

Winter 2018

Classification of Arbitrary Motion into a Canonical Basis

Michael Moger

University of New Hampshire, Durham

Follow this and additional works at: <https://scholars.unh.edu/thesis>

Recommended Citation

Moger, Michael, "Classification of Arbitrary Motion into a Canonical Basis" (2018). *Master's Theses and Capstones*. 1258.
<https://scholars.unh.edu/thesis/1258>

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.

CLASSIFICATION OF ARBITRARY MOTION INTO A CANONICAL BASIS

BY

MICHAEL MOGER

Bachelor of Science, University of New Hampshire, 2015

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

in

Applied Mathematics

December 2018

This thesis has been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics by:

Kevin M. Short, Professor in Mathematics
Thesis Director

Mark E. Lyon, Associate Professor in Mathematics
Co-Director of Integrated Applied Mathematics Program

Rita A. Hirschweiler, Professor in Mathematics
Graduate Student Coordinator

on August 17, 2018

Approval signatures are on file with the University of New Hampshire Graduate School.

Acknowledgements

I would like to take this time to personally thank anyone and everyone who has helped me with this thesis over the past three years, including (but not limited to): Laszlo for introducing me to the project and transitioning me to his former position. Barry, Brian, and everyone at Lamprey Networks, Inc for the opportunity to conduct research over the summer of 2016. Professor Short for aiding me in more ways than I can express. Professor Lyon for stepping in as a committee member and the constructive feedback. The entire Math and Stats department at UNH for all of the support, teaching, and guidance during my tenure at this fine university.

Thank you, sincerely.

Contents

Committee Members	i
Acknowledgements	ii
Table of Contents	iv
List of Tables	v
List of Figures	viii
Code Listings	ix
Abstract	x
Introduction	1
0 Preliminaries	3
0.1 Empatica E4 Overview	3
0.2 Data Acquisition Method	4
1 Data Processing Techniques	7
1.1 Window Functions	7
1.1.1 Different Window Types	8
1.1.2 Convolution and Resampling	9

1.2	Spectral Methods	12
1.2.1	Fourier	12
1.2.2	Wavelets	14
1.3	Algorithms and the ML Connection	15
1.3.1	Savitzky-Golay	15
1.3.2	Singular Value Decomposition	16
1.3.3	Matching Pursuit	19
2	Applying Dynamic Data Analysis on E4 Data	23
2.1	Pre-conditioned Data	23
2.1.1	Smoothing and Filtering	23
2.1.2	Neighborhood Search	26
2.1.3	Applying SVD to each motion	29
2.1.4	Creating a Dictionary	33
2.2	Supervised Learning	35
2.2.1	Classification of a Known Input	37
3	How to Decompose Arbitrary Motions Into a Canonical Basis	41
3.1	Semi-Supervised Learning	41
3.1.1	Data Exploration	42
3.1.2	Classification of a Semi-Random Input	44
3.2	Unsupervised Learning	47
3.2.1	Data Exploration	48
3.2.2	Classification of a Random Input	51
3.3	Limiting Factors	54
3.4	Summary and Conclusion	58
	References	60

List of Tables

1	Index for collected samples, initial	5
2.1	Index for collected samples, final	34
2.2	MP metrics for supervised rolling data	39
3.1	MP metrics for semi-supervised falling data	46
3.2	MP metrics for unsupervised random data	52
3.3	MP metrics for unsupervised random data (default dictionary)	56
3.4	Summary of conditions for MP alg on each learning method	58
3.5	Summary of results for MP alg on each learning method	59

List of Figures

1	Sleep sample accelerometer reading	6
2	Sleep sample heart rate reading	6
3	Sleep sample EDA reading	6
4	Sleep sample temperature reading	6
1.1	Visualization of each window function for $n=25$	10
1.2	Application of convolution with input signal	11
1.3	2x2 example of SVD decomposition of a real, square matrix	17
2.1	Smoothing an accelerometer data signal with a Gaussian window function . .	24
2.2	Smoothing an accelerometer signal with Savitzky-Golay filtering	25
2.3	Close-up of smoothing with x-axis only	26
2.4	Geometry of the problem	27
2.5	Spin SVD using a centroid of points, step 1	29
2.6	Spin SVD using a centroid of points, step 2	29
2.7	Spin SVD using a centroid of points, step 3	30
2.8	Spin SVD using a centroid of points, step 4	30
2.9	Roll SVD using a centroid of points, step 1	30
2.10	Roll SVD using a centroid of points, step 2	30
2.11	Roll SVD using a centroid of points, step 3	31
2.12	Roll SVD using a centroid of points, step 4	31

2.13	Flip SVD using a centroid of points, step 1	31
2.14	Flip SVD using a centroid of points, step 2	31
2.15	Flip SVD using a centroid of points, step 3	31
2.16	Flip SVD using a centroid of points, step 4	31
2.17	Up-down SVD using a centroid of points, step 1	32
2.18	Up-down SVD using a centroid of points, step 4	32
2.19	Back-forth SVD using a centroid of points, step 1	33
2.20	Back-forth SVD using a centroid of points, step 4	33
2.21	Left-right SVD using a centroid of points, step 1	33
2.22	Left-right SVD using a centroid of points, step 4	33
2.23	Plot of SVD-generated dictionary for Matching Pursuit	34
2.24	Supervised sample for MP	35
2.25	MP results for normalized rolling signal	37
2.26	Normalized rolling signal residual plot	40
3.1	Pseudo-random sample accelerometer reading (falling)	42
3.2	Savitsky-Golay filtering of falling data	43
3.3	Falling SVD using a centroid of points, step 1	43
3.4	Falling SVD using a centroid of points, step 4	43
3.5	Different angle of step 4 above	44
3.6	MP results for normalized falling signal	45
3.7	Normalized falling signal residual plot	47
3.8	Random input signal	48
3.9	Random input signal, filtered	49
3.10	Random SVD using a centroid of points, step 1	50
3.11	Random SVD using a centroid of points, step 4	50
3.12	Different angle of step 4 above	50
3.13	MP results for normalized random signal	51

3.14	Normalized random signal residual plot	53
3.15	MP results for normalized random signal using all atoms of our custom dic- tionary	54
3.16	MP results for normalized random signal with default dictionary	55
3.17	Normalized random signal residual plot with default dictionary	57

Code Listings

1.1	Convolution Matlab function	9
1.2	Resample Matlab function	11
1.3	Savitzky-Golay Matlab function	16
1.4	SVD Matlab function	17
1.5	MP Dictionary Matlab function	19
1.6	Matching Pursuit Matlab function	21
2.1	Gaussian window smoothing code	24
2.2	Savitzky-Golay filtering	24
2.3	k-NN Search Matlab function	26
2.4	Variables for 3-D SVD options	28
2.5	Local 3-D SVD using the centroid	28
2.6	Variables for 2-D SVD options	32
2.7	Matching Pursuit code	36
2.8	Plotting residuals	38

ABSTRACT

CLASSIFICATION OF ARBITRARY MOTION INTO A CANONICAL BASIS

by

Michael Moger

University of New Hampshire

December 2018

The Empatica E4 wristwatch utilizes four sensors to capture medical data from its user - an accelerometer, a plethysmograph, an electro-dermal activity sensor, and an infrared thermophile. Utilizing these sensors, the device can provide detection-based feedback for patients suffering from various ailments. However, each sensor is coupled with the other readings, so any raw data will have a degree of noise accompanying the actual signal. After detailing a conceptual and programming knowledge of various industry-standard data processing techniques, we follow the appropriate steps to take in order to clean up a noisy E4 data signal, starting with supervised basis signals and ending with unsupervised, random samples. We conclude with a discussion of how one can decompose arbitrary motions into a canonical basis for proper data analysis, providing insight based on our results.

Introduction

Reducing motion artifacts in medical data is necessary to clean up an otherwise noisy signal so doctors and patients can correctly analyze and interpret their results. In the development of the Empatica E4 wristwatch, it has been found that the combination of an accelerometer (ACC), a plethysmograph (PPG), an electro-dermal activity (EDA) sensor, and an infrared thermophile (TEMP) is very effective at collecting and analyzing medical data for uses such as seizure and fall detection.

Chapter 0 is broken up into two main sections. In section 0.1, we consider the power of the E4 by outlining the capabilities and drawbacks of the device, followed by suggestions on how to fix some of its major issues. We will also highlight the current research being performed with similar devices as a means for comparison and motivation. Since the method of data acquisition is crucial to understanding our approach of the problem, section 0.2 lays out exactly that. Each of the three Euclidean dimensions is isolated, as well as combinations of two directions (flip, spin, or roll), into a control set of data.

Chapter 1 serves two purposes; the first of which is to provide researchers with a list/proxy for the industry-standard data processing techniques, and the second is to demonstrate the computation necessary to apply these methods. In it, we will explore windowing (such as Gaussian or Kaiser functions), spectral methods (such as the FFT or DWT), dynamical systems theory (such as k-NN or SVD), and various filtering algorithms (such as Savitzky-Golay or Matching Pursuit) in extensive conceptual and implementary detail.

Chapter 2 is where we apply the aforementioned techniques on data collected from the

E4, starting with the control set. As an added measure, the control set was constructed from the best data collection that was discussed in Chapter 0. From smoothing - including trying out different windows/pre-processing - to filtering, the results from control data serve as a benchmark for an ideal analysis.

Chapter 3 is posed, primarily, as a test to replicate the entire process in Chapter 2 (same variables and same filtering coefficients, adjusted for data size) on randomly collected data. We investigate varying levels of randomness, finishing with a discussion on some of the limitations of the proposed solution.

Chapter 0

Preliminaries

This chapter will serve as a brief but complete description of the device that was used for data collection. Please note that the Empatica E4 is a research device that is constantly being updated, so some of this information may change in future revisions.

0.1 Empatica E4 Overview

The E4 is a wrist-worn medical device from Empatica that offers real time computerized biofeedback thanks to its four powerful sensors [1]. The real productivity in the sensors come from the E4's built-in feature set: combining off-the-shelf components with proprietary design and options for storage and analysis of end user data. As opposed to older methods of physiological data collection, the E4 has all of its sensors integrated into the band, a design choice that benefits from being both seamless and inconspicuous. Users (researchers, patients, and doctors alike) have the option of either transmitting their data in real time via a Bluetooth interface or storing their sessions in the device's internal flash memory. Regardless of data acquisition, user data can be viewed online (or on mobile) with zoom and compare options. For researchers and developers, the Empatica API allows for further desktop and mobile integration of the E4 into their respective infrastructures [1].

Boasting a form-fitting design (weighing in at 25g, 110-190mm wrist measurement, with

dimensions: 44x40x16mm) with impressive specifications (20-36 hr battery life, 60+ hrs of flash memory, splash resistance, and an official medical device certification) [1], the E4 defines what it means to be a powerful and easy-to-use health monitoring device. A long press of the devices' lone button serves as the on/off switch, while a short press marks a certain timestamp in the data with a tag. These tags are ideally useful for determining changes in activity, such as when used to indicate periods of sleep, exercise, or leisure.

Despite all of the power inside of the E4, there are certain flaws in the design. The PPG sensor notably uses a proprietary artifact removal technique based on a combination of wavelengths (instead of the common technique of exploiting a single wavelength with ACC data), meaning that the signal users see is already pre-filtered and split into different components of data that a PPG normally gives. On the flip side, the 3-axis accelerometer signal arrives unfiltered, which is perplexing since a single sensor is simply not able to accurately track the motion of a rigid body [3, 5]. Their Embrace watch, marketed specifically as a device to aid seizure-prone individuals, includes a gyroscope as a secondary motion-tracker (angular velocity). However, since I performed all of my testing on the E4, this paper will highlight the process of filtering out excess noise from the ACC signal, and to reinforce/introduce the prevailing theme, finding the best possible way to clean up a noisy, coupled system [3-6].

0.2 Data Acquisition Method

Due to variable coupling, motion bias was always a huge threat to data collection. For this reason, initial testing of the device was done during sleep to mitigate motion bias. Periods of little-to-no movement permitted access to some of the more interesting aspects of EDA data, specifically when researching a phenomena known as "sleep storms". Tags were recorded (to the best of my ability) whenever I awoke or was stirring in bed, unable to fall asleep. From these collections, motion certainly did not seem to be an issue.

The next step in the data acquisition process was to consider a user who would want to track their data throughout the day which requires a much more efficient knowledge of the dynamical system present in the E4. To achieve this, we would need an extensive amount of sample data in a Euclidean coordinate system. The goal was to create a library of known samples and use these to test out algorithms to properly filter out the noise. Since we have an idea of what the signal should look like - based on the x, y, and z directions in which the motion was recorded - this makes the process of locating an efficient basis easier. This eventually lead to the decision to record six (6) types of data: roll (xy), flip (yz), spin (xy), back/forth (x), left/right (y), and up/down (z), with more focus given to actions involving two directions.

As with most data collection sprees, the initial batch was poor. The problem was the method; not only was it non-fluid motion but I was always directly touching the watch, causing some latency issues. After some thought, I came upon an arcane solution thanks to the help of three household objects: a ruler, a drumstick, and some duct tape. I taped the wrist-side of the watch to the ruler and then liberally applied tape to the drumstick until there was enough where the device would fit securely through. After choosing the principle axes, the data came flowing in.

index	signal
1-3	bf
4-6	flip
7-9	lr
10-12	roll
13-15	spin
16-18	ud

Table 1: Index for collected samples, initial

Let's take a look at some sample data, in particular a sleep cycle I created early on in the project, lasting about 7 hours. Using the Empatica web-app, which provides storage and some initial data cleansing up front, we can view the data. As touched on briefly, the PPG signal is converted into three separate readings: blood volume pulse, heart rate, and

inter-beat interval. Since this is sample taken while asleep, the accelerometer is notably stagnant, but what is remarkable about the sample is the EDA spike about an hour before awakening, known as a "sleep storm." Should a similar spike be combined with an oscillating of falling ACC level, this would be indicative of seizure [2].

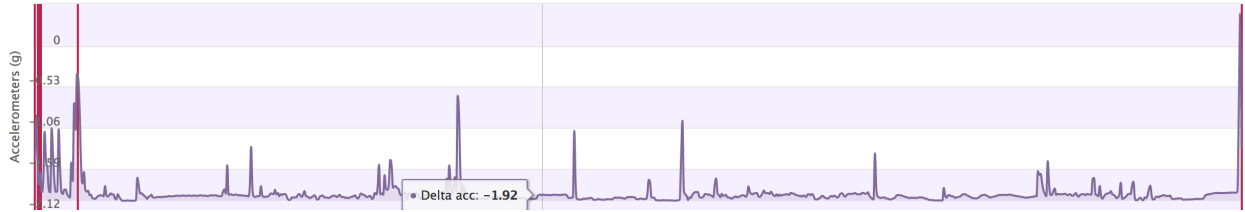


Figure 1: Sleep sample accelerometer reading

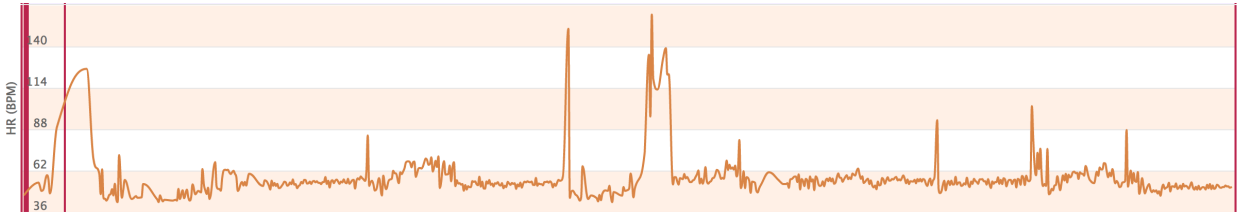


Figure 2: Sleep sample heart rate reading

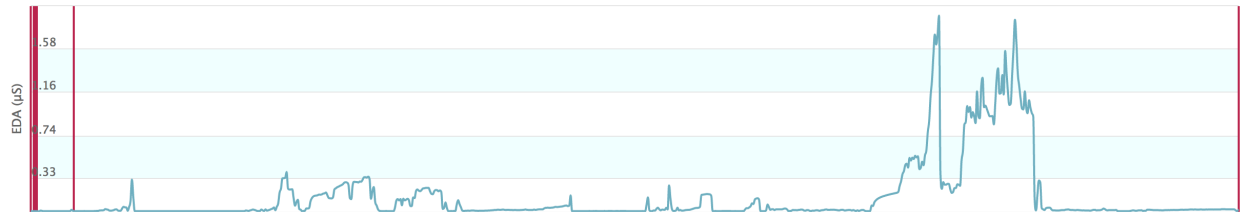


Figure 3: Sleep sample EDA reading

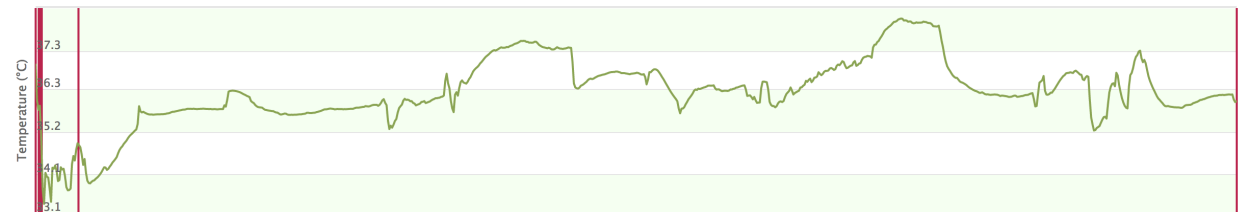


Figure 4: Sleep sample temperature reading

Chapter 1

Data Processing Techniques

The first step of any analysis with data is clean-up. In fact, data pre-processing can take up to 80% of a project's timeline (which certainly proves true in practice). This is a process that involves a lot of trial-and-error, some luck, and a keen eye for discrepancies in the data. Data collected from the E4 could be categorized as either signal processing or time series analysis, and since this paper will not be going into detail about the forecasting techniques of the latter, we'll focus on the former. Signal processing is a common tool used for continuous or discrete data that can come from many sources, such as the medical data we'll be investigating soon, or sound waves found in audio production [7]. The following sections will serve as a guide, in both theory and computation, to standard practices when working with different types of data.

1.1 Window Functions

By definition, a window function is a function that is zero-valued outside of a chosen interval. These special functions are important because when convoluted with a given signal, the overlapping result is a view of the data "through the window." This idea has been applied to many different disciplines, including spectral analysis, bandwidth, and filter design. Each window has different properties, so it is crucial to understand the intricacies of each type

before attempting to apply them to data. In all of the functions, let N represent the width (in samples) of a discrete-time window function w , then:

$$w[n], \quad 0 \leq n \leq N - 1$$

For ease of comprehension, our study will focus on three main categories of window functions, starting with polynomial ones.

1.1.1 Different Window Types

Polynomial windows are functions that originate from B -spline forms, of which are manipulations of the rectangular window, defined as $w(n) = 1$. The rectangular window, while the simplest, is useful in zeroing out all but N values of a signal (i.e. a boolean process) and serves as the basis for understanding all of the other functions.

If we combine two $\frac{N}{2}$ -width rectangular windows, we would arrive at the triangular window, defined as:

$$w(n) = 1 - \left| \frac{n - \frac{N-1}{2}}{\frac{L}{2}} \right|, \quad \text{for } L = N, N + 1, \text{ or } N - 1.$$

Another polynomial window to explore is the Welch function, of which has a single parabolic section, and is defined as:

$$w(n) = 1 - \left(\frac{n - \frac{N-1}{2}}{\frac{N-1}{2}} \right)^2.$$

Cosine window functions have the benefit of already being common signals, as the the general cosine, or Hamming, window is defined as:

$$w(n) = \alpha - \beta \cos \left(\frac{2\pi n}{N-1} \right), \text{ where } \alpha \text{ and } \beta \text{ are real coefficients between 0 and 1.}$$

American mathematician Richard W. Hamming is credited in the naming due to his particular choice for $\alpha = 0.54$ and $\beta = 1 - \alpha = 0.46$, of which minimizes the nearest side lobe of the signal. Its counterpart, the Hann window, is similar in construction, but instead chooses $\alpha = \beta = 0.5$, giving the following definition:

$$w(n) = 0.5 \left(1 - \cos \left(\frac{2\pi n}{N-1} \right) \right) = \text{hav} \left(\frac{2\pi n}{N-1} \right), \text{ where } \text{hav} \text{ is the haversine function.}$$

Higher-order cosine windows are a series representation of the generalized Hamming window.

$$w(n) = \sum_{k=0}^K \cos \left(\frac{2\pi kn}{N} \right).$$

Since $w(n)$ of this form will only have $2k + 1$ non-zero discrete Fourier transform (DFT) components, these windows are ideal for a scenario requiring windowing by convolution in the frequency domain [REF]. Blackman, Nuttall, Flat Top, and Rife-Vincent (plus other hybrid types) windows are all variants of this type, with the key difference being the coefficient values and number of terms in the series. Again, now we will perform some visualization routines with the cosine windows.

Even better than cosine windows are ones that you can adjust. These window functions form a range of options of smoothing in the frequency domain, all starting with the most common one, the Gaussian (or Normal distribution) window:

$$w(n) = e^{-\frac{1}{2} \left(\frac{n-(N-1)/2}{\sigma(N-1)/2} \right)^2}, \quad \sigma \leq 0.5.$$

1.1.2 Convolution and Resampling

In all of the above examples, we use one unified approach in application: convolution. A convolution of two vectors, u and v , represents the area of overlap under the points as v slides across u . This process is equivalent to multiplying polynomials whose coefficients are the elements of u and v .

```

1 function w = conv(u,v,shape)
2     % Inputs:
3     %   u, v - Vectors to be convoluted.
4     %   shape - Subsection, specified as full, same, or valid.
5     %
6     % Outputs:
7     %   w - Output of the convolution of u and v.

```

Code Listing 1.1: Convolution Matlab function

Let's plot a number of different window functions of length 25:

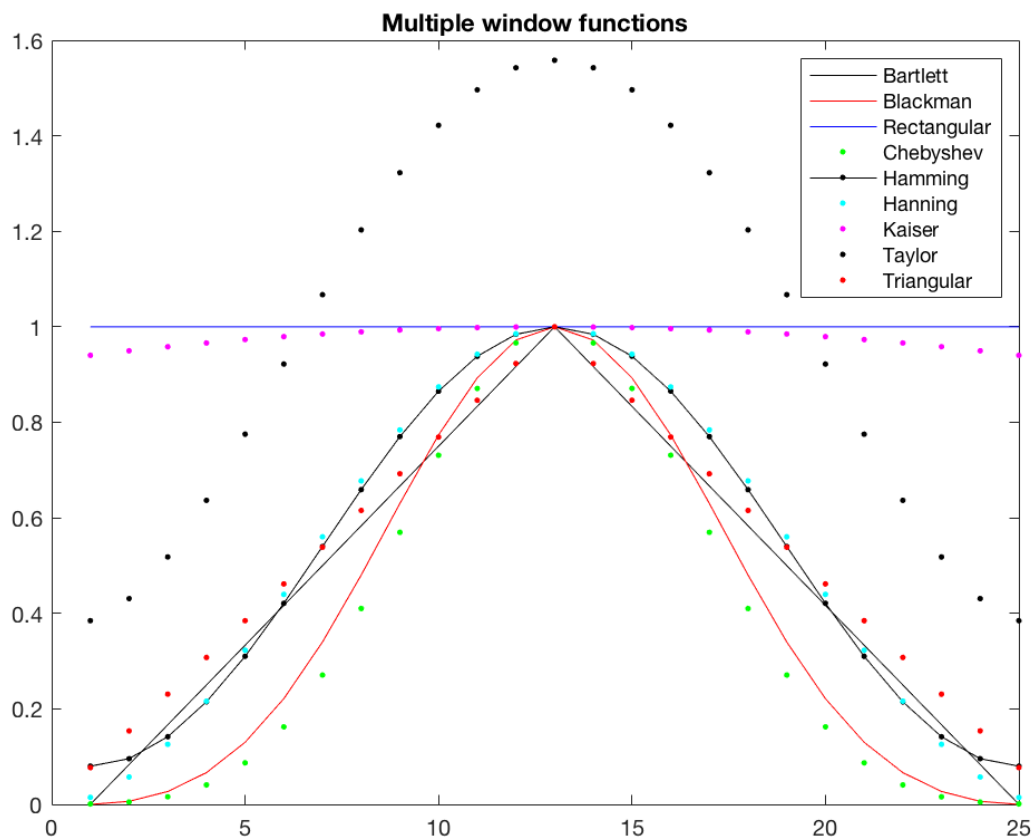


Figure 1.1: Visualization of each window function for $n=25$

And now we can apply these to a random input signal via convolution. This acts as a smoother or de-noiser for the accelerometer peaks.

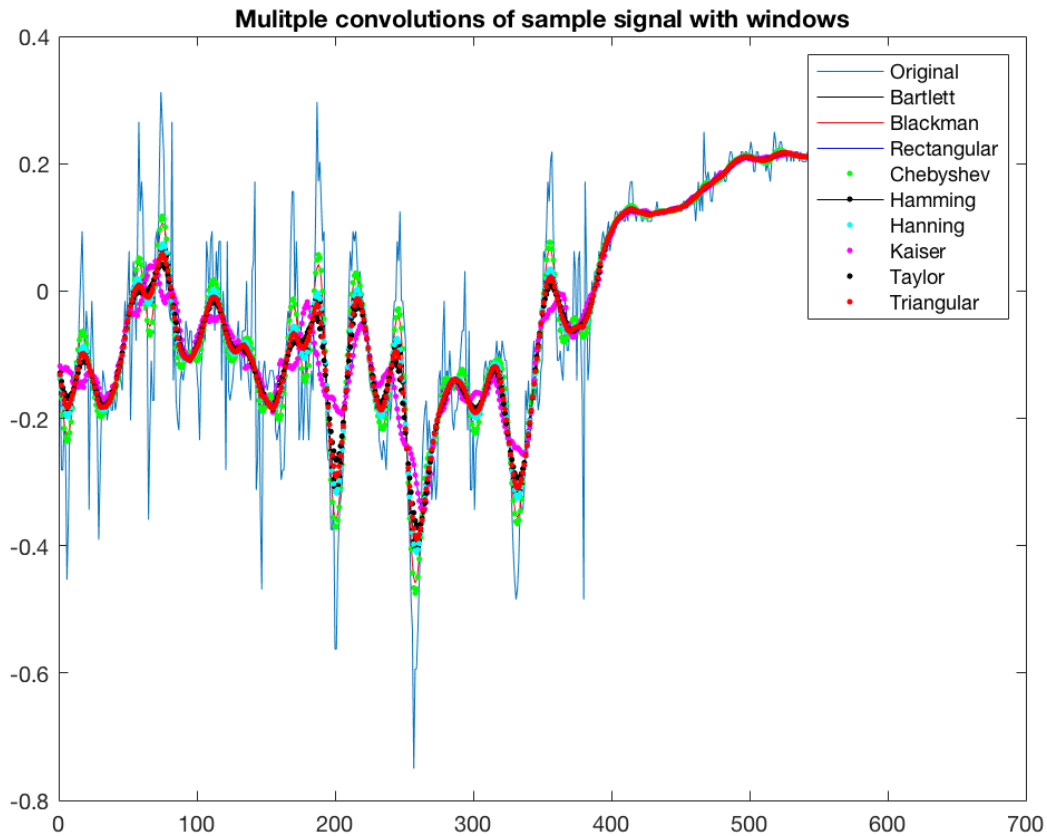


Figure 1.2: Application of convolution with input signal

For each polynomial multiplication or convolution, the length of the input signal is increased by $n - 1$ where n is the window size. This is important to note, as it slightly shifts our signal view of focus. We can mitigate any latency issues by resampling the signal into the original window size. Details about resampling can be found with Matlab's `resample`, as listed below:

```

1 function y = resample(x,p,q)
2     % Inputs:
3     %     x — Input signal, either vector or matrix.
4     %     p, q — Resampling factors. Positive ints.
5     %
6     % Options:

```

```

7      %      n – Neighborhood term number for nn interpolation.
8      %      Length of filter is proportional for n.
9      %      beta – Shape parameter of Kaiser window used to design the filter.
10     %      b – FIR filter coefficients. Has odd length and linear phase.
11     %      method – Interpolation method. Default is linear; options include
12     %      pchip and spline.
13     %
14     %      Outputs:
15     %      y – Resampled signal. If x has length N, y has length N * (p/q).

```

Code Listing 1.2: Resample Matlab function

1.2 Spectral Methods

Time-frequency data, commonly found in the medical world, is simplest when there is a periodicity present. However, that seldom occurs in the entire spectra, so we attempt to find areas that are more-easily explained via frequency inspection. The general approach to analyzing this type of data is with spectral methods, two of which we will explore in depth: the classical Fourier series and the more modern wavelet.

1.2.1 Fourier

If you’ve taken one course in applied mathematics or linear algebra, there’s no doubt that a very popular (and powerful) concept was introduced: the Fourier series and it’s associated transform. An integral transform that dates back to the 1820’s, the idea is to take a time-dependent function and express it in terms of it’s frequency spectrum. Here’s the definition plus the inverse transform:

$$\hat{f}(z) = \int_{-\infty}^{\infty} f(t)e^{-2\pi izt} dt \implies f(t) = \int_{-\infty}^{\infty} \hat{f}(z)e^{2\pi itz} dz,$$

where $z \in \mathbb{R}$ denotes frequency and t denotes time for an integrable function $f : \mathbb{R} \mapsto \mathbb{C}$.

There is a limiting case with this however, and it deals with Heisenberg's uncertainty principle. Notably, one cannot have absolute precision on both frequency and temporal domains. Classical Fourier transform allows for a measurement with zero bandwidth that equates with an exact size for frequency, but at the cost of not knowing when that particular frequency occurred. In essence, it's a trade-off between time and frequency. A specific case of this comes up later, in our discussion of the spectrogram and using the short-time Fourier transform (STFT) to generate it.

Even though data collection in continuous in nature, discrete means are necessary to properly compute Fourier transforms with Matlab or some other program. Luckily, the FT is easily made discrete by the following definition:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n / N}, \quad k \in \mathbb{Z}.$$

The usual domain is $[0, N - 1]$ due to periodicity, but others include $[-\frac{N}{2}, \frac{N}{2} - 1]$ for N even or $[-\frac{N-1}{2}, \frac{N-1}{2}]$ for N odd. However, a traditional discrete Fourier transform (DFT) is a monster of a calculation - $\mathcal{O}(n^2)$, so the best way to compute a FT is with the famous fast Fourier transform (FFT). Originally invented by Gauss in the early 1800's, then later rediscovered and developed by James Cooley and John Tukey in 1965, the FFT splits a DFT calculation into 2 (or more) problems. Variants of the algorithm include ones by Bluestein and Rader, both of which use a convolution-based approach. As part of the FFTPACK, the FFT algorithm has been precisely tuned and optimized to $\mathcal{O}(n \log(n))$ calculation time, where n is the sample size. This huge computational improvement has signified the importance of the algorithm.

1.2.2 Wavelets

For decades, Fourier analysis has dominated the applied mathematics world, and for good reason too. It's a stones-throw away from simple series that evolves into a method that supplies an overload of computational might for such a straightforward theory. However, what happens if we want a more precise time-domain measurement? Fourier, again, is limited by it's choice of frequency precision. Enter wavelets, which take advantage of the uncertainty principle's intermediary clause. A wavelet transform allows for a rough estimate of both time and frequency domains, simultaneously. Due to this advantage, they have steadily grown in popularity as a method for signal processing (especially for image compression and filter design) since their discovery by Haar in 1909.

$\psi(t)$ denotes the "Mother wavelet" such that $\psi(t) \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$, ie. $\psi(t)$ is absolutely and square integrable:

$$\int_{-\infty}^{\infty} |\psi(t)| dt < \infty, \int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty.$$

Being in this space guarantees zero mean and squared norm of one for the Mother wavelet. For a wavelet to be usable in either a continuous or discrete transform, a scaling factor is necessary. Let $(a, b) \in \mathbb{R}$, then

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right).$$

It is also possible to define a wavelet in terms of a "Father wavelet," usually denoted as the scaling function $\phi(t)$.

Like window functions, there are multiple types of wavelets that can be used for analysis, the first of which is the Haar wavelet. Here's the definition:

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

with scaling function

$$\phi(t) = \begin{cases} 1, & 0 \leq t < 1 \\ 0, & \text{elsewhere.} \end{cases}$$

This particular function, while being the simplest and also discontinuous (hence non-differentiable), is historically important as being the first wavelet - though Haar did not refer to it as such. Cases that utilize the Haar wavelet include signals with sudden transitions or binary outputs.

Despite the additional benefits of having dual-precision on both the frequency and time domains, wavelets were not used in any low-level method aside from matching pursuit (which utilizes wavelet-packet and sinusoidal bases to represent any input signal based on a redundant dictionary). Instead, our matching pursuit implementation will be using a custom dictionary built by collecting samples on the E4. We will then compare our results using custom dictionary and the default dictionary, testing for accuracy and ease of understanding.

1.3 Algorithms and the ML Connection

Alongside window functions, we will be investigating some data analysis algorithms to aid us in cleansing, filtering, and finding the best possible basis for any input. In addition, we'll start to make the connection to machine-learning techniques from the world of Data Science, as this paper's main goal is quite similar to problems found in that realm.

1.3.1 Savitzky-Golay

By definition, the Savitzky-Golay filter for one-dimensional data (x_j, y_j) where $j = 1, \dots, n$ is:

$$Y_j = \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} C_i y_{j+i} \quad \text{for} \quad \frac{m-1}{2} \leq j \leq n - \frac{m-1}{2},$$

where C_i are the set of m convolution coefficients. Essentially, SG is a digital filter that can be applied to smooth a set of data (ie. increasing the signal-to-noise ratio without reaching distortion). To do so, this process utilizes convolution, or fitting of adjacent data points with

polynomials in a least-squares method. If the data are uniformly spaced, then the least-squares equations can be solved analytically with a single set of convolution coefficients, C_j . In turn, these coefficients provide the best estimate of a smoothed signal from the original data [8].

Quick Matlab snippet outlining the built-in `sgolayfilt` function:

```

1 function y = sgolayfilt(x,order,framelen,weights,dim)
2     % Inputs:
3     %     x - Signal to be processed.
4     %     order - Polynomial order. Must be less than the frame length.
5     %     framelen - Frame length. Must be odd.
6     %     weights - Weighting vector of real, positive weights used during
7     %         least squares.
8     %     dim - Dimension along which the filter operates. If dim is not
9     %         specified, the filter operates on dim 1 for column vectors,
10    %         and dim 2 for row vectors.
11    %
12    % Outputs:
13    %     y - Filtered data object.

```

Code Listing 1.3: Savitzky-Golay Matlab function

Savitzky-Golay was chosen out of others in the family of low-pass filters due to its particular property of maintaining more high-frequency components of a given signal [8], which is something we desire here given that accelerometers have defining peaks we'd like to keep intact. Details outlining a comparison between window smoothing and Savitzky-Golay filtering will be provided in Chapter 2. Overall, the two are quite similar, but give separate edge-defining characteristics.

1.3.2 Singular Value Decomposition

To continue with the analysis, after smoothing comes finding the best possible coordinate system for the data. Once discovered, these coordinates give an accurate picture of the

motion the watch (and presumably, the user) is undergoing. In order to provide a worthwhile measurement for future results, it is crucial to find such a basis. The best way to do so is with the singular value decomposition (SVD). This transform takes a matrix $M \in \mathbb{R}_{m,n}$ and decomposes it into two unitary matrices, U and V , plus a square matrix, Σ , containing the singular values of M . Pictorially, this looks like the following:

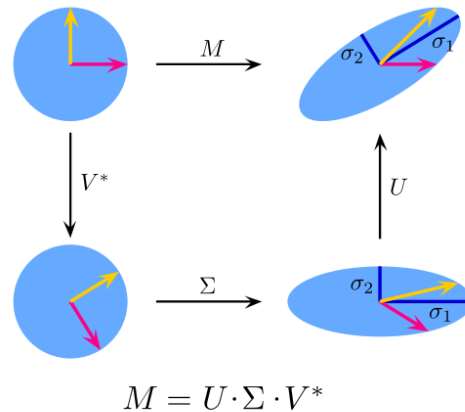


Figure 1.3: 2x2 example of SVD decomposition of a real, square matrix

The basic idea behind the singular value decomposition is a coordinate transform along the primary basis functions hidden in the raw data. We start in a standard Euclidean space with unit vectors $e_1 = (1, 0)$ and $e_2 = (0, 1)$ for \mathbb{R}^2 . Decomposition of a matrix M from (e_1, e_2) to (σ_1, σ_2) entails an initial rotation, scaling along the coordinate axes, and a final rotation [11].

```

1 function [U,S,V] = svd(A)
2     % Inputs:
3     %     A - Matrix to be decomposed. Must be square or rectangular.
4     %
5     % Choice-Value:
6     %     'econ' - Apply an economy-size decomposition.
7     %     if m > n, only the first n cols of U are computed, S is nxn
8     %     if m = n, svd(A,'econ') == svd(A)
9     %     if m < n, only the first n cols of V are computed, S is mxm

```

```

10    %    0 - Apply a slightly different economy-sized decomp.
11    %    if m > n, svd(A,0) == svd(A,'econ')
12    %    if m <= n, svd(A,0) == svd(A)
13    %
14    % Outputs:
15    %    U - Left singular vectors in the form of a mxn unitary matrix.
16    %    Cols of U corresponding to non-zero singular values form a
17    %    set of orthonormal basis vectors for range(A).
18    %    S - Square matrix with singular values on the diagonal.
19    %    V - Right singular vectors in the form of a nxn unitary matrix.
20    %    Cols of V which do not correspond to non-zero singular values
21    %    form a set of orthonormal basis vectors for null(A).

```

Code Listing 1.4: SVD Matlab function

For a square, symmetric, and positive definite matrix M , $\text{eig}(M) = \text{svd}(M)$, where $\text{eig}(M)$ is the Eigen-decomposition ($Mv = \lambda v$) for M . The difference between these decompositions is in the mapping: Eigen goes from a vector space to itself and SVD goes from a vector space to another vector space, generally one with a different dimensionality. Given our problem space - a set of non-orthogonal, non-linear dynamics that approximate human motion - we benefit greatly from SVD to be able to map collected samples into a localized coordinate system for analysis.

For completeness, here is the $m \times n$ singular value decomposition for a matrix A [11].

$$A = U \Sigma V^T$$

$$= \left(\begin{array}{c|c|c|c} \begin{array}{c} u_1 \\ \vdots \\ u_r \end{array} & \begin{array}{c} u_{r+1} \\ \vdots \\ u_m \end{array} & \begin{array}{c} \vdots \\ \vdots \end{array} & \begin{array}{c} \vdots \\ \vdots \end{array} \\ \hline \text{col}(A) & \text{null}(A) & & \end{array} \right) \left(\begin{array}{ccc} \sigma_1 & & 0 \\ & \ddots & \\ & & \sigma_r \\ & & & 0 \\ & & & & \ddots \\ 0 & & & & & 0 \end{array} \right) \left(\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \right) \begin{array}{l} v_1^T \\ v_r^T \\ v_{r+1}^T \\ v_n^T \end{array} \left. \begin{array}{l} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \right\} \begin{array}{l} \text{row}(A) \\ \text{null}(A) \end{array}$$

1.3.3 Matching Pursuit

Matching pursuit (or MP for short) is a sparse-approximation algorithm for classification of a signal via an input dictionary. The most useful implementation of such a method is for approximating the motions of a non-linear dynamical system in terms of a known, usually redundant, basis. This easily analogizes to the very active field of data science and machine-learning algorithms for predictive analytics in the form of a "supervised learning" exercise [9, 10]. Instead of a signal as the function to-be-determined, we have some matrix containing features and their values attempting to shed insight on a response, using different functions and logic to calculate a prediction.

For MP of a function $f(t)$ in a Hilbert space H and dictionary D with N components, the setup is as follows:

$$f(t) \approx \hat{f}_N(t) = \sum_{n=1}^N a_n \phi_n(t),$$

where a_n is the amplitude of each component $\phi_n \in D$. To start out, we need to construct the dictionary for which we compare an input signal. For this, we'll use wmpdictionary through Matlab's Wavelet toolbox [7].

```
1 function mpdict = wmpdictionary(N, 'Name', 'Value')
```

```

2      % Inputs:
3      %      N - Length of input signal.
4      %
5      % Name-Value pairs:
6      %      'lstcpt' - A cell array of cell arrays with valid sub-dictionaries.
7      %      Valid ones include: {'wavelet family', N}, 'dct', 'sin', 'cos', '
      %      poly',
8      %      'RnIdent' (shifted Kronecker delta). Default is {'sym4',5} ,
9      %      {'wpsym4',5}, 'dct', 'sin'.
10     %      'addbeg' - Prepended sub-dictionary (MxN matrix).
11     %      'addend' - Appended sub-dictionary (MxN matrix).
12     %
13     % Outputs:
14     %      mpdict - Dictionary signal for sparse approximation.

```

Code Listing 1.5: MP Dictionary Matlab function

For simplified cases, the dictionary chosen replicates the signal you're trying to approximate, ie a smooth continuous input would be represented best with a Fourier basis, while one would use a Wavelet basis for smooth input with isolated discontinuities. However, with real-world data, it can be sparsely represented by any basis, so we would want to construct a dictionary using vectors which span different bases. This leads to a set of atoms which are not linearly independent, so the MP solution of an input is not unique - there may be other combinations of dictionary atoms which sparsely represent the signal. To avoid this issue, we introduce the idea of redundancy (dictionary atoms form a linearly independent set) such that x can be expanded by a set of atom vectors that adapt to the time-frequency or time-scale characteristics of x . This restriction, along with the notion of completeness (dictionary atoms span the entire signal space) allow us to utilize MP effectively. The challenge is how to choose the optimal N -term expansion of x in a given dictionary. With this in mind, let's consider the base matching pursuit algorithm.

Let $\phi = \{\phi_k\}$ be a dictionary of unit-norm atoms and f an input signal. Define $R^0 f = f$

as the starting residual. Select the atom from ϕ such that the inner product is maximized. Denote that atom as ϕ_p . Update the residual by subtracting the orthogonal projection of $R^0 f$ onto the space spanned by ϕ_p : $R^1 f = R^0 f - \langle R^0 f, \phi_p \rangle \phi_p$. Iterate for each atom, as such:

$$R^{n+1} f = R^n f - \langle R^n f, \phi_k \rangle \phi_k$$

and stop once the norm of the residual and f converge to 0 [7, 8].

Here is the base algorithm using the Wavelet toolbox's `wmpalg`:

```

1 function [yfit,R,coeff,iopt,qual] = wmpalg('mpalg',Y,mpdict,'Name','Value')
2     % Inputs:
3     % 'mpalg' - Algorithm type (basic, orthogonal, or weak).
4     % Y - Signal for matching pursuit (vector).
5     % mpdict - MP dictionary, constructed via wmpdictionary.
6     %
7     % Name-Value pairs:
8     % 'itermax' - Integer fixing the max number of iterations.
9     %     Default is 25.
10    % 'maxerr' - Cell array with the norm and max relative error (%).
11    %     Norms are 'L1', 'L2', or 'Linf'. Rel err is 100 * ||R|| / ||Y||.
12    % 'typeplot' - Type of plot to produce. Options are 'none', 'one',
13    %     'movie', or 'stepwise'
14    % 'stepplot' - Number of iterations between plots (for movie or step).
15    %
16    % Outputs:
17    % yfit - Adaptive greedy approximation of Y
18    % R - Residual after MP terminates.
19    % coeff - Expansion coefficients in mpdict. Dictionary atoms
20    %     weighted by coeff yield yfit.
21    % iopt - Column indices of the selected mpdict atoms.
22    % qual - Proportion of retained signal energy for each iteration
23    %     in MP. q_k = ||a_k||^2 / ||Y||^2, where a_k is the vector of

```

Code Listing 1.6: Matching Pursuit Matlab function

Utilization of MP as a classification method requires a precise definition of the type of algorithm, of which there are three to choose from [7]. 1) Basic: dictionary atoms are not mutually orthogonal vectors, so subtracting subsequent residuals from the priors can mix non-orthogonal components to the span. 2) Orthogonal: the residual is always orthogonal to the span of the atoms already selected. Therefore, convergence is guaranteed in d steps equal to the dimensionality of the signal. 3) Weak orthogonal: for computational efficiency, it uses a weaker maximization criteria for the inner product, such that $|\langle x, \phi_p \rangle| \geq \alpha \max_k |\langle x, \phi_k \rangle|$ where $\alpha \in (0, 1]$.

To quantify the fit of a matched pursuit, we can utilize the output metrics, just like with machine learning for predictive analytics. Using `wmpalg` allows us the following options:

- **RESIDUALS** plot of observed - predicted vs observed; how far off are we?
- **QUALITY OF FIT** how it adapts to training; does it generalize?
- **FEATURE IMPORTANCES** dictionary atom weight; which features are important?

These concepts may require some background knowledge about statistical learning, which can be found in the wonderfully written and presented "Introduction to Statistical Learning" by Hastie and Tibshirani. The text covers the beginnings of machine learning and data science through linear algebra-based statistical models. In short, we are attempting to put some measure on the goodness of fit of our matching pursuit model, which will be trained on data from a specific basis [9, 10].

Chapter 2

Applying Dynamic Data Analysis on E4 Data

2.1 Pre-conditioned Data

Now that we've shown a suite of possible methods for data processing, it's time to apply them directly on data collected from the E4. To recap, we're going to start with a pre-conditioned set in which we used the best possible acquisition scenario: isolating the three-dimensional Euclidean coordinates, along with combinations of two directions (flip, spin, and roll). From there, the first step of analysis should always be clean-up.

2.1.1 Smoothing and Filtering

Based on our discussion in chapter 1, we are going to employ a Gaussian window function of length 11 to smooth the accelerometer data. The choice of window size comes from an observation of the periodicity in random data.

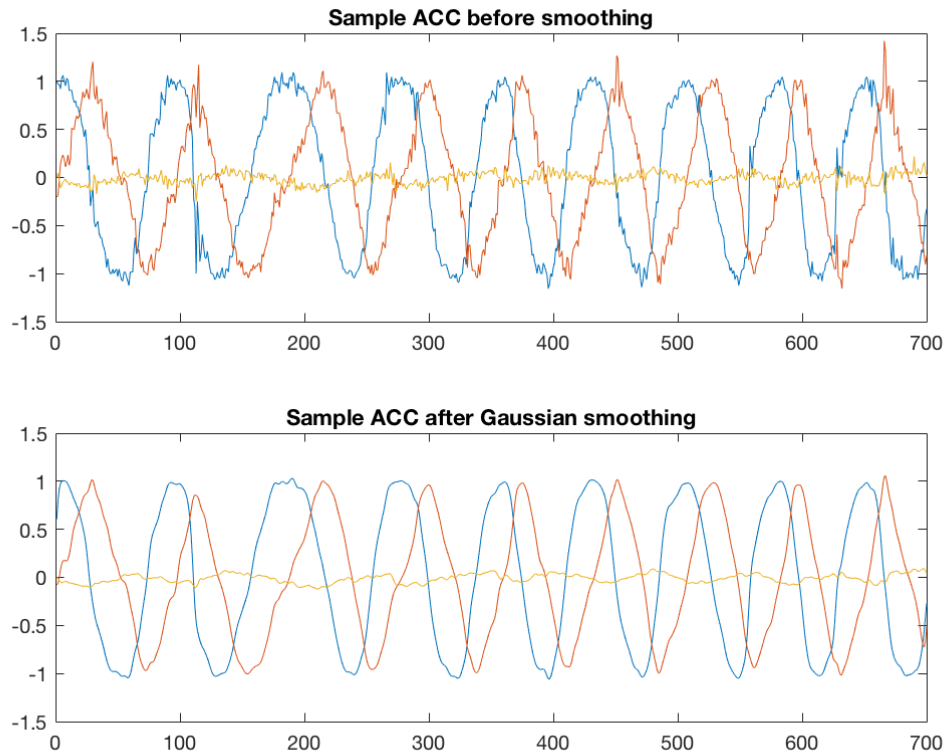


Figure 2.1: Smoothing an accelerometer data signal with a Gaussian window function

```

1 obj = E4_session('xy_7');           % initiate csv read
2 x = obj.ACC.data;                   % grab the ACC data
3 for i = 1:3
4     G = gausswin(11);               % choose an 11 point window
5     gaussFilt = G/sum(G);           % normalize
6     SV(:,i) = conv(x(:,i), gaussFilt); % convolute with Gaussian

```

Code Listing 2.1: Gaussian window smoothing code

Now we will compare the Gaussian smoothing with a Savitzky-Golay filter, using a third-order spline polynomial and a frame size of 15.

```

1 sV = sgolayfilt(x,3,15);

```

Code Listing 2.2: Savitzky-Golay filtering

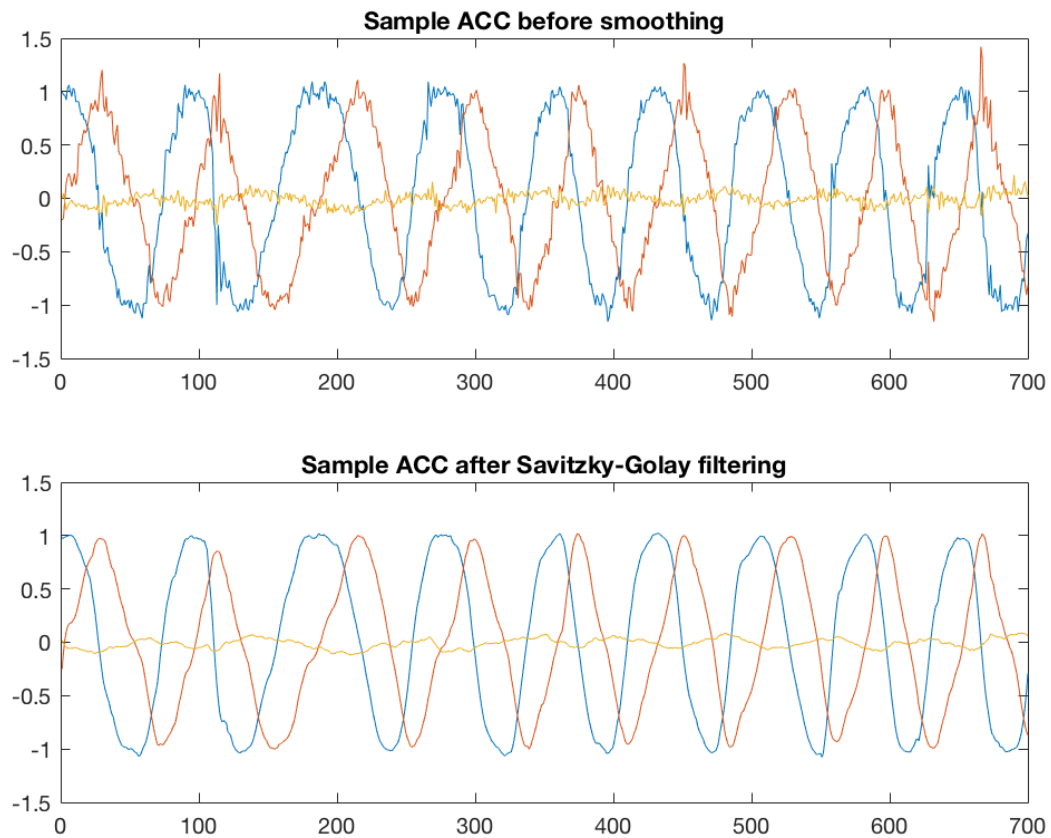


Figure 2.2: Smoothing an accelerometer signal with Savitzky-Golay filtering

As you can see, the two smoothing filters give a very similar result, the main distinction being found in the peaks. Recall that Savitzky-Golay preserves high frequencies as a defining property, and thus will be used in all future analyses. Here's a direct comparison of the x-axis of each method used above:

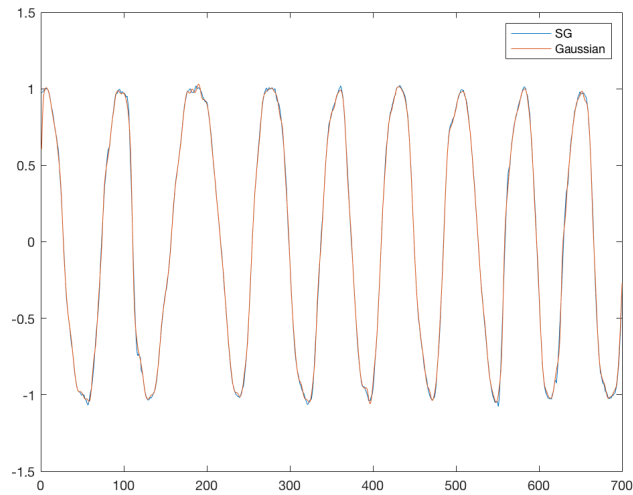


Figure 2.3: Close-up of smoothing with x-axis only

2.1.2 Neighborhood Search

After smoothing the data, we would like to identify the best possible coordinate system for the motions at play. The best way to do so is with the SVD, which we discussed previously.

As with our smoothing options, there are a few ways to set up this particular SVD after constructing a sequence of groups containing nearby data points. To do so, we will utilize the k-nearest neighbors search algorithm (with Matlab's built-in *knnsearch* function) to create a matrix M containing a starting point plus its 24 closest neighbors. For more information about *knnsearch*, see below for a brief description of the function. We will be using the Euclidean distance in the following code snippets.

```

1 function IDX = knnsearch(X,Y, 'Name', 'Value')
2     % Inputs:
3     %   X - Matrix of points or to-be nearest neighbors.
4     %   Y - Matrix of query points.
5     %
6     %   Name-Value pairs:
7     %   'k' - Specifying the number of neighbors to be found in X for
8     %       each point in Y. Default is 1.

```

```

9      %      'NSMethod' – Search method. Either kdtree or exhaustive.
10     %      Kdtree only valid for the following distance metrics:
11     %      'euclidean', 'cityblock', 'minkowski', or 'chebychev'.
12     %      'Distance' – Specifies the distance metric. Default is 'euclidean'.
13     %      see "help knnsearch for the full Name–Value pair list.
14     %
15     %      Outputs:
16     %      IDX – Indices in X denoting the located nearest neighbors.

```

Code Listing 2.3: k-NN Search Matlab function

The figure below simply shows a random data set, assuming motion in two directions so that we expect a circular pattern to appear if scaled down to \mathbb{R}^2 .

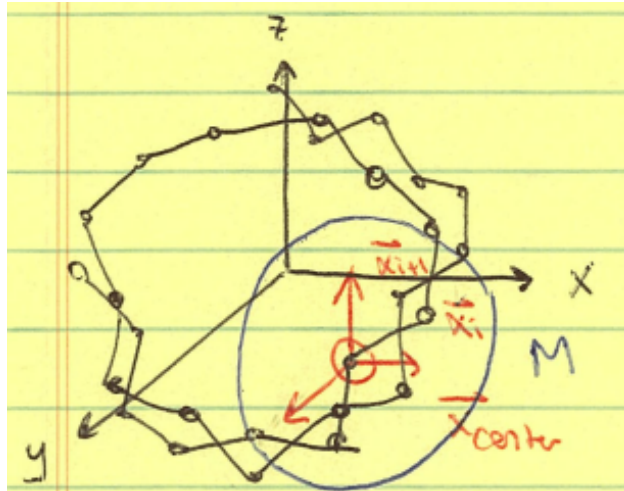


Figure 2.4: Geometry of the problem

To apply SVD on this data, M , let x_j , x_{j+1} , and x_{center} be the starting point, next nearest neighbor, and the center point of that knn cluster, respectively. To find the centroid of M , we take the average and create an appropriately-sized matrix where each row is the center, subtracting it from M to get a matrix, $M2$, of 25 nearby vectors that are now centered around the origin. Now we apply SVD on $M2$ to find its singular values and associated vectors. Next, we project out the third dimension and multiply $M2$ by v such that $v(:, 3) = 0$. To return to a 3-space, we apply the inverse transform (namely, v^T) to $M2 * v^T$ [11]. Finally,

the data needs to be relocated to its original position, so we'll add the center back to each vector in the final matrix. This script repeats for each vector in the smoothed data, until a local SVD is generated for the j^{th} row, each of which has an associated matrix of nearest neighbors. The code snippet below demonstrates this option for SVD on nearest neighbors after we apply a smoothing filter.

Assume these variables in each of the SVD options below:

```

1 obj = E4_session('session');           % initiate session csv read
2 if exist('x','var') == 0                % check if we can repeat the process
3     x = obj.ACC.data;                   % grab the ACC data
4 else x = MN;                            % assign x to be the recent coord sys
5 sV = sgolayfilt(x,3,15);                % run Savitzky-Goly filter to smooth ACC
6 tt = 1:700; d = length(tt);            % portion of data that fits best
7 plot3(x(tt,1),x(tt,2),x(tt,3))         % smoothed ACC data
8 Mnew = zeros(d,3); y = zeros(25,3);    % next M matrix and associated tangents
9 S = zeros(1,d); T = zeros(1,d);        % matrices to hold singular values

```

Code Listing 2.4: Variables for 3-D SVD options

It should be noted that the goal here is to localize to the underlying motion present in each session, so just doing one pass through is not enough. The 'if exist' portion of the code allows for us to apply the algorithm multiple times until we are confident that the best possible coordinate system has been found.

```

1 for j = 1:d
2     X = sV(j,:);                        % starting point for the search
3     idx = knnsearch(sV,X,'k',25);       % searches for 25 nearest neighbors
4     M = sV(idx(1:25),:);                % nearest neighbor smoothed value
5     center = sum(M)/numel(M);            % centroid
6     N = ones(25,1)*center;              % matrix of centroid points
7     M2 = M-N;                           % M matrix minus its center
8     [u,s,v] = svd(M2,0);                % svd to find best coord system
9     v2 = v; v2(:,3) = 0;                % projecting out the 3rd dimension
10    M3 = M2*v2;                          % coordinate transform

```



```

11     M4 = M3*v2'; % rotation back to 3d (using v')
12     M5 = M4+N; % put back centroid
13     Mnew(j:j+24,:) = M5; % matrix of M5's
14     MN = Mnew(1:d,:); % cut off the end

```

Code Listing 2.5: Local 3-D SVD using the centroid

2.1.3 Applying SVD to each motion

Finally, we will visually compare each of the local SVD options to coax out that "best" coordinate system that describes the underlying motion beneath the noise. Do note that each of these used the spin data, so we expected a circular motion localized to the xy-plane. Savitzky-Golay filtering was applied to smooth out the signal prior to analysis. The light blue depicts the smoothed ACC data, while the black dots are the actual motion, given by the SVD output. Sequentially, with each pass through the process, the localized dynamical system becomes more visible and apparent.

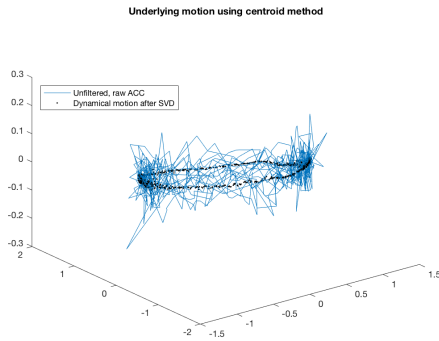


Figure 2.5: Spin SVD using a centroid of points, step 1

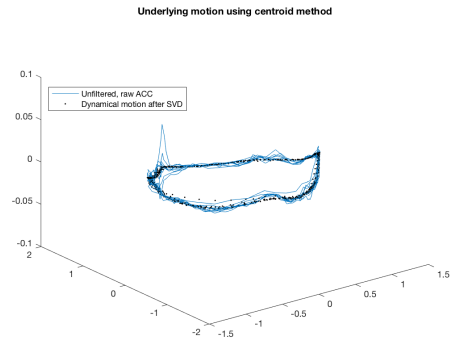


Figure 2.6: Spin SVD using a centroid of points, step 2

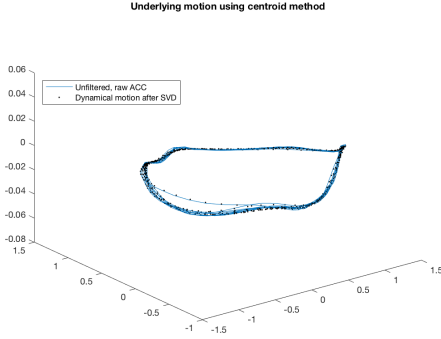


Figure 2.7: Spin SVD using a centroid of points, step 3

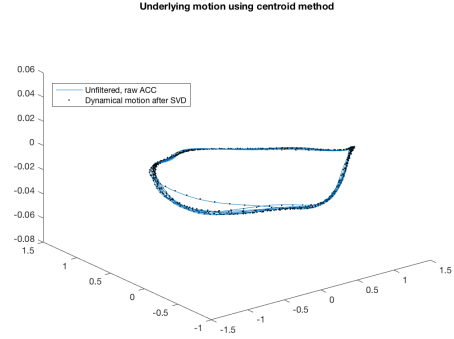


Figure 2.8: Spin SVD using a centroid of points, step 4

As you can see, with each new application of the algorithm, the actual underlying motion starts to poke through. Further passes yield similar results to step 4, but just slightly smoother, so we will stop at that point and assert the following: $\prod_x^\infty \text{SVD}_x(\text{signal}) = \text{actual motion}$, such that an infinite number of "local coordinate checks" produces the actual motion that is being collected by the accelerometer. Now, we will explore the other canonical motions and find out if this dynamic is present under those as well.

The xz-plane depicts a rolling motion. The reason for the bending and twisting of the dynamic comes from the data-collection phase, where the apparatus was not completely stabilized. In future revisions/research, a more exact and true apparatus will be used to nullify the y-component to depict a better roll.

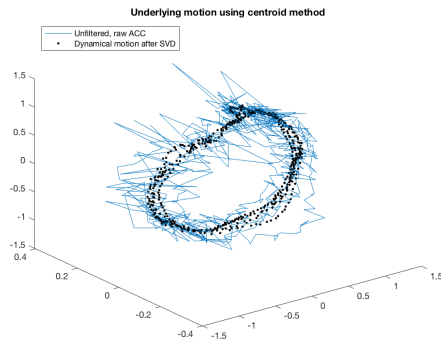


Figure 2.9: Roll SVD using a centroid of points, step 1

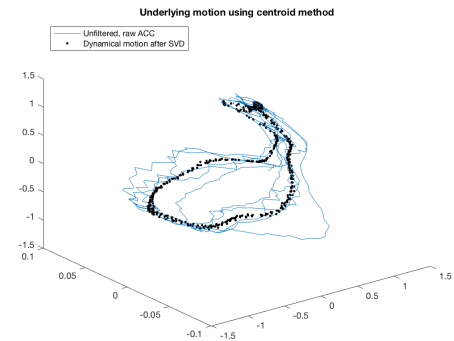


Figure 2.10: Roll SVD using a centroid of points, step 2

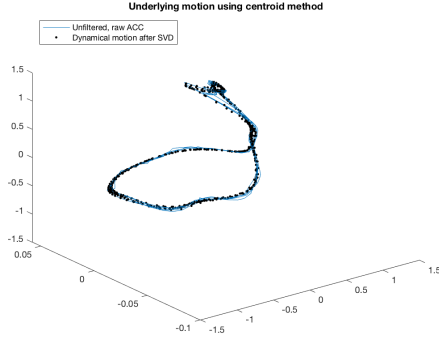


Figure 2.11: Roll SVD using a centroid of points, step 3

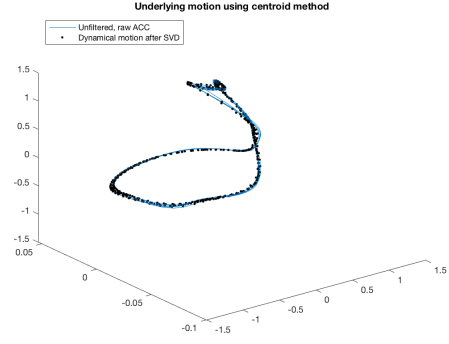


Figure 2.12: Roll SVD using a centroid of points, step 4

The same bending and twisting logic also applies to the yz-plane flipping motion. In both cases, we can clearly see 2d rotation in the major axes of each motion.

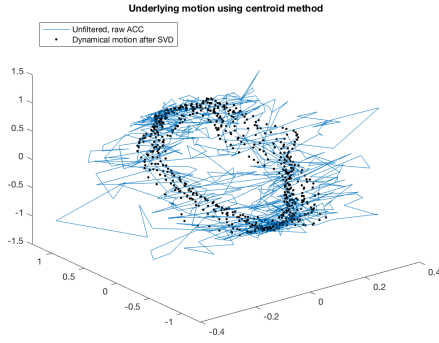


Figure 2.13: Flip SVD using a centroid of points, step 1

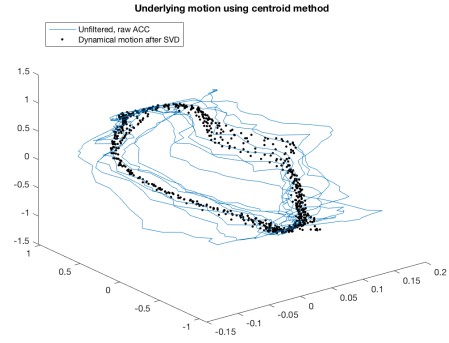


Figure 2.14: Flip SVD using a centroid of points, step 2

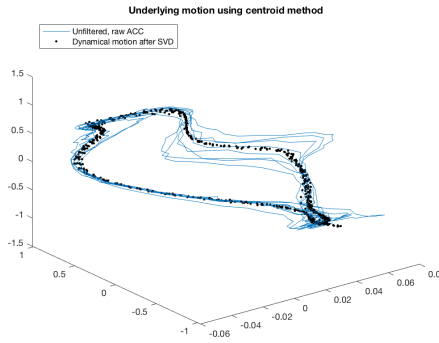


Figure 2.15: Flip SVD using a centroid of points, step 3

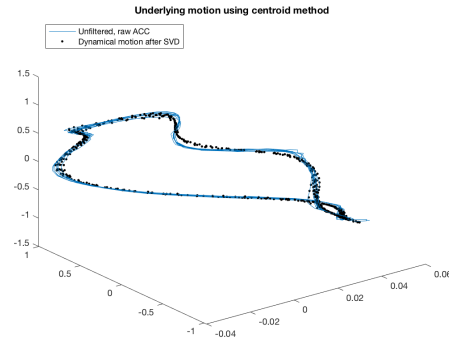


Figure 2.16: Flip SVD using a centroid of points, step 4

For the lateral motions (up-down, back-forth, left-right), a 2d SVD is utilized, with one dimension removed and another being projected out as we apply iterative transformations to the input signal. To choose the appropriate dimension to nullify we calculate the minimum variance of the input signal and remove that column - thus stabilizing the dynamic.

```

1 obj = E4_session('session');           % initiate session csv read
2 x = obj.ACC.data; v = min(var(x));      % grab the data, calculate min variance
3 for k = 1:3
4     if var(x(:,k)) == v                  % check if col has min variance
5         x(:,k) = [];                    % remove col if this is the case
6 sV = sgolayfilt(x,3,15);                % run Savitzky-Goly filter to smooth ACC
7 tt = 1:700; d = length(tt);            % portion of data that fits best
8 plot(x(tt,1),x(tt,2))                  % smoothed ACC data
9 Mnew = zeros(d,2); y = zeros(25,2);    % next M matrix and associated tangents
10 S = zeros(1,d); T = zeros(1,d);       % matrices to hold singular values
11 ... %continue with centroid method on 2 dimensions

```

Code Listing 2.6: Variables for 2-D SVD options

Each lateral plane depicts a different motion. The z-plane equates to up-down, y-plane is left-right, and x-plane is back-forth. For brevity, we will only include the first and final iterative SVD result.

Up-down:

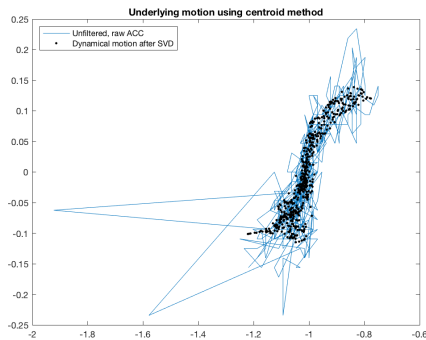


Figure 2.17: Up-down SVD using a centroid of points, step 1

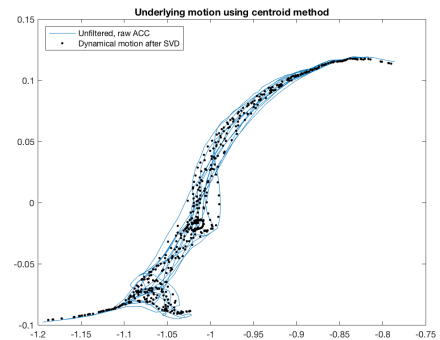


Figure 2.18: Up-down SVD using a centroid of points, step 4

Back-forth:

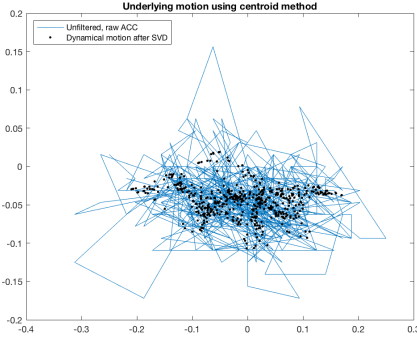


Figure 2.19: Back-forth SVD using a centroid of points, step 1

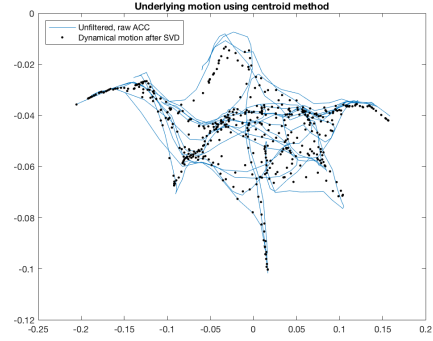


Figure 2.20: Back-forth SVD using a centroid of points, step 4

Left-right:

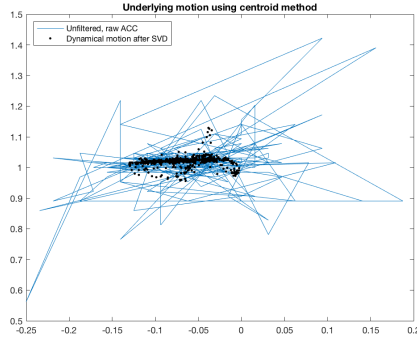


Figure 2.21: Left-right SVD using a centroid of points, step 1

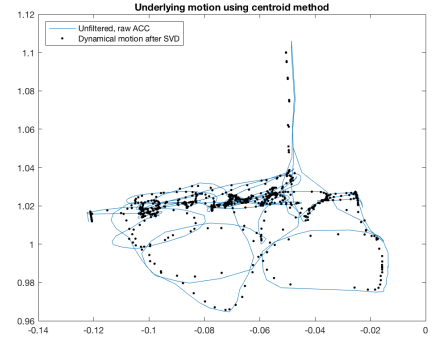


Figure 2.22: Left-right SVD using a centroid of points, step 4

2.1.4 Creating a Dictionary

A few more steps before designing the dictionary - first we have to account for randomness in the data-collection process by including interaction terms between the major axes of motion. Note that the neighborhood search SVD process removed some components of the motion via mapping down to a smaller dimensionality. To do so, we'll apply matrix multiplication first, then our NS-SVD scheme on the output (to maintain independence). Second, we normalize each component with $\frac{signal_i}{\|signal_i\|}, i \in signal$, giving us our final set of

atoms.

index	signal	index	signal	index	signal	index	signal
1-3	bf	19-21	bflr	37-39	udflip	55-57	fliproll
4-6	flip	22-24	bfud	40-42	udroll	58-60	flipspin
7-9	lr	25-27	lrud	43-45	udspin	61-63	rollspin
10-12	roll	28-30	bfflip	46-48	lrflip	63-66	all
13-15	spin	31-33	bfroll	49-51	lrroll		
16-18	ud	34-36	bfspin	52-54	lrspin		

Table 2.1: Index for collected samples, final

As a sanity check, we can compute the row-reduced echelon form of our matrix. This is immediately analogous to solving a system of linear equations with the goal of maintaining dimensionality of the original matrix, namely in the form of $rank(A)$. Once we are certain of linear independence among the dictionary atoms, we can run `wmpdictionary` and start the process of signal classification via matching pursuit.

If we plot the resulting dictionary from `wmpdictionary`, it looks like this:

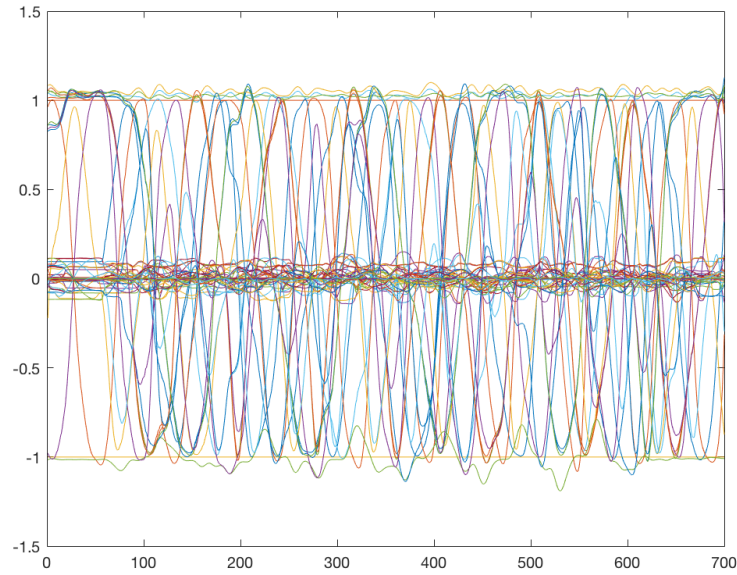


Figure 2.23: Plot of SVD-generated dictionary for Matching Pursuit

This plot doesn't hold any meaningful significance except to show some redundancy and

complexity of the input dictionary, plus perhaps some sense of fulfillment from all of the data collection process to make it to this point.

2.2 Supervised Learning

With the dictionary created, we're going to start applying matching pursuit on some data. Following along with some machine-learning concepts, we always begin by seeing how a model fits to training data. So let's use a sample of the signals used to create the dictionary, expecting the prediction to be perfectly accurate. As a reminder to the reader, these samples will have already been cleaned with Savitzky-Golay. Here's the rolling dictionary:

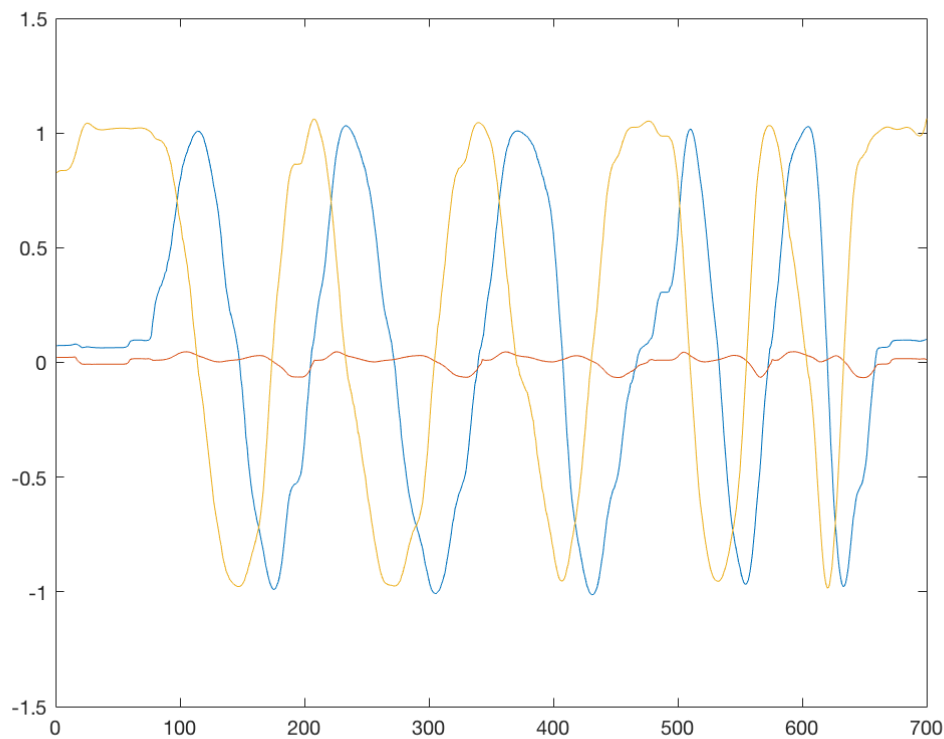


Figure 2.24: Supervised sample for MP

This process to apply matching pursuit on sample data first normalizes, then uses a cutoff point in the data to detect any points where we lose clarity (such as where $x, y, z = -2$),

and then resamples the signal to fit inside an appropriate length for fitting. We set the max number of iterations to 25 in order to mimic our original basis (without the additional interaction terms). Code snippet for our falling motion is below.

```
1 obj = E4_session('roll_svd');           % initiate session csv read
2 x = obj.ACC.data;                       % grab the sample
3 x = x/sum(x);                           % normalize the sample
4 d = length(tt);                         % portion that fits best
5 cut = find(x(1:end),2) == -2.000);      % find the cutoff point
6 y = resample(x,d,cut-1);                % resample(signal,length,cutoff)
7 y = y(tt,:);                            % cut off the end
8 [yfit,res,coeff,idx,qual] = wmpalg('WMP',y,xx,'typeplot','one','stepplot',2);
```

Code Listing 2.7: Matching Pursuit code

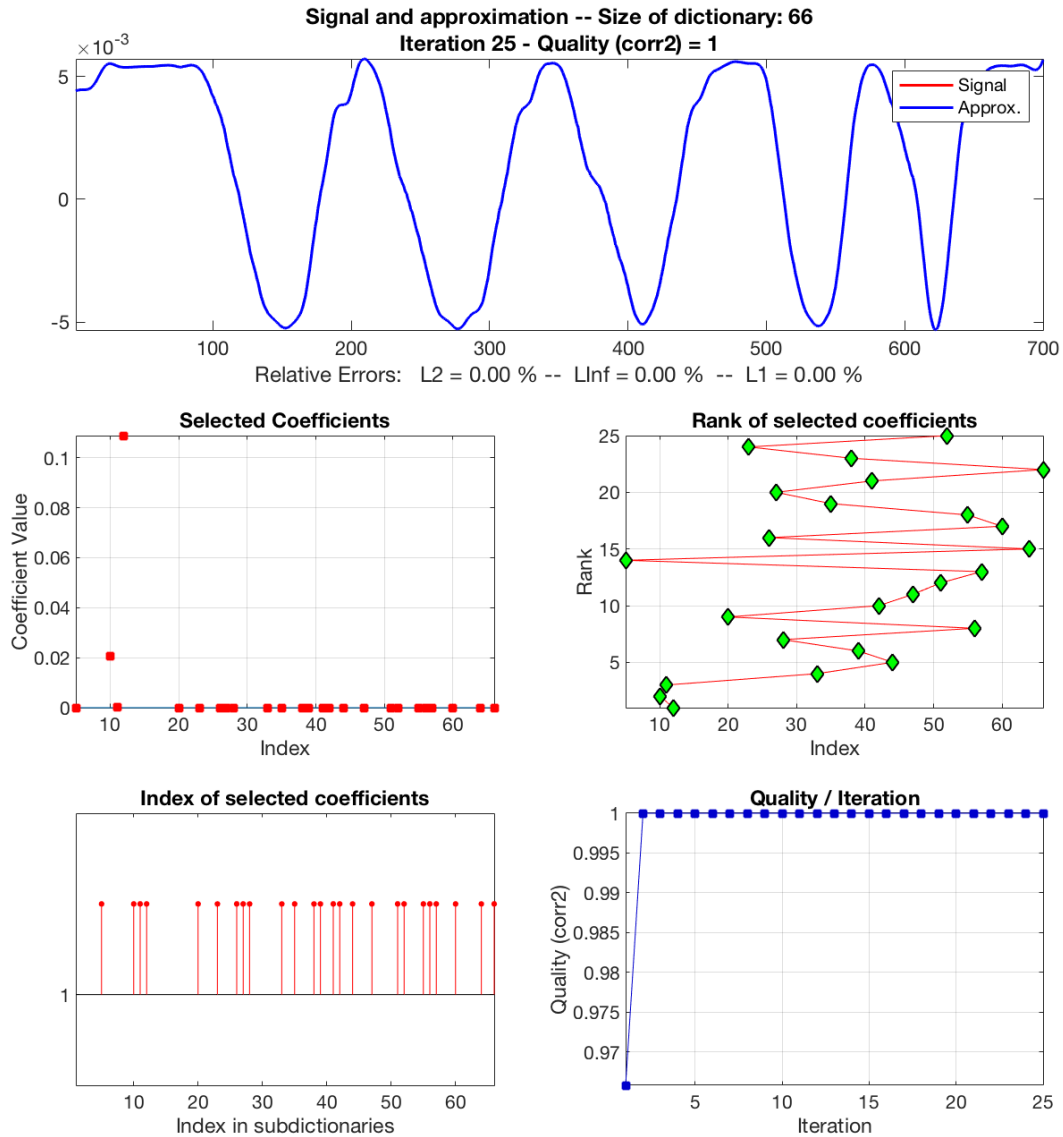


Figure 2.25: MP results for normalized rolling signal

2.2.1 Classification of a Known Input

Before we get into some results with the matching pursuit output plots, let's discuss the pieces of wmpalg. Our goal was to determine which dictionary atoms are contributing to the input signal - the first and most direct path to that answer is through the atom coefficients. These coefficients can be thought of as the most influential basis signal(s) to classify an

input. Connected to the coefficients is the index selector, which chooses which coefficient(s) to base the matched pursuit signal on.

We can also investigate the quality of the fit, or more precisely, the correlation between the signal and the dictionary atoms as you increase the number of iterations. This is easiest to understand when viewing the plots output from matching pursuit. Lastly, and perhaps, the most crucial component for analysis is the residual plot, which compares the observed values with the offset between actual and predicted. Along with quality, it can verify whether the model fits appropriately (given any assumptions made) [9, 10].

Plotting these residual plots is quite easy using Python packages, as seen here:

```
1 import pandas as pd
2 import seaborn as sb
3 import matplotlib.pyplot as plt
4
5 raw = pd.read_csv('data.csv')
6 res = pd.read_csv('residuals.csv')
7
8 fig = plt.figure()
9 ax = sb.regplot(x=res.x, y=raw.x)
10 ax.set_xlabel('Residual values')
11 ax.set_ylabel('Raw data')
12 ax.set_title('Residual plot')
13 fig.savefig("residual_plot.png")
```

Code Listing 2.8: Plotting residuals

As expected, the results for matching pursuit on a known sample are perfectly accurate, with atoms 10-12 chosen for classification (precisely the ones used to create the rolling sub-dictionary). Looking at the output coefficients and quality shows that MP sparsely represented the input signal perfectly in 2 iterations, estimating $y = 0.1084\phi_{12} + 0.0205\phi_{10} + 0.0001\phi_{11}$.

atom	weight	cumulative quality of fit
12	0.024282	0.9658
10	-0.0047335	1
11	0.0019433	1
33	0.0015364	1
44	-0.0036901	1
39	0.019895	1
28	0.0056626	1
56	-0.010656	1
20	0.0040329	1
42	0.0078637	1
47	0.0046697	1
51	-0.0069905	1
57	-0.0046011	1
5	0.0043878	1
64	-0.0040896	1
26	-0.0062983	1
60	-0.0064075	1
55	0.00204	1
35	0.0028547	1
27	0.0057625	1
41	0.0017381	1
66	0.0053186	1
38	-0.0017064	1
23	-0.0025586	1
52	-0.0014733	1

Table 2.2: MP metrics for supervised rolling data

As such, the residuals are diminished and dispersed randomly around $x = 0$. This is the property we want, as it indicates that our model assumption is correct and that the fit is accurate. Note the scale on the x-axis is $1e^{-15}$, which is essentially 0 due to floating point precision.

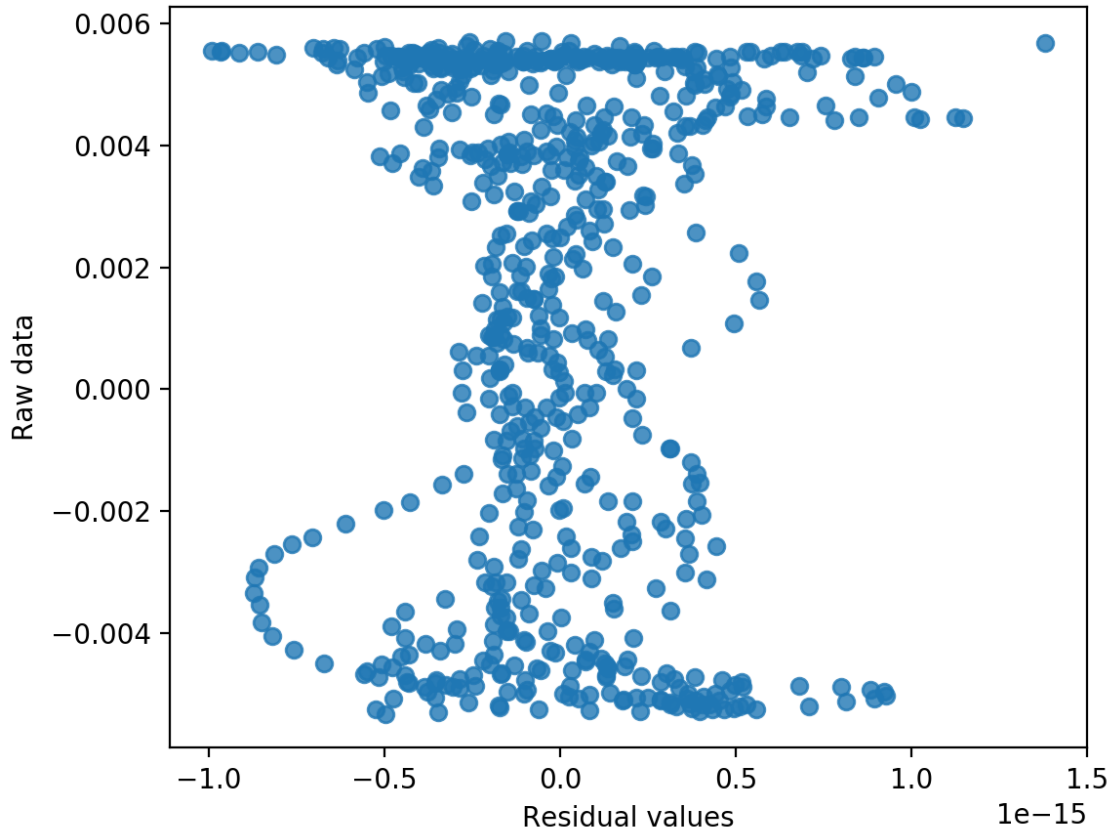


Figure 2.26: Normalized rolling signal residual plot

As we just saw, decomposing pre-determined motions using a basis of well-defined motions with lower dimensionality than the original data-space is quite intuitive. But what about arbitrary, random motions? Is the process as intuitive and simplistic? Let's find out.

Chapter 3

How to Decompose Arbitrary Motions Into a Canonical Basis

This final chapter will serve as a collection of all challenges and issues of applying some of the processes we've discussed on truly random data. The structure will be similar to that of chapter two, though a conscious choice was made here to focus on results and visualization rather than the mathematical details - as they were explored extensively in previous sections.

3.1 Semi-Supervised Learning

In chapter 2, we utilized a training set which was used in the creation of our basis dictionary, and the results showed, giving us a proof-of-concept to scale this to more random data. We'll start small a signal that aims to emulate walking, spinning, and falling - though not quite the same data as what was used for the dictionary. In machine-learning terms, this is called semi-supervised learning. The idea here is that the new sample is not an exact replica of the training data, but an approximation of what would happen if you pasted certain atoms next to each other [9, 10]. If MP is successful here, it shows that our method generalizes well to data it hasn't necessarily seen, but has an idea of what labels to expect.

3.1.1 Data Exploration

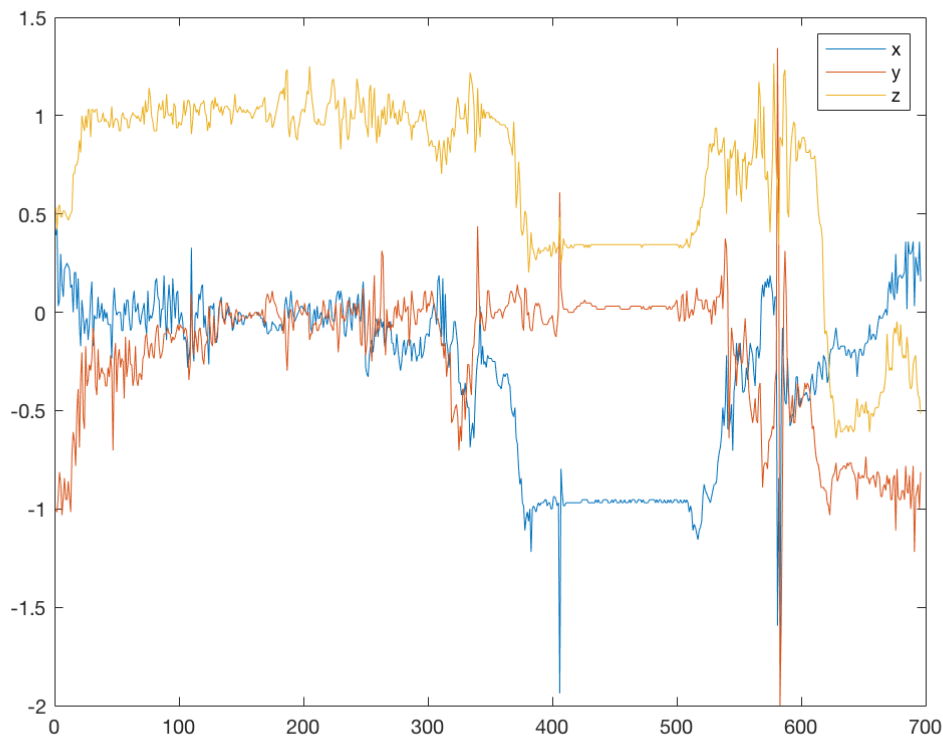


Figure 3.1: Pseudo-random sample accelerometer reading (falling)

The portion at $x = [0, 300)$ is a walking sequence, followed by a quick spin at $x = (300, 400)$, and finally a fall to the floor at $x = (400, 500)$. This leaves $x = (500, 583]$ to involve getting up and removing the E4. For our MP implementation, we include a routine to identify and remove data following cutoff points. From there, we resample the data to maintain an equal length to the dictionary. Using this signal, let's perform some cleansing and filtering to get a real sense of the accelerometer behavior. We'll apply the same Savitsky-Golay filter and explore the iterative SVD to determine a proxy for the underlying motion.

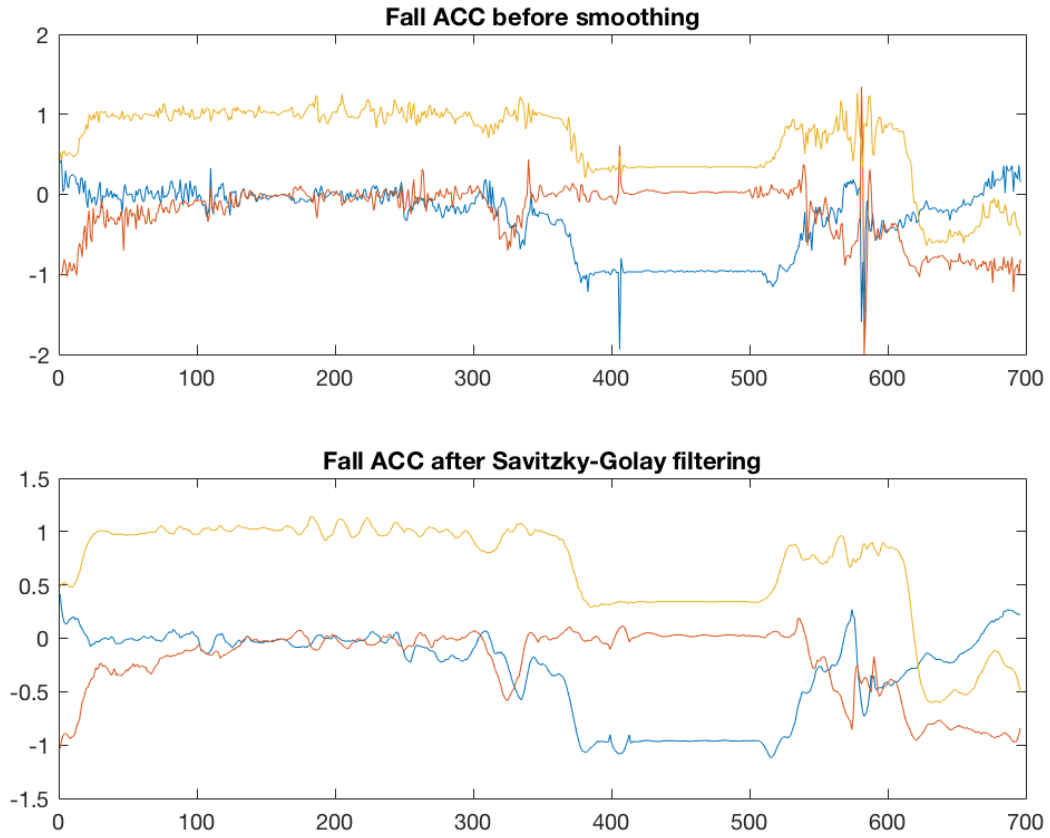


Figure 3.2: Savitzky-Golay filtering of falling data

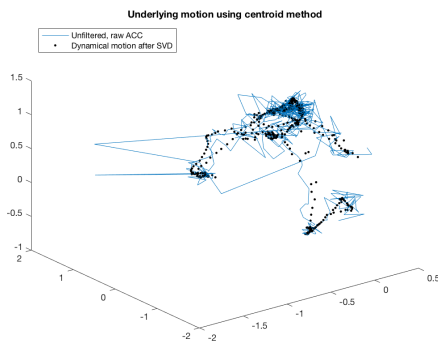


Figure 3.3: Falling SVD using a centroid of points, step 1

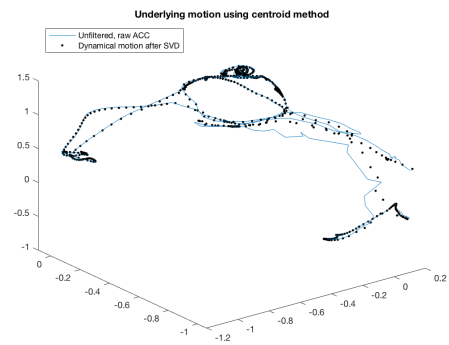


Figure 3.4: Falling SVD using a centroid of points, step 4

If we rotate the angle on the final image, we can clearly see a circle forming around the time of a spin and fall taking place - a definite indication that we will be able to detect that

signal from a portion of the input dictionary. It is also interesting how the walking portion of the sample nearly matches that of the left-right and back-forth dictionary signals.

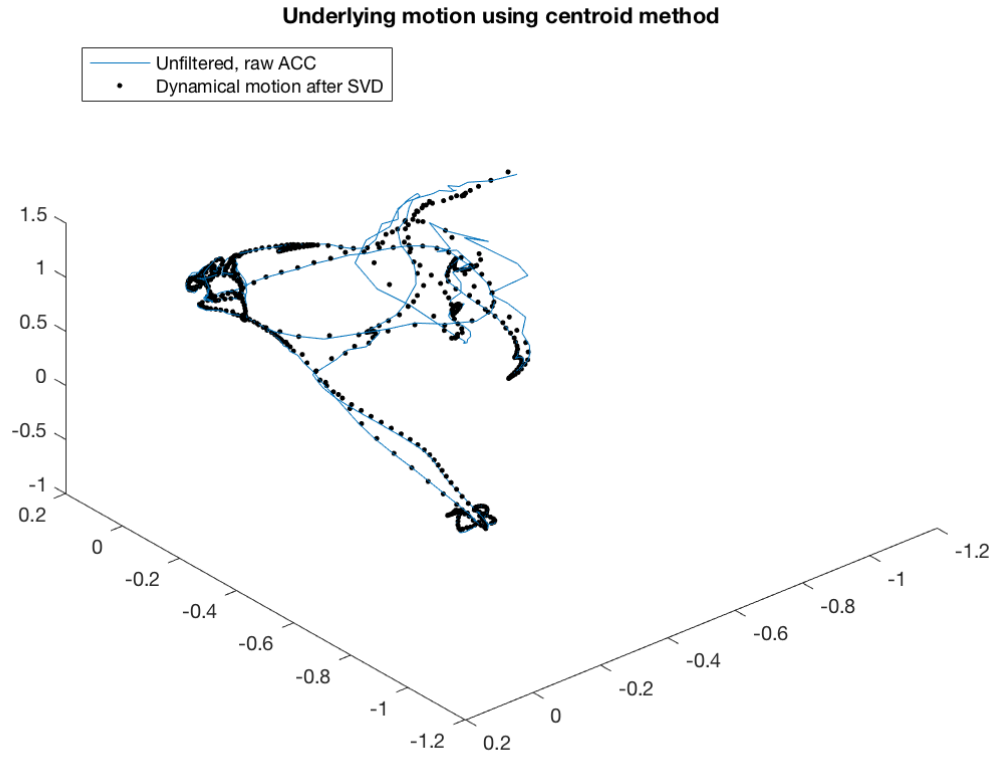


Figure 3.5: Different angle of step 4 above

3.1.2 Classification of a Semi-Random Input

As with our supervised learning example in the previous chapter, the steps to apply matching pursuit are the same, so we can jump straight to the plots and output.

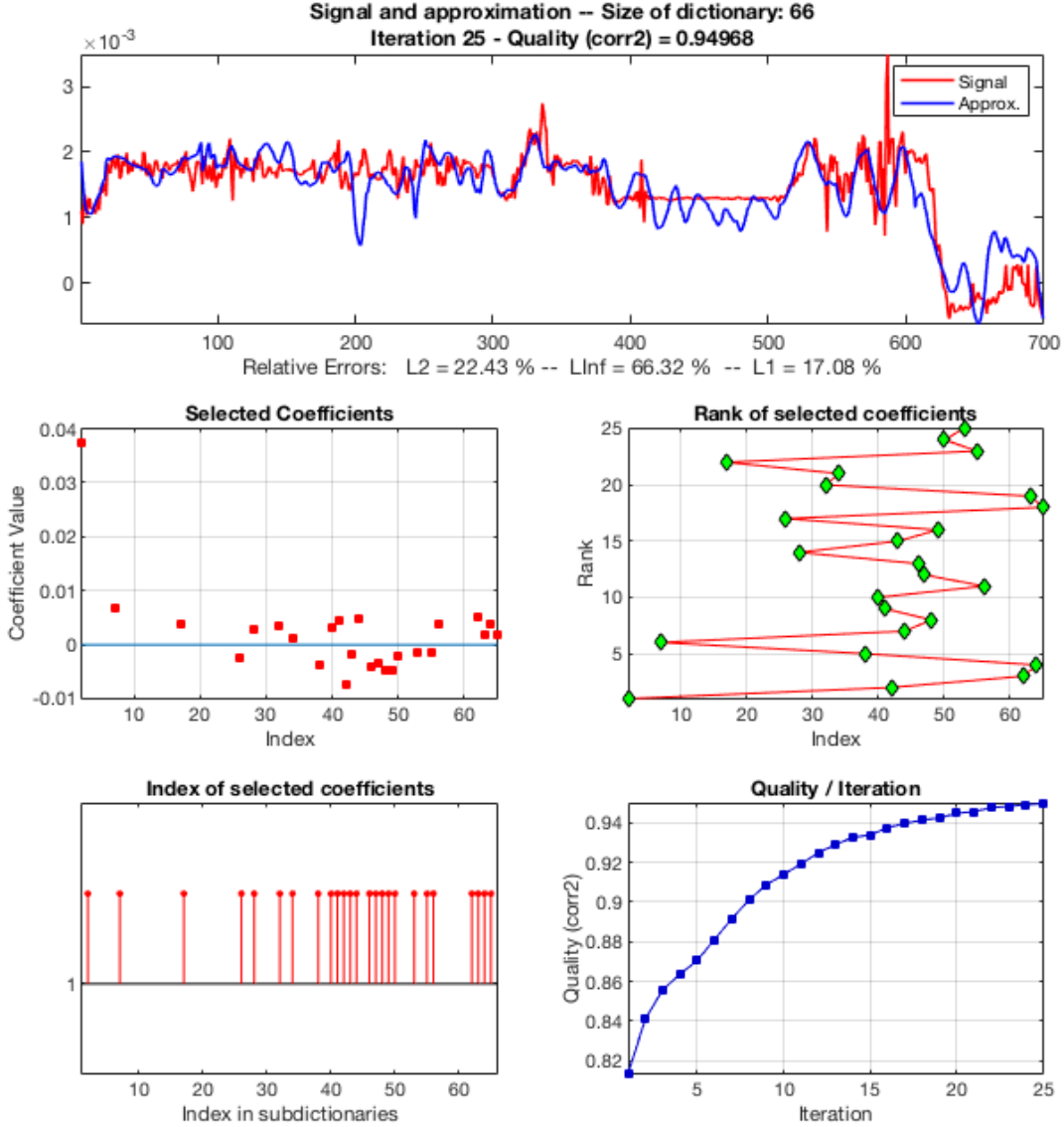


Figure 3.6: MP results for normalized falling signal

Top five atoms chosen for classification: 2 (bf), 42 (ud-roll), 62 (rollspin), 64 (all), and 38 (udflip), which represent some of the interactions present in the data - especially the back-forth and spinning motions. What is likely occurring here is due to the actual dynamics of the falling and standing up portion of the signal: the combined terms are hinting at additional factors being a necessary component of signal classification. Per the coefficients

and quality table, our falling signal was approximated up to 90% in 8 iterations.

atom	weight	cumulative quality of fit
2	0.024282	0.81357
42	-0.0047335	0.84114
62	0.0019433	0.85573
64	0.0015364	0.8637
38	-0.0036901	0.87063
7	0.019895	0.88117
44	0.0056626	0.89169
48	-0.010656	0.90111
41	0.0040329	0.90885
40	0.0078637	0.91394
56	0.0046697	0.91928
47	-0.0069905	0.92491
46	-0.0046011	0.92899
28	0.0043878	0.93264
43	-0.0040896	0.93388
49	-0.0062983	0.93755
26	-0.0064075	0.9397
65	0.00204	0.94137
63	0.0028547	0.94241
32	0.0057625	0.94477
34	0.0017381	0.94561
17	0.0053186	0.94762
55	-0.0017064	0.94829
50	-0.0025586	0.94913
53	-0.0014733	0.94968

Table 3.1: MP metrics for semi-supervised falling data

Again, we have residuals which are diminished and dispersed randomly around $x = 0$ (aside from 2 outliers); a clear sign of an accurate fit.

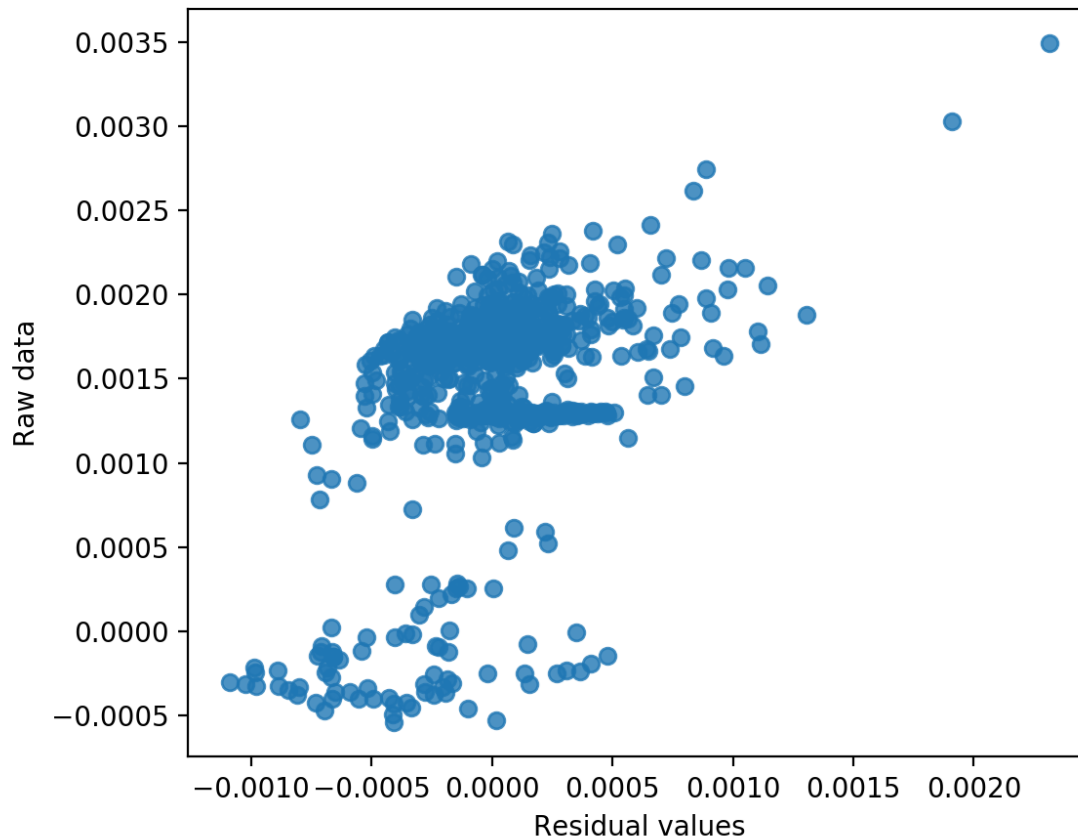


Figure 3.7: Normalized falling signal residual plot

3.2 Unsupervised Learning

Now with a solid grasp of generalization achieved, let's move on to the real test: completely unlabeled, random input. We'll apply the same exact process, with the caveat of not knowing the true dynamic of motion. Instead, we'll attempt to infer behavior based on the basis dictionary signals we created, verifying with matching pursuit.

3.2.1 Data Exploration

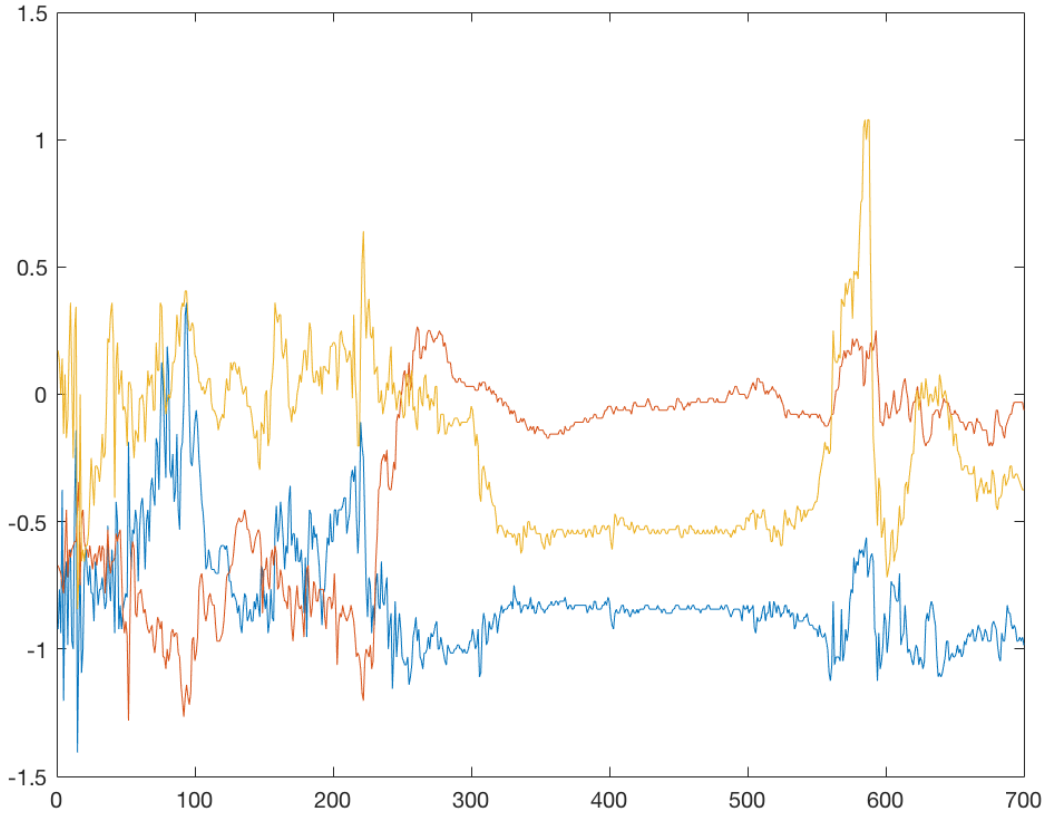


Figure 3.8: Random input signal

Taking an educated guess at this data, I'd suggest that the portion from $x = [0, 250)$ is a combination of coupled lateral motions. Whether these manifest as well-defined spins or flips is hard to discern. From $x = (250, 550)$, some interesting behavior develops; a steep rise in left-right motion, followed by near-stagnation of all axes. This could be indicative of a reflex causing a sudden movement, then silence. And finally, for $x = (500, 700]$, all signals perk up (yz being stronger, perhaps a flip with some back-forth artifacts?) before returning to a constant state. Let's explore the data some more to see if certain patterns make themselves clear.

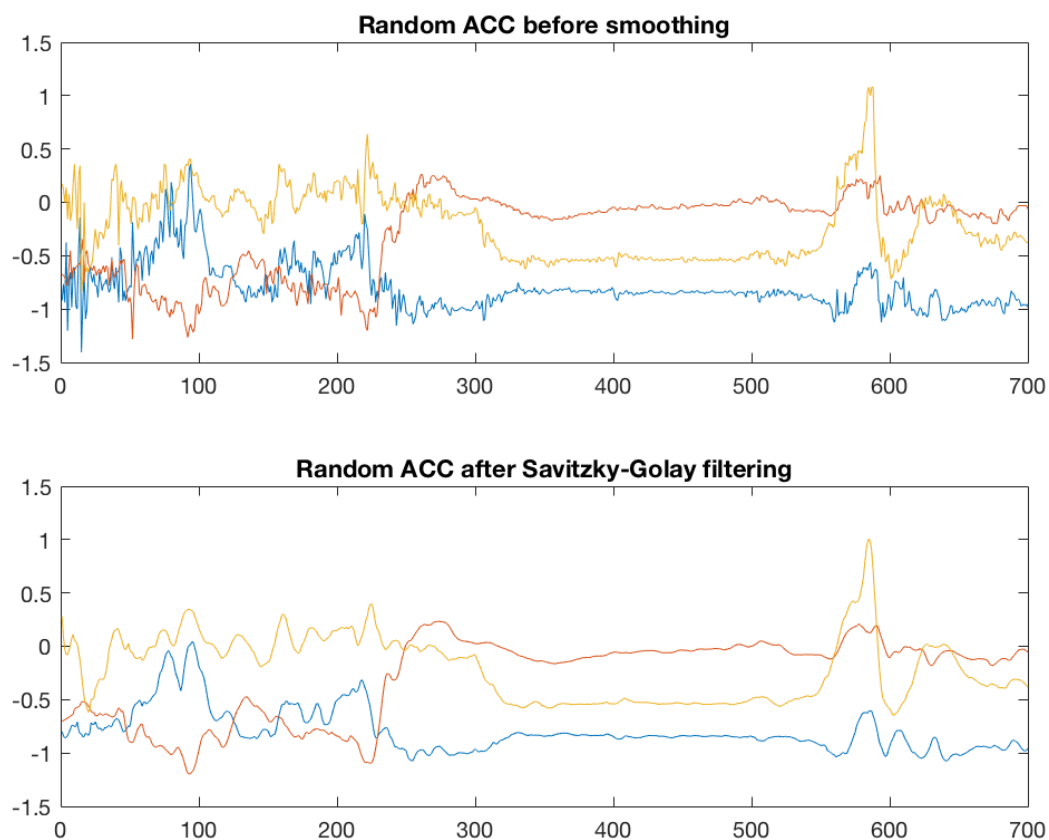


Figure 3.9: Random input signal, filtered

After applying Savitzky-Golay, it seems that the $x = (500, 700]$ portion is more indicative of a spin with heavy up-down artifacts, due to the relative mirroring of the x and y signals at the tail-end of the sample.

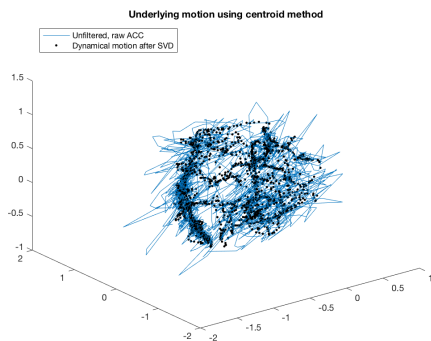


Figure 3.10: Random SVD using a centroid of points, step 1

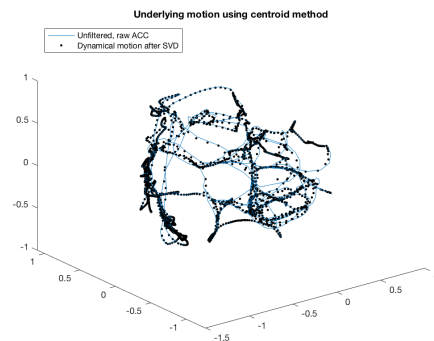


Figure 3.11: Random SVD using a centroid of points, step 4

Nothing obvious here, but suppose we rotate it like we did with the semi-supervised sample. This gives the data more perspective, and could explain the late spike in the z -component (the portion of the image toward the left-hand side).

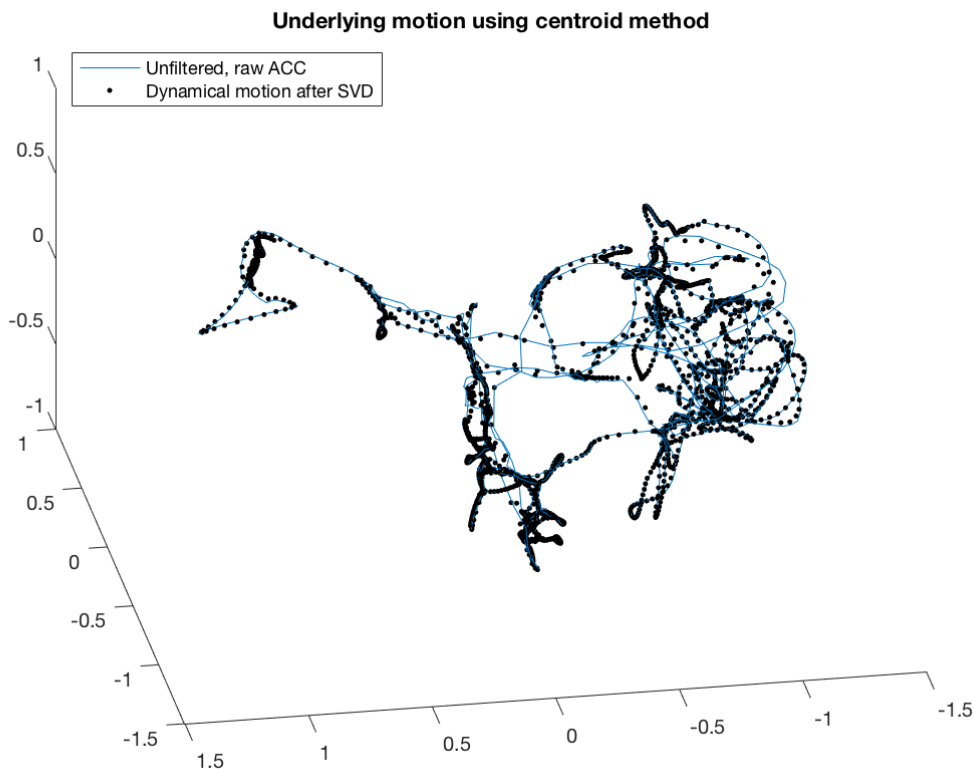


Figure 3.12: Different angle of step 4 above

Nonetheless, the sample remains unlabeled. For future research, a video recording of each sample for manual tagging would be greatly beneficial for random input signals. This would give us an important benchmark in how matching pursuit performs, without needing the guesswork.

3.2.2 Classification of a Random Input

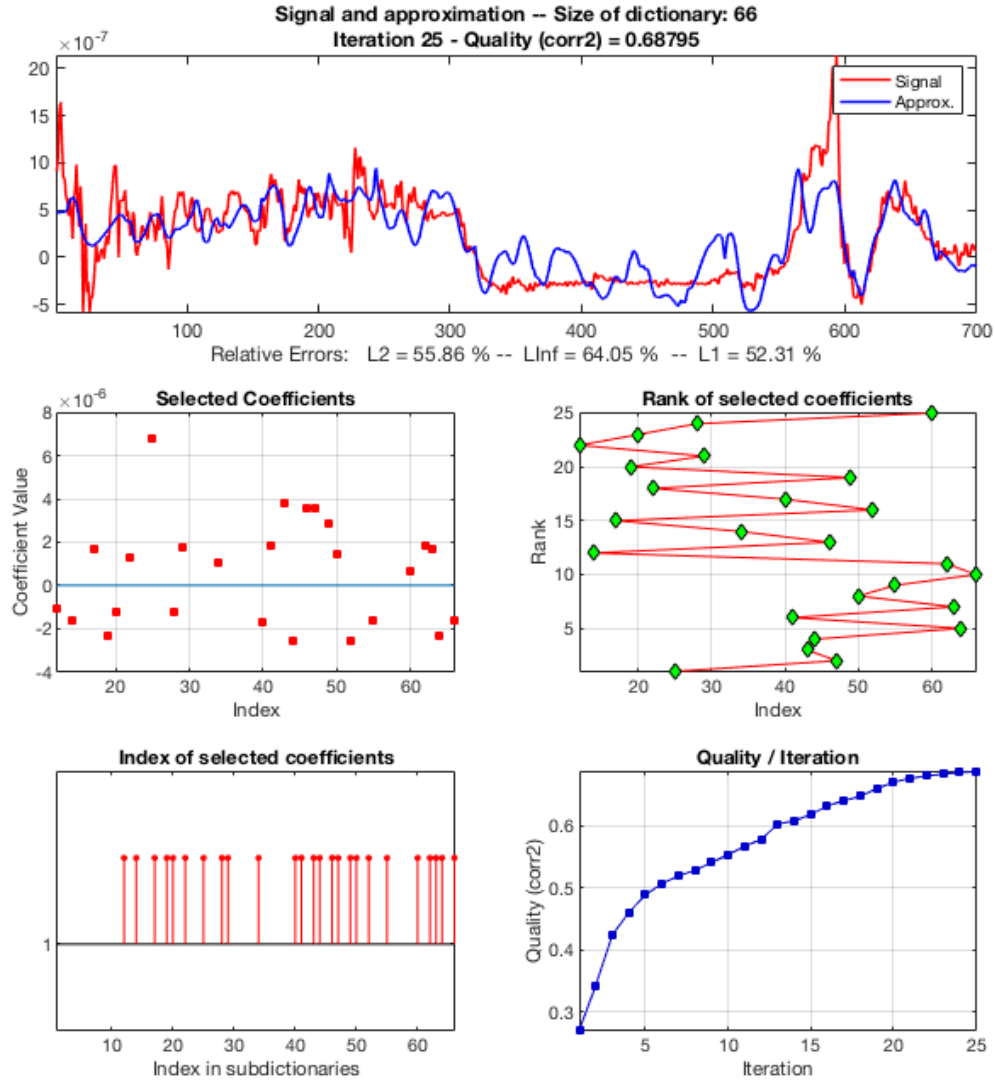


Figure 3.13: MP results for normalized random signal

Top five atoms chosen for classification: 25 (lr-ud), 47 (lr-flip), 43 (ud-spin), 44 (ud-spin), and 64 (all), which represent a lot of the interactions in the data. This is telling because we are not aware of the true labels here, and it is real-world data we’re attempting to describe with a sparse dictionary. In terms of coefficients and quality, our random signal was approximated up to 68% in 25 iterations. If we increased the number of iterations, the variance in the input could be explained, however at the expense of interpretability.

atom	weight	cumulative quality of fit
25	3.0199E-06	0.26976
47	2.4879E-06	0.34218
43	5.2201E-06	0.4237
44	-2.2532E-06	0.46024
64	-2.3684E-06	0.48875
41	2.8208E-06	0.50688
63	3.9229E-06	0.51932
50	3.6866E-06	0.52872
55	-1.3833E-06	0.5409
66	-4.0142E-06	0.55382
62	4.359E-06	0.56733
14	-4.2519E-06	0.57844
46	4.2165E-06	0.60309
34	1.263E-06	0.60866
17	3.5364E-06	0.61852
52	-2.2609E-06	0.63277
40	-4.1962E-06	0.64107
22	3.302E-06	0.64839
49	3.7426E-06	0.66053
19	-2.5989E-06	0.67125
29	2.6658E-06	0.67727
12	-1.1614E-06	0.68112
20	-1.1902E-06	0.68418
28	-1.2972E-06	0.68681
60	6.6311E-07	0.68795

Table 3.2: MP metrics for unsupervised random data

Since we have a slightly misclassified signal, our residual plot should be slightly off as well. Despite still being somewhat randomly dispersed around $x = 0$, there is a definite trend of larger values (likely contributing to spikes in the data) of having larger residuals.

In statistics, we would interpret this residual trend as a sign of needing additional factors in our model.

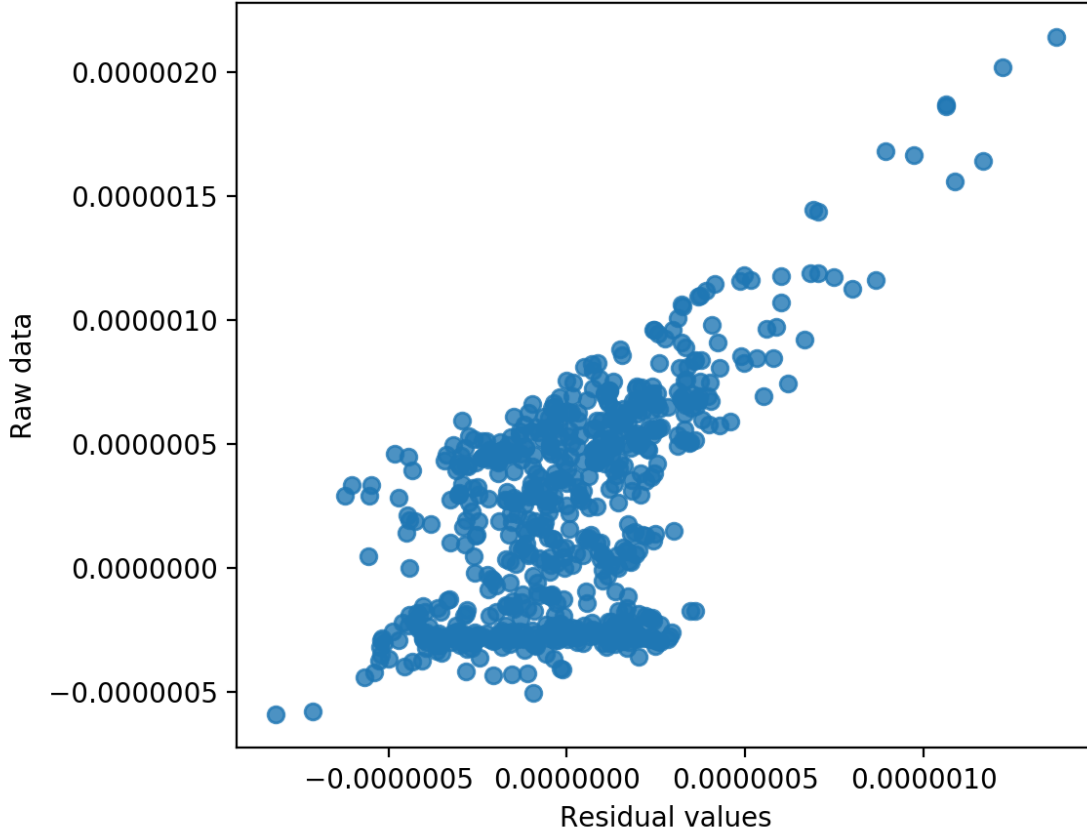


Figure 3.14: Normalized random signal residual plot

For completeness, here is matching pursuit on a random input using all of the atoms in our dictionary (setting `itermax` to be 66). The additional factors will help fill in gaps in the original prediction and enhance the residual plot and quality (up to 80%), but will not affect the top five contributing atoms, as the iteration process is optimize to always choose the best atom (one that maximize $|\langle x, \phi_k \rangle|$ and minimize $\|f - D_k\|^2$) in sequence until either a) convergence is achieved, b) `itermax` is reached, or c) there are no more atoms to use.

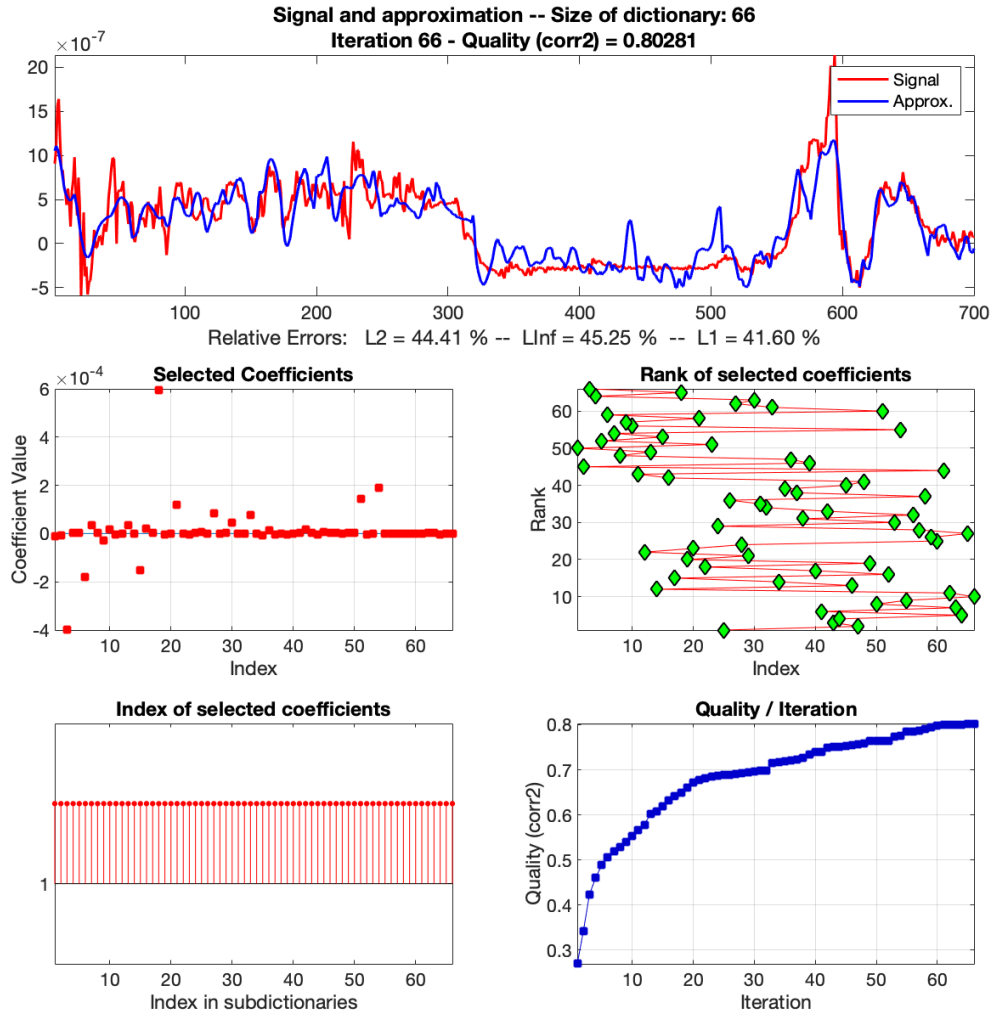


Figure 3.15: MP results for normalized random signal using all atoms of our custom dictionary

3.3 Limiting Factors

Given the promising results on a totally random input with both the dictionary we created and the default for `wmpalg`, I'd like to pose two questions. 1) Is our dictionary complete and/or redundant enough to generalize to all cases? 2) Is our dictionary sparse enough so that atom coefficients are uniquely identifiable?

These questions seem to highlight a divide between accuracy and interpretability [10].

On one hand, if we use the most robust dictionary possible, we'd expect nearly perfect classification accuracy. But on the other, if we keep the dictionary small, we'd expect nearly perfect precision in the atoms chosen for each assignment. Suppose we were to use a default library of basis functions instead of building our own. Would the results perform exceptionally better? At what potential cost? Let us consider this using our unsupervised, random sample.

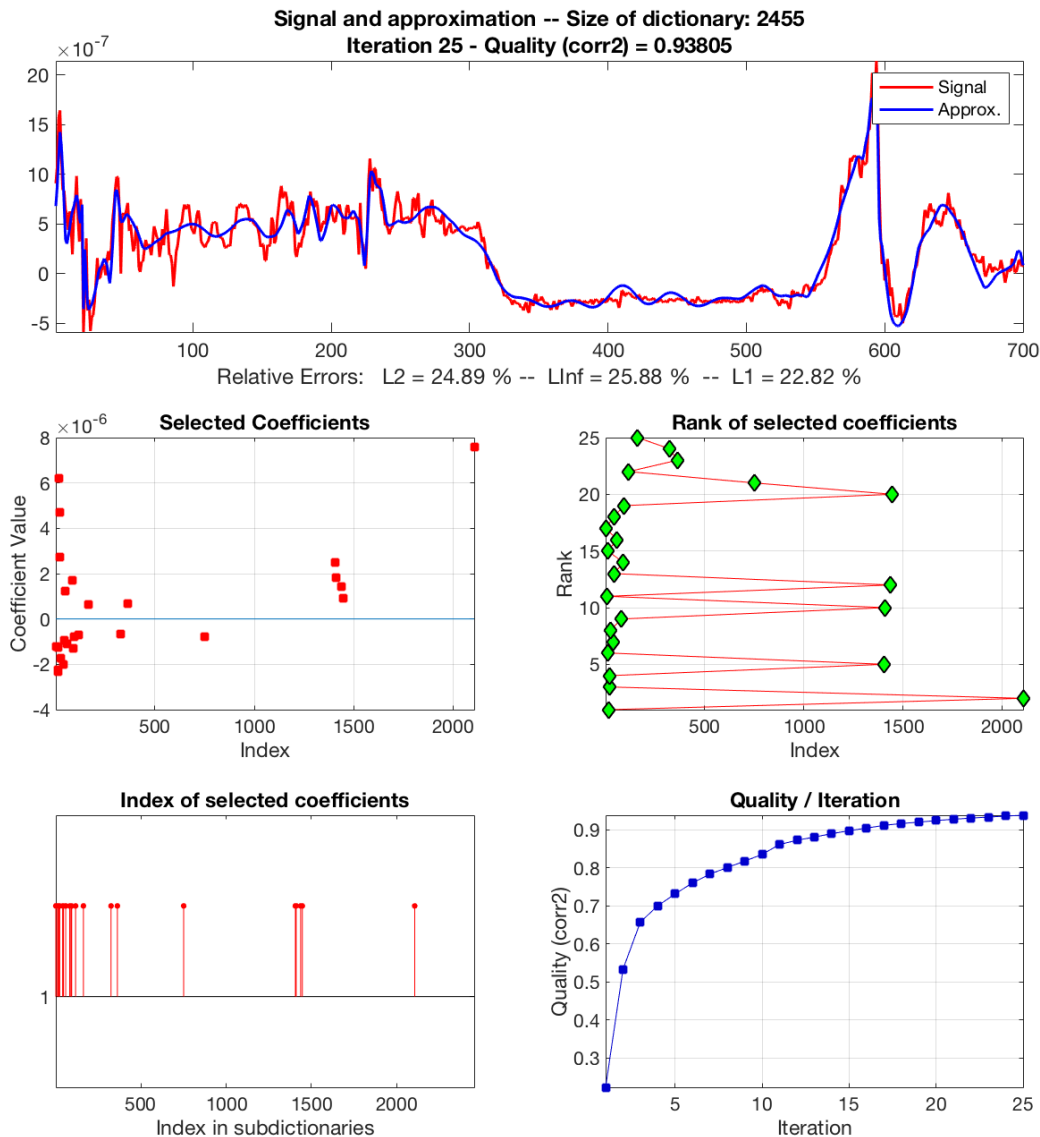


Figure 3.16: MP results for normalized random signal with default dictionary

Per the coefficients and quality table, our random signal was approximated up to 93% in 25 iterations, quite a few iterations less than our test with the entirety of our custom dictionary.

atom	weight	cumulative quality of fit
18	6.7742E-06	0.2227
2106	8.2557E-06	0.53328
20	3.8668E-06	0.6573
22	1.3428E-06	0.70004
1406	4.507E-06	0.73116
11	-2.5954E-06	0.76035
41	-2.0216E-06	0.78336
24	-1.7507E-06	0.80066
82	1.59E-06	0.81751
1411	2.4026E-06	0.83508
10	-2.4215E-06	0.86113
1437	8.8946E-07	0.8722
46	1.2764E-06	0.88072
89	-1.181E-06	0.88958
12	-1.2379E-06	0.89741
59	-1.0928E-06	0.90432
1	-1.1983E-06	0.91141
42	-1.0434E-06	0.91623
94	-8.0068E-07	0.91991
1447	8.9412E-07	0.92403
751	-7.8209E-07	0.92756
117	-7.1627E-07	0.93053
362	6.861E-07	0.93326
325	-6.6001E-07	0.93579
163	6.2429E-07	0.93805

Table 3.3: MP metrics for unsupervised random data (default dictionary)

With a better fit, we would also expect a more randomly dispersed set of residuals, as seen here.

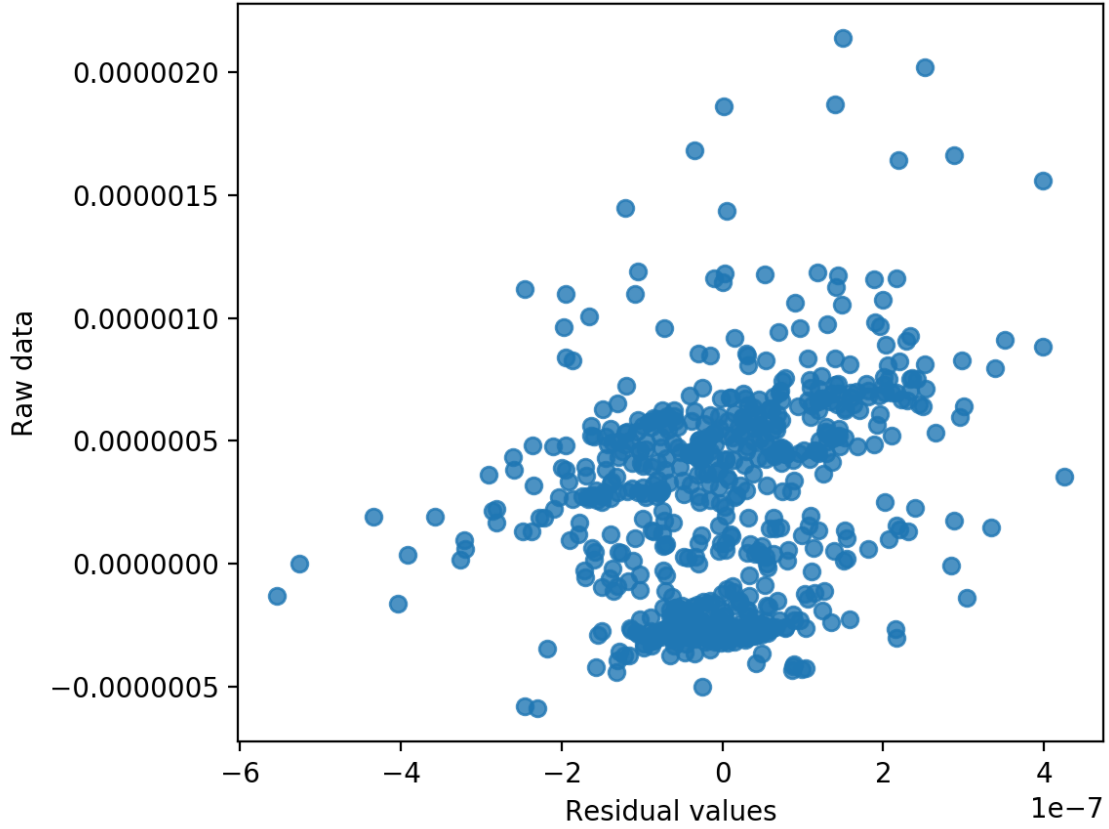


Figure 3.17: Normalized random signal residual plot with default dictionary

As shown from the two results, the default dictionary performs better in terms of quality of fit and percentage points, but lacks the precision that a specifically constructed basis provides. In short, we would lose a significant chunk of interpretability in the model if we blindly utilize the default basis. Of course, the `wmpdictionary` default is an *orthogonal* basis, which already is a huge leap from our non-orthogonal basis used in the paper [7, 8, 11]. For this method to become more robust to any accelerometer input, many more samples would need to be collected, normalized, and applied to the analysis.

Since the techniques presented in this paper were dependent on manual data collection (somewhat mitigated by isolating axes), optimized questions, future research, and more data (that is video-tagged and labeled) is absolutely necessary to achieve the goal of applying

real-time detection algorithms on raw accelerometer data. Some code for a streamed data (which still has some latency) has been developed, but our focus was to present a method for classification of pre-collected samples.

Other sources of limitations include the prevailing sense of scale used. All dictionary atoms were either collected at a fixed length, or resampled to meet that length. This meant that all matching pursuit calls had to be on the same scale, without adjusting for phase space. Ideally, one could pull out specific portions of an input signal to assign a single classification for that snippet (repeating this process over different window lengths), but nothing of the sort was developed for this research. This concept is closely related to the idea of "Frames".

3.4 Summary and Conclusion

To summarize our matching pursuit task, we have the input conditions and output results tables:

Data Type	Label	Source	Expectation
Supervised	Labels known	Single basis signal	Perfect metrics
Semi-supervised	Labels recognized	Approximate concat of basis signals	Correct atoms, quick convergence
Unsupervised	No labels	Completely random	None
Unsupervised (default)	No labels	Completely random	Better accuracy over custom dictionary

Table 3.4: Summary of conditions for MP alg on each learning method

Data Type	Accuracy	Convergence	Notes
Supervised	Correct atoms, perfect metrics	Immediate, within 2 iterations	Proof-of-concept
Semi-supervised	Mostly correct atoms (bf, spin)	90% within 8 iterations	Shows generalization to similar data
Unsupervised	Based on observational guess, atoms are in the right ballpark	Slow, 68% after full bout	Additional generalization, shows some weaknesses
Unsupervised (default)	No idea	Faster, 90% fit after 20 iterations	Accuracy at the expense of interpretability

Table 3.5: Summary of results for MP alg on each learning method

In conclusion, we have outlined a successful method for data collection, filtering/cleansing, classification, and verification of results using raw accelerometer signals from a medical watch. From this, we have come up with matching pursuit models that are 70-95% accurate on incoming raw data. Much of this success is due to the emphasis placed upon building a suite of standard practices and applications for data analysis, along with extensive searches for the best possible coordinate system for the dynamics (through a long process of trial-and-error during the acquisition phase). We have leveraged statistical learning and data science techniques through different learning methods, model metrics, and discussions about accuracy versus interpretability.

References

- [1] Garbarino, M., Lai, M., Bender, D., Picard, R. W., Tognetti, S. (2014). Empatica E3 - A wearable wireless multi-sensor device for real-time computerized biofeedback and data acquisition. In 2014 EAI 4th International Conference on Wireless Mobile Communication and Healthcare (Mobihealth) 39-42. Abstract, PDF, BibTeX.
- [2] Poh, M. , Loddenkemper, T. , Reinsberger, C. , Swenson, N. C., Goyal, S. , Sabtala, M. C., Madsen, J. R. and Picard, R. W. (2012), Convulsive seizure detection using a wrist-worn electrodermal activity and accelerometry biosensor. *Epilepsia*, 53: e93-e97. doi:10.1111/j.1528-1167.2012.03444.x
- [3] T. Gibbs, Peter Wood, Levi Asada, Harry. (2005). Active motion artifact cancellation for wearable health monitoring sensors using collocated MEMS accelerometers. *Proceedings of SPIE - The International Society for Optical Engineering*. 5765. 10.1117/12.600781.
- [4] John-Olof Nilsson, Amit K Gupta, and Peter Händel. Foot-mounted inertial navigation made easy. In *International Conference on Indoor Positioning and Indoor Navigation*, volume 27, page 30th, 2014.
- [5] E. Cope. *Estimating Human Movement Using a Three Axis Accelerometer*. Mar. 9, 2009, 79 pages.
- [6] Luinge H.J., *Inertial Sensing of Human Movement*, Ph.D. thesis, University of Twente. 2002, pp. 1-88.
- [7] The Mathworks, Inc. (2017). *Signal Processing and Wavelet Toolbox: User's Guides* (r2017a).

- [8] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1992. Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing. Cambridge University Press, New York, NY, USA.
- [9] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. An Introduction to Statistical Learning: With Applications in R. Springer Publishing Company, Incorporated.
- [10] Hastie, T., Tibshirani, R., and Friedman, J. 2001. The Elements of Statistical Learning. Springer New York Inc., New York, NY, USA.
- [11] Trefethen, Lloyd N. and Bau, David. Numerical Linear Algebra. : SIAM, 1997.