformation matrix V because the norm of a is minimized only if b is a minimal singular vector of $\hat{Z}$, and as we have already mentioned the norm of b is 1. Now that we have the parameter estimate (essentially the Z transform of the denominator at roots of unity) of the function we need to find function values at the roots of unity of the new function. This allows us to go through a similar process again to find the coefficients of a. In order to get function values at the roots of unity for the estimated function we take the inverse FFT of b. Then the FFT of the function values of the estimated system multiplied by the roots of unity is evaluated to get the coefficients of a. This whole process can be summarized by the code snippet provided by [1]. In a perfect noisless world with infinite computational precision this would always be perfect, exact, and far more complicated than necessary. Unfortunately, there is always output noise, computers only have finite precision, and all the extra complexity is needed to deal with real systems. The true beauty of *ratdisk* comes from its ability to clean up the noise or manifest some form of

```
col = fft(fj)/(N+1);                % column of Toeplitz matrix
row = conj(fft(conj(fj)))/(N+1);    % row of Toeplitz matrix
Z = toeplitz(col,row(1:n+1));       % the Toeplitz matrix
[U,S,V] = svd(Z(m+2:N+1,:),0);      % singular value decomposition
b = V(:,n+1);                       % coefficients of q
qj = ifft(b,N+1);                   % values of q at zj
ah = fft(qj.*fj);                   % coefficients of p-hat
a = ah(1:m+1);                      % coefficients of p
pj = ifft(a,N+1);                   % values of p at zj
```

Figure 2.2 A code snippet describing the basic working principles of *ratdisk* taken from [1]

underlying symmetry by its clever usage of the singular value decomposition. For details see [1].

CHAPTER 3. METHODS AND PROCEDURES

3.1 Evaluating the transfer function estimate

Describing how well a transfer function fits the given data using the mean square error or MSE

$$\frac{\sum_{i=1}^{N} r_i^2}{N}$$

where r_i is the residual at each point and N is the number of samples in the observation. This works great when comparing different identification methods on the same data but does not work so well when comparing across systems. In order to compare across systems some sort of normalization must be used. In order to stay consistent with the system identification toolbox we will use the same normalization method known as the normalized root mean squared error and turn that into a percent fit [4].

$$\%fit = \left(\frac{\|y - \hat{y}\|}{\|y - \bar{y}\|}\right) \cdot 100$$

Where $\|*\|$ is the L2 norm, y is the observation, $\hat{y}$ is the prediction, and $\bar{y}$ is the mean of the observation.

Identifying minimum phase systems using the *ratdisk* function is very effective. It is capable of producing a near perfect reproduction of the original system even when noise is present as shown by this quote from Omeje's work [2]

> The analyses of the numerical results in chapter five shows that this technique gives better model estimation in all of our test functions than the MATLAB®

> system identification toolbox. It is seen from the tables of results that the percentage fit to estimation is over 99.5% for each of the functions tested using this technique whereas the corresponding percentage fit using system identification toolbox is below 88%.

This fitting is done without over fitting and is done using a simple random binary signal. For a detailed analysis of the use of *ratdisk* on minimum phase systems see [2]

3.1.1 Input signal structure

Typically, when identifying a system, an input signal that excites all dynamics present in that system is required. Standard input signals such as the unit step, ramp, and parabola can work for simple systems or those that are already well described. These signals allow for qualitative comparison between different systems but adding noise into the mix makes different qualities difficult to distinguish regardless of what type of system is under consideration. White noise is typically employed to excite even hidden modes, but several input optimization algorithms exist that guarantee good identification results [5]. Omeje implemented a binary input signal that switches from positive one to negative one in a somewhat random manner with great results. This signal is then repeated several times in order to transition the system from the transient domain into the stationary domain and average out any noise. Unfortunately, white noise and random binary signals often causes unbounded outputs with nonminimum phase systems. It is apparent from experimentation that adding a small amount of noise to the binary switching signal allows for stable outputs on nonminimum phase systems. This is known as a pseudorandom binary signal.

One of the main features of the pseudorandom input signal is the switching probability which ranges from zero to one. The switching probability is one of the main features that

optimal input design adjusts in order to get good results [5]. This work uses a method that is not guaranteed to be optimal but it usually works. Testing each switching probability in individual tests leads to great results in simulation because something close to every available frequency is tested directly. Running these tests in a real-world system would take a significant amount of time. All of the systems tested used a 0.1 second sample time which means that a single test would take about 3 minutes, 10 tests would take about 30 minutes, and 100 tests would take 300 minutes. As such this pseudorandom signal has been modified to sweep through several switching probabilities as it searches for the best fit. This signal changes the switching probability every 32 samples and is repeated in reverse starting at the midpoint. Further investigation led to the realization that having zero as the input at the beginning and end of the input signal also led to improved results. In more statistical methods that characterize the noise, like those available in the system identification toolbox, the zero inputs would help in determining the noise characteristics of the unexcited system. The sweeping pseudorandom signal provides sufficient variation in the input signal to find a good fit without resorting to more computationally intense or complex schemes such as those found in [5]. The sweeping pseudorandom signal has poor results on minimum phase systems when using *ratdisk* for the identification. Most often the most accurate approximation of minimum phase systems with a sweeping PRBS has many spurious poles and zeros that do not approximately cancel each other out.

3.1.2 Sample duration

In the transient domain nonminimum phase system often appear to be unstable because they can have unbounded output with many input signals. As such a nonminimum phase system that has stable poles can easily be misclassified if the test duration is sufficiently small. This identification algorithm runs for the same amount of time as those run by Omeje [2], 2048 samples, allowing for any apparent instability to work itself out. If

this entire simulation were to be run on a real-world system, it would take about 3.4 minutes based on the 0.1 second sample time. Systems with smaller sample times like those found in electrical circuits could be evaluated much more quickly, while systems on larger time scales like those found in thermal systems would take significantly longer. The observation matrix can become singular as the matrix gets larger so the identification method must be able to handle singular matrices or limit the duration of the test.

3.1.3 Expected transfer function order sweep

The *ratdisk* function (and most other identification methods) requires an expected order for the numerator and denominator of the system. This algorithm does a sweep through all possible combinations for causal systems up to eighth order. These results are then evaluated based on the MSE. The attempt with the lowest MSE is the one that is the closest match for the actual system. Typically, the identified system with the lowest MSE matches the order of the actual system. In some circumstances, especially with nonminimum phase systems having short experiment times, the algorithm will provide spurious poles and zeros. These pole zero pairs are often called Froissart doublets and approximately cancel each other out. *ratdisk* has a method for dealing with these extra poles and zeros but setting a good tolerance value has been elusive. Testing several tolerance values on each system is time consuming without improving the results significantly. There is probably a possible method to set a tolerance that allows the user to set a distance between the doublets that are canceled but finding the proper method is not achieved in this work. *ratdisk* favors numerator and denominator orders that are separated by one. Even going as far as just adding extra zeros if there are fewer than are necessary for a one sample delay. Interestingly this does not typically make the approximation less accurate as can be seen in system 4 in the results section.

3.1.4 The dangers of overfitting

In a generic data set increasing the order of the approximation reduces the error, drastically for each new order on the low end, and less so for higher order approximations. Excessively high order approximations can account for every minute change in the data, but also incorporate any noise into the model. Overfitting is a problem because high order approximations are difficult to work with and because incorporating noise into a model is not good practice. Interestingly, the errors tend to increase with higher order fits from *ratdisk* with all the systems tested in this work. An example is given below in table This table shows the MSE for the entire system order sweep of the first system tested. All the blank entries were either not tested because they are not causal or did not produce any usable results.

Table 3.1 A table of MSE for various system orders using *ratdisk*

order	1	2	3	4	5	6	7	8
0	0.2709							
1	0.6501	0.0025						
2			9.7473e+52					
3		inf	inf	1.1794e+20				
4					inf			
5	inf					4.5209e+27		
6							5.3816e+41	
7			inf					1.2391e+34

3.1.5 Unstable system exclusion

In this algorithm any systems with poles that have an absolute value larger than one, or outside the unit circle, are considered to have infinite MSE because their outputs are unbounded. This work only explores stable system identification, but there is nothing stopping this method from working on unstable systems. In many instances where spurious poles and zeros are present in the best estimation, the estimation that matches

the actual order of the system is unstable. The extra poles and zeros add just enough flexibility to the estimate to overcome this issue. Some may consider this to be a case of over fitting, but it is a drastic reduction in order from the ETFE and should not be considered as such.

3.1.6 Effect of output noise

Small amounts of output noise have little effect on retrieving system information for simple systems. Some nonminimum phase systems are much more sensitive to noise. In these tests we set the noise to be band limited white noise with various standard deviations to test the capabilities of *ratdisk*. Determining the noise level is not as simple as adding noise with a certain preset standard deviation because some systems/inputs cause large outputs. Resultantly, the standard deviation of the output noise is set to be a percentage of the root mean square, or RMS, of the noiseless output signal. 8% RMS noise seemed to be a good threshold for keeping the systems identifiable. 8% RMS noise corresponds to a signal to noise ratio of 12.5 or about 11 dB. Adding more noise led to poor results for all available methods. The level of noise that is considered acceptable is different depending on the system. For voice recognition a SNR of 10 dB is considered very low, but still reasonable to identify, and an SNR of -5 dB is nearly impossible to identify without specialized methods [6]. The minimum SNR for useable Wi-Fi signals is about 15 dB [7].

3.1.7 Experimental method

All the simulated systems were excited using a sweeping PRBS with 1% input noise for 2048 samples. The first and last chunk of 32 samples are zero and the signal is reversed and repeated at the midpoint. This means that each input signal is different, but they all come from the same settings. Output data is produced by a standard ARX simulator using the discrete time transfer function provided by the user. The output has white noise added

with a standard deviation of 8% of the RMS value of the system The noisy output of the system is fed into *ratdisk*, the default system identification toolbox method, and a filtered version of the system identification toolbox. The system identification toolbox uses some form of Gauss-Newton seeking algorithm, but the details are hidden behind the copyright of MathWorks. The identified system is then compared to the noisy system using the mean squared error and percent fit. The MSE and percent fit of the identified filtered system are compared to the filtered data and should not be directly compared to the other identification methods. Due to the random nature of the inputs and the difficulty in identifying some systems some of the data comes from simulations that were executed 10 or more times in one run through. The results are the best of that set. After running the simulations several more times it is clear that these are not the best possible results, but any results that were remarkably bad were thrown out. This helps illustrate the need to run systems several times if the identification results are poor and should not be construed to mean that these results are cherry picked from only the best possible results.

CHAPTER 4. RESULTS

4.1 Evaluation of identification methods on various systems

Each of these systems is tested using 8% RMS noise added to the output and are fed through several identification methods. The input and output signals are provided as well as the *ratdisk* approximation to the system. Due to the sample rate and limitations on the switching probability the input and output frequencies all fall in the range from .01 to 100 rad/s. A table of approximate poles and zeros is provided for each system.

4.1.1 System 1

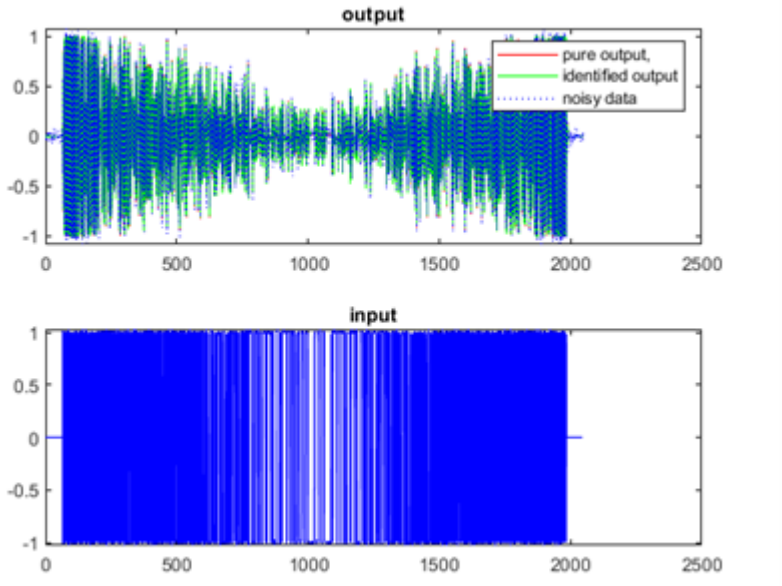


Figure 4.1 Input and output data for system 1

Table 4.1 A table of approximate poles and zeros for system 1

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	-0.5033 + 0.0536i	0.9411 + 0.2320i	-0.5085 + 0.1051i	-.5
Pole 2	-0.5033 - 0.0536i	0.9411 - 0.2320i	-0.5085 - 0.1051i	-.5
Zero 1	1.4928	-3.2392	1.4346	1.5
MSE	0.0014	5.4242e-05	0.0015	
Fit to data	92.2155	29.4298	92.0045	

The true transfer function for system 1 is:

$$H_1(z) = \frac{z - 1.5}{z^2 + z + 0.25}$$

and its zeros and poles are given in the last column of table 4.1. This first test shows a few interesting things about the identification methods and their measures of fit. It is clear that both the system identification toolbox and *ratdisk* are very capable of handling simple nonminimum phase systems. Feeding filtered data into the system identification toolbox usually leads to poor identification which is clear from this test. Both the system identification toolbox and *ratdisk* correctly identify this system as nonminimum phase and get very close to finding the actual poles and zeros of the system. This system has a small output and reacts more to high frequency inputs.

4.1.2 System 2

The true transfer function for system 2 is:

$$H_2(z) = \frac{0.07707\, z^2 - 0.1705\, z + 0.09429}{z^3 - 2.715\, z^2 + 2.456\, z - 0.7408}$$

and its zeros and poles are given in the last column of table 4.2. This second system exhibits some interesting identification behavior. *ratdisk* correctly identifies the qualities

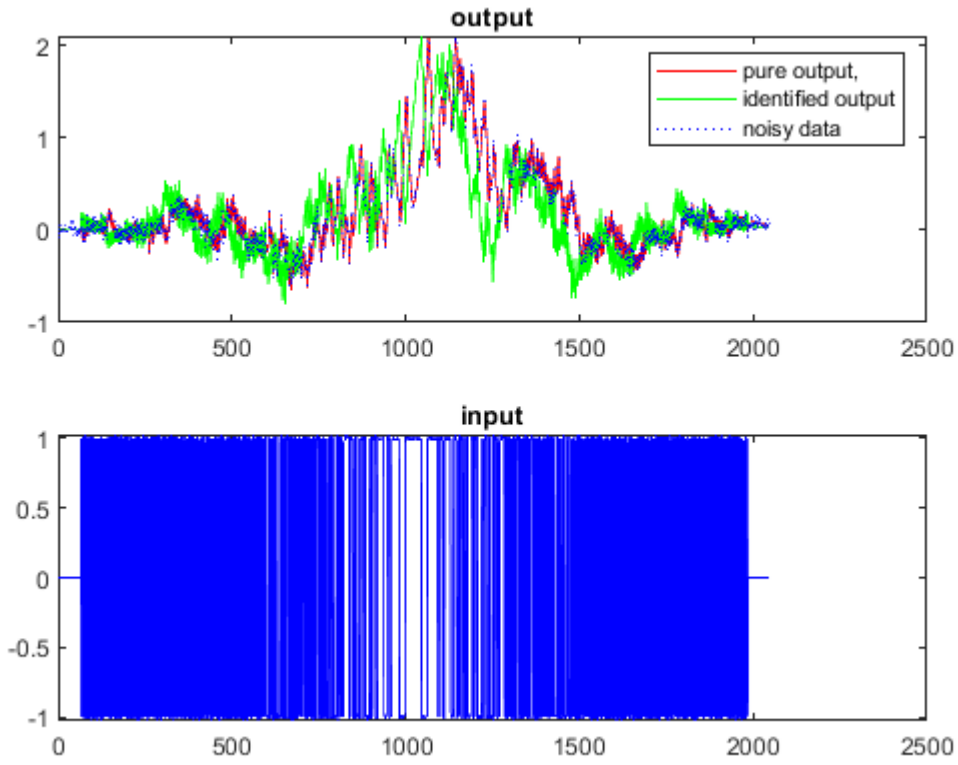


Figure 4.2 Input and output data for system 2

of the system but misses the exact values of the second and third poles. The system identification toolbox misclassifies one pole and one zero but gets much closer on the second and third poles. Filtering out any frequencies over 5 radians per second allows the system identification toolbox to correctly identify all aspects of the system. The results of the filtered system are great, but the MSE and percent fit are based on the filtered data and should not be directly compared to the other two methods.

4.1.3 System 3

The true transfer function for system 3 is:

$$H_3(z) = \frac{-0.7691\,z^3 - 1.359\,z^2 + 1.616\,z + 0.6008}{z^4 - 1.2\,z^3 + 0.3309\,z^2 - 0.6484\,z + 0.6065}$$

Table 4.2 A table of approximate poles and zeros for system 2

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	-0.9996 + 0.0000i	0.9915 + 0.0000i	0.9829 + 0.0000i	0.9915 + 0.0000i
Pole 2	0.8142 + 0.2463i	0.8665 + 0.0703i	0.5929 + 0.7417i	0.8617 + 0.0674i
Pole 3	0.8142 - 0.2463i	0.8665 - 0.0703i	0.5929 - 0.7417i	0.8617 – 0.0674i
Zero 1	-0.9987	1.1331	0.7267 + 0.5964i	1.1165
Zero 2	0.9068	1.0879	0.7267 - 0.5964i	1.0958
MSE	0.2502	3.4937e-05	0.1626	
Fit to data	-5.9500	98.3036	14.5825	

and its zeros and poles are given in the last column of table 4.3.

Table 4.3 A table of approximate poles and zeros for system 3

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	-0.3334 + 0.7462i	0.9336 + 0.1820i	-0.3478 + 0.7489i	-0.3333 + 0.7479i
Pole 2	-0.3334 - 0.7462i	0.9336 - 0.1820i	-0.3478 - 0.7489i	-0.3333 - 0.7479i
Pole 3	0.9333 + 0.1836i	0.9963 + 0.0794i	0.9303 + 0.1830i	0.9333 + 0.1834i
Pole 4	0.9333 - 0.1836i	0.9963 - 0.0794i	0.9303 - 0.1830i	0.9333 - 0.1834i
Zero 1	-2.4959	1.0250 + 0.0000i	-2.6174	-2.4858
Zero 2	1.0248	0.9962 + 0.0793i	1.0256	1.0253
Zero 3	-0.3078	0.9962 - 0.0793i	-0.4147	-0.3065
MSE	0.0953	0.0025	0.1192	
Fit to data	92.0547	92.9131	91.1158	

This third system is another example of both *ratdisk* and the system identification toolbox estimating something very close to the true system. Filtering the data leads to less accurate identification.

4.1.4 System 4

The true transfer function for system 4 is:

$$H_4(z) = \frac{z^2 + 2.5\,z + 2.5}{z^3 - 2.451\,z^2 + 2.129\,z - .6666}$$

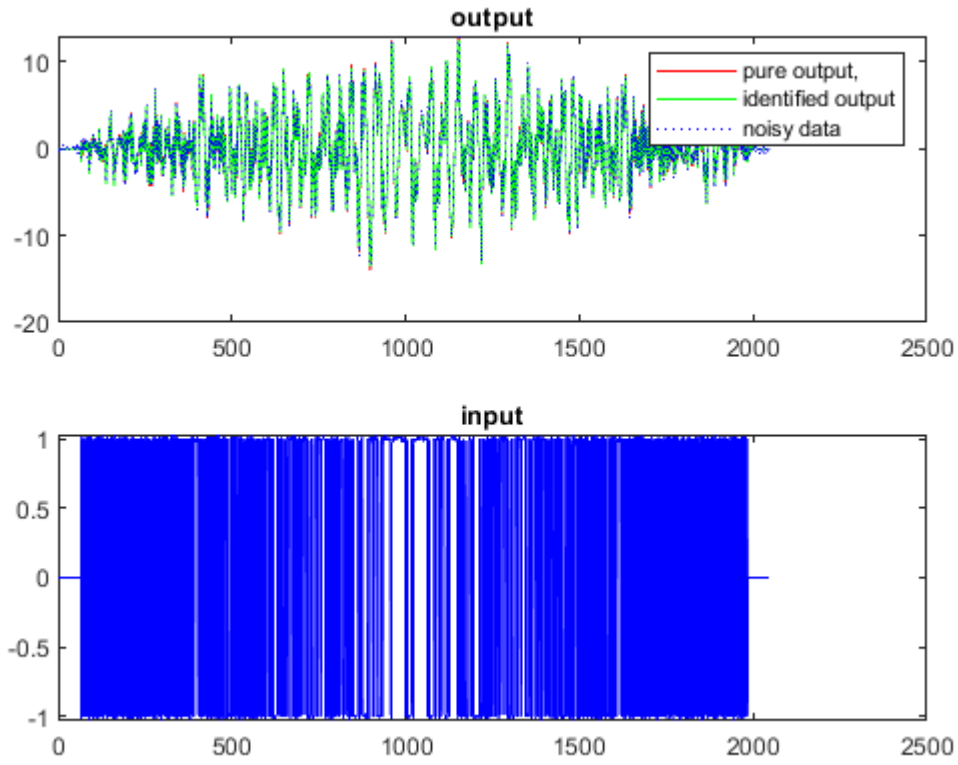


Figure 4.3 Input and output data for system 3

and its zeros and poles are given in the last column of table 4.4. This fourth system shows the classic example of Froissart doublets and how they can sometimes make the identification more accurate even when the zeros are not perfectly identified.

4.1.5 System 5

The true transfer function for system 5 is:

$$H_5(z) = \frac{z + 1.1}{z^2 + z + 1}$$

and its zeros and poles are given in the last column of table 4.5. With system 5 the system identification toolbox is the clear winner. It appears that *ratdisk* has a little bit of trouble

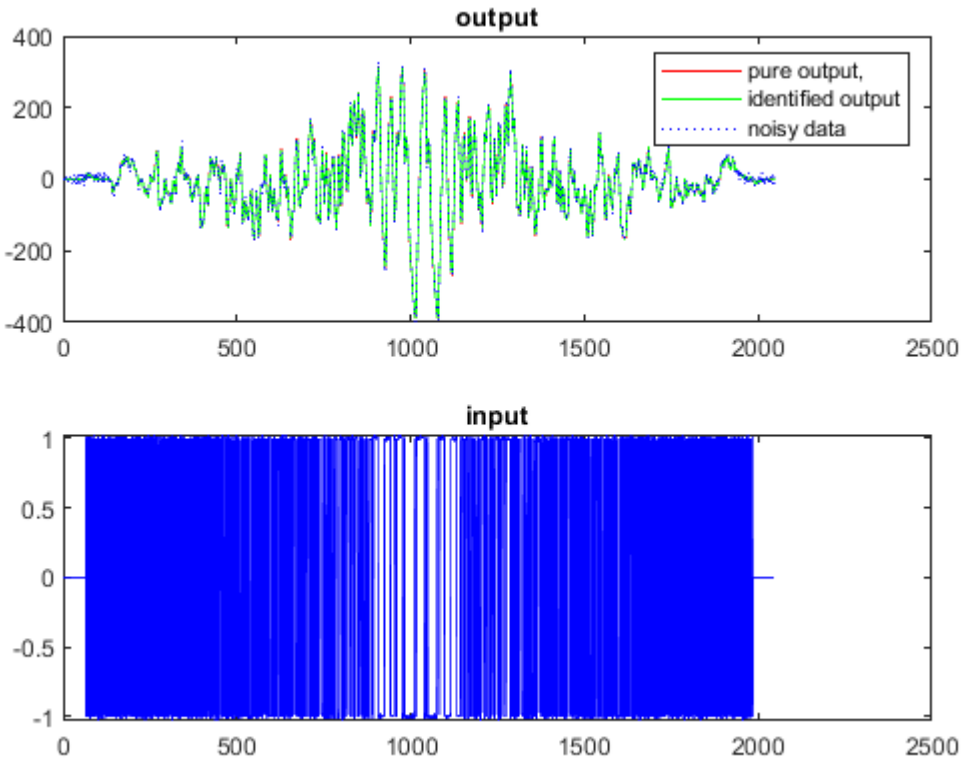


Figure 4.4 Input and output data for system 4

matching the system perfectly when the poles are directly on the unit disk, but the predicted poles and zeros are very close to the true system.

4.1.6 System 6

The true transfer function for system 6 is:

$$H_6(z) = \frac{8\,z^6 + 6\,z^5 + 12\,z^4 + 20\,z^3 + 5\,z^3 + 6\,z^2 + 7}{z^7 + z^6 + z^5 + z^4 + z^3 + z^2 + z + .5}$$

and its zeros and poles are given in the last column of table 4.6. This sixth and final example shows that both *ratdisk* and the system identification toolbox are very capable in

Table 4.4 A table of approximate poles and zeros for system 4

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	0.9426 + 0.0000i	-0.9977	0.9427 + 0.0000i	0.9427 + 0.0000i
Pole 2	0.7547 + 0.3730i	0.9426	0.7462 + 0.3812i	0.7542 + 0.3720i
Pole 3	0.7547 - 0.3730i	0.9068	0.7462 - 0.3812i	0.7542 - 0.3720i
Pole 4			-0.7699 + 0.6238i	
Pole 5			-0.7699 - 0.6238i	
Zero 1	-0.9120 + 1.1901i	1.7151	-0.0274 + 1.5043i	-1.2500 + 0.9682i
Zero 2	-0.9120 - 1.1901i	0.9086	-0.0274 - 1.5043i	-1.2500 - 0.9682i
Zero 3			-0.5863 + 0.6969i	
Zero 4			-0.5863 - 0.6969i	
MSE	60.0064	1.5615	69.5380	
Fit to data	91.9458	98.0393	91.3297	

Table 4.5 A table of approximate poles and zeros for system 5

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	-0.5000 + 0.8660i	0.9978	-0.4994 + 0.8650i	-0.5000 + 0.8660i
Pole 2	-0.5000 - 0.8660i	-0.108	2 -0.4994 - 0.8650i	-0.5000 - 0.8660i
Zero 1	-1.1009	0.9965	-1.1742	-1.1000
MSE	0.3426	0.0119	6.9525	
Fit to data	92.2926	45.8383	65.2792	

dealing with higher order systems. The expected system order sweep was expanded to 15 for both the numerator and denominator for *ratdisk*.

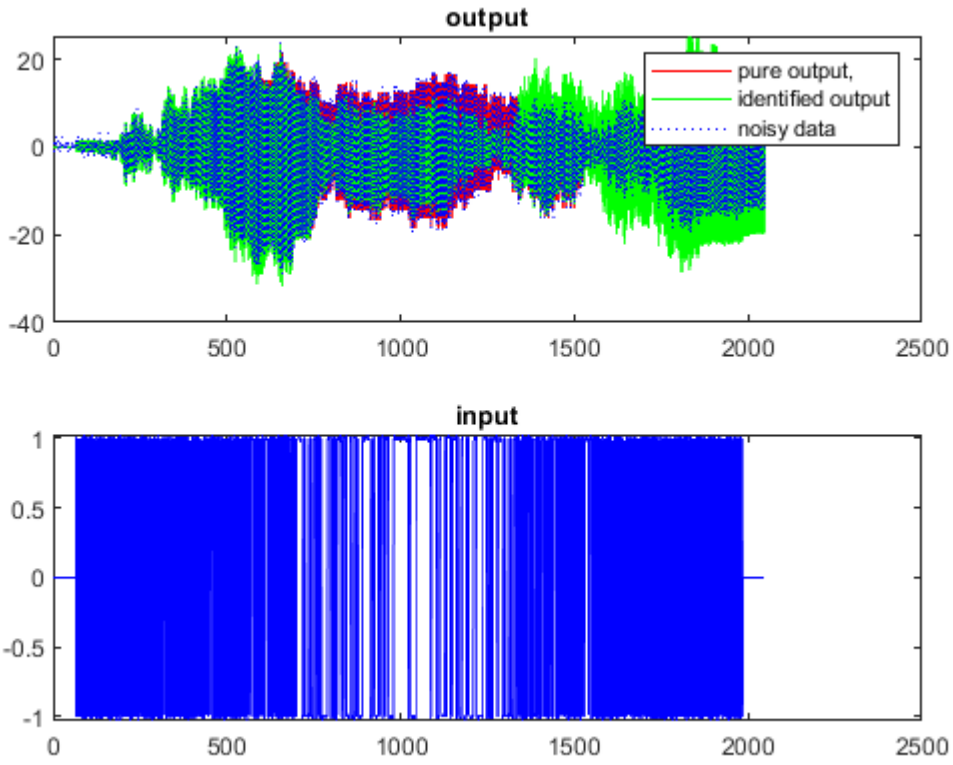


Figure 4.5 Input and output data for system 5

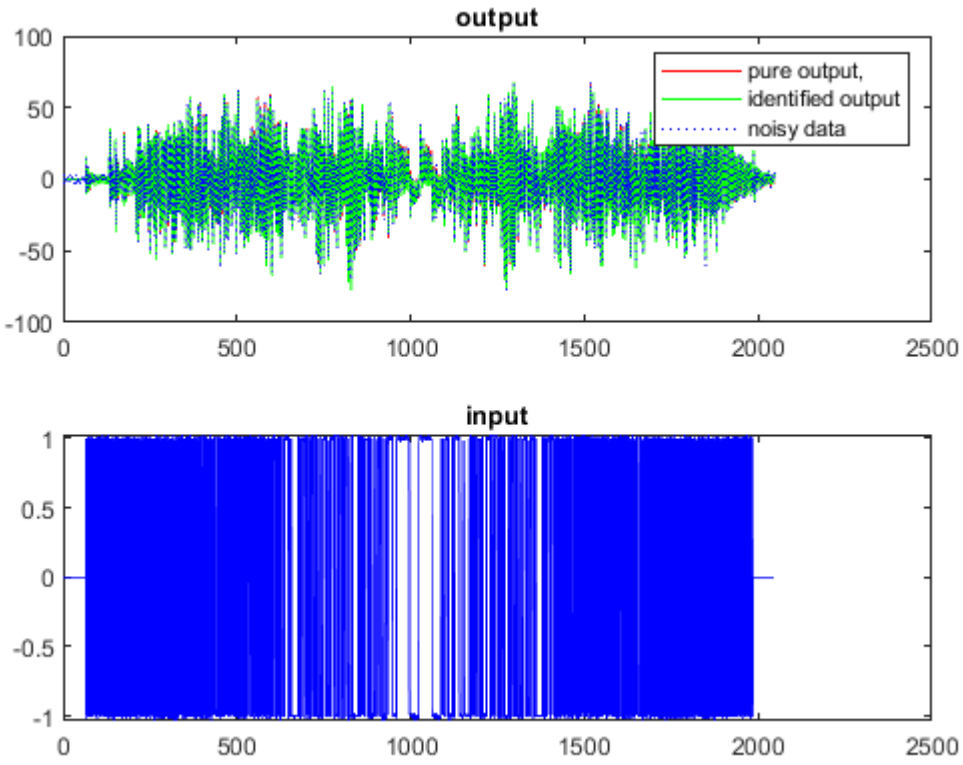


Figure 4.6 Input and output data for system 6

Table 4.6 A table of approximate poles and zeros for system 6

	Unfiltered Sys ID	Filtered Sys ID	*ratdisk*	True System
Pole 1	0.6613 + 0.7341i	-1.0000 + 0.0000i	0.6610 + 0.7336i	0.6614 + 0.7340i
Pole 2	0.6613 - 0.7341i	0.6132 + 0.7814i	0.6610 - 0.7336i	0.6614 - 0.7340i
Pole 3	-0.0954 + 0.9433i	0.6132 - 0.7814i	-0.0966 + 0.9445i	-0.0954 + 0.9434i
Pole 4	-0.0954 - 0.9433i	0.9862 + 0.0000i	-0.0966 - 0.9445i	-0.0954 - 0.9434i
Pole 5	-0.6812 + 0.5278i	0.9702 + 0.0792i	-0.6823 + 0.5204i	-0.6834 + 0.5268i
Pole 6	-0.6812 - 0.5278i	0.9702 - 0.0792i	-0.6823 - 0.5204i	-0.6834 - 0.5268i
Pole 7	-0.8097 + 0.0000i	0.7636 + 0.0000i	-0.8712 + 0.0000i	-0.7651 + 0.0000i
Zero 1	0.2025 + 1.3102i	-0.8909 + 0.0000i	0.1871 + 1.2885i	0.2020 + 1.3150i
Zero 2	0.2025 - 1.3102i	1.7578 + 0.0000i	0.1871 - 1.2885i	0.2020 - 1.3150i
Zero 3	-0.9965 + 0.0000i	0.9583 + 0.0954i	-0.9672 + 0.0248i	-1.0000 + 0.0000i
Zero 4	-0.8662 + 0.0000i	0.9583 - 0.0954i	-0.9672 - 0.0248i	-0.8266 + 0.0000i
Zero 5	0.3354 + 0.6985i	0.9782 + 0.0000i	0.3351 + 0.7098i	0.3362 + 0.6964i
Zero 6	0.3354 - 0.6985i	0.9280 + 0.0000i	0.3351 - 0.7098i	0.3362 - 0.6964i
MSE	3.2983	0.2464	3.8209	
Fit to data	91.8446	80.5935	91.22	

CHAPTER 5. Conclusion

Each of the identification methods explored in this thesis perform well in different circumstances. The *ratdisk* algorithm is very capable in the world of system identification in most situations,and handles noise with few issues even when working with nonminimum phase systems. The system identification toolbox is also quite capable at identifying nonminimum phase systems and provides extra tools, like filtering, to condition the data prior to identifying the system. Based on all the test performed in this research it is clear that using several identification methods allows us to achieve useful results from various types of systems. Using different methods also provides a way to gauge how well a model matches the data compared other models.

5.1 Suggestions for further research

The adaptive Antoulas—Anderson, or AAA, algorithm seems to be the next logical step in system identification by rational approximation. This algorithm uses a method of barycentric fitting which is capable of doing everything *ratdisk* does while achieving slightly better accuracy for details see [8] Currently there are no easily available methods that produce rational approximations with specifically chosen orders for the top and the bottom of the fraction, the top and bottom of the fraction are either the same or off by exactly one. NREL has hinted that they will release an extension of the AAA algorithm that can be used with any chosen order for both numerator and denominator [9]. Hopefully this will clear up some of the spurious poles and zeros that are still present in *ratdisk*. Any

identification methods that use an optimal input signal or any new form of identification are also worth exploring.

Bibliography

[1] P. Gonnet, R. Pachon, and L. N. Trefethen, *Robust Rational Interpolation and Least-Squares*, Oxford University Mathematical Institute Std., 2011.

[2] A. V. Omeje, "Linear system identification via rational function approximation of empirical transfer function estimates," Master's thesis, Idaho State University, 2019.

[3] J. B. Hoagg and D. S. Bernstein, "Nonminimum-phase zeros - much to do about nothing - classical control - revisited part ii," *IEEE Control Systems Magazine*, vol. 27, no. 3, pp. 45–57, jun 2007, doi: 10.1109/MCS.2007.365003.

[4] T. M. Works. Goodness of fit. Accessed 28-Dec-21. [Online]. Available: https://www.mathworks.com/help/ident/ref/goodnessoffit.html

[5] C. BRIGHENTI, "On input design for system identification: Input design using markov chains," Master's thesis, Stockholm, Sweden, 2009.

[6] S. Fattah, W.-P. Zhu, and M. O. Ahmad, "An approach to arma system identification at a very low signal-to-noise ratio," *IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 4, 2005.

[7] M. Heath. (2020, jul) Wi-fi setup guide: What is a good signal level or signal-to-noise ratio (snr) for wi-fi? Accessed 28-Dec-21. [Online]. Available: https://www.increasebroadbandspeed.co.uk/what-is-a-good-signal-level-or-signal-to-noise-ratio-snr-for-wi-fi

[8] O. S. YUJI NAKATSUKASA and L. N. TREFETHEN. The aaa algorithm for rational approximation. [Online]. Available: https://arxiv.org/pdf/1612.00337.pdf

[9] L. Monzón, W. Johns, S. Iyengar, M. Reynolds, J. Maack, and K. Prabakar, *A Multi-Function AAA Algorithm Applied to Frequency Dependent Line Modeling*, National Renewable Energy Laboratory Std., 2020, nREL/CP-5D00-75381. [Online]. Available: https://www.nrel.gov/docs/fy20osti/75381.pdf

APPENDIX. MATLAB code

```
% script for ratdisk arx identification:

clc;
close all;
clear all;
disp('various results based on switching prob. in rpbs')
%resp = 'y';

% define a stable arx model
% The characteristic equation's coeff. are [ 1  a]
%a=[1 .25];
%b=[.1 -.15];%nmp
%a=[.005 .08 .006 .08 .005];
%a=[-2.715 2.456 -.7408];
%b=[.07707 -.1705 .09429];%nmp
    %a = [-1.2 0.3309 -0.6484 0.6065];
    %b =  [-0.7691 -1.359 1.616 0.6008];%nmp Version 7.0 of the
 CONTSID Toolbox (A. Padilla, H. Garnier, and M. Gilson, 2015)-System
 1
    a = [-2.451 2.129 -.6666];
    b =  [.0644 .0249 -.0045];
   %a=[1 1];
   %b=[1 .5];
     %a = [ -2.451 2.129 -.6666];
     %b =  [1  2.5 2.5]; %nmp
   %b=[1 1.1];%nmp
    %a=[1 1 1 1 1 1 .5 ];
    %b=[8  6 12  20 5  6 7 ];%nmp



%Set iter to more than one if you want to run multiple simulations in
 a row
iter=10;

for in=1:iter

    p =.1;%1/iter*in-.01;% input('pls. enter switching prob. p in
 (0,1), p = ');
    N =64; %input('pls. enter N(as power of 2): ');
    M =15; %input('pls. enter M(as power of 2, less than N): ');
    n = 1;%input('pls. enter n(as numerator order of the ARX Model):
 ');
    m = 2;%input('pls. enter m(as denomiator order of the ARX Model):
 ');
    c='d';%lower(input('Select option from -> (a) No noise (b) Input
 noise (c) output noise (d) both : ','s'));

    u{in}  = prbs(N,p);
    uc = ones(N,M);

    ucT{in} = uc.*u{in};
```

```
    for ii=1:M

    p =1/(1*M)*ii-.01;
    O{ii}  = prbs(N,p);
    end
    ucT{in}=cell2mat(fliplr(O));


    if c=='a' || c=='c'
    ucT{in} = reshape(ucT,N*M,1);% this gives us M repetitions of the
N-periodic signal
    noiseT{in}=reshape(noiseT{in},N*M,1);
    elseif c=='b' || c=='d'
    sigmai = .01;% input('enter std. dev. for input noise? : ');
    ucT{in}=ucT{in}+sigmai*randn(size(ucT{in}));
    ucT{in}=[zeros(N,1),(ucT{in}),fliplr(ucT{in}),zeros(N,1)];

    M=(2*M+2);
    ucT{in} = reshape(ucT{in},N*M,1);

    else
    end


    %%%%%%%%%%%%%%%
    %%%%%%


    % for input to arx_sim
    ucfft{in} = fft(ucT{in});
    %ucfft{in}(ucfft{in}==0) = rms(ucfft{in});



    if c=='a' || c=='b'
    yc{in} = arx_sim(a,b,N*M,ucT{in});

    elseif c=='c' || c=='d'
    sigmaset = .08;%input('enter std. dev. for output noise? : ');
    yc{in} = arx_sim(a,b,size(ucT{in},1),ucT{in});
    sigma2{in}=sigmaset*rms(yc{in});
    yc{in} =yc{in}+ sigma2{in}*randn(size(ucT{in}));

    else
    end



    ycfft{in}=fft(yc{in});

    etfN_M{in} = ycfft{in}./ucfft{in};
  etfN_M{in}(isinf(etfN_M{in})) = nan;
  etfN_M{in}=fillmissing(etfN_M{in},'pchip');
```

```
  %we are now using the entire ETFE


% now fetch the eft values at the N roots of unity, by taking every
 Mth
% data point in the N*M element etfN_M

    etf_N{in}=zeros(N,1);
    for k = 1:N
        jj=(k-1)*M+1;
        etf_N{in}(k) = etfN_M{in}(jj);

    end
    % do a safety check: we may have accidentally missed one of the
    % N frequencies in our u : this would show up as an Inf value in
    % abs(etf_N) If this occors we just try again
   if max(abs(etf_N{in})) == Inf

       continue
   end

   %Find the best estimation with different numerator and denominator
   %degrees. Most often we just search through 8 by 8, but when
 testing
   %higher order systems a higher order can be used.
    MSE{in}=NaN(8,8);
    for ii=1:8
        for jj=0:ii-1
    [r,num,den,mu,nu,poles] = ratdisk(etfN_M{in},jj,ii,M*N-1);

    % so the numerator and denominator match our convention (highest
    % power to lowest power, we will convert them:
    % Note: num and den are column vectors. We want row vectors, so
    % transpose and then fliplr and normalize our denominator
    bratdisk = fliplr(num');
    aratdisk = fliplr(den');
    bratdisk = 1/aratdisk(1)*bratdisk;
    aratdisk = 1/aratdisk(1)*aratdisk;
   % disp('poles of estimated transfer function:')
    Pest{in,ii,jj+1}=roots(aratdisk);
    Pest{in,ii,jj+1};
   % disp('zeros of estimated transfer function:')
    Rest{in,ii,jj+1}=roots(bratdisk);
    Rest{in,ii,jj+1};
    %disp('hit enter to continue')

    %simulate the true system without noise
    ycT{in} = arx_sim(a,b,size(ucT{in},1),ucT{in});

    try

       %Simulate the estimated system
            ze=bratdisk;
```

```
    ycP{in,ii,jj+1} =
 arx_sim(aratdisk(2:length(aratdisk)),ze,size(ucT{in},1),ucT{in});
        MSE{in}(ii,jj+1) = (sum((yc{in}-ycP{in,ii,jj+1}).^2)/
(N*M)); %The MSE is always positive which makes using the minimum MSE
 an easy tool to use.
        est=Pest{in,ii,jj+1};
        %Exclude unstable systems

        for kk=1:numel(est)
        if abs(est(kk))>1
      % MSE{in}(ii,jj+1)=inf;
        end

    end


    %%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%
    %notice the slight difference from what Vitus had for the percent
 fit

    Perc_Predictn{in}(ii,jj+1) = 100*(1 -(norm(yc{in}-ycP{in,ii,jj
+1})/norm(yc{in}-mean(yc{in}))));

    catch

    end
        end
    end
    %This finds the best MSE out of the different order numerators and
    %denominators
    [ ii,jj ] = minmat( MSE{in} );
    indbest{in}=[ii,jj];

    %%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%









     end


     truepoles=roots([1 a]);
    truezeros=roots(b);

bestMSE=NaN(2,iter);
```

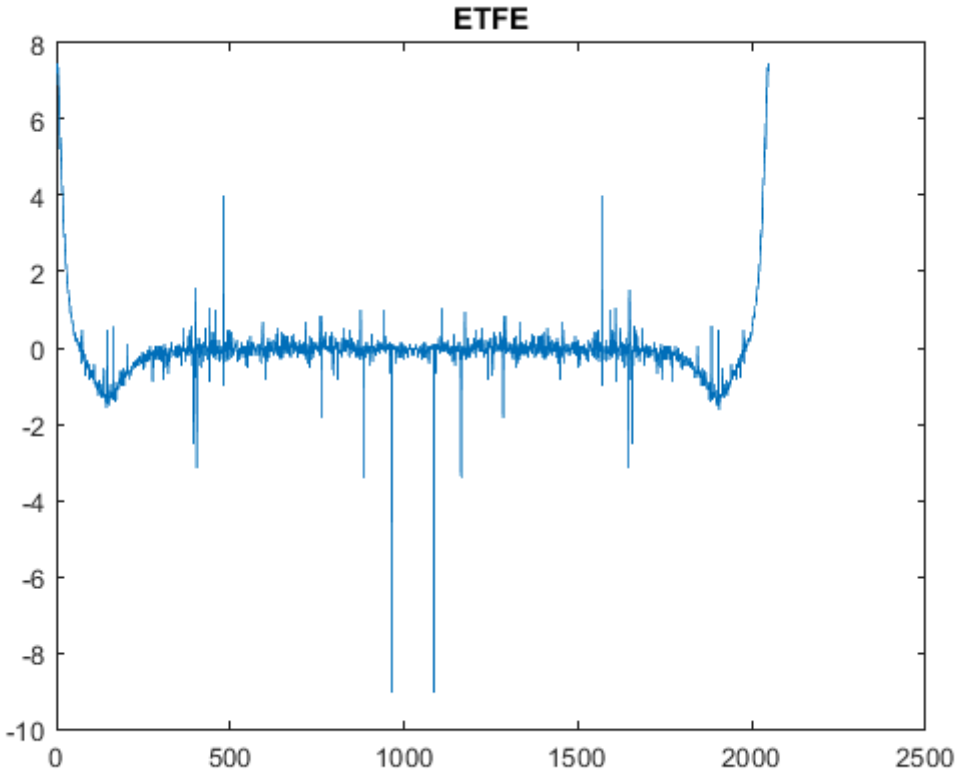
```
%For situations where more than one simulation are excicuted in one
 run of
%the script this section finds the best of all of the runs.
for mm=1:iter
    try
    bestMSE(1,mm)=MSE{1,mm}(indbest{1,mm}(1),indbest{1,mm}(2));

    catch
    end
end
[t,in]=minmat(bestMSE)

figure(1)

    plot(1:size(ucfft{in},1),etfN_M{in})
    title('ETFE')

    figure(5)

    subplot(2,1,1);
      plot([1:length(ucT{in})],ycT{in},'r',
[1:length(ucT{in})],ycP{in,indbest{in}(1),indbest{in}
(2)},'g',1:length(ucT{in}),yc{in},'b:')
    title('output');
    legend('pure output,','identified output','noisy data');
    subplot(2,1,2);
    plot([1:length(ucT{in})],ucT{in},'b')
    title('input');

    %Display the needed information for comparison for the ratdisk
 method
    POratdisk=Pest{in,indbest{in}(1),indbest{in}(2)}
    ZEratdisk=Rest{in,indbest{in}(1),indbest{in}(2)}
    mseratdisk=MSE{in}(indbest{in}(1),indbest{in}(2))
    Fitratdisk=Perc_Predictn{in}(indbest{in}(1),indbest{in}(2))

%create a data object for sysid toolbox.
mydata=iddata(yc{in},ucT{in},.1);
%Filter the data with a set frequency. this was determined using the
 sysid
%app and cutting off what appears to be noise. filters both input and
%output.

     mydataf = idfilt(mydata,5,0.031831);
     %create transfer function objects based on system identification
 toolbox
     tf1 = tfest(mydata, length(a), length(b), 'Ts', 0.1);
     tf2 = tfest(mydataf, length(a), length(b), 'Ts', 0.1);
      %Display the needed information for comparison for the sysid
 method
     PO=roots(tf1.Denominator)
      ZE=roots(tf1.Numerator)
      fit=tf1.Report.Fit.FitPercent
      mse=tf1.Report.Fit.MSE
```

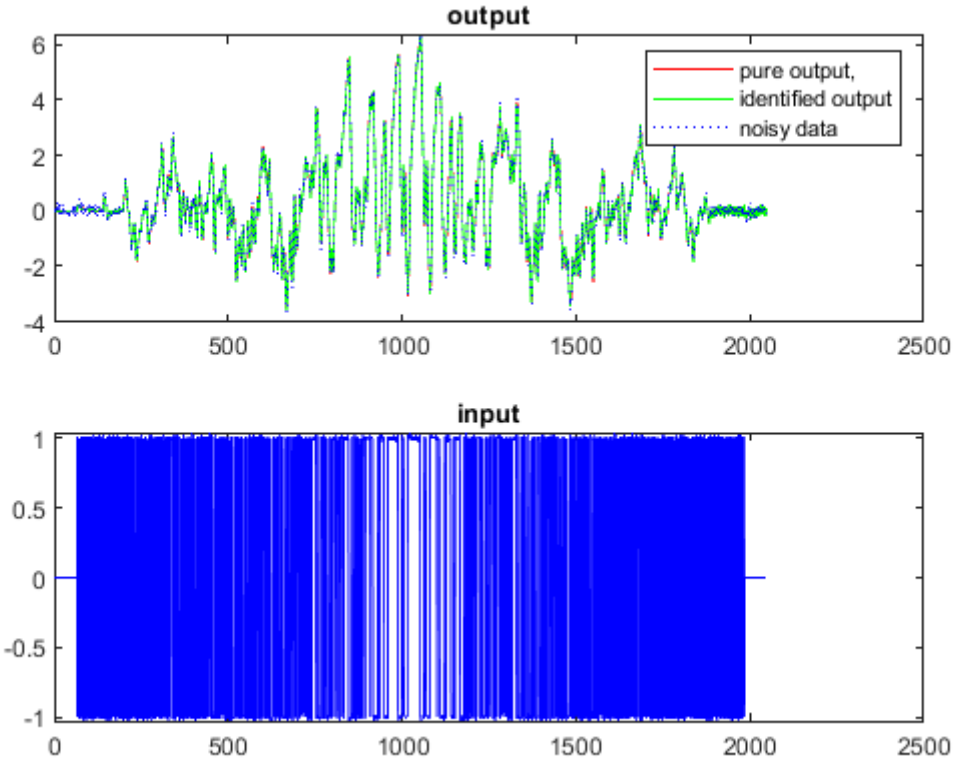
```
      %Display the needed information for comparison for thefiltered
 sysid method
     POf=roots(tf2.Denominator)
     ZEf=roots(tf2.Numerator)
     Fitf=tf2.Report.Fit.FitPercent
     MSEf=tf2.Report.Fit.MSE
     %Show the true poles and zeros in continuous time
    tf3=d2c(tf(b,[1,a],.1),'tustin')
    contzer=roots(tf3.Numerator{1})
    contpol=roots(tf3.Denominator{1})
```

```
various results based on switching prob. in rpbs

t =

     1


in =

     9

Warning: Imaginary parts of complex X and/or Y arguments ignored.

POratdisk =

  -0.9759 + 0.2177i
  -0.9759 - 0.2177i
   0.1829 + 0.9832i
   0.1829 - 0.9832i
   0.7421 + 0.3701i
   0.7421 - 0.3701i
   0.9440 + 0.0000i


ZEratdisk =

   0.1592 + 0.9798i
   0.1592 - 0.9798i
  -0.8149 + 0.5519i
  -0.8149 - 0.5519i
  -0.9758 + 0.0000i
   0.2818 + 0.0000i


mseratdisk =

    0.0255


Fitratdisk =

   89.9640
```

```
PO =

   0.9420 + 0.0000i
   0.7511 + 0.3758i
   0.7511 - 0.3758i


ZE =

   -0.3355
    0.0098


fit =

   92.1047


mse =

    0.0158


POf =

   -0.9999
    0.9940
    0.9399


ZEf =

    3.0747
    0.9940


Fitf =

   98.0707


MSEf =

   5.5765e-04


tf3 =

  -0.005603 s^3 - 0.3291 s^2 + 3.394 s + 108.6
  ---------------------------------------------
        s^3 + 4.233 s^2 + 26.88 s + 14.6

Continuous-time transfer function.
```

```
contzer =

   -63.4747
    20.0000
   -15.2682


contpol =

   -1.8215 + 4.6275i
   -1.8215 - 4.6275i
   -0.5903 + 0.0000i
```





Published with MATLAB® R2020a

```
function y = arx_sim(a,b,N,u,y0)
% arx_sim runs an arx model of the form:
% y(k) + a(1)y(k-1) + ... + a(n)y(k-n) = b(1)u(k-1) + ... + b(m)u(k-m)
% for N time steps k = 1,...,N with exogenous input u and optional IC
 y0
% usage:
% y = arx_sim(a,b,N,u,y0)
% y will be an N x 1 vector of system outputs
% or:
% y = arx_sim(a,b,N,u) in which case y0 = zeros(N,1) is assumed.
%
% Notes:
%  0) the coeff. vector a is NOT the coeff.'s of the char. poly,
%     rather [ 1 , a]  is.  I.e., we normalize the leading coeff.
%     of y(k) in the arx model to be 1.
%  1) it must be the case that m <= n or an error will be thrown.
%  2) u must be a vector with N values, or an error will be thrown.
%  3) N must be >= n, or an error will be thrown.
n = length(a);
m = length(b);
if m > n
    error('m must be <= n');
end
if N < n
    error('N must be >= n');
end
y = zeros(N,1);
if nargin == 5
    y(1:n) = y0;
end
for k = n+1:N
    y(k) = -dot(a,y(k-1:-1:k-n)) + dot(b,u(k-1:-1:k-m));
end
return
```

```
Not enough input arguments.

Error in arx_sim (line 18)
n = length(a);
```

Published with MATLAB® R2020a

```
function [ a,b ] = minmat( c )
as=size(c);
total_ele=numel(c);
[~,I]=min(c(:));
r=rem(I,as(1));
a=r;
b=((I-a)/as(1))+1;
if a==0
    a=as(1);
    b=b-1;
else
    a=r;
    b=b;
end
end
```

```
Not enough input arguments.

Error in minmat (line 2)
as=size(c);
```

Published with MATLAB® R2020a

```
function u = prbs(N,p)
% prbs creates an psuedo-random binary sequence alternating between
% 1 and -1, with switching probability p in [0 ,1]
% The closer p is to 1, the more likely a switch in state will occur.
%
% usage: u = prbs(N,p)
% where: N: length of u (i.e., if t = 1:N, u = [u(1),...,u(N)]
%        p: switching probability in [0,1]
u = ones(N,1);
r = rand(N,1);
Mask = (r < p);
s = 1-2*Mask;
for k = 2:N
    u(k) = s(k)*u(k-1);
end
return
```

```
Not enough input arguments.

Error in prbs (line 9)
u = ones(N,1);
```

Published with MATLAB® R2020a

```
function [r,a,b,mu,nu,poles,residues] = ratdisk(f,m,n,N,tol)
%  Input: Function f or vector of data at zj = exp(2i*pi*(0:N)/(N+1))
%           for some N>=m+n.  If N>>m+n, it is best to choose N odd.
%         Maximal numerator, denominator degrees m,n.
%         An optional 5th argument specifies relative tolerance tol.
%           If omitted, tol = 1e-14.  Use tol=0 to turn off
 robustness.
% Output: function handle r of exact type (mu,nu) approximant to f
%         with coeff vectors a and b and optional poles and residues.
% P. Gonnet, R. Pachon, L. N. Trefethen, January 2011

if nargin<4, if isfloat(f), N=length(f)-1;
  else N=m+n; end, end                          % do interpolation if no
 N given
N1 = N+1;                                       % no. of roots of unity
if nargin<5, tol = 1e-14; end                   % default rel tolerance
 1e-14
if isfloat(f), fj = f(:);                       % allow for either
 function
else fj = f(exp(2i*pi*(0:N)'/(N1))); end        %   handle or data vector
ts = tol*norm(fj,inf);                          % absolute tolerance
M = floor(N/2);                                 % no. of pts in upper
 half-plane
f1 = fj(2:M+1); f2 = fj(N+2-M:N1);              % fj in upper, lower
 half-plane
realf = norm(f1(M:-1:1)-conj(f2),inf)<ts;       % true if fj is real
 symmetric
oddN = mod(N,2)==1;                             % true if N is odd
evenf = oddN & norm(f1-f2,inf)<ts;              % true if fj is even
oddf  = oddN & norm(f1+f2,inf)<ts;              % true if fj is odd
row = conj(fft(conj(fj)))/N1;                   % 1st row of Toeplitz
 matrix
col = fft(fj)/N1; col(1) = row(1);              % 1st column of Toeplitz
 matrix
if realf, row = real(row);                      % discard negligible imag
 parts
  col = real(col); end
d = xor(evenf,mod(m,2)==1);                     % either 0 or 1
while true                                      % main stabilization loop
  Z = toeplitz(col,row(1:n+1));                 % Toeplitz matrix
  if ~oddf & ~evenf                             % fj is neither even nor
 odd
    [U,S,V] = svd(Z(m+2:N1,:),0);               % singular value
 decomposition
    b = V(:,n+1);                               % coeffs of q
  else                                          % fj is even or odd
    [U,S,V] = svd(Z(m+2+d:2:N1,1:2:n+1),0);     % special treatment for
 symmetry
    b = zeros(n+1,1); b(1:2:end) = V(:,end); % coeffs of q
  end
  if N > m+n && n>0, ssv = S(end,end);          % smallest singular value
```

```
 else ssv = 0; end                          % or 0 in case of
interpolation
 qj = ifft(b,N1); qj = qj(:);               % values of q at zj
 ah = fft(qj.*fj);                          % coeffs of p-hat
 a = ah(1:m+1);                             % coeffs of p
 if realf a = real(a); end                  % discard imag. rounding
errors
 if evenf a(2:2:end) = 0; end               % enforce even symmetry
of coeffs
 if  oddf a(1:2:end) = 0; end               % enforce odd symmetry of
coeffs
 if tol>0                                   % tol=0 means no
stabilization
   ns = n;                                  % no. of singular values
   if oddf|evenf, ns = floor(n/2); end
   s = diag(S(1:ns,1:ns));                  % extract singular values
   nz = sum(s-ssv<=ts);                     % no. of sing. values to
discard
   if nz == 0, break                        % if no discards, we are
done
   else n=n-nz; end
 else break                                 % no iteration if tol=0.
 end
end                                         % end of main loop
nna = abs(a)>ts; nnb = abs(b)>tol;          % nonnegligible a and b
 coeffs
kk = 1:min(m+1,n+1);                        % indices a and b have in
 common
a = a(1:find(nna,1,'last'));                % discard trailing zeros
 of a
b = b(1:find(nnb,1,'last'));                % discard trailing zeros
 of b
if length(a)==0 a=0; b=1; end               % special case of zero
 function
mu = length(a)-1; nu = length(b)-1;         % exact numer, denom
 degrees
r = @(z) polyval(a(end:-1:1),z)...          % function handle for r
       ./polyval(b(end:-1:1),z);
if nargout>5                                % only compute poles if
 necessary
  poles = roots(b(end:-1:1));               % poles
  t = max(tol,1e-7);                        % perturbation for
 residue estimate
  residues = t*(r(poles+t)-r(poles-t))/2;   % estimate of residues
end
```

```
Not enough input arguments.

Error in ratdisk (line 11)
if nargin<4, if isfloat(f), N=length(f)-1;
```

Published with MATLAB® R2020a