School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

A Computer Model of Electrocardiogram Signals

by

Andrew Chin Hou Au

Thesis submitted as a requirement for the degree of

Bachelor of Software Engineering

Submitted:	December 2019	Student ID:	z5020593
Supervisor:	Dr. Socrates Dokos	Topic ID:	N/A

Abstract

This thesis attempts to implement an open source computer model of electrocardiogram signals (ECGs). While there are open source implementations of computer models available they do not include extensions such as noise filtering or parameter estimation, which are typically only implemented in proprietary software like MATLAB. This thesis implements a computer model of ECGs capable of performing parameter estimation with an Extended Kalman Filtering that can serve as a basic introduction to an open source computer model.

Results show that the Extended Kalman Filter is capable of estimating the parameters of a dynamic model of electrocardiogram signals, but requires proper initial estimates, more observations and control over sampling rates to reliably produce the parameters of the model.

Contents

Chapter 1

Introduction

Computer modelling of Electrocardiogram Signals (ECGs) is a field concerned with the generation of synthetic ECG signals. This is achieved with a dynamic mathematical model that describes the motion of an ECG. A computer model implements the dynamic model and is then able to simulate ECG signals with varying parameters such as noise, sampling frequency and heart rate. Consequently, a computer model is able to generate regular and irregular heart signals.

Signal processing techniques are valuable for the clinical diagnosis of heart conditions such as arrhythmia. The development of such algorithms are motivated by cases like Holter monitoring (24hr monitoring) where constant monitoring of a patient's heart condition is infeasible.

Computer models of ECGs facilitate the evaluation of algorithms used for automatic diagnosis of heart conditions in patients. Algorithms such as R-peak detection, [1, 2], QT-interval detection [3] and the derivation of heart rate and respiratory rate [4,5] are usually tested against ECGs obtained from patients or from the Physionet database [6]. ECGs obtained from these sources have fixed noise levels and sampling frequencies, and obtaining ECGs with specific properties from patients can be difficult. Computer models facilitate the evaluation of such algorithms by providing synthetic ECG signals that can be used to evaluate the behaviour of algorithms under different conditions.

Other applications of computer modelling of ECGs include noise filtering and parameter estimation. Noise filtering is used to reduce noise from ECG signals and has a special use in reducing background noise from fetal ECGs, where the signal itself is weaker than background noise. Parameter estimation is an alternate filtering method used to estimate the parameters to generate an ECG signal from a dynamic model. This can be utilized to generate noiseless ECG signals.

Several dynamic models utilizing different backgrounds of mathematics have been proposed to model ECG signals. Proposed dynamic models include [7–11]. Existing implementations of noise filtering and parameter fitting are extended from [7] and include [8], [12], [13].

Implementations of computer models and their extensions utilize proprietary software such as MATLAB, which is difficult to obtain due to licensing requirements. Open source implementations of computer models [14] exist, but lack features introduced in extensions to computer model. An implementation of a computer model that can generate regular and irregular ECG signals as well as estimate the parameters to produce a signal would be a good base for an open source implementation of a computer model for ECG signals.

This thesis aims to develop an open source alternative to a dynamic ECG model that can generate regular and irregular ECG signals as well as perform parameter estimation on imported ECG samples. Such an implementation would serve as an introduction to an open source computer model of ECGs with better accessibility.

Chapter 2 contains background information required to understand ECGs and a literature review of ECG models and methods of performing parameter estimation. Chapter 3 contains work performed in this thesis followed by Chapter 4, which contains an evaluation of the work done. The report then ends with Chapter 5 which contains a conclusion and possibilities for future work.

Chapter 2

Background

2.1 ECG Signal Collection and Usage

An ECG is a record of electrical activity that occurs through a heartbeat that is used in diagnosing heart conditions. ECG collection occurs by placing electrodes onto the surface of the skin at various positions across the heart and the limbs. The potential difference between electrodes is recorded and produces the ECG signal. ECG data is collected by an electrocardiograph, which is printed onto paper or displayed on a screen. Modern electrocardiographs typically have screens and have built-in interpretation algorithms for ECG intervals.

ECGs can be used to diagnose heart conditions such as arrhythmia or stroke, which occurs when the heart beats too slow, fast or irregular. Some arrhythmia are harmless, although some like ventricular fibrillation can be life-threatening.

Figure 2.1: Modern electrocardiograph machine

2.2 ECG Morphology

A single heartbeat can be observed as a series of deflections from the baseline of an ECG. Deflections on the ECG represent the electrical activity produced by the heart as it initiates muscle contraction. A single cycle of the ECG is associated with a number of deflections that are traditionally labelled as P, Q, R, S and T.

The ECG may be divided into the following sections:

- P-wave: a small voltage deflection caused by the depolarization of the atria prior to atrial depolarization.
- PQ-interval: the time between the start of atrial depolarization and start of ventricular depolarization.
- QRS-complex: the largest amplitude portion of the ECG that represents atrial repolarization and ventricular depolarization. Atrial repolarization is not observed as ventricular repolarization has greater amplitude.

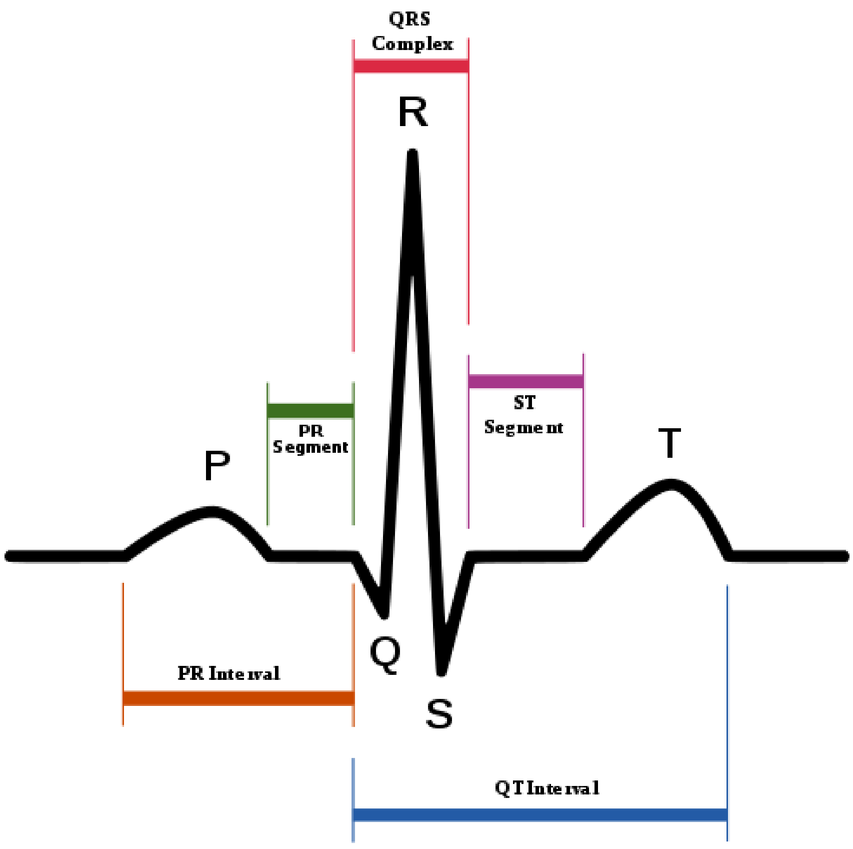


Figure 2.2: Morphology of one PQRST-complex of the ECG

- QT-interval: the time between the onset of ventricular depolarization and the end of ventricular repolarization.
- ST-interval: the time between the end of the S-wave and the beginning of the T-wave.
- T-wave: end of ventricular repolarization, where the heart prepares for the next heartbeat. The period between the T-wave and the P-wave is the same as the baseline for a healthy heart.

2.3 Dynamic Models of ECG Signals

Several ECG models utilizing different mathematical backgrounds have been proposed over time. Of the models, [7] is the most commonly cited and is often used in extensions for modelling ECGs.

2.3.1 ECG Model based on Ordinary Differential Equations

McSharry et al. proposed a synthetic ECG generator based on three dynamic equations of motion. The model generates a quasi-periodic trajectory in a three dimensional (3D) state space with coordinates (x, y, z). The trajectory revolves around a unit radius around the (x, y) plane, which corresponds to a single heartbeat. The points P, Q, R, S and T are described by events placed at fixed angles along the unit circle labelled as $\theta_P, \theta_Q, \theta_R, \theta_S$ and θ_T.

The equations of motion are given by three ordinary differential equations:

$$\dot{x} = \alpha x - \omega y$$

$$\dot{y} = \alpha y + \omega x$$

$$\dot{z} = - \sum_{i \in P,Q,R,S,T} a_i \Delta\theta_i \exp\left(- \frac{\Delta\theta_i^2}{2b_i^2} \right) - (z - z_0)$$

where $\alpha = 1 - \sqrt{x^2 + y^2}$, $\Delta\theta_i = (\theta - \theta_i) \bmod 2\pi$, $\theta = \text{atan2}(y, x)$ and ω is the angular velocity of the trajectory as it moves around the x, y plane. The constants a_i and b_i for $i = 1..5$ are based on the morphology of a healthy ECG.

Index (i)	**P**	**Q**	**R**	**S**	**T**
Time (sec)	-0.2	-0.05	0	0.05	0.3
θ_i (rads)	-$\pi/3$	-$\pi/12$	0	$\pi/3$	$\pi/2$
a_i	1.2	-5.0	30.0	-7.5	0.75
b_i	0.25	0.1	0.1	0.1	0.4

Baseline wander of ECG signals has been modeled with z_0 and respiratory frequency f_2:

$$z_0 = A \sin(2\pi f_2 t), \quad A = 0.15mV, \quad f_2 = 0.25Hz$$

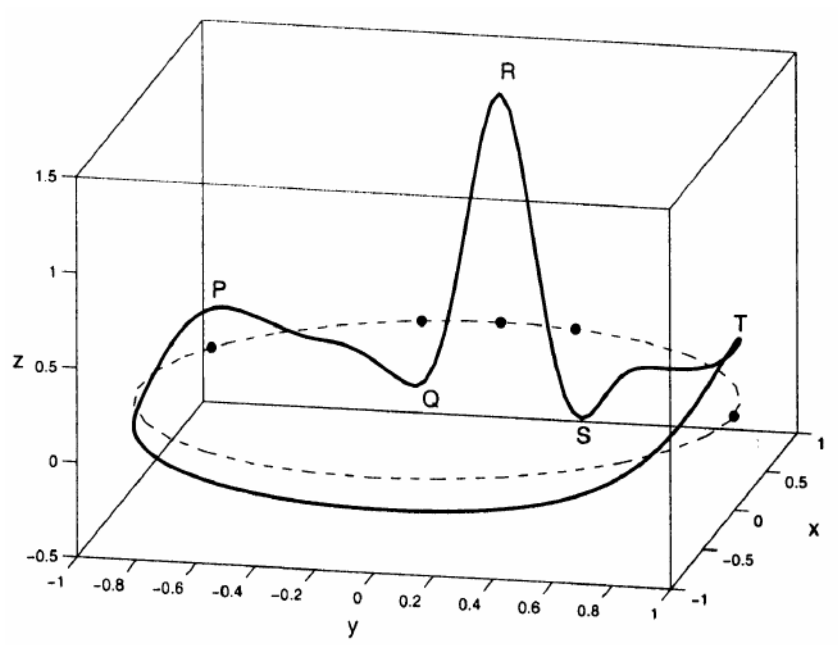


Figure 2.3: Typical trajectory generated by dynamic model

The model introduced by McSharry has an open source implementation [14] that implements the model. Alternate forms of the model are used in [8] and [15]. Extensions to the model include model-based filtering, parameter fitting [12] and compression [16]. Other authors have also used this model in extensions, such as the simulation of fetal ECGs [8], hidden Markov model [17] to account for beat-type changes as well as a wave-based model accompanied by Bayesian Filtering and Kalman smoother [18]. Implementations of these extensions are done in MATLAB.

The prevalence of the McSharry model is attributed to its relative simplicity and flexibility in generating a variety of ECG signals.

2.3.2 ECG Model based on Polar Coordinates

R.Sameni suggested a modification of the McSharry model in [8], where the model is presented in polar form.

$$\begin{aligned} r' &= r(1-r) \\ \theta' &= \omega \\ z' &= -\sum_{i \in P,Q,R,S,T} a_i \Delta\theta_i \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2}\right) - (z - z_0) \end{aligned}$$

This form uses the same parameters for a_i, b_i and θ_i as [7] and presents them in a simpler format. Since the second and third differential equations are independent from the value r, the first differential equation can be omitted without affecting the z value.

The simplified discrete version of the polar form (which can be solved with Euler's method) is presented by:

$$\begin{aligned} \theta[k+1] &= \theta[k] + \omega \cdot \Delta \\ z[k+1] &= -\sum_{i \in P,Q,R,S,T} \Delta \cdot a_i \Delta\theta_i \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2}\right) + z[k] + N \cdot \Delta \end{aligned}$$

where Δ is the sampling time and $\Delta\theta_i = (\theta - \theta_i) \bmod(2\pi)$. N is a random additive noise that models sources of noise in ECG signals such as baseline wander.

The polar form is a simpler modification over [7] since it removes an entire differential equation and focuses only on the z state. The polar form tracks the angle of the trajectory instead of calculating the angle in [7] with $\theta = \text{atan2}(y, x)$. This has benefits as it can provide a second measurement for Kalman Filtering.

2.3.3 ECG Model based on Geometrical Features

P. Kovacs [9] proposed a model based on the geometrical features which gives strict mathematical control over the ECG signal.

The model utilizes 15 base points $x_1, ..., x_{15}$ to find a spline S that satisfies:

$$S^{(i)}(x_k) = f^{(i)}(x_k), \quad (k = 1, ..., 15;\ i = 0, 1, 2)$$

where $f : \mathbb{R} \to \mathbb{R}$ is a time-varying function that represents the ECG curve.

There are 3 values associated with each base point, which makes the model require up to more than 50 parameters.
The base points can be classified into two different classes:

- Diagnostic: represents the base points required for PR, QRS, QT, ST.
- Geometric: represents the positions and amplitudes of P, Q, R, S, T.

Hermite interpolation is used to find the polynomial approximation of the curve. The model uses profiles of lower and upper bounds to create the ECG signal.

The use of profiles can be used to generate characteristic waves of an ECG. When used with the many parameters the model can provide strict control over ECG signals. However, the number of control points makes this model difficult to implement. In addition to the existing parameters, additional parameters would need to be introduced in order to simulate ECGs of heart conditions. Furthermore, the implementation of extensions of interest to clinical diagnosis such as feature extraction, parameter fitting and noise filtering is uncertain for this model.

2.3.4 ECG Model based on Fourier Series

The work of J. Kubicek [10] aimed to elucidate the use of the Fourier series in producing ECG signals and arrhythmia's.

The model describes P and T waves as a composition of repeating sine waves and the QRS complex as ascending and descending triangle waves. Each part is further divided into 11 parts and described by mathematical functions using coefficients a, b, c, d, k, l, m, n to describe the shifts and slopes of the functions.

Model of P and T waves

P and T waves can be approximated by a parabolic function $f(t) = -t^2$.

$$\begin{aligned}
a_0 &= \frac{1}{\pi}\int_c^{c+2\pi} -t^2 dt = \frac{1}{\pi}\left[-\frac{t^3}{3}\right]_c^{c+2\pi} = -\frac{2}{3}\pi^2 \\
a_k &= \frac{1}{\pi}\int_{c+2\pi}^{c} -t^2 \cos(kt)\, dt \\
b_k &= \frac{1}{\pi}\int_{c+2\pi}^{c} t \sin(tx)\, dx = 0 \\
f(t) &= 4\sum_{k^2}^{\infty} \frac{(-1)^k}{k^2} \cos(kt)
\end{aligned}$$

Model of QRS complex

Waves of QRS can be divided into falling and rising functions $f(t) = \pm t$.

$$\begin{aligned}
a_0 &= \frac{1}{\pi}\int_c^{c+2\pi} \pm t\, dt = \frac{1}{\pi}\left[\frac{\pm t^2}{2}\right]_c^{c+2\pi} = 0 \\
a_k &= \frac{1}{\pi}\int_{c+2\pi}^{c} \pm t \cos(kt)\, dt = 0 \\
b_k &= \frac{1}{\pi}\int_{c+2\pi}^{c} -t \sin(kt)\, dx = \pm((-1)^{k+1}\frac{2}{k}) \\
f(t) &= \pm 2\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \sin(kt)
\end{aligned}$$

The model is able to simulate various arrhythmia (tachycardia, bradycardia, atrial flutter). The aim of the model is to be used as a basis for detecting pathological phenomena in ECG and as a proof of theory.

It is worth noting that there are multiple studies involving the use of the Fourier series (and FFT) in ECG analysis. These include detection of cardiac arrhythmia [19], automatic feature extraction of ECGs [20] and noise filtering features [21]. The features included indicate that there is potential for development in this model and its extensions. Unlike other models however, the Fourier series is periodic. This is in contrast to the quasi-periodic nature of the McSharry model, which varies slightly per cycle and would be better for simulating other conditions of heart signals.

2.4 Filtering Noisy ECG Signals using EKF

R.Sameni developed an Extended Kalman Filter (EKF) in MATLAB in [13] for filtering noisy ECG signals.

The Kalman filter is capable of estimating the hidden states of a system with a linear or non-linear model when provided with measurements of the system. The Kalman filter is an optimal estimator under some conditions in the minimum mean square error sense. This means that the filter will eventually converge on some states of a system.

For non-linear models (which is the case for [7]), an Extended Kalman Filter needs to be developed. The Extended Kalman Filter is no longer an optimal estimator, and will diverge if the process model is incorrect or if an improper initial state is given. Nevertheless, the Extended Kalman Filter remains a powerful estimator and is the de-facto standard for navigation systems and GPS.

For a discrete nonlinear system with the state vector x_k and observation vector y_k the dynamic model may be formulated as:

$$\begin{cases} \underline{x}_{k+1} = f(\underline{x}_k, \underline{w}_k, k) \\ \underline{y}_k = g(\underline{x}_k, \underline{v}_k, k) \end{cases}$$

where $\underline{w}_k$ and $\underline{v}_k$ are the process and measurement noises respectively with covariance matrices $Q_k = E\{\underline{w}_k \underline{w}_k^T\}$ and $R_k = E\{\underline{v}_k \underline{v}_k^T\}$.

The initial state estimate of the state $\underline{x}_0$ is defined as $\bar{\underline{x}}_0 = E\{\underline{x}_0\}$ with $P_0 = E\{(\underline{x}_0 - \bar{\underline{x}}_0)(\underline{x}_0 - \bar{\underline{x}}_0)^T\}$

In order to use an Extended Kalman Filter, it is necessary to derive a linear approximation of the model near a desired reference point. This approximation will lead to the following linear estimate:

$$\begin{cases} \underline{x}_{k+1} \approx f(\hat{\underline{x}}_k, \hat{\underline{w}}_k, k) + A_k(\underline{x}_k - \hat{\underline{x}}_k) + F_k(\underline{w}_k - \hat{\underline{w}}_k) \\ \underline{y}_k \approx g(\hat{\underline{x}}_k, \hat{\underline{v}}_k, k) + C_k(\underline{x}_k - \hat{\underline{x}}_k) + G_k(\underline{v}_k - \hat{\underline{v}}_k) \end{cases}$$

where

$$A_k = \frac{\partial f(\underline{x}, \hat{\underline{w}}_k, k)}{\partial \underline{x}}\bigg|_{\underline{x}=\hat{\underline{x}}_k} \qquad F_k = \frac{\partial f(\hat{\underline{x}}_k, \underline{w}, k)}{\partial \underline{w}}\bigg|_{\underline{w}=\hat{\underline{w}}_k}$$
$$C_k = \frac{\partial g(\underline{x}, \hat{\underline{v}}_k, k)}{\partial \underline{x}}\bigg|_{\underline{x}=\hat{\underline{x}}_k} \qquad G_k = \frac{\partial g(\hat{\underline{x}}, \underline{v}, k)}{\partial \underline{v}}\bigg|_{\underline{v}=\hat{\underline{v}}_k}$$

In order to implement the EKF, the time propagation is done using the original nonlinear equation, while the Kalman filter gain and the covariance matrix are calculated from the linearized equations. So the EKF may be summarized as follows:

$$\hat{\underline{x}}_{k+1}^- = f(\hat{\underline{x}}_k^+, \underline{w}, k)\bigg|_{\underline{w}=\underline{0}}$$
$$P_{k+1}^- = A_k P_k^+ A_k^T + F_k Q_k F_k^T$$

and the measurement propagation equations are:

$$\hat{\underline{x}}_k^+ = \hat{\underline{x}}_k^- + K_k\big[\underline{y}_k - g(\underline{x}_k^-, \underline{v}, k)\big|_{\underline{w}=\underline{0}}\big]$$
$$K_k = P_k^- C_k^T \big[C_k P_k^- C_k^T + G_k\big]^{-1}$$
$$P_k^+ = P_k^- - K_k C_k P_k^-$$

where $\hat{\underset{\sim}{x}}_k^- = \hat{E}\{\underset{\sim}{x}_k | \underset{\sim}{y}_{k-1}, \underset{\sim}{y}_{k-2}, ..., \underset{\sim}{y}_1\}$ is an estimate of the state vector, in the k^{th} stage, using the observations $\underset{\sim}{y}_1$ to $\underset{\sim}{y}_{k-1}$, and $\hat{\underset{\sim}{x}}_k^+ = \hat{E}\{\underset{\sim}{x}_k | \underset{\sim}{y}_{k-1}, \underset{\sim}{y}_{k-2}, ..., \underset{\sim}{y}_1\}$ is an estimate of this state vector after adding the k^{th} observations $\underset{\sim}{y}_k$.

P_k^- and P_k^+ are defined in the same manner to be the estimates of the covariance matrixes, in the k^{th} stage, before and after using the k^{th} observation, respectively.

The linearized model with respect to the state variables x, y and z are:

$$\begin{cases} \dot{x} = F(x, y, z) \\ \dot{y} = G(x, y, z) \\ \dot{z} = H(x, y, z) \end{cases}$$

$$\begin{array}{lll}
\frac{\partial F}{\partial x} = \frac{-2x^2 - y^2}{\sqrt{x^2 + y^2}} + 1 & \frac{\partial F}{\partial y} = \frac{-xy}{\sqrt{x^2 + y^2}} - \omega & \frac{\partial F}{\partial z} = 0 \\
\frac{\partial G}{\partial x} = \frac{-xy}{\sqrt{x^2 + y^2}} + \omega & \frac{\partial G}{\partial y} = \frac{-2y^2 - x^2}{\sqrt{x^2 + y^2}} + 1 & \frac{\partial G}{\partial z} = 0
\end{array}$$

$$\frac{\partial H}{\partial x} = \sum_{i \in P,Q,R,S,T} \frac{a_i y}{x^2 + y^2} \exp(-\frac{\Delta\theta_i^2}{2b_i^2})[1 - \frac{\Delta\theta_i}{b_i^2}]$$

$$\frac{\partial H}{\partial y} = \sum_{i \in P,Q,R,S,T} \frac{-a_i x}{x^2 + y^2} \exp(-\frac{\Delta\theta_i^2}{2b_i^2})[1 - \frac{\Delta\theta_i}{b_i^2}]$$

$$\frac{\partial H}{\partial z} = -1$$

Finally, the EKF requires a relationship between the states and measurements (the function g). An ECG measurement contains only one variable z, which is the amplitude of the signal. The observation may thus be related as follows:

$$s_k = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \underset{\sim}{X}_k + \underset{\sim}{v}_k$$

where $\underset{\sim}{X}_k = \begin{bmatrix} x_k & y_k & z_k \end{bmatrix}^T$ with $R_k = E\{\underset{\sim}{v}_k \underset{\sim}{v}_k^T\}$.

Although the EKF is not an optimal estimator, it still proves a powerful filter and is suitable as a base for noise filtering. The described method is used for noise filtering but can be extended to perform parameter estimation by adding variables into the state and adjusting the linearized model.

2.5 Filtering Noisy ECG Based on a Modified ECG Model

R.Sameni later introduced a modified ECG model in polar form [8] which was previously explained. This method uses the same EKF equations used above but with a different model and linearization.

$$\theta[k+1] = \theta[k] + \omega \cdot \Delta$$
$$z[k+1] = -\sum_{i \in P,Q,R,S,T} \Delta \cdot a_i \Delta\theta_i \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2}\right) + z[k] + N \cdot \Delta$$

where Δ is the sampling time and $\Delta\theta_i = (\theta - \theta_i)\ \mathrm{mod}(2\pi)$. N is random additive noise that models sources of noise in ECG signals (such as baseline wander).

In this procedure θ and z are the state variables and the process noise consists of $\omega, a_i, b_i, \theta_i$ and N.

The linearized model with respect to the state variables are:

$$\frac{\partial F}{\partial z} = 0 \qquad \frac{\partial F}{\partial \theta} = \frac{\partial G}{\partial z} = 1$$
$$\frac{\partial G}{\partial \theta} = \sum_{i \in P,Q,R,S,T} -\Delta \cdot a_i [1 - \frac{\Delta\theta_i}{b_i^2}] \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2}\right)$$

Meanwhile the linearization of the model with respect to noise is:

$$\frac{\partial F}{\partial a_i} = \frac{\partial F}{\partial b_i} = \frac{\partial F}{\partial \theta_i} = \frac{\partial F}{\partial N} \qquad \frac{\partial F}{\partial \omega} = \Delta$$
$$\frac{\partial G}{\partial a_i} = -\Delta \cdot \Delta\theta_i \exp\big(-\frac{\Delta\theta_i^2}{2b_i^2}\big)$$
$$\frac{\partial G}{\partial b_i} = -\Delta \cdot a_i \frac{\Delta\theta_i^3}{b_i^3} \exp\big(-\frac{\Delta\theta_i^2}{2b_i^2}\big) \qquad \frac{\partial G}{\partial \omega} = 0$$
$$\frac{\partial G}{\partial \theta_i} = \Delta \cdot a_i [1 - \frac{\Delta\theta_i}{b_i^2}] \exp\big(-\frac{\Delta\theta_i^2}{2b_i^2}\big) \qquad \frac{\partial G}{\partial N} = \Delta$$

In addition to the ECG observation (the amplitude of the signal z), the phase may also be added as a second observation. θ is a periodic value that starts from $\theta = 0$ at the R-peak and ends with the next R-peak. Then the points lying between two R-peaks can be linearly assigned a phase between 0 and 2π, which forms a second observation.

The measurement function g that relates state to measurement thus becomes:

$$\begin{bmatrix} \varphi \\ s_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \theta_k \\ z_k \end{bmatrix} + \begin{bmatrix} v1_k \\ v2_k \end{bmatrix}$$

where φ is the phase observations, s_k is the amplitude of the measured ECG signal and $R_k = E\{[v1_k, v2_k] \cdot [v1_k, v2_k]^T\}$.

The EKF requires a proper estimate of the initial state. The angular frequency ω may be estimated by $\omega = \frac{2\pi}{T}$, where T is the R-R peak period in each ECG cycle. The values for a_i, b_i and θ_i may be calculated by taking the mean of a phase wrapped signal and using an optimization method such as the least-squares method to obtain the estimated values.

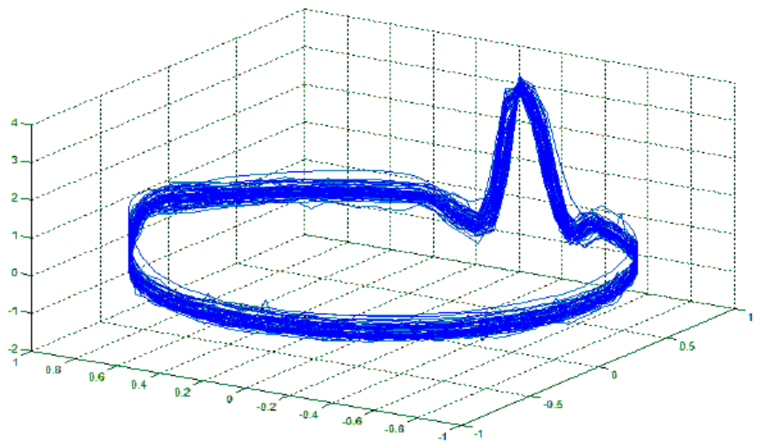


Figure 2.4: Phase-wrapped ECG Signal

This method is an improvement over the previous method by using a simpler model, providing a second observation and automatic estimation of the initial state. This method handles noise filtering, but can be extended to perform parameter estimation by moving variables from the process noise into the state and adjusting the linearized model.

Chapter 3

Implementation

The open source application is implemented with Python3, which was chosen for its simplicity and wealth of supporting libraries for mathematics and GUI development. The application implements a dynamic ECG model capable of generating synthetic ECG signals and parameter estimation with an Extended Kalman Filter.

3.1 Software

SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, the core packages of SciPy consists of NumPy, SciPy library, matplotlib, IPython, SymPy and pandas. The SciPy library provides user-friendly and efficient numerical routines for numerical integration, interpolation, optimization, linear algebra and statistics. It is a widely used library used for math and science comparable to the functionality that MATLAB provides.

NumPy is the fundamental package from SciPy that nearly all aspects of the ecosystem uses to implement its features. NumPy provides a powerful N-dimensional array object to model arrays, vectors and matrices. NumPy also provides a number of highly useful methods for reading text, mathematical functions (sine, cosine, etc), generating random numbers and more.

matplotlib is the plotting library from SciPy which produces interactive environments to generate a variety of plots such as histograms, bar charts, errorcharts, scatterplots and more. The matplotlib library also comes with features that enable full control over line styles, font properties and axes properties through an object oriented interface (or series of functions similar to MATLAB).

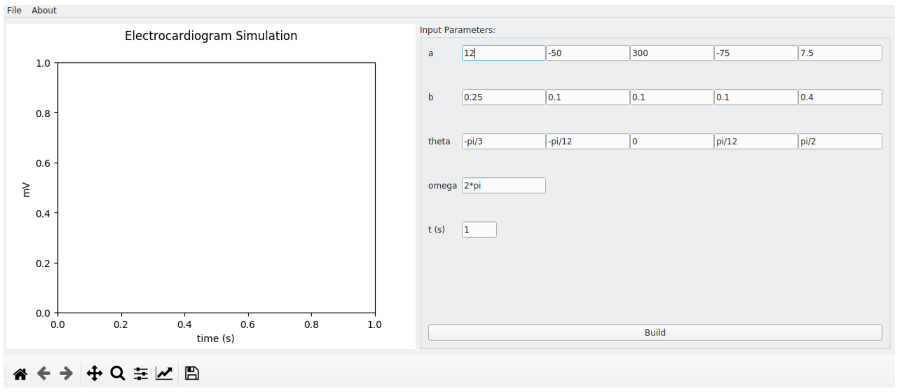
SymPy is a Python library for symbolic mathematics that is part of the SciPy ecosystem. The library aims to become a full-featured computer algebra system that is an alternative to other systems such as Mathematica or Maple while keeping the code simple and extensible.

numexpr is a fast numerical expression evaluator for NumPy. Mathematical expressions written as strings containing variables and a number of math operators can be evaluated into math. Numexpr has multi-threaded capabilities for accelerated computations and is optimized for operations on expressions involving arrays (like "3*a+4*b").

Qt is a modern open-source cross-platform widget toolkit for creating graphical user interfaces (GUIs). Qt comes with a variety of widgets and layouts for different purposes. It's central feature is a signal and slot based mechanism that allows widgets to communicate with another. Qt requires a commercial license for commercial usage but is free for open-source. Qt has bindings for various languages including C++ and Python. Qt offers **PyQt5** and **PySide** bindings for Python. The implementation uses **PyQt5**.

3.2 GUI

PyQt5 was used to implement a graphical user interface (GUI) for user interaction.



```
a      12      -50      300   -75    7.5
b      0.25    0.1      0.1   0.1    0.4
theta  -pi/3   -pi/12   0     pi/12  pi/2
omega  2*pi
t (s)  1
```

Figure 3.1: GUI containing dynamic parameters of ECG model

3.2.1 Input Validation and Evaluation

A form for inputting the 16 parameters of the dynamic ECG model was created. An extra input for time (in seconds) to specify how many seconds of an ECG to generate is also available.

Inputs have input validation to prevent invalid input. Validation is conducted with the PyQt validator property using regular expressions. The time input for seconds of an ECG to generate only accepts numbers, while the parameters for a_i and b_i accepts numbers and decimals.

Inputs involving θ_i and ω allow the use of mathematical operators $(+, -, /, *)$ and the letters "*pi*" in addition to the constraints applied to a_i and b_i. This allows basic input of mathematical expressions involving mathematical operators and *pi*.

```
pi_validator = QRegExpValidator(QRegExp(r"[\d.+\-*/()pi\s]*"))
dbl_validator = QDoubleValidator(decimals=8)
t_validator = QRegExpValidator(QRegExp(r"\d*"))
```

For convenience mathematical expressions involving variables should follow the same rules as algebra. e.g. $2a = 2 * a$. A custom replacer function was used to replace instances of *pi* depending on the mathematical operators around it.

```
pi_regx = re.compile(r"(pi\s*|[0-9.]+)(?=(\s*pi|[0-9.]))")
def pirepl(word):
    def repl(matchobj):
        if matchobj.group(1).strip() == 'pi':
            return "{}*".format(matchobj.group(1))
        elif matchobj.group(2).strip() == 'pi':
            return "{}*".format(matchobj.group(1))
        else:
            return matchobj.group(0)

    return re.sub(pi_regx, repl, word)
```

For instance, $2pi$ becomes $2 * \pi$.

Input forms in PyQt5 use strings, which must be converted into numbers (specifically floats) for evaluation. Numexpr is a third-party open-source package that can perform fast numerical expression evaluation on strings with operators and variables into numbers. Numexpr doesn't handle the keyword *pi* by default, but accepts a map of custom expressions to values. The value returned by numexpr is contained in an array, which can be flattened into a scalar value with NumPy. Combining the custom replacer mentioned above allows evaluation of expressions such as "$2pi$" to 6.28318530718.

```
val = ne.evaluate(_pirepl(val), {"pi": np.pi})
return float(np.asscalar(val))
```

Giving numexpr an invalid expression would cause an error that stops the program. If an error occurs while evaluating an expression the error is caught, an error window appears and the program can continue.

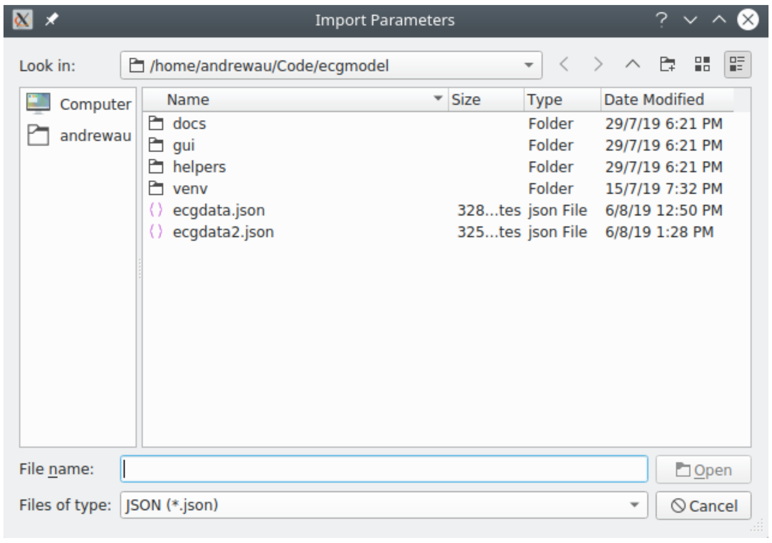
3.2.2 Import and Export Functionality

Common functionality for importing and exporting ECG samples and parameters to the dynamic was introduced.

When importing or exporting parameters, a typical save file window is opened. Exported parameters are taken from the main application form and saved to a file using the JSON file format. This format was chosen for its human readability and the ease of generating and parsing the format. All parameters in the application form (excluding time) are saved inside arrays as string values.

```
{    "a": ["1.2", "-5.0", ...],
     "b": ["0.25", "0.1", ...],
     "evt": ["-pi/3", "-pi/12", ...],
     "omega": ["2*pi"]    }
```

Only JSON files are shown when attempting to import parameters. When a JSON file is imported the array contents are read, evaluated and then written to the main form of the application. If the file contains the wrong amount of parameters, cannot be evaluated to a mathematical expression or number an error message is displayed and the form is not filled.



/home/andrewau/Code/ecgmodel

Name	Size	Type	Date Modified
docs		Folder	29/7/19 6:21 PM
gui		Folder	29/7/19 6:21 PM
helpers		Folder	29/7/19 6:21 PM
venv		Folder	15/7/19 7:32 PM
ecgdata.json	328...tes	json File	6/8/19 12:50 PM
ecgdata2.json	325...tes	json File	6/8/19 1:28 PM

Files of type: JSON (*.json)

Figure 3.2: Import window for exported parameters

Importing and exporting signals uses the CSV file format (column separated values). This format is readable and useful for storing tabular data. The exported CSV file has the same format as CSV files obtained from the Physionet ATM service.

CSV files from Physionet contain a header for each column of comma separated values. Each column contains time and amplitude for the ECG. When importing CSV files, numpy.genfromtext() is used to automatically obtain the data needed. Headers in the file are skipped as the information isn't needed.

```
'Elapsed time', 'ECG1'
'seconds', 'mV'
0.000, −0.165
..., ...
```

To allow importing of a variable length of time, a form is implemented using a combination of PyQt5 QDialog and QSlider widgets. The form is given the maximum time from the imported sample to display a slider. The selected time length from the slider is taken from the imported sample using NumPy style indexing and then projected onto the canvas.

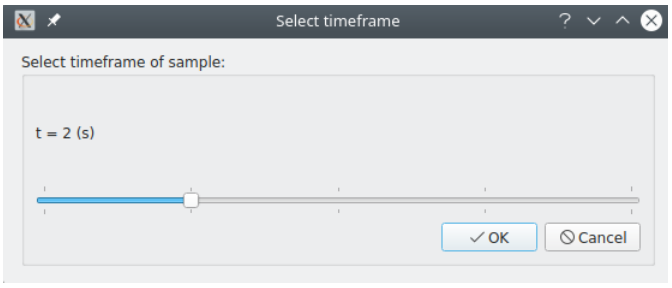


Figure 3.3: Slider for selecting time length of ECG

3.3 Dynamic ECG Model

The implementation of a dynamic ECG model is based off of the McSharry model [7], which consists of a system of differential equations to be solved with an ordinary differential equation solver.

The ordinary differential equation described in [7] was solved with the scipy.solve_ivp() function. The function takes a model, time span, initial value and evaluation times to solve the differential equations.

The original model was slightly modified by multiplying the z value by ω and the a_i values by 10. This causes the model to produce signal amplitudes similar to those found in real patient ECG signals. The model to use with the function is written as follows:

```python
def ecg_model(X, a, b, evt, omega=2*np.pi, z0=0):
    x, y, z = X
    dX = np.zeros(3)
    theta = np.arctan2(y, x)
    dtheta = [(theta - ei) for ei in evt]

    alpha = 1 - np.sqrt(x**2 + y**2)
    dX[0] = (alpha*x) - (omega*y)
    dX[1] = (alpha*y) + (omega*x)
    dX[2] = -(z - z0) - sum(
        ai * omega * dthi * np.exp(-(dthi**2)/(2*bi**2))
        for ai, bi, dthi in zip(a, b, dtheta)
    )
    return dX
```

By default solve_ivp will choose the times to evaluate the solution of the system and will produce uneven plots. A smoother plot can be produced by increasing the number of points to evaluate. The following assigns a linear space of $tf * 100$ points over a time interval of 0 to tf seconds for the function to evaluate.

```
def solve_ecg(a, b, evt, omega, tf=1):
    if not tf:
        tf = 1

    y0 = np.array([-1, 0, 0])
    teval = np.linspace(0, tf, num=tf*100)
    fun = lambda t, y: ecg_model(y, a, b, evt, omega)
    sol = solve_ivp(fun=fun, t_span=(0, tf), y0=y0, t_eval=teval)
    return sol
```

3.4 Plotting

Matplotlib is a plotting library for SciPy that comes with a MATLAB-like interface with functions and an object-oriented interface for full control over elements such as line styles, fonts and axes. The object-oriented interface uses an Axes object to contain plots and axis. Plots are stored inside Line objects while labels, legends and units are stored inside Axis objects.

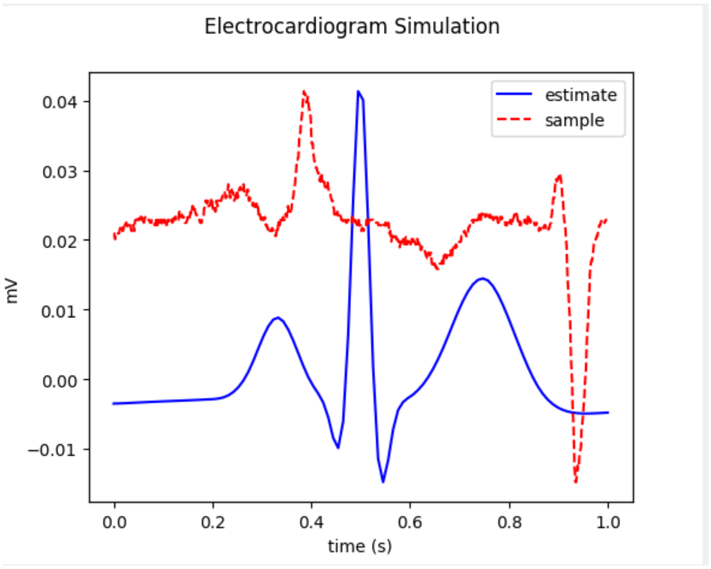


Figure 3.4: Plot of imported sample and generated ECG

The application requires a method to remove existing plots as well as provide labels and legends for plots. Plots must exist inside the same Axes objects and use the same Axes object. To achieve this plotted ECG signals are given names to be used as handlers. Generated ECG signals are named ”estimate” while imported ECG signals are named ”sample”. To remove a plot we search through an Axes object for the corresponding name.

```
def removePlot(self, name):
    for l in self.ax.get_lines():
        if l.get_label() == name:
            l.remove()
        self.redraw_axes()
```

Matplotlib uses the names of Line objects to generate legends for plots. Each time a plot is created or removed the Axes object has its legend recreated and its axis re-scaled to the existing plots. Legends are recreated each time to prevent lingering legends after removing plots.

```
def redraw_axes(self):
    self.ax.relim()
    self.ax.autoscale_view()

    axe_hndl, axe_lbl = self.ax.get_legend_handles_labels()
    if self.ax.get_legend():
        self.ax.get_legend().remove()
    if axe_lbl:
        self.ax.legend(handles=axe_hndl, labels=axe_lbl)
    self.ax.figure.canvas.draw()
```

3.5 Parameter Estimation

Parameter estimation is implemented with the McSharry model model and Extended Kalman filter equation explored in 2.4.

3.5.1 Designing the Process Model

The Extended kalman filter uses the original non-linear equation for time propagation and the linearized equations for kalman gain and covariance matrix calculation. The scipy.solve_ivp() function can be used to solve an ordinary differential equation, but is difficult to modify for use in an EKF.

A simpler alternative is to use Euler's method, which is a simple Runge-Kutta method for solving first order differential equations. This was utilized in [15] to perform Extended Kalman Filtering on ECG signals. The ECG model can be represented with a discrete model (which is the same as applying Euler's method):

$$x[k+1] = (1+\alpha \cdot dt)x[k] - \omega \cdot dt \cdot y[k]$$

$$y[k+1] = (1+\alpha \cdot dt)y[k] + \omega \cdot dt \cdot x[k]$$

$$z[k+1] = -\sum_{i \in P,Q,R,S,T} a_i \cdot \omega \cdot dt \cdot \theta_i \exp\left(-\frac{\Delta\theta_i^2}{2b_i^2}\right) - ((dt-1)z[k] - hz_0)$$

where dt is the change in time between two measurements.

The EKF must track 19 variables x, y, z, a_1, b_1, θ_1 ... θ_5 and ω in its state. These variables are dynamic parameters to the model and remain constant over time.
The discrete non-linear method is hence:

```
Xk = np.zeros(19)
Xk[0] = (1+alpha*dt)*x - (omega*dt*y)
Xk[1] = (1+alpha*dt)*y + (omega*dt*x)
Xk[2] = -((dt-1)*z - (dt*z0)) - sum(
    ai * omega * dt * dthi * np.exp(-(dthi**2)/(2*bi**2))
    for ai, bi, dthi in zip(a, b, dtheta))
```

```
Xk[3:19] = [*a, *b, *theta, omega]
```

The linearization of the model occurs with respect to the state and noise variables. For a state consisting of 19 variables the linearized model becomes a 19x19 matrix.

SymPy is used to symbolically define the discrete model and return the linearized model. The model is defined with symbols and given a state to be differentiated against. The differentiated result can then be converted into a function with sp.lambdify().

```
def state_jacobian():
    ...
    m = sp.Matrix([F, G, H, *a, *b, *e, omega])
    state = sp.Matrix([x, y, z, *a, *b, *e, omega])
    j = m.jacobian(state)
    return sp.lambdify([dt, x, y, z, *a, *b, *e, omega, z0], j)


f = state_jacobian()
f(dt, *xk, z0)
```

Assuming that noise is additive to each state variable the linearized model with respect to noise becomes a simple 19x19 diagonal matrix.

```
F = np.asmatrix(np.eye(19))
```

3.5.2 Designing the Measurement Model

The measurement model is the function relating states to observations. The original model uses $\begin{bmatrix}0 & 0 & 1\end{bmatrix} \underline{X}_k^T + \underline{v}_k$. For a model with 19 states this is simply:

$$g = \begin{bmatrix}0 & 0 & 1 & 0 & ... & 0\end{bmatrix} \underline{X}_k^T + \underline{v}_k$$

The linearization of the measurement model with respect to the state becomes $\begin{bmatrix}0 & 0 & 1 & 0 & ... & 0\end{bmatrix}$. Finally, the linearized measurement model with respect to additive noise to the measurement is a simple constant.

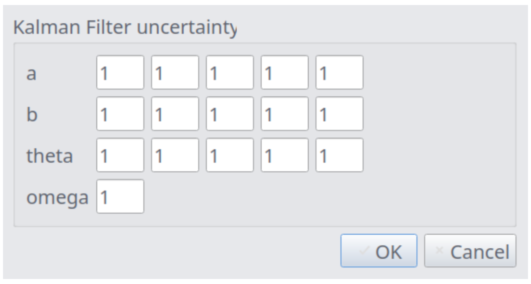
3.5.3 Initial Conditions

The EKF requires an initial state X_0 and covariance P_0.

The initial state variables x, y, z are fixed to $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. The remaining parameters are taken from the GUI form in the application.

The initial covariance P_0 refers to the accuracy of the initial estimate. The value of P_0 can be set to a diagonal matrix, with each value in the diagonal representing the uncertainty in a variable. An individual value set to 0 represents absolute certainty in the initial estimate, whereas values of 1 and higher represent low to high uncertainty.

A short form allowing the change of individual covariances is implemented using simple PyQt Dialog and Entry classes. This form is shown before parameter estimation begins. By default, each value in the form is set to 1.


Kalman Filter uncertainty

a	1	1	1	1	1
b	1	1	1	1	1
theta	1	1	1	1	1
omega	1				

Figure 3.5: Form for modifying initial uncertainty

3.5.4 Filter Implementation

Parameter estimation passes the imported ECG signal, initial state x_0 and initial covariance P_0 to a function which performs extended kalman filtering.

The filter iterates through each time and amplitude observation from the imported signal for a round of state prediction and measurement update. When the filter runs out of observations the state variables $x, y, z, a_1, b_1, \theta_1, ...\theta_5$ and ω are returned and their values are filled into the main application form.

The prediction and update steps convert the state into a column matrix for calculation and array for storage. This is to provide simpler indexing and slicing for functions.

The prediction step uses the original non-linear ECG model in discrete form, which requires the change in time dt from the previous step. For the initial step the value of dt is 0. For every other step of the filter the value of dt is taken by subtracting the time of the current step from the time of the previous step.

The implementation of the extended kalman filter assumes that the model does not contain noise. This changes the covariance update equation in the predict step to:

$$P_{k+1}^{-} = A_k P_k^{+} A_k^T + F_k F_k^T$$

Noise is similarly assumed to not exist in the measurement model. This changes the measurement function $g()$ to:

$$g = \begin{bmatrix} 0 & 0 & 1 & 0 & ... & 0 \end{bmatrix} \underset{\sim}{X}_k^T + 0$$

The remainder of the filter implementation follows the equations described in 2.4.

```
def parameter_est(ts, ys, a, b, evt, omega, p0, z0=0):
    xk = np.array([-1, 0, 0, *a, *b, *evt, omega])
    pk = np.asmatrix(np.eye(19)*p0, dtype="float")

    A = state_jacobian()  # linearized model function
    Fk = np.asmatrix(np.eye(19), dtype="float")

    g = np.matrix([0, 0, 1,*[0 for i in range(16)]])
    Gk = np.matrix([1])
    Ck = np.matrix([0, 0, 1, *[0 for i in range(16)]])

    xs = []
    t_old = ts[0]
```

```
for tk, yk in zip(ts, ys):
    dt = tk - t_old

    # perform state prediction
    x_hat = discrete_ecg_model(dt, xk)
    x_hat = np.asmatrix(x_hat).T

    # perform covariance update
    Ak = A(dt, *xk, z0)
    p_hat = Ak * pk * Ak.T + Fk * Fk.T

    # perform measurement update
    gk = g*x_hat + 0  # vk

    zk = yk - gk
    S = Ck * p_hat * Ck.T + Gk
    K = p_hat * Ck.T * S.I
    pk = p_hat - K*Ck*p_hat

    xk = x_hat + K * zk
    xk = np.array(xk.T, dtype="float")[0]
    xs.append(xk)

    # set old time to current time
    t_old = tk

return (ts, [i[2] for i in xs[:]], xs[-1][3:19])
```

Chapter 4

Evaluation

To test the effectiveness of parameter estimation we test different signals imported into our application.

4.1 Test Setup

Three pre-generated signals with known parameters and length of 6 seconds is generated and tested against the implemented Extended Kalman Filter for parameter estimation.

Parameter estimation is performed with 1 second of the sample, followed by the full 6 seconds of the sample.

The initial state is fixed to the default parameters of a regular ECG signal, and then changed to a value closer to the actual signal. For estimations that diverge, we adjust the initial covariance and observe the changes.

4.2 Results

The following three signals were generated:

1. ECG with a larger P wave amplitude ($a_1 = 50$).
2. Sample with multiple modified attributes ($a_1 = 20, a_2 = -40, a_3 = 290, b_1 = 0.5$).
3. ECG with ω set to 4π.

Each signal was tested with the described test setup. The parameters produced by estimating the full sample can be found in the Appendix.

Estimating 1 second of the first signal produced a signal that had a similar shape but was inaccurate. Taking a longer sample at 6 seconds showed the signal diverging from the sample, owing to the poor initial estimate given to the filter. Adjusting the initial covariance of the a_1 variable from 1 to 200 saw an improvement in the estimate, which while wasn't accurate, did not diverge. Using an initial state that was closer to the sample ($a_1 = 40$) produced a signal that was close to the sample signal, but with different parameters. This suggests that there can be multiple solutions satisfying a signal.

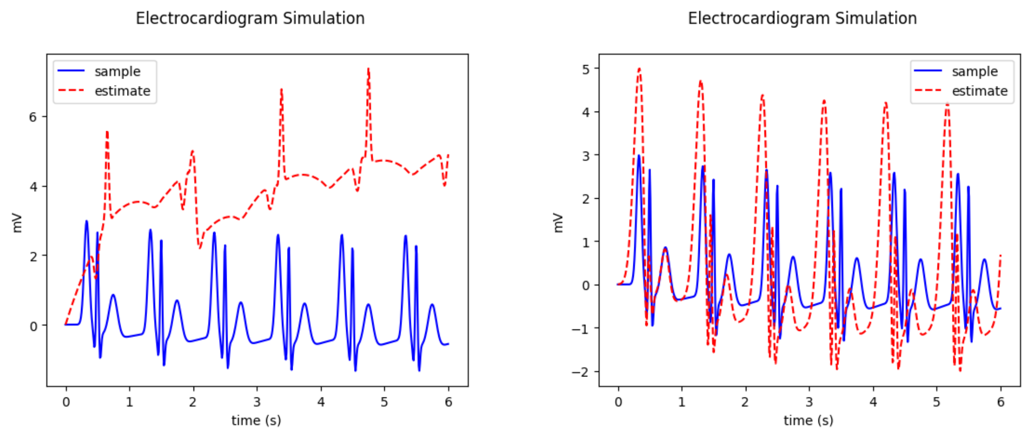


Figure 4.1: Divergent signal and signal generated by increasing covariance.

The estimate of the second signal generated a signal very close to the sample, owing to how close the values of the default initial state is to the sample. The full sample shows that the signal eventually converges on a reliable parameter estimate resembling the sample.

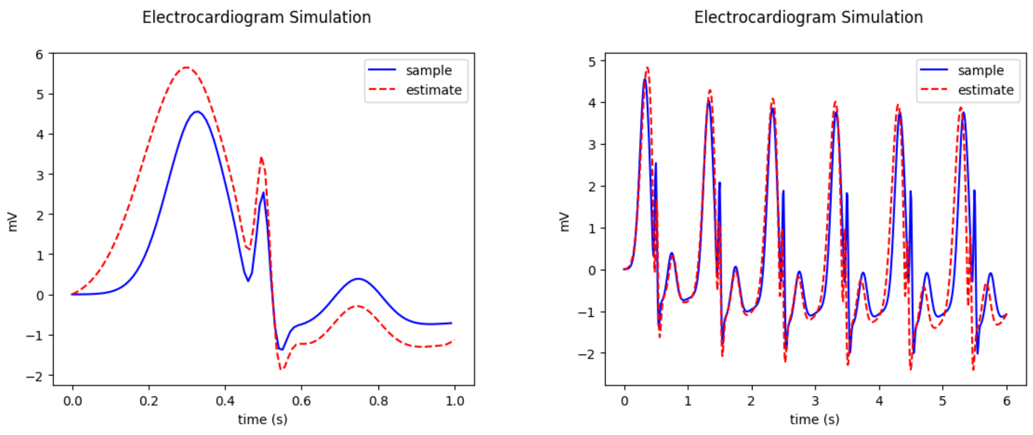


Figure 4.2: Signal generated by taking 1 second vs 6 seconds.

Parameter estimation struggled with the third signal generated with $\omega = 4\pi$. Providing the default state and increasing covariance for ω produced a diverging signal with a 1 second sample.

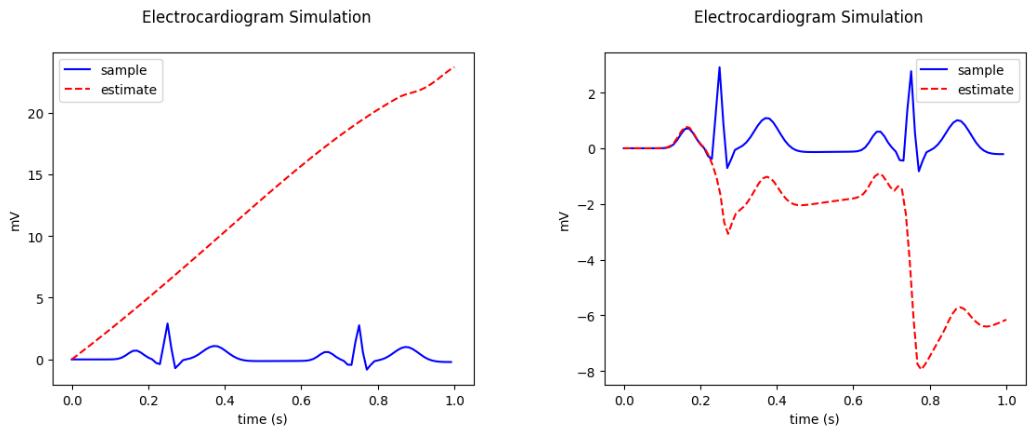


Figure 4.3: Divergent signal with default initial value and disjointed signal with $\omega = 4\pi$.

Providing a close measurement with $\omega = 3.9\pi$ and $\omega = 4\pi$ produced a more accurate estimate. Interestingly, the signal produced with the exact value of $\omega = 4\pi$ produced a disjointed signal for a one second sample. Performing parameter estimation on 6 seconds of the sample with initial state $\omega = 4\pi$ produced a more reliable signal that did not diverge as there were more observations for the EKF to refine its estimate.

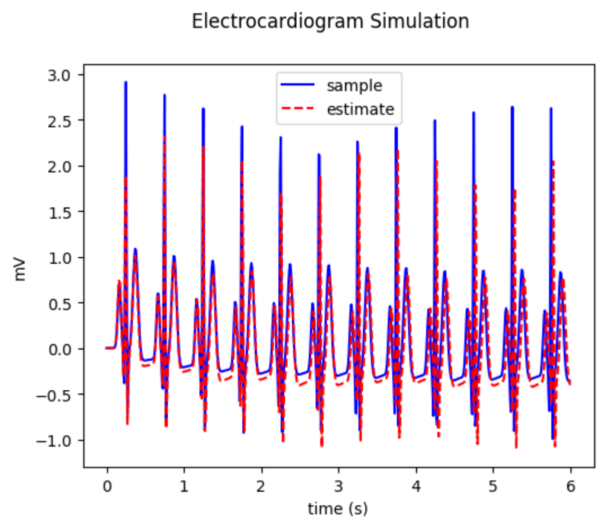


Figure 4.4: Signal produced with a longer sample and better initial estimate ($\omega = 4\pi$).

The Extended Kalman Filter is capable of estimating hidden states but requires a proper initial estimate and a proper process model to prevent the estimate from diverging. The lack of the two properties is likely to produce a diverging signal. The kalman filter also requires a sample of sufficient length to produce a better estimate of the parameters. A longer sample however would not remedy an improper process model or poor initial estimate.

It should also be noted that the prediction step uses Euler's method to predict the next state. Euler's method is effective but requires a sufficiently small step size to obtain an accurate prediction. The tested samples were generated by evaluating the ECG model with scipy.solve_ivp() at $100 * tf$ points linearly assigned between 0 and tf seconds. This effectively gives the signals a step size of $1/100$. Using a larger step size would produce inaccurate (though qualitatively correct) predictions, but would cause diverging due to an inaccurate process model.

While the usage of covariance values can be used to rectify a poor initial estimate, it would be more reliable to automatically find a proper initial estimate prior to kalman filtering.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis focuses on introducing an open source computer model of electrocardiograms with better accessibility for users. The following features were implemented:

- A computer model capable of generating regular and irregular synthetic ECG signals based on a system of differential equations was implemented. This signal would be plotted inside a canvas for user interaction and observation.

- A graphical user interface for user interaction was introduced. The user interface is able to import/export signals and parameters of an ECG signal. Forms for importing timeframe as an ECG signal, inputting dynamic parameters and initial uncertainty for parameter estimation is available.

- Parameter estimation for the dynamic parameters of an imported ECG signal was introduced as an extension to the model, although has some pre-requisites to produce a reliable estimate.

5.2 Future Work

Following on from this project, there are more possibilities to implement for future work:

1. **Noise Simulation** The current implementation does not simulate sources of noise from ECGs, such as baseline wander. It is possible to customize this in the future and introduce random noise to ECGs.

2. **Live plotting** The current implementation generates a static ECG signal. In the future, seeing live plotting of the dynamic model as well as step-by-step parameter estimation would provide interesting insight in how the signal is generated over time.

3. **Automatic estimation of initial state for Parameter Fitting** The current implementation of parameter estimation uses values filled by the user as initial state. The Extended Kalman Filter requires a proper initial state to provide an accurate parameter estimate. To prevent divergent behaviour for parameter estimation, automatic estimation of initial parameters should occur prior to estimation. This process would require a sufficient R-peak estimation method, phase wrapping around a sample ECG signal and potentially a change in model to utilize a second phase observation as mentioned in 2.5.

Bibliography

[1] J. Pan and W. J. Tompkins, "A real-time qrs detection algorithm," *IEEE Transactions on Biomedical Engineering*, vol. BME-32, pp. 230–236, March 1985.

[2] D. T. Kaplan, "Simultaneous qrs detection and feature extraction using simple matched filter basis functions," in *[1990] Proceedings Computers in Cardiology*, pp. 503–506, Sep. 1990.

[3] G. B. Moody, R. G. Mark, M. A. Bump, J. S. Weinstein, A. D. Berman, J. E. Mietus, and A. Goldberger, "Clinical validation of the ecg-derived respiration (edr) technique," *Computing in Cardiology*, vol. 13, 01 1986.

[4] G. B. Moody, R. G. Mark, A. Zoccola, and S. Mantero, "Derivation of respiratory signals from multilead ecgs," *Computers in Cardiology*, vol. 12, 01 1985.

[5] G. B. Moody, R. G. Mark, M. A. Bump, J. S. Weinstein, A. D. Berman, J. E. Mietus, and A. Goldberger, "Clinical validation of the ecg-derived respiration (edr) technique," *Computing in Cardiology*, vol. 13, 01 1986.

[6] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13). Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[7] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith, "A dynamical model for generating synthetic electrocardiogram signals," *IEEE Transactions on Biomedical Engineering*, vol. 50, pp. 289–294, March 2003.

[8] R. Sameni, M. B. Shamsollahi, C. Jutten, and M. Babaie-Zade, "Filtering noisy ecg signals using the extended kalman filter based on a modified dynamic ecg model," in *Computers in Cardiology, 2005*, pp. 1017–1020, Sep. 2005.

[9] P. Kovács, "Ecg signal generator based on geometrical features," *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica*, vol. 37, 01 2012.

[10] J. Kubicek, M. Penhaker, and R. Kahankova, "Design of a synthetic ecg signal based on the fourier series," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1881–1885, Sep. 2014.

[11] P. Dolinský, I. Andras, L. Michaeli, and D. Grimaldi, "Model for generating simple synthetic ecg signals," *Acta Electrotechnica et Informatica*, vol. 18, pp. 3–8, 09 2018.

[12] G. D. Clifford and P. E. McSharry, "Method to filter ecgs and evaluate clinical parameter distortion using realistic ecg model parameter fitting," in *Computers in Cardiology, 2005*, pp. 715–718, Sep. 2005.

[13] R. Sameni, M. B. Shamsollahi, and C. Jutten, "Filtering electrocardiogram signals using the extended kalman filter," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp. 5639–5642, Jan 2005.

[14] J. Healey, G. D. Clifford, L. Kontothanassis, and P. E. McSharry, "An open-source method for simulating atrial fibrillation using ecgsyn," in *Computers in Cardiology, 2004*, pp. 425–427, Sep. 2004.

[15] O. Mohammed Assam, K. Chafaa, M. Ghanai, L. Moreno, and D. Rojas, "Ecg denoising using extended kalman filter," pp. 1–6, 01 2013.

[16] G. D. Clifford, A. Shoeb, P. E. McSharry, and B. B. Janz, "Model-based filtering, compression and classification of the ecg," 2005.

[17] R. Andreão, B. Dorizzi, and J. Boudy, "Ecg signal analysis through hidden markov models," *IEEE transactions on bio-medical engineering*, vol. 53, pp. 1541–9, 09 2006.

[18] O. Sayadi, M. B Shamsollahi, and G. D Clifford, "Synthetic ecg generation and bayesian filtering using a gaussian wave-based dynamical model," *Physiological measurement*, vol. 31, pp. 1309–29, 10 2010.

[19] H. Gothwal, S. Kedawat, and R. Kumar, "Cardiac arrhythmias detection in an ecg beat signal using fast fourier transforms and artificial neural network," *Journal of Biomedical Science and Engineering*, vol. 4, pp. 287–294, 01 2011.

[20] A. F. Haque, M. Hanif, M. Kiber, M. T. Hasan, and H. ++, "Automatic feature extraction of ecg signal using fast fourier transform," *Journal of Bangladesh Electronic Society*, vol. 09, 06 2009.

[21] D. Sadhukhan and M. Mitra, "Ecg noise reduction using fourier coefficient suppression," in *Proceedings of The 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC)*, pp. 142–146, Jan 2014.

Appendix

Signal 1 Parameters										
Signal Values						Estimated Values				
a	50	-50	300	-75	75	40.18	-50	300	-75	7.45
b	0.25	0.1	0.1	0.1	0.4	0.2789	0.0892	0.1012	0.0982	0.3995
θ	$\pi/3$	$\pi/12$	0	$\pi/3$	$\pi/2$	-1.0210	-0.2050	0.0773	0.3478	1.6574
ω	2π					6.3483				

Table A.1: Parameters estimation of signal 1 using initial state: $a_1 = 40$

Signal 2 Parameters										
Signal Values						Estimated Values				
a	20	-40	290	-75	75	13.22	-50	300	-75	7.32
b	0.5	0.1	0.1	0.1	0.4	0.6416	0.2682	0.1275	0.1234	0.4035
θ	$\pi/3$	$\pi/12$	0	$\pi/3$	$\pi/2$	-0.7499	0.1387	0.1349	0.0167	1.7548
ω	2π					6.3878				

Table A.2: Parameters estimation of signal 2 using default initial state of ECG model

Signal 3 Parameters										
Signal Values						Estimated Values				
a	12	-50	300	-75	75	11.73	-50.39	300	-75	7.32
b	0.25	0.1	0.1	0.1	0.4	0.2561	-0.0038	-0.1273	0.1754	0.4026
θ	$\pi/3$	$\pi/12$	0	$\pi/3$	$\pi/2$	-0.9756	-0.9728	0.0703	0.1113	1.641
ω	4π					12.5394				

Table A.3: Parameters estimation of signal 3 using initial state $\omega = 3.9\pi$