JAMES COOK UNIVERSITY

SCHOOL OF ENGINEERING AND PHYSICAL SCIENCES

EG4011/EG4012

Realtime Empirical Mode Decomposition for Intravascular Bubble Detection

Li-aung Yip

Thesis submitted to the School of Engineering and Physical Sciences in partial fulfilment of the requirements of the degree of

Bachelor of Engineering (Electrical and Electronic) - Bachelor of Science (Mathematics)

September 2010

Statement of Access

I, the undersigned, the author of this thesis, understand the James Cook University will make it available for use within the University Library and, by microfilm or other means, allow access to users in other approved libraries. All users consulting with this thesis will have to sign the following statement:

> In consulting this thesis I agree not to copy or closely paraphrase it in whole or in part without the written consent of the author; and to make proper written acknowledgement for any assistance which I have obtained from it.

Beyond this, I do not wish to place any restriction on access to this thesis.

_____ _____

Li-aung Yip Date

Sources Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Li-aung Yip

Abstract

The automatic detection of circulating gas embolisms using continuous-wave Doppler ultrasound would be useful for medical diagnosis and research. The best monitoring site for CW Doppler gas embolism detection is the precordial region, but not much success has been obtained in automatic distinction of embolic signals against precordial background noise. A promising method due to Chappell, using the novel *empirical mode decomposition* (EMD), was investigated.

It was found that converting Chappell's method to a real-time system would require a real-time EMD algorithm, but no good implementations were available. Since the EMD could have many applications for real-time signals processing, a real-time EMD implementation was created; the main difficulty was in handling the EMDs end effects in a block-processing system. The program has been released for public use, testing, and feedback; the author hopes it will become the base for improved implementations.

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

Front Matter

Formal Acknowledgements

This thesis has been made possible by help from:

- My supervisor, Dr. Owen Kenny. (An amazingly tolerant man.)
- Dr. Denise Blake and Mr. Peter Bisaro, Townsville Hospital Hyperbaric Unit, for providing the Doppler ultrasound recordings I used in my research.
- Dr. Michael Chappell, U. Oxford, for giving me pointers on who to talk to about obtaining Doppler ultrasound recordings. (I promised him a postcard, but I never found his address.)
- Dr. Shaun Belward - for letting me borrow freely from his collection of numerical mathematics texts (under threat of a failing mark for MA3605 if he didn't get them back... I appreciate it anyway.)
- Phil Turner - for liberally dispensing his own brand of sage (witty? *evil*? evil sounds right) advice whenever asked.

This document was written in LyX/LaTeX, because *Real Documents* Are Written In LaTeX. The LaTeX layout file is based on Jamie Steven's `phdthesis.sty` layout[1], heavily customised and enhanced by master wizards Bronson Philippa and Christopher Jordan (thanks, guys!)

Sarah and Amender helped proofread, so any spelling or grammar eerors there fault. Blame them.

Personal Acknowledgements

I would like to extend personal thanks to my family and friends for putting up with me, even when I'm sleep deprived and cranky (increasingly common in my old age).

- *Mum and Dad:* Thanks for keeping me in food, housing and Internet for 21 years. I'll pay you back eventually...
- *Billy Rittson:* for being the wise, calm, guiding influence in my life (Billy? The *responsible adult*?! Now that's *just not right*. Get me a bigger shiny thing, I says.)
- *Bronson, Zatta (*Adrian *Zatta) and Michelle:* Holy moly, we made it! We're *real engineers* now! (is that scary or what?)
- *Sarah:* because hugs and cake fix everything! Now we just have to teach her how to cook...

[1] `http://www-ra.phys.utas.edu.au/~jstevens/code_thesis_style.html`

- *Jenelle:* Who *can* cook Chinese food that doesn't taste like the walls at the Red Lantern... but only on a Sunday.
- *Wes:* For not completely freaking out after I crashed your car... into a reflector post... while reversing. A winner is me!
- *Random acknowledgements:* Anyone who's ever shared their knowledge on the Internet, or contributed to free software, with no expectations of repayment; Google (BDFL!), Tux, XKCD, Wikipedia, Everything2, TvTropes, Old Gold chocolate, Nerf guns, Sheaffer, Parker, Lamy, Pelikan, Diamine, and Noodler's Legal Lapis.

xkcd #208, "Regular Expressions."
(Randall Munroe - xkcd.com)

Software Licensing

The MATLAB code developed for this thesis has been made publically available under the terms of the Apache Public License, version 2.0, which may be found at: `http://www.apache.org/licenses/LICENSE-2.0`. The full text of the license is reproduced below.

The latest version of the code may obtained from the author's website `http://www.penwatch.net/`, or by emailing the author: `liaung.yip@ieee.org`. Questions, comments and improvements are welcome.

Apache License Version 2.0

Apache License Version 2.0, January 2004

`http://www.apache.org/licenses/`

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and

You must cause any modified files to carry prominent notices stating that You changed the files; and

You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Nomenclature

Barotrauma rupturing of the lungs due to overpressure.

CW Doppler continuous-wave Doppler ultrasound.

Decompression illness (DCI) umbrella term for DCS and barotrauma.

Decompression sickness (DCS) severe medical condition resulting from the precipitation and expansion of gas bubbles in the body during depressurisation.

Empirical mode decomposition (EMD) a technique for time-domain decomposition of signals into intrinsic mode functions (IMFs.)

Extrema the local minima and maxima of a signal, i.e. the turning points.

Fast Fourier Transform (FFT) method for analysing frequency content of a signal; usually synonymous with the short-time FFT, which analyses frequency content over time.

Gas embolism a gas bubble $> 25\mu m$ in the circulatory system.

Intrinsic mode function (IMF) an oscillatory signal component extracted by the EMD.

Middle cerebral artery (MCA) large artery in the brain.

Precordial region the chest area near the heart.

PW Doppler pulsed-wave Doppler ultrasound.

Subclavian vein either of the two large veins which return blood from the arms (shoulder area).

Stationary, nonstationary signals signals which repeat for all time are stationary; signals which evolve over time are non-stationary.

Contents

1 Background **1**

1.1 Bubbles 1

1.2 Detection of bubbles 3

1.3 Summary 7

2 Literature Review **8**

2.1 Automatic Systems for analysis of CW Doppler audio 8

2.2 Chappell's Algorithm 10

2.3 Empirical mode decomposition 13

2.4 Summary 17

3 Methodology **18**

3.1 Implementation Goals 18

3.2 Implementation method 19

3.3 Verification 19

4 Implementation **20**

4.1 Implementation Architecture 20

4.2 Buffer object 21

4.3 IMF object 22

4.4 EMD Object 27

4.5 Testing Framework 27

4.6 Summary 27

5 Testing and Verification **29**

5.1 Testing parameters 29

5.2 Test vectors 30

5.3 Test Vector 0: Basic functionality check 31

5.4 Test Vector 1 : Execution speed vs. Input Size 33

5.5 Test Vector 2: Execution Speed vs. Number of IMFs extracted 36

5.6 Test Vector 3 : Execution Speed vs. Number of Sifting Iterations 37

5.7 Test Vector 4 : Comparison of output with other implementations 38

5.8 Summary 41

6 Summary and Conclusions **42**

References **44**

A MATLAB implementation of Offline EMD **47**

B User Manual: Online EMD program **49**

B.1 Buffer object 50

C Testing Framework **52**

C.1 EMD_Test() usage 52

C.2 EMD_Test() example 53

C.3 Standard Test Settings 54

List of Figures

1.1 Doppler effect. From URL `http://www.centrus.com.br/DiplomaFMF/SeriesFMF/doppler/capitulos-html/imagens-cap-01/fig-02.jpg` 5

1.2 CW Doppler ultrasound instrument. 5

2.1 A simplified explanation of Chappell's method. 11

2.2 How the empirical mode decomposition assists in detecting anomalous features. 12

2.3 An illustration of the EMD concept. 14

4.1 Streaming EMD – cascading "waterfall" structure of calculations 20

4.2 Streaming data architecture - `Buffer`, `EMD` and `IMF` objects. 21

4.3 End effects in blockwise IMF calculation (exaggerated for illustrative effect.) . 22

4.4 Cubic splines (left) vs. Hermite splines (right.) 23

4.5 End effects and their reduction via "mirrorisation." 25

4.6 Details of the mirrorisation process for the end of the interpolation interval. . . 25

4.7 Overlapping block calculations to reconstruct the "true" IMF without end effects. 26

5.1 First 5000 samples of an online EMD calculated with default settings. 32

5.2 Profiler data - 20 runs of basic test using 'Hermite' interpolation method 32

5.3 Execution speed for different input lengths. 35

5.4 Execution speed for different input lengths. 36

5.5 Execution speed for different numbers of sifting iterations. 38

5.6 Output of online and offline implementations for Signal #1 (simple). 39

5.7 Output of online and offline implementations for Signal #3 (real world) 40

1 Background

This thesis was initially motivated by the problem of detecting circulating bubbles (*gas embolisms*) in the human body. These bubbles can cause a wide variety of symptoms (up to and including death), so research into methods for their detection and characterisation are of great interest. In particular, *automatic* systems for bubble detection would be useful for medical and research applications.

We will now outline the nature of gas embolisms, and current methods for their detection.

1.1 Bubbles

The presence of *small* gas bubbles in the body is harmless; deliberate injections of gas microbubbles can even be clinically useful [1, 2]. It is the *larger* bubbles > $25\mu m$ which cause problems.

Gas bubbles of this size can be introduced into the body in several ways, including:

- **Invasive procedures:** hypodermic injections contaminated with air, compressed air accidents, surgical procedures[3], etc.
- **Decompression:** Breathing pressurised air drives a large amount of inert gas into the blood. If subsequent decompression is *gradual*, the gas is harmlessly eliminated via the lungs, but rapid decompression causes the gas to precipitate as bubbles *in situ* [4].

- **Barotrauma:** Keeping breath held while decompressing will rupture the lungs due to overpressure, driving air into the bloodstream. A mere 1-1.3 metres ascent in water (with breath held) can cause this to occur [5, 6]. [1]

Gas bubbles in the body can lead to serious medical complications [7, 8, 6, 5]:

- During rapid decompression, bubbles form in the body and expand (Boyle's Law[2]), causing decompression sickness (*the bends*.) In addition to causing extreme pain, the bubbles damage surrounding tissues as they expand.
- Gas bubbles in the bloodstream > 25μm (*gas embolisms*) can lodge in blood vessels, cutting off the blood supply to parts of the body. Large gas embolisms do this *very* effectively. Obstruction of the pulmonary circulation causes "the chokes"; obstruction of the coronary artery causes heart attack; obstruction of the blood supply to the brain can cause neurological damage or stroke. Less than 1mL of gas bubbles (total) can cause death [8, p120].

The only treatment for air embolisation is recompression in a hyperbaric chamber. Delays in giving treatment lead to increased risks of permanent injury or death [8, 9].

1.1.1 SCUBA Diving

In SCUBA diving literature, decompression sickness (the bends) and barotrauma are grouped together under the heading of *decompression illness* or DCI.

The development of safe dive protocols to avoid DCI has historically been by experimental research - if a diver suffered DCI after executing a particular *dive profile*, that dive profile was considered to be unsafe. Data from many dives gave enough information to build dive tables, defining the envelopes for safe diving profiles. Even so, the dive tables are not applicable to all individuals; many factors may predispose a diver to decompression sickness, so that even "safe" dive profiles can lead to DCS .

The symptoms of DCI will usually be immediately obvious - bloody froth from the lungs indicates barotrauma, while the bends, chokes, itches, or staggers are all indicators of DCS [6]. This is not always the case, however. Sometimes the onset of symptoms after a dive can occur well after a diver has reached the surface - see the case studies in [8, p156] for examples.

1.1.2 The need for automatic bubble detection systems

Non-invasive means of detecting and characterising gas emboli in the human body have several applications. Firstly, portable bubble detection equipment would be useful in *diagnosing* cases of gas embolisation in divers before the onset of symptoms, facilitating pre-emptive

[1] Interesting note: Hypothetically, if you are ever thrown out of an airlock into space, you must avoid barotrauma by breathing *out*. You can survive without air for three minutes, provided that you *still have lungs to breath with afterwards*.

[2] Boyle's law: $PV = nRT$. A decrease in gas pressure results in expansion of the gas.

treatment. Secondly, bubble detection and characterisation is useful for *research* purposes, such as the development of better dive protocols, or investigating the mechanisms of bubble formation.

JCU, for example, is currently cooperating with Queensland Health in researching the effect of helicopter vibrations on increased precipitation of bubbles into the bloodstream. This is significant because patients with DCI are often flown to treatment facilities via helicopter; it is hypothesised the vibrations could be precipitating more bubbles into the blood, *worsening* the patient's condition.

Usual methods for bubble detection rely on use of medical ultrasound, with the ultrasound data being analysed by trained personnel in real time [8, 10]. (This will be discussed more below.) There are two problems, however, arising from human factors:

1. Human involvement introduces subjectivity, which is a problem for objective research [11].
2. Constant attention is required, which limits the amount of time that can be spent monitoring for bubbles. Low densities of emboli, which can still be clinically significant, may not be detected [3].

An automatic bubble detection system would have two main benefits. Firstly, automatic methods would give far more consistent and repeatable results than humans; secondly, an automatic system would not require constant attention , so the detection of low emboli densities can be achieved by monitoring for days or hours at a time.

Finally, it should be noted that automatic bubble detection techniques may have wider applications. Solid embolisms in the bloodstream (blood clots, fat droplets) are just as clinically important as gas embolisms; detecting these is *similar* to detecting gas emboli, but more difficult [3].

1.2 Detection of bubbles

Medical ultrasound instruments are the primary means of embolus detection in practical use [10]. Several other ways of detecting gas embolisms in humans have been explored, including methods based on the measurement of electrical impedance or optical sensors [12]; these methods remain experimental in nature, and have not found practical applications.

1.2.1 Ultrasound principles

The basic principles of ultrasound are simple. A piezoelectric *transducer* generates a pulse (or continuous wave) of high-frequency vibrations - a sound wave - and this sound wave propagates through the body. Echoes are produced whenever a difference in acoustic impedance (density) is encountered [13]; the echoes can be picked up with another transducer, and yield information about structures inside the body [12, 14, 15] :

- The *amplitude* of the echoes give information about the size and composition (density) of the reflectors. Reflectors with greater differences in density return a greater *fraction* of incident energy; bigger reflectors return larger echoes because they intercept more energy in total.
- The *lag* between transmitting the pulse and recieving the echoes tells us the *distance* to the reflectors.
- The *frequency shift* of the echo tells us the velocity of the reflector (Doppler effect - Figure 1.1)

Different ultrasound instruments are constructed to interpret the echo information in different ways .

- CONTINUOUS-WAVE (CW) DOPPLER uses two transducers: one continuously transmitting a sound wave at frequency f_0, and another constantly receiving the echoes. The focal point of the two transducers defines the instrument's "sensitive zone" [16, p534]. (Figure 1.2.)
  The received echoes are demodulated so the echoes at f_0 (stationary objects) are removed and the Doppler shifts are in the audible range. The output is a continuous audio signal, which is listened to by a human expert. This method can determine the amplitude of the echoes (corresponding to reflector size/composition) and Doppler frequency shift (corresponding to reflector velocity), but no depth information is obtained.
- PULSED-WAVE (PW) DOPPLER is analogous to time-domain reflectometry. A single transducer is used alternately to send a pulse of ultrasound, and then recieve the echoes. It can be considered a depth-selective version of continuous-wave Doppler, returning information about the *depth* of reflectors in addition to their size and velocity [16, p535].
- ULTRASOUND IMAGING is exactly analogous to radar or echolocation. Pulses of ultrasound are used (as in PW Doppler), but an entire phased *array* of transducers are used to receive the echoes (not just one), yielding echo *direction* information in addition to depth and amplitude.
  The output of an ultrasound imaging instrument is generally a 2D image showing a cross-section of the body. More expensive instruments can also produce 2D colour Doppler overlays and 3D images, as well as replicating the functionality of CW or PW Doppler at the same time.

Other, more specialised, ultrasound instruments exist. One of these is the DUAL-FREQUENCY ULTRASOUND, which uses two transmitting transducers; one at a "pump" frequency which induces resonance in certain objects of interest only, and one at an "image" frequency as in normal ultrasound [17]. These instruments are not widespread and they will not be further considered.

1.2.2 Ultrasound bubble detection

Embolisms are detected by using ultrasound to monitor a large blood vessel. Suitably large blood vessels include the subclavian veins (arm/shoulder), femoral arteries (thighs), middle cerebral artery (brain), or the precordial region (heart) [10].

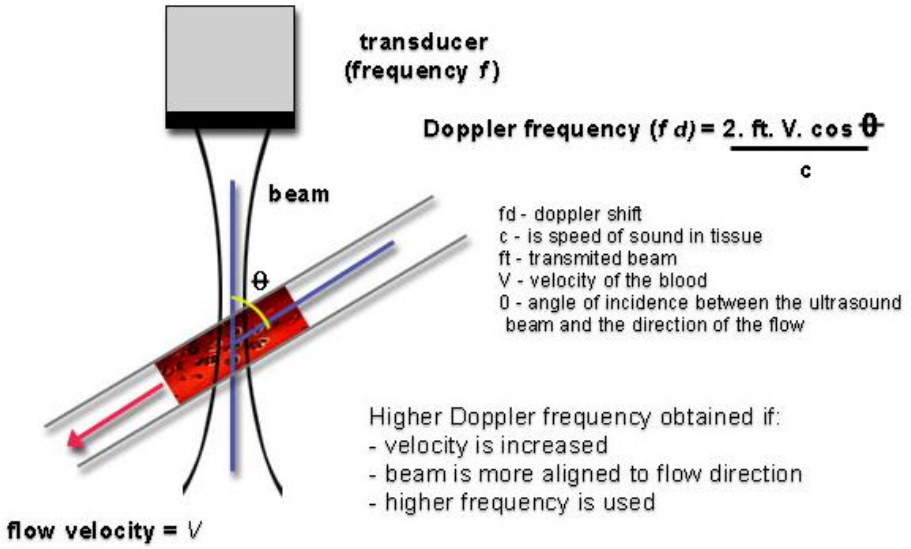


Figure 1.1: Doppler effect. From URL `http://www.centrus.com.br/DiplomaFMF/SeriesFMF/doppler/capitulos-html/imagens-cap-01/fig-02.jpg`

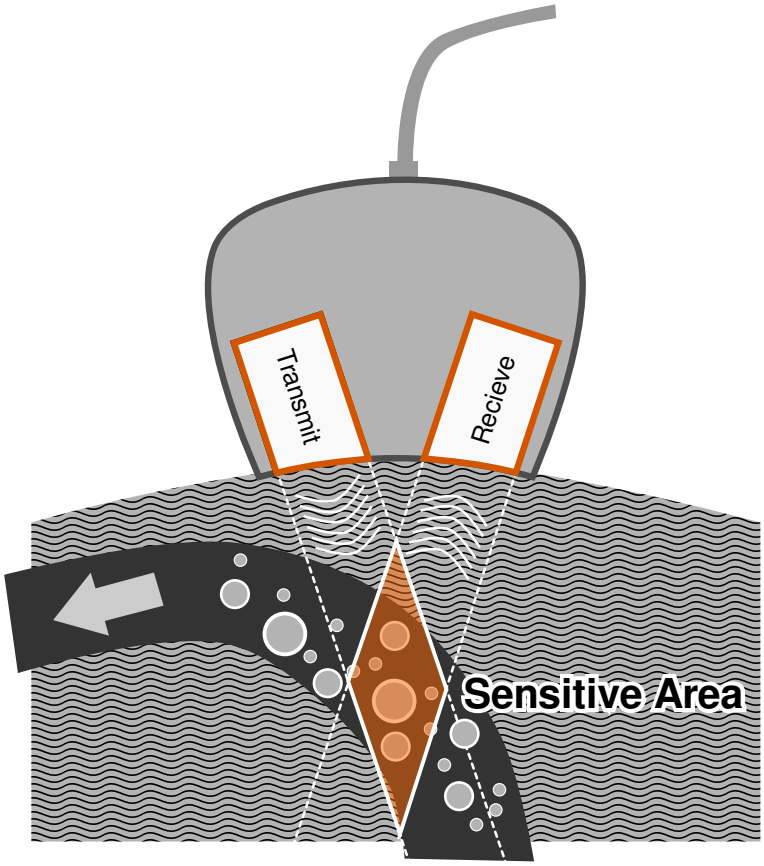


Figure 1.2: CW Doppler ultrasound instrument.

The choice of monitoring site is important because bubbles in the blood are mostly trapped during the blood's passage through the lungs [8, p131]. Monitoring of the subclavian vein, for example, will only intercept bubbles that *originate* in the arms. Similarly, monitoring the femoral artery will only detect bubbles originating in the legs, and monitoring the MCA will only detect bubbles that form in the brain.

The precordial region (heart), however, is the crossroads for all circulating blood in the body. *All* blood passes through the heart before the lungs; therefore, if we monitor for bubbles in the heart, this is in some sense[3] equivalent to monitoring the *entire* circulation. Therefore monitoring of the precordial region is preferred where possible [10, 18].

CW Doppler instruments are the *most commonly used* because of their wide availability, low expense, and robust construction; this is offset by the level of skill needed to correctly position the instrument and interpret the audio signal that is produced [10].

PW Doppler is rarely used, because although the instrument theoretically gives good depth discrimination, it is more expensive and much harder to operate than simple CW Doppler, and gives little advantage in practice [10]. Dual-frequency ultrasound units, while *designed* for bubble detection, are similarly rarely used due to their expense and relatively recent development [17, 12].

Ultrasound imaging is a possibility [12, 19, 13], but the trans-thoracic echocardiography (TTE) technique is an invasive procedure; non-invasive Doppler ultrasound is typically preferred.

1.2.3 CW Doppler bubble detection procedures

In CW Doppler monitoring, the instrument is placed so that the sensitive zone is aligned with a large blood vessel. (Figure 1.2.) The output signal is then interpreted aurally (by ear.)

Bubbles in the circulation are exceedingly good reflectors of sound [14, 20], and move with the same speed as the blood; therefore, circulating gas embolisms return high-amplitude reflections with a Doppler shift. (Figure 1.1.)

Typical bubbles typically take about 10ms [18] to transit the monitored volume, producing transient "chirps" or "warbles" in the Doppler audio output; bubbles as small as 40μm can be detected by listening for these distinctive *embolic signals* [8, p121].

Though bubbles produce the loudest echoes (for their size), *any* moving object in the monitored area will produce Doppler-shifted echoes as well (*background noise*.) For example, the blood itself contains millions of blood cells in motion; while each blood cell only reflects a small amount of sound, the sum of the echoes from many blood cells is significant, and manifests in the output signal as a quiet, periodic "whoosh" [8].

For monitoring of the extremities (subclavian vein, middle cerebral artery, femoral artery, etc.) this "whoosh" is the main source of background noise; the bubble signals are usually clearly audible above it, making detection of bubbles in these parts of the body comparatively easy [10].

[3] Some bubbles may dissipate or become stuck before they reach the heart. [REF]

Recall from above, however, that monitoring of the extremities may miss some bubbles after they are filtered out by the lungs. We would therefore like to monitor the precordial region, through which all blood (and therefore all bubbles) must pass. This is complicated by the high level of background noise in the precordial region; according to Nishi, et. al. [10], "the signal in the precordial region is extremely noisy, with contributions from the blood flow, heart valve action, and heart wall motion."

These background noises can sound very similar to embolic signals [18], and differentiating them is difficult. It takes well-trained personnel to produce reliable and consistent results [10].

1.3 Summary

The presence of bubbles in the human bloodstream, termed *gas emboli*, can be potentially fatal. Ultrasound instruments are capable of detecting bubbles circulating in the blood; the most commonly used is the continuous-wave Doppler ultrasound, which is cheap and widely available.

The output of the CW Doppler instrument is an audio signal which is interpreted aurally (by ear); the difficulty of interpreting the output is dependent on the monitoring site. The extremities are easy to monitor for bubbles, but some bubbles may be missed. The heart (precordial region) is therefore the best monitoring site, as all bubbles will pass through the heart, but the high level of background noise in the precordial region makes interpretation of the Doppler signal difficult.

An automatic system for bubble detection would be desirable for both medical and research purposes. We will discuss the current state of automatic bubble detection systems in the next section.

2

Literature Review

2.1 Automatic Systems for analysis of CW Doppler audio

The basic challenge in designing a CW Doppler based automatic bubble detection system is to correctly detect the presence of *embolic signals* (ES) in a Doppler audio signal. This is complicated by the presence of background noise due to the normal operation of the body.

For monitoring of the extremities, the background noise is relatively quiet and this is not an issue; but measurements of bubbles at the extremities do not generalise to the rest of the body. We would like to monitor the precordial region because bubble formation in *any* part of the body can be detected there.

The problem is that the precordial region has many sources of background noise which may overpower the ES, and which may *sound* very similar to ES as well - i.e. they may have very similar time-frequency characteristics. This makes implementation of automatic bubble detection methods for the precordial region much more difficult than for the extremities.

2.1.1 Fourier/filter-bank techniques

Conventional time-frequency techniques, based on the FFT or frequency-selective filters, perform well when analysing signals that contain relatively little background noise.

- As early as 1977, Kisman was able to use the plain FFT to obtain good bubble detection rates with Doppler probes implanted *directly adjacent* to the vein of interest, thereby

minimising the presence of artifacts in the signal [21]. While this work showed that automatic detection of emboli was possible, the highly idealised experiment involved invasive procedures and the processing of a signal with virtually no artifacts, neither of which apply to our case.

- Tufan, et. al. developed a program that utilised the Teager Energy Operator and a simple bandpass filter, which reliably detected embolic signals in the subclavian vein [11]. Again, this signal is relatively low in background noise, which makes detection of ES easy.
- Markus, Cullinaine and Reid used the conventional FFT in an online program to detect emboli from trans-cranial Doppler measures [22]. This program was shown to perform about as well as a panel of human experts [3]. Once again, this signal has very low background noise.

The literature abounds with many more Fourier-based systems that can successfully detect embolic signals in the cerebral circulation or peripheral veins, where there is relatively little background noise [12].

2.1.2 Wavelet techniques

The discrete wavelet transform (DWT) is another method for the time-frequency analysis of signals. Like the Fourier transform, which decomposes signals into a weighted sum of sinusoids, the DWT decomposes signals into a weighted sum of *wavelets*. Unlike the Fourier representation of a signal, which retains only frequency information, the wavelet representation contains both time and frequency information; this makes it more suited to the analysis of nonstationary signals [23] (i.e. signals which change with time.)

Since Doppler audio signals are an example of a nonstationary signal, there have been several attempts to apply the DWT:

- Lui, et. al. implemented a wavelet-transform based analyser which detected very large gas embolisms injected into dogs [24]. This was found to be useful in context, but does not relate well to detecting the small bubbles typical of DCS in humans.
- Aydin, et. al. used the DWT to analyse very short samples of transcranial Doppler audio signals. Each sample contained one feature in isolation - an artifact, Doppler speckle, or an embolic signal; their algorithm was able to correctly classify the vast majority of 300 such samples [25].

While there have been no publications about applying the wavelet transform to the analysis of precordial Doppler recordings, this does not imply that the wavelet transform is unsuitable [12]. Chappell even suggests that it may be an avenue of future research [18].

2.1.3 Other techniques (not using CW Doppler)

Eftedal et. al. [19, 12] were able to implement a system that used high-resolution ultrasound imaging of the precordial region to detect bubbles *graphically* (as opposed to Doppler audio.) It was found that even relatively untrained personnel were able to count individual bubbles in the ultrasound images, and a computer program for doing this automatically was also developed.

The high resolution of the images was made possible by a technique known as trans-esophagic echocardiography (TEE), in which the ultrasound probe is inserted down the throat; this allows placement of the probe very close to the heart. Unfortunately this procedure carries a risk of perforating the oesophagus, so the research was limited to animal experiments only.

2.1.4 Empirical mode decomposition

The *empirical mode decomposition* (EMD) is a relatively new method for analysing non-stationary, non-linear signals [26]. Like the Fourier or wavelet transforms, the empirical mode decomposition reduces a time signal into a set of basis signals; unlike the Fourier or wavelet transforms, however, the basis functions are *derived from the data itself*. Each basis function of the EMD, known as an intrinsic mode function (IMF), captures the repeating behaviour of the signal at some particular time scale.

There has been one attempt by Chappell et. al.[27, 18] to apply the EMD to the off-line detection of emboli in the precordial region. Their approach relies on the regular occurrence of artifacts compared to the transient nature of the embolic signals.

Chappell claims that this method achieves excellent sensitivity when run on precordial Doppler audio recordings (nearly all bubbles are detected), though specificity (false positives) are a problem [18]. As this is the only known successful attempt at analysing precordial Doppler audio, we will turn our attention to the specifics of Chappell's method.

2.2 Chappell's Algorithm

2.2.1 Principle of operation

Chappel's algorithm, for offline analysis of precordial Doppler recordings, is described in his paper [18]. Although the details are somewhat complicated, the basic concept is remarkably simple. It consists of four main stages: *heartbeat detection*, *empirical mode decomposition*, *statistical comparison of heartbeats* and *feature classification*.

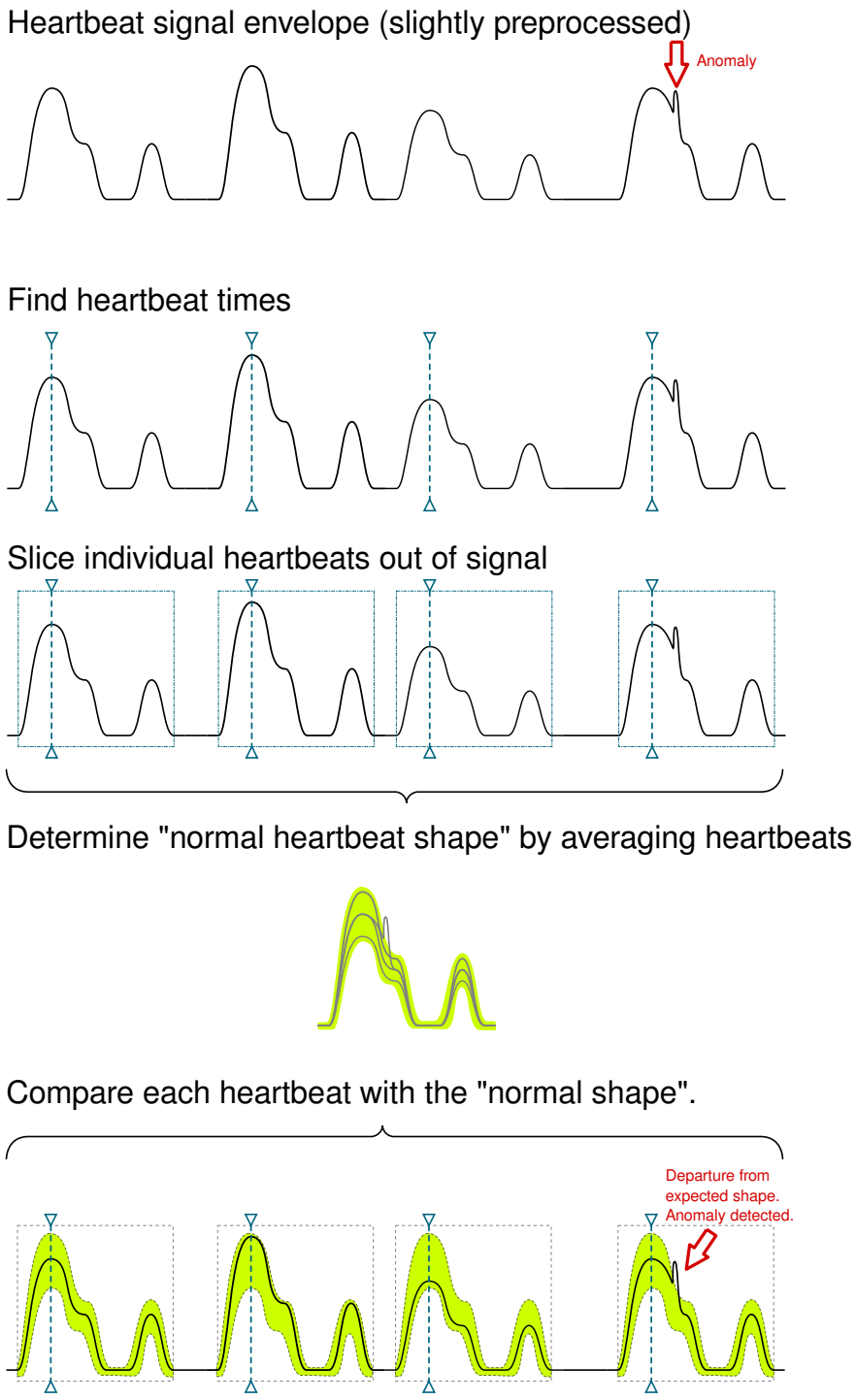
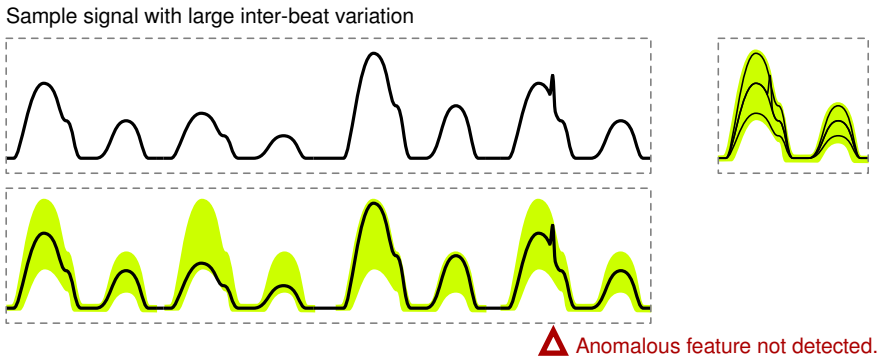
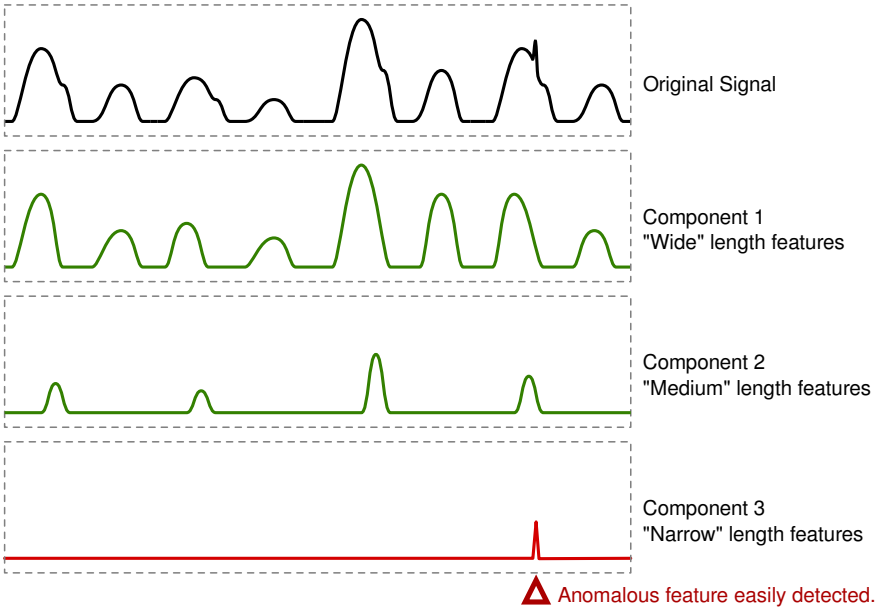


Figure 2.1: A simplified explanation of Chappell's method.

The principle of operation is illustrated in Figure 2.1. The premise is that bubble-free precordial Doppler recordings will contain many heartbeats which should *look* the same - i.e. their *envelopes* should be very similar. The many individual heartbeats in the signal are detected (*heartbeat detection*) and averaged together to estimate what an average heartbeat should look like, as well as the margin of variation that can be expected.



(a) The simple algorithm fails to work where the heartbeats have the same shape, but very different heights.



(b) Separation of components using empirical mode decomposition.

Figure 2.2: How the empirical mode decomposition assists in detecting anomalous features.

Once we have an idea of what the normal heartbeat should look like, we can compare this to each of the original heartbeats to see if any of the heartbeats contain unusual features (*statistical comparison*.) Small deviations will be ignored if they do not vary too far from the expected shape, but large excursions will be detected as *anomalous* features that cannot be part of a normal heartbeat. These anomalies are flagged for *feature classification* into either bubble or non-bubble signals.

The problem with this approach is that if the individual heartbeats vary widely as a matter of course, small features of interest may be lost against the background of normal variations in the signal. Figure 2.2a shows this occurring in a highly idealized signal where the last heartbeat contains an obvious anomaly. Because the heartbeats vary so much in amplitude, the margin of expected variation is very wide and the anomaly does not exceed the threshold for detection.

The root problem is that the variability in the large repeating features overpowers the small features of interest. To resolve this, Chappell applied the EMD to decompose the Doppler audio into components based on the time-scales (widths) of features in the signal. Figure 2.2b

is an idealized illustration of this concept. The large repeating features all have the same width (regardless of amplitude) and are separated away from the very short-duration anomaly in the last heartbeat; the simple procedure outlined above is then applied to *each component*. This effectively isolates the large variations into their own components, while the anomaly in the third component is easily detected and flagged for investigation.

2.3 Empirical mode decomposition

As mentioned above, Chappell was effectively able to increase the signal to noise ratio for anomalous feature detection by decomposing the signal based on the *time scales* of the features, irrespective of their amplitudes. To do this Chappell used the empirical mode decomposition, a signals-processing algorithm first developed by Huang, et. al. in 1998 [26]. We will describe this algorithm now.

2.3.1 Motivation

The root problem with traditional signal processing techniques is their limited time-frequency resolution.

Consider the FFT as an example. We can either choose a small FFT size and obtain good time resolution, or a long FFT size and obtain good frequency resolution, but not both. We will always be uncertain about either a feature's timing in the signal or its frequency content. Since bubbles and normal precordial Doppler features are so close in time and frequency, distinguishing them requires accuracy in *both*.

The wavelet transform allows both high time and frequency resolution, and was suggested by Chappell as a possible approach. However, while the wavelet transform can yield both time and frequency precision, this comes at great computational cost; this often makes it unsuitable for real-time applications.

The empirical mode decomposition is an alternative to these techniques. The FFT and wavelet transforms attempt to transform a signal from the time domain to the frequency domain; the EMD does away with the concept of a "frequency domain", dealing with the time domain only. Therefore "lack of frequency resolution" is not an issue.

2.3.2 Basics

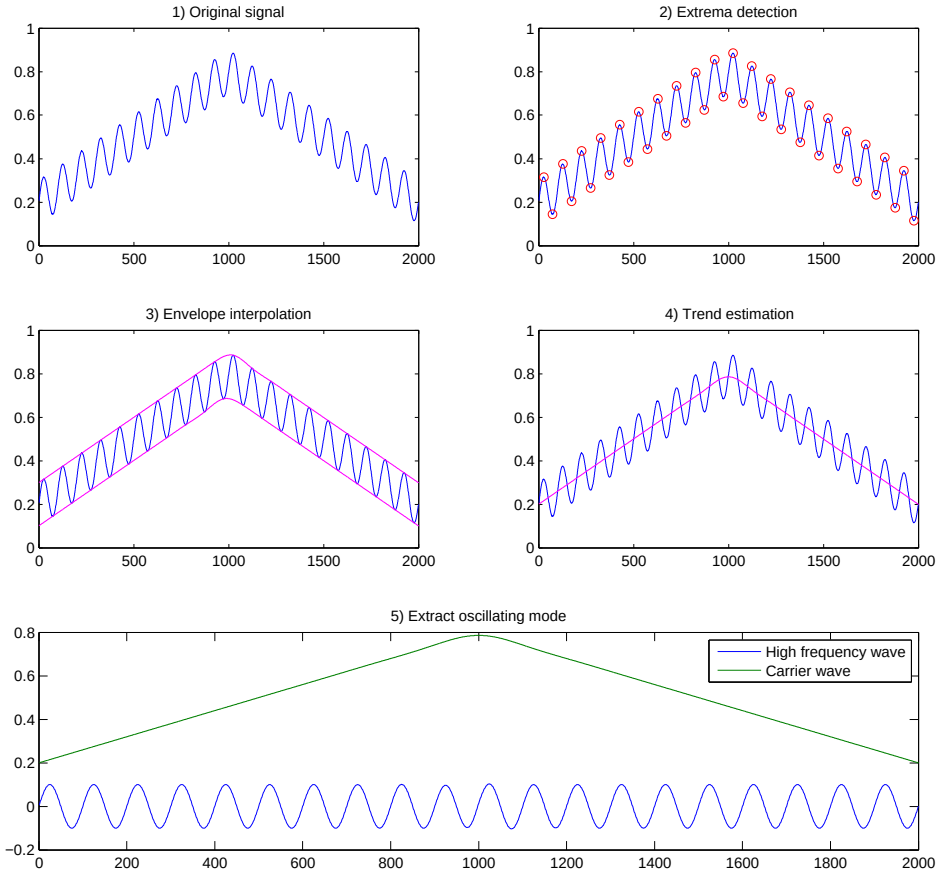


Figure 2.3: An illustration of the EMD concept.

To illustrate the concept of the EMD, consider the signal marked (1) in Figure 2.3. It is clear by inspection that it consists of a high-frequency sine wave riding atop a triangular "carrier" wave. Huang observed that the *extrema* (local minima and maxima) of the signal (2) define an upper and lower *envelope* for the signal (3), and that the average of these two envelopes might give the average trend (4) of the signal (ignoring the highest-frequency oscillations.) Subtraction of the trend from the signal will therefore isolate the highest frequency component (5).

In general, signals will consist of more than just one oscillatory component and simple carrier wave; complex signals will have *"one undulation is riding on top of another, and they, in turn, are riding on still other undulations, and so on,"* as Huang puts it [26]. The idea of the EMD is to repeatedly apply the above process to separate out the fastest oscillatory mode, then the next fastest, and so on until the signal has been entirely broken down into simple oscillatory components, which Huang calls *intrinsic mode functions* (IMFs.)

2.3.3 EMD Algorithm

The EMD algorithm is remarkably elegant. We will give pseudocode first, then explain its operation.

Algorithm 1 Offline EMD algorithm.

EMPIRICAL-MODE-DECOMPOSITION($c(t)$)

```
1  r(t) = c(t)
2  n = 1
3  repeat
4      [IMF_n(t), r(t)] = EXTRACT-MODE( r(t) )
5      n = n + 1
6  until max(r(t)) < ε OR residue is monotone
7  return IMF_1 ... IMF_k , r(t)
```

$c(t)$	original signal, to be decomposed into intrinsic mode functions (IMFs)
$IMF_1 ... IMF_k$	the intrinsic mode functions extracted from $c(t)$. The number of IMFs is not known in advance.
$r(t)$	residue after IMF extraction.

EXTRACT-MODE($x(t)$)

```
 1  d_0(t) = x(t)
 2  n = 0
 3  repeat                                              // Sifting loop
 4      if d_n(t) is a valid IMF, or n > MAX-ITERATIONS
 5          IMF = d_n(t)
 6          r(t) = x(t) − d_n(t)
 7          return IMF, residue
 8      else
 9          [(t_max, v_max)] = FIND-MAXIMA(d_n(t))
10          [(t_min, v_min)] = FIND-MINIMA(d_n(t))
11          e_max(t) = INTERPOLATE([(t_max, v_max)])
12          e_min(t) = INTERPOLATE([(t_min, v_min)])
13          ρ_n(t) = MEAN(e_max, e_min)
14          d_{n+1}(t) = d_n(t) − ρ_n(t)
15          n = n + 1
```

$d_n(t)$	The "detail" of the signal - an estimation of the IMF. $d_n(t)$ is refined by successive iterations of the sifting loop until it becomes an IMF.
(t_{min}, v_{min})	The times and amplitudes of the local minima in the signal. (resp. maxima.)
$e_{min}(t)$	The lower envelope of the signal, obtained by interpolation between the minima. (resp. upper envelope and maxima.)
$\rho_n(t)$	The average of the upper and lower envelopes, which estimates the "carrier signal" or "trend" in $d_n(t)$ - i.e. the "low frequency" part of $d_n(t)$. Subtracting $d_{n+1}(t) = d_n(t) - \rho_n(t)$ removes the low frequency part from $d_n(t)$.

Here $c(t)$ is the signal to be decomposed into intrinsic mode functions. We repeatedly apply the EXTRACT-MODE function, first to the input $c(t)$, then iteratively to the residue from the last extraction. We continue extracting IMFs until the residue is monotonic or "sufficiently small".

EXTRACT-MODE operates by repeatedly *de-trending* $d_n(t)$ - that is, starting with $d_0(t) = x(t)$, repeatedly estimate and subtract $\rho_n(t)$ from $d_n(t)$ to produce $d_{n+1}(t)$; this process is known as *sifting*. Eventually, we will end up with a $d_n(t)$ which has zero mean - that is, the carrier wave has been completely removed, and $d(t)$ is the pure "high frequency" component of the original signal. At this point we have extracted the IMF and return the results $IMF_k(t) = d_n(t)$ and $r(t) = x(t) - d_n(t)$.

The description above is purposely broad. Several details are left up to the implementer:

- The method of determining if $d_n(t)$ is an IMF - i.e. how to test that it has zero local mean.
- The method of interpolating the extrema to obtain $e_{max}(t)$ and $e_{min}(t)$.

The usual implementation determines if $d_n(t)$ is zero mean by demanding that the carrier signal $\rho_n(t) \approx 0$ to within some arbitrary tolerance ϵ. Alternately, we can approximate this by demanding that $d_n(t)$ has the same number of zero crossings as extrema (± 1), which forces $d_n(t)$ to be "centred around zero," a looser condition for zero-mean [26, 28].

In still other implementations, it was observed that just running a fixed number of sifting iterations produces is sufficient to produce valid IMFs most of the time. In these implementations, the number of sifting iterations is just fixed *a priori*, and the zero-meanness of the IMF is not explicitly checked [28].

The method of interpolating the extrema to obtain the signal envelopes is also subject to choice. The usual choice is the cubic splines method [26, 29, 28], which is simple but has some drawbacks [29, 30]. Other choices will be discussed in Chapter 4.

2.3.4 EMD applications and implementations

The EMD is becoming an increasingly used tool for signal analysis where the usual time-frequency techniques fail to produce adequate results. It is easy to find papers about the application of the EMD to analyse financial data[31], ionospheric disturbances [32], and seismographic data[33], amongst other things.

A number of implementations exist, some of which are proprietary and some of which are freely available.

- Huang, et. al. implemented a proprietary version at NASA, which is available under license.[1] This was the version used by Chappell [18].
- Alan Tan's HHT – free download from The Mathworks. [2] This is a very simple implementation similar to the implementation in Appendix A.

[1] `http://www.nasa.gov/centers/goddard/news/dynadx_diagnostic.html`
[2] `http://www.mathworks.com/matlabcentral/fileexchange/19681-hilbert-huang-transform`

- Manuel Ortiguera's – `rParabEMD`, available from the Mathworks website.[3] This implementation is an accompaniment to the paper [30], which attempts to address some problems with the EMD - notably difficulty in accurate location of extrema in the data, and the problematic end effects at either edge of the IMFs.
- Flandrin, Rilling, Goncalves – `pack_emd`, accompanying their paper "On the EMD and its algorithms" [28]. Available from Patrick Flandrin's website.[4] There are several different implementations and demonstrations included, notably including a "local sifting" implementation that claims to avoid "oversifting" the IMF's, and of most interest, a proof-of-concept *online* EMD implementation.

To implement Chappell's method in real-time will require a correspondingly real-time implementation of the EMD. I have only found two references to real-time implementations of EMD. One was due to Meeson [29], who discussed some theoretical aspects of how the online EMD might be computed, but did not produce or publish an implementation. The other was due to Rilling et. al, who published a working proof of concept.

Rilling's proof of concept was a quick demonstration produced for the NSIP 2003 conference. It is available on the Internet as noted above, as the file `package_emd/EMDs/emd_online.m` within the archive `pack_emd.zip`. Unfortunately it is not of "production" quality; it lacks documentation (to the point of having nearly no comments), the variable names are so obfuscated as to defy translation, and the implementation style is strongly reminiscent of "spaghetti code". It is not very robust: it works with the particular examples selected by the authors, but not with a simple sine wave input.

A better real-time EMD implementation will be required before Chappell's method can be implemented in real time.

2.4 Summary

The problem of automatic bubble detection is worthy of study because its solution opens up new avenues for treatment and research into decompression sickness and emboli in general. Automatic embolus detection methods have been successfully implemented to process the "quiet" signals from the *peripheral* veins, with some achieving comparable performance with humans. However the significant problem of non-invasive, automatic bubble detection in the *precordial* region (using CW Doppler or otherwise) remains unsolved.

The problem with automatic CW Doppler ultrasound in the precordial region is the similarity of normal heart sounds and bubble sounds. So far the most promising progress towards solving this problem has been made by Chappell, who used the empirical mode decomposition to process the Doppler audio, but only in offline operation. We would like to implement this in real time, but this cannot proceed until a real-time EMD implementation exists.

Such an implementation would be worthwhile beyond the context of automatic bubble detection; there are many signals processing problems that would benefit from the availability of a realtime EMD implementation.

[3] `http://www.mathworks.com/matlabcentral/fileexchange/21409-empirical-mode-decomposition`
[4] `http://perso.ens-lyon.fr/patrick.flandrin/emd.html`

3

Methodology

The literature review identified a need for an online implementation of the empirical mode decomposition, i.e. one that operates on a stream of data on real time. Since an online EMD implementation could potentially be applied to a wide range of problems, the production of such an implementation would be a worthy undertaking.

We therefore pose the following questions:

- How can the empirical mode decomposition be adapted for use with streaming data instead of recorded data?
- What are the performance characteristics of such a streaming-data implementation? In particular, can it be made to run in *real time*?
- Will a streaming-data implementation produce comparable results to the standard off-line implementation?

These questions will be answered by implementing an online EMD program and performing thorough testing.

3.1 Implementation Goals

The objective is to implement the streaming-data EMD in a way that is *a)* immediately useful for real-time applications and *b)* a good base for future improvements.

The criteria for the implementation to be immediately useful are:

- **Real time performance.** The definition of "real-time" is loose, but for the EMD to be useful in audio processing applications (such as Doppler bubble detection), it should be able to handle a 22050Hz audio stream in real time.
- **Accuracy.** The online EMD must be functionally equivalent to the offline EMD in that they produce the same (or at least very similar) IMF decompositions.
- **Easy to use.** If the basic user manual is more than two pages long, the program is too complicated to use.

The criteria to be "a good base for future improvement" are more subjective. The code must be easy to understand and easy to modify; this will be accomplished by following software engineering best practises [1] :

- Practise modular design; small pieces of code are easier to understand and easier to debug.
- Avoid premature optimisation, which complicates the code and often gives no real benefit. "Prototype before polishing."
- Explicitly check the code is doing what it is supposed to (by using `assert()`, etc.) If inconsistencies are detected, crash immediately; this aids debugging.

3.2 Implementation method

It was decided early on that the implementation would be in MATLAB. While MATLAB is not as fast as compiled languages such as C/C++, D or Ocaml, its large libraries, expressive syntax for numerical mathematics, and excellent integrated debugging capability make it the best choice for rapid prototyping of signals processing algorithms.

In terms of the actual algorithmic details, my objective is not to reinvent the wheel; as much as possible will be drawn from what prior work there is. Meeson's paper [29], Rilling's proof-of-concept implementation [28], and Rato's paper [30] are rich sources of ideas, though none will lead to a complete implementation on their own.

3.3 Verification

Verification of the implementation will require thorough testing to establish:

1. Performance characteristics.
2. Correctness of output.

The design of the test signals, test vectors, and testing procedure will be deferred to Chapter 5.

[1] These are paraphrasings of Eric S. Raymond's "UNIX Philosophy," a list of rules for good software design. See his book "The Art of UNIX programming," which is freely available on the Internet.

The unwritten rules of code optimisation:
#1 : Don't.
#2 : Don't... *yet. (Experts only.)*
#3 : ***Profile*** *before you optimise.*

Unknown (from c2.com)

4

Implementation

4.1 Implementation Architecture

The realtime EMD calculation can be modelled as a cascade of sifting operations [29, 28]. For the moment, consider the first sifting operation in the chain. We wait until sufficient data has accumulated, then perform sifting to compute a *block* of the first IMF, simultaeneously producing a block of the corresponding residue. The residue from the first sifting is then used as the input to the second sifting, and so on.

Since there is some lag between input between input to a sifting operation and a block of the IMF being produced, the input data propagates through the system like a waterfall (Figure 4.1). Note that it may take a significant time between the input of some data with timestamp t,

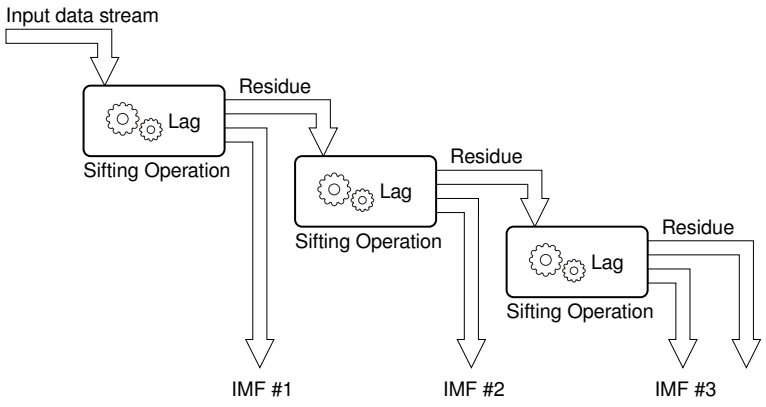


Figure 4.1: Streaming EMD – cascading "waterfall" structure of calculations

and the kth IMF being calculated for time t [28]. For many applications we are only interested in the first few IMFs - the calculation of these progresses quickly and the lag is of little concern.

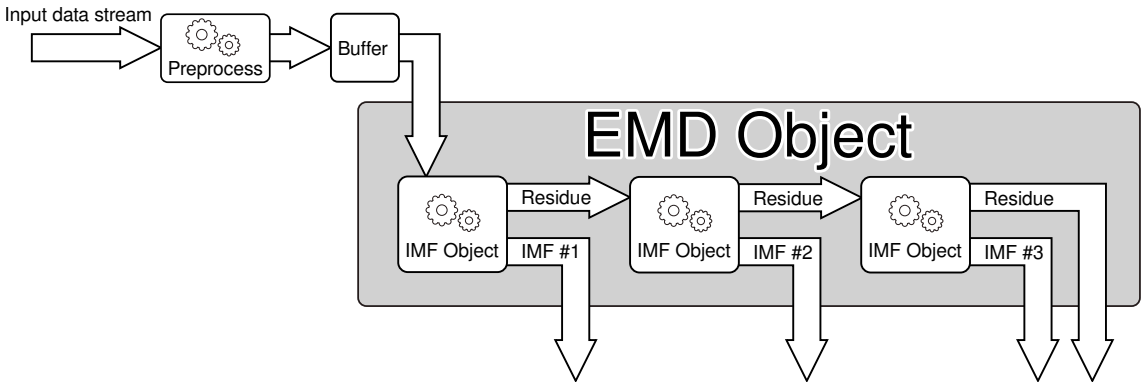


Figure 4.2: Streaming data architecture - `Buffer`, `EMD` and `IMF` objects.

The waterfall analogy suggested a natural form for calculation of the EMD, based on a cascade of `IMF` objects (Figure 4.2). Each `IMF` object has only *one* responsibility: to accept some input data stream and perform blockwise extraction of the first IMF and residue. Chaining many of these IMF blocks together, using the residue output of each `IMF` block as the input to the next, results in streaming calculation of the entire EMD (with some time lag for the higher-order modes.)

The collection of `IMF` objects is controlled by an `EMD` object, which has *one input* - the data from an input `Buffer` - and *one set of outputs* - the calculated IMF's and final residue. The end user can treat the implementation as a "black box": the only things they need to concern themselves with are providing the stream of input data to the `Buffer` (and hence to the `EMD` object), and extracting the IMFs once they are calculated.

In terms of the implementation, the design of the EMD object is trivial; its only job is to sequentially tell the IMF blocks when new data is available. The design of the `Buffer` object is also relatively trivial. The real challenge is designing a sifting algorithm that extracts an IMF block-by-block - i.e. using *only local information* - while obtaining the same result as would be obtained by *offline* sifting.

4.2 Buffer object

The Buffer object was implemented as a circular buffer, which keeps its memory usage low. It also supports pre-processing of the input data with any MATLAB filter object; this allows seamless filtering and sampling rate conversion (decimation or interpolation) of the input stream before the data is stored to the `Buffer`.

The design of the `Buffer` object is not particularly interesting, so it will not be expanded on here. See Appendix B.1 for documentation and usage examples.

4.3 IMF object

The only responsibility of the IMF object is to calculate the *first IMF and residue* of some data stream. In general, this is accomplished by waiting for input data to accumulate until we have enough to compute a small *block* of the IMF and residue, using the same *sifting* process as in the offline algorithm. We compute further blocks of the IMF and residue as data becomes available, and append these to the IMF/residue we have already calculated; this continues as long as new input data is presented.

The catch is that computation of each block must be able to pick up exactly where the previous block left off, with smooth joins between blocks. Two factors complicate this:

- Each sifting iteration changes the IMF slightly. If different numbers of sifting iterations are applied to each block, the IMF will no longer be continuous across the block boundaries.
- The sifting process produces *end effects*, which can cause the output to be discontinuous between blocks.

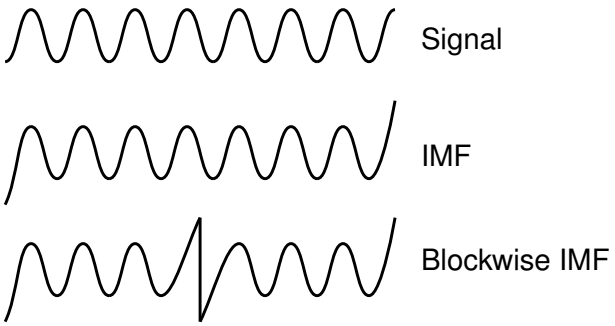


Figure 4.3: End effects in blockwise IMF calculation (exaggerated for illustrative effect.)

The first problem is solved by fixing the number of sifting iterations *a priori*. According to Rilling, et. al. [28], as few as four to ten sifting iterations usually suffice to extract meaningful IMFs. Using this few sifting iterations may even be advantageous, as oversifting until the stop conditions are strictly met can destroy the physical meaning of the extracted IMFs [28]. This implementation uses 10 sifting iterations by default; this can be configured to trade off IMF extraction quality for increased speed. [1]

The problem of end effects, however, has no simple solution; the problem is fundamental to the sifting process. Recall that the trend function $\rho(t)$, to be removed from each iteration of the IMF candidate $d(t)$, is calculated by *interpolating* the extrema of $d(t)$. Since these extrema do not extend all the way to the ends of the data, the envelope has to be *extrapolated* at the ends. Since cubic splines are bad extrapolators [30], this leads to large errors of the type seen in 4.5.

The end effects are not problematic for the offline EMD, but they introduce discontinuities into the blockwise IMF output (Figure 4.3.) If the discontinuities produce spurious extrema in the residue, then subsequent IMF extractions from that residue will be severely corrupted.

[1] See the `'IMF:SIFT_BLOCK:NUM_SIFTING_ITERATIONS'` config option.

Several strategies for reducing end effects have been implemented in the sifting code, `IMF.Sift_Block();`

1. Using Hermite splines instead of cubic splines for envelope interpolation.
2. Extending the interpolating points past the interval by "mirrorisation"
3. Calculating more data than needed, then trimming the excess.

4.3.1 Hermite splines

In most implementations of the EMD, the envelopes are interpolated from the extrema using *cubic splines*. These are a piecewise polynomial interpolant, where an individual cubic polynomial (*spline*) is fitted between each pair of points being interpolated [34]. When the second derivative is forced to be continuous between neighbouring splines, the total curvature is minimised (which is to say that cubic splines can be considered the "smoothest *possible*" interpolant [35, p114].)

Fitting a cubic spline to $n+1$ points involves solving for $4n$ coefficients - four for each of n individual splines. With cubic splines, the solution for each of these coefficients (and thus the shape of the curve) depends on the entire set of points to be interpolated; this is undesirable because adding a single extra interpolating point will change the *entire curve*.

By definition, the stream of data in an online application will be constantly adding new points to interpolate; the dependence of each cubic spline on the entire data set is therefore problematic.

Meeson suggested the use of Hermite splines [29], which are very similar to cubic splines; but instead of forcing the second derivative to be continuous, the first derivative is fixed at each point [36]. Since the first derivative for point i can be estimated by averaging the gradients between points $i-1 \leftrightarrow i$ and $i \leftrightarrow i+1$, each spline can be calculated using *local information only*. This means that adding new points to the end of the data only changes the very last spline; the others remain static; additionally, end effects will only appear in the *last spline*.

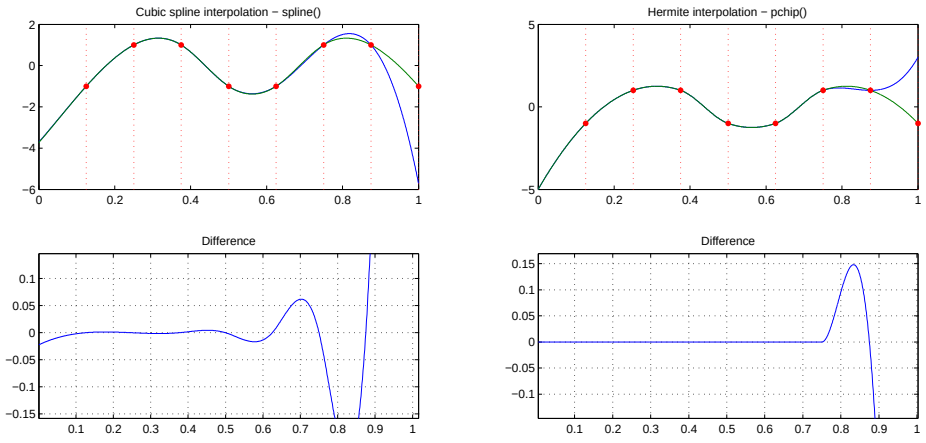


Figure 4.4: Cubic splines (left) vs. Hermite splines (right.)

The difference is illustrated in Figure 4.4. To simulate the effects of adding a single new point to be interpolated, a cubic splines interpolant was fitted first to all the data points except the last, and then to the entire set. The difference between the two curves is shown at bottom left; it can be seen that adding one point has changed the entire extent of the curve. When this is repeated for Hermite splines, however, the curves are exactly the same until the last segment.

Having established the superiority of Hermite splines, I adapted the `pchip_tx()` example[2] from [37] – a reference implementation of MATLAB's built in `pchip()`[3]. Two modifications were made.

- Firstly, `pchip_tx()` used a complicated weighted geometric mean between adjacent derivatives to determine the slopes at each point, which produced strange effects; this was replaced with a simpler calculation.
- Secondly, `pchip_tx()` was designed to be "shape preserving" in that turning points in the data are assumed to have zero derivative. Since this might lead to inaccurate representation of the signal envelope, this feature was removed.

The end result was a function called `hermite_spline()` which has identical usage to MATLAB `spline()` or `pchip()`, but customised for envelope interpolation with limited end effects.

The sifting function `Sift_Block()` uses `hermite_spline()` by default, but the `'IMF:SIFT_BLOCK:ENVELOPE_INTERPOLATION_METHOD'` option can be used to configure this at runtime.

4.3.2 Mirrorisation

"Mirrorisation" is a technique suggested by Rilling, et. al. [28] to reduce the end effects due to extrapolation. The concept is simple: instead of extrapolating the *spline* to cover the ends of the interval, extrapolate the *data* past the end of the interval. This is done by making a mirror-copy of the extrema, then interpolating with both the mirrored extrema and the original extrema (Figure 4.5.)

The effect is to constrain the spline from "shooting off;" it will now transition smoothly across the ends of the interval. Note that this does not guarantee the envelopes for subsequent blocks will *line up exactly*, as that would require knowledge of the future signal, but the discontinuity should be limited.

The way in which the extrema are mirrored is chosen to represent the data. Consider the end of the interpolation interval; if the data appears to be trending upwards, an extra maxima is inserted to ensure the interpolated upper envelope $e_{max}(t)$ continues to rise; similarly, an extra minima will be added to continue a downwards trend. The extrema are then mirrored around the inserted extrema. If the data appears to be holding steady, the extrema are mirrored without inserting any extra.

[2] See the "NCM file collection" at `http://www.mathworks.com/moler/chapters.html`.
[3] short for "Piecewise Cubic Hermite Interpolating Polynomial." See `doc pchip` for more information

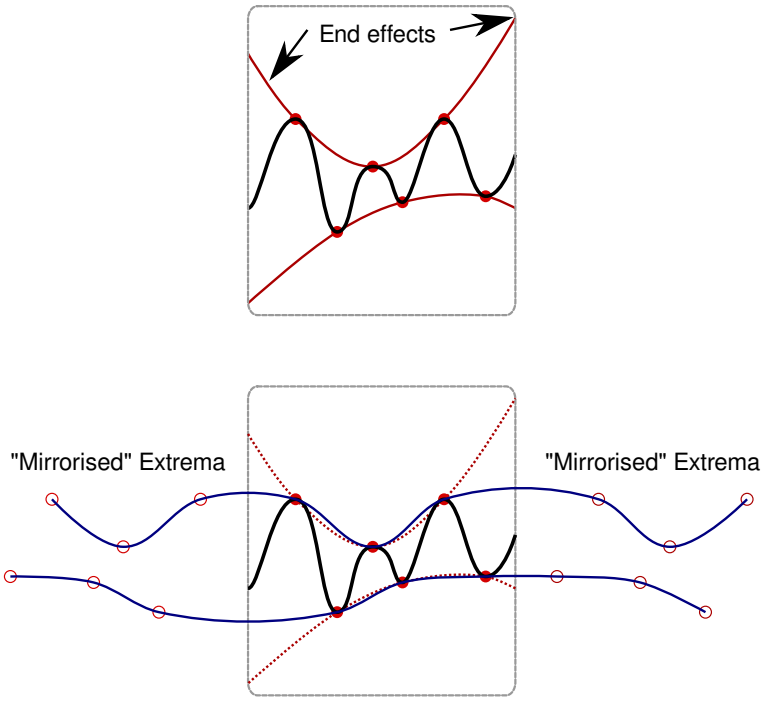


Figure 4.5: End effects and their reduction via "mirrorisation."

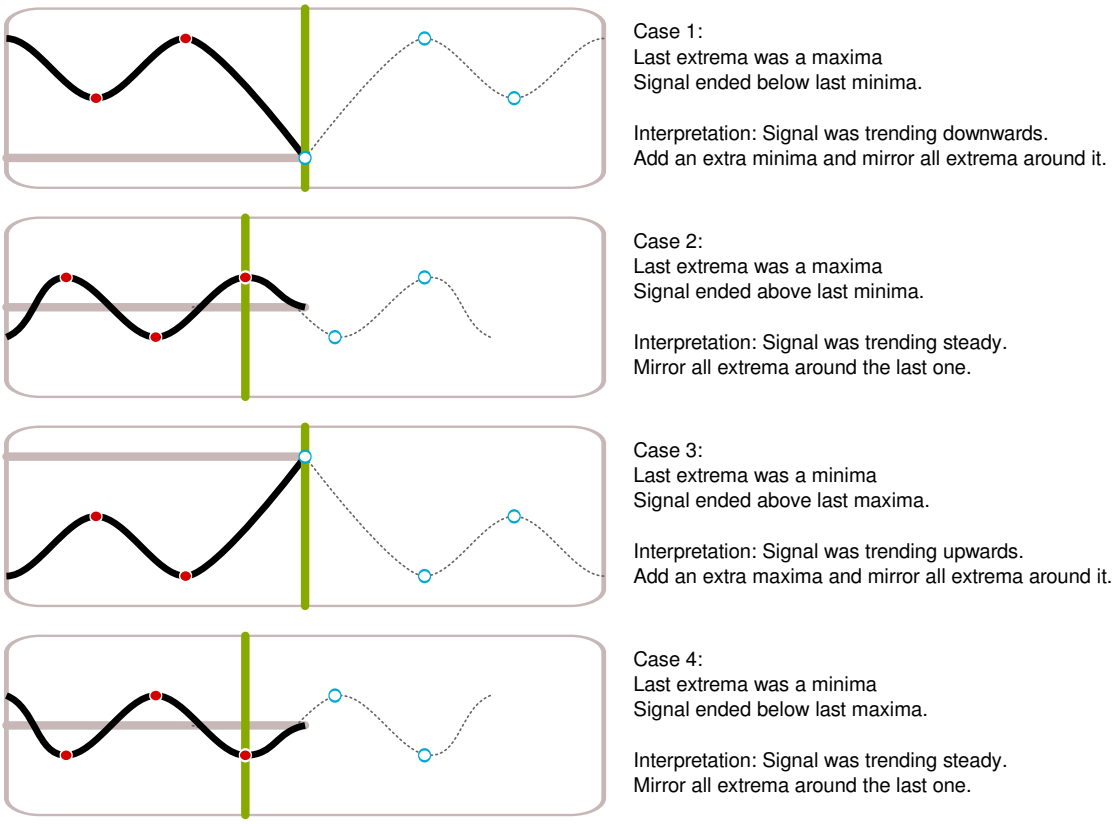


Figure 4.6: Details of the mirrorisation process for the end of the interpolation interval.

This process is shown graphically in Figure 4.6. The same process applies to the start of the interpolation interval, except (naturally) in reverse.

The algorithmic implementation of the mirrorisation process (Figure 4.6) is contained in `@IMF/Sift_Block.m` as the `Mirrorize_Extrema()` function. It can be turned on or off by using the `'IMF:SIFT_BLOCK:MIRRORISE_EXTREMA'` flag.

4.3.3 Calculating excess data

Even with mirrorisation and Hermite splines instead of cubic splines, there will still be end effects in the interpolation. This is a natural consequence of having local data only : we "don't know what comes next," so we can never interpolate it.

The use of Hermite splines, however, *should* limit the end effects to the first and last interpolation intervals. If we are willing to overlap block IMF calculations and throw away the ends of each block, we should be able to eliminate end effects nearly completely; the concept is illustrated in Figure 4.7.

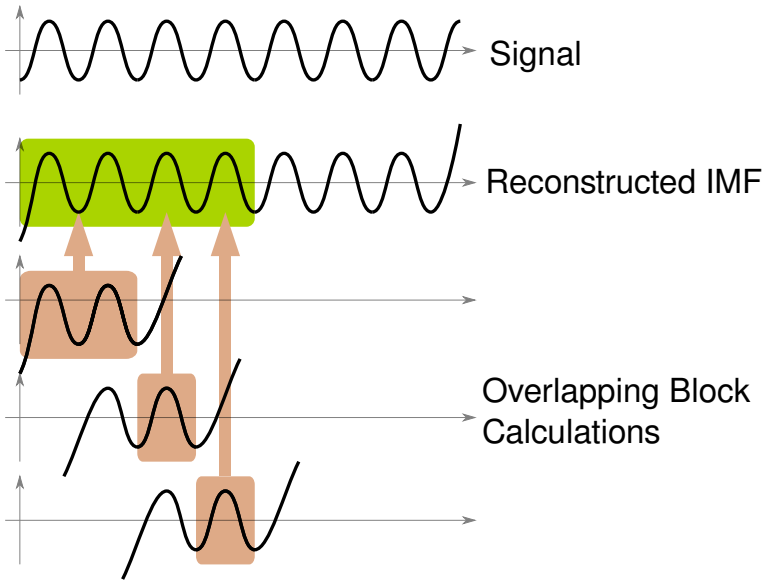


Figure 4.7: Overlapping block calculations to reconstruct the "true" IMF without end effects.

By default, the program will retrieve "ten extrema worth" of previous data to pad the beginning of the block, perform the sifting process, and then discard the "last three extrema worth" of data from the right hand side. Since this padding and discarding increases the amount of work that must be done per unit input, there may be a performance penalty. The larger the block size processed, however, the smaller this penalty should be.

- The `'IMF:SIFT_BLOCK:NUM_EXTREMA_PRECEDING'` configuration option controls the amount of historic data appended to the beginning of blocks before sifting.
- The `'IMF:SIFT_BLOCK:NUM_RIGHT_TRIM_EXTR'` configuration option controls the amount of each IMF block that is trimmed off the right side before appending to the stored IMF.

4.4 EMD Object

Operation of the EMD object is fairly straightforwards (see Appendix B for usage instructions.)

At present, the EMD object causes a *fixed* number of IMFs to be extracted from the input stream ; this means that the number of IMFs required for interpretation of a particular signal must be known and configured *a priori*.

4.5 Testing Framework

4.5.1 Features

An automated testing framework was written to enable quick, repeatable testing of the algorithm. Its main features are as follows:

- Fixed configuration options for each test.
- Repeated testing to determine average runtime.
- Saves of EMD outputs (IMFs and residues) for later review.
- Saves figures as high-resolution `.pdf`.
- All command window output, including any MATLAB errors or warnings, are captured to a `.txt` log file. (This can be used for debugging purposes.)
- Error-tolerance - If a MATLAB error (i.e. `OutOfMemory`) occurs during a particular test, that test is skipped. This allows batch testing runs to continue even if some tests produce errors.

The design of the testing framework is trivial and needs no explanation. Usage instructions, examples, and the default configuration options for the testing framework are given in Appendix C.

4.6 Summary

An streaming-data version of the EMD was implemented. The chief difficulty in the implementation was preventing the inclusion of end effects in the computed IMF and residue. The inclusion of end effects would mean the IMF obtained would be inaccurate (as compared the the IMF obtained by offline sifting). Sharp discontinuities due to end effects also manifest as spurious extrema in the residue, which severely corrupts the calculation of the next IMF.

A variety of measures were taken to eliminate (or at least reduce) the end effects. Hermite interpolation was substituted for cubic splines; the extrema were “mirrorised” to extend the

data; and an overlap-and-discard calculation method was implemented to chop off what end effect remained. The effectiveness of these techniques will be evident during testing.

The implementation was designed to have a particularly elegant architecture, and was coded in a style which eschews complicated optimisations in favour of clarity. Given the public release of the code to the world, this will hopefully facilitate easy understanding and improvement of the code by future contributors.

Verification of the implementation will be performed in the next section.

5

Testing and Verification

5.1 Testing parameters

5.1.1 Testing environment

The MATLAB code was tested using the following environments:

Desktop

MATLAB version	MATLAB 7.10.0.499 (R2010a) 32-bit (glnx86)
Operating System	Arch Linux, kernel 2.6.35-ARCH #1 SMP PREEMPT
CPU	Intel(R) Core(TM)2 Duo CPU E7400 @ 2.80GHz
RAM	4Gb

Laptop

MATLAB version	MATLAB 7.10.0.499 (R2010a) 32-bit (glnx86)
Computer	Lenovo Thinkpad X200s
Operating System	Ubuntu Linux 10.04, kernel 2.6.32-24-generic-pae #43-Ubuntu SMP
CPU	Intel(R) Core(TM)2 Duo CPU L9400 @ 1.86GHz
RAM	4Gb

Timing is performed using the `tic` and `toc` functions in the pattern `tic; timed_operation(); toc;` (as recommended in the MATLAB documentation.)

5.1.2 Test Signals

Three test signals were chosen to test the implementation.

- **Test signal #1:** Elementary synthesised signal. Sum of three sinusoids - $\sin(100 \times 2\pi t) + \sin(300 \times 2\pi t) + \sin(500 \times 2\pi t)$.
- **Test signal #2:** Artificial heartbeat data. Simulated 80BPM heartbeat signal with random variation in beat time and amplitude, and added noise. See `Heartbeat_Test_Data.m` for details.
- **Test signal #3:** Real-world signal : sample from the DCIEM precordial Doppler audio studies, originally by DCIEM Canada and provided to me by the Townsville Hospital Hyperbaric Unit.

All test signals 30 seconds long at 22050Hz sampling rate.

REMARKS: In general, synthesised signals will have relatively "simple" behaviours and decompose into a small number of IMFs. Real-world signals are much more complex; electrocardiogram heartbeat signals, for example, are hypothesised to exhibit fractal time characteristics [38]. Therefore, the test signal set contains examples of both synthetic and "real world" signals.

Some purposely simplistic signals are used to test the robustness of the implementation. Rilling's proof-of-concept on-line EMD, for example, works well on complex signals, but crashes when analysing a sine wave. Additionally, the IMFs from simple signals should have very particular forms - the sum of three sine waves, for example, should decompose back into three sine waves (plus or minus some end effects.)

5.2 Test vectors

The testing vectors will be designed to answer the following questions:

- **Speed:** How fast does the program run? How does the program's execution time change with...
  - increasing input size? Is the execution time $O(n)$, $O(n^2)$, $O(2^n)$... ?
  - increasingly complex signals?
  - the number of IMFs extracted?
  - the number of sifting iterations?
- **Correctness:** How does the program's output compare to offline implementations?
- **Robustness:** Does the program run on all inputs, or are there certain input signals which will cause it to crash?

The test vectors are as follows:

- **Test vector 0:** A preliminary test run with simple signal #1, to validate basic functionality.
- **Test vector 1:** Test runs on various lengths of data to determine speed and scalability. The tests will be repeated for each of the three test signals, which will determine how the runtime varies with signal complexity.
- **Test vector 2:** Test runs on fixed lengths of data to determine how the run time varies with number of IMFs extracted.
- **Test vector 3:** Test runs on fixed lengths of data to determine how the run time varies with number of sifting iterations.
- **Test vector 4:** Small test runs on 1-second data samples to characterise the accuracy of the online EMDs output as compared to reference offline implementations.

5.3 Test Vector 0: Basic functionality check

5.3.1 Hypothesis

The EMD program should correctly decompose the sum of three sinusoids back into three sinusoidal components. Assuming the program design is correct, no end effects should appear in the middle of the components.

5.3.2 Procedure

The input signal was 10 seconds of Signal #1. The test was repeated 20 times with standard settings - 2000 sample input blocks, 5 IMF's extracted, 10 sifting iterations, and Hermite interpolation of the envelopes. (See appendix for standard settings.) The test rig was run using the MATLAB profiler to assess performance of the code.

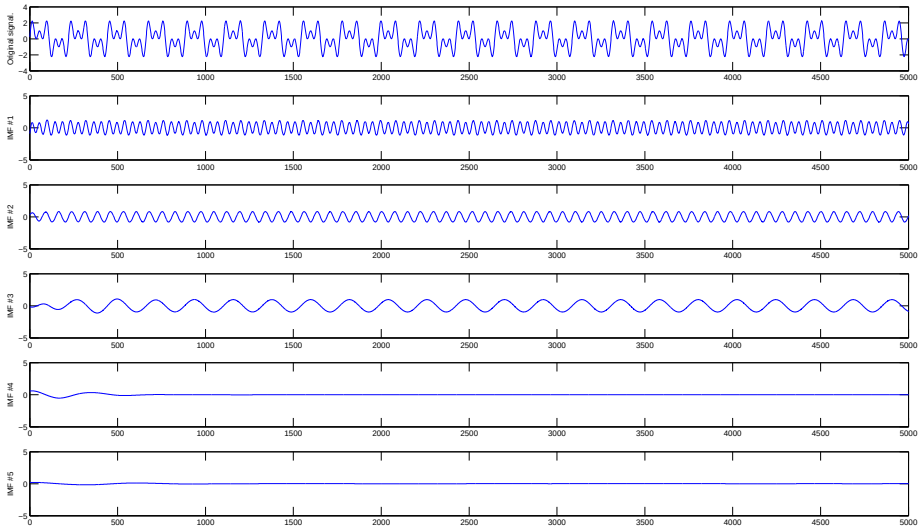
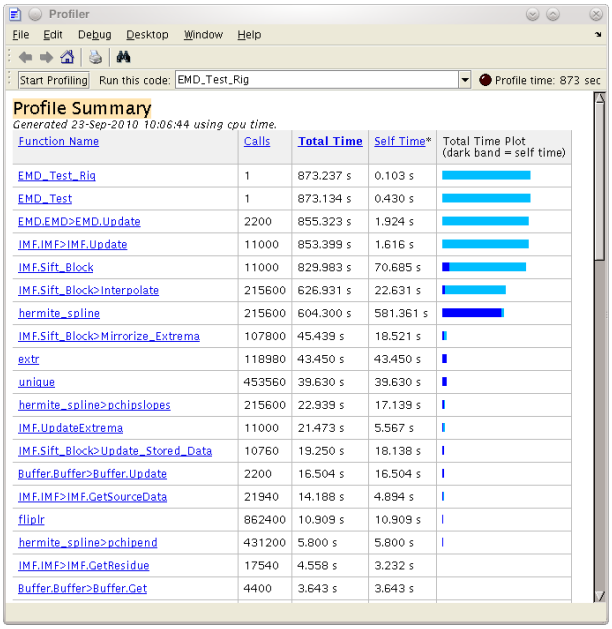


Figure 5.1: First 5000 samples of an online EMD calculated with default settings.

5.3.3 Results



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
EMD_Test_Rig	1	873.237 s	0.103 s	
EMD_Test	1	873.134 s	0.430 s	
EMD.EMD>EMD.Update	2200	855.323 s	1.924 s	
IMF.IMF>IMF.Update	11000	853.399 s	1.616 s	
IMF.Sift_Block	11000	829.983 s	70.685 s	
IMF.Sift_Block>Interpolate	215600	626.931 s	22.631 s	
hermite_spline	215600	604.300 s	581.361 s	
IMF.Sift_Block>Mirrorize_Extrema	107800	45.439 s	18.521 s	
extr	118980	43.450 s	43.450 s	
unique	453560	39.630 s	39.630 s	
hermite_spline>pchipslopes	215600	22.939 s	17.139 s	
IMF.UpdateExtrema	11000	21.473 s	5.567 s	
IMF.Sift_Block>Update_Stored_Data	10760	19.250 s	18.138 s	
Buffer.Buffer>Buffer.Update	2200	16.504 s	16.504 s	
IMF.IMF>IMF.GetSourceData	21940	14.188 s	4.894 s	
fliplr	862400	10.909 s	10.909 s	
hermite_spline>pchipend	431200	5.800 s	5.800 s	
IMF.IMF>IMF.GetResidue	17540	4.558 s	3.232 s	
Buffer.Buffer>Buffer.Get	4400	3.643 s	3.643 s	

Figure 5.2: Profiler data - 20 runs of basic test using 'Hermite' interpolation method

The total runtime for the test was 873 seconds for 20 runs (desktop computer.) The average runtime was 43.61 seconds to run over the 10 seconds of 22050Hz data ($\bar{x} = 43.61$s, $\sigma = 1.44$s), which is 4.3x slower than real time.

The outputs were mostly as expected (Figure 5.1). End effects are seen at the beginning of each IMF, which leads to transients in IMFs #4 and #5 before all the IMFs settle into the correct extraction of the sinusoids. No end effects are seen in the middle of the IMFs, which means

that the measures taken to prevent their appearance between blocks have been successful (at least for this test signal.)

Profiler results are shown in Figure 5.2.

5.3.4 Discussion

This preliminary result is encouraging; it shows that the algorithm runs within an order of magnitude of real-time on a fairly average computer, *even before any optimisation of the code has been attempted*. The necessary speedup to real-time could potentially be obtained just by using a faster computer, or downsampling the data (say to 5kHz) before performing the EMD.

In the context of Chappell's method of bubble detection, the data is de-noised by applying a 12.5ms moving average before the EMD is calculated. This is more or less equivalent to applying a lowpass filter, so downsampling the input signal (with a FIR lowpass filter and decimator) may actually be *justified*.[1]

The profiler results show which areas of the code would benefit most from optimisation.

- ~**70%** of the execution time was spent performing the `hermite_spline()` interpolation.
- 4.5% of execution time is spent in the `unique()` function. This function is only used for debugging purposes - the calls to `unique()` could be removed for an instant 5% performance gain.

It is notable that the `hermite_spline()` function is a custom version of MATLAB's built in `pchip()` function. Might using the `pchip()` (or `spline()`) function instead lead to faster performance?

5.4 Test Vector 1 : Execution speed vs. Input Size

5.4.1 Hypotheses

1. The streaming EMD only concerns itself with small pieces of data at a time. Therefore its execution time should scale linearly with increasing input size.
2. If the number of IMFs is fixed, more complex signals should require no more work to decompose than simple signals. The execution speed should not be dependent on signal complexity.

[1] (The necessary code for applying an FIR decimating filter is already implemented in the `Buffer` class, and is quite easy to use; see Appendix X for usage examples.)

5.4.2 Experiment

Record the run time of the EMD while permuting the following variables:

- Signal length: range between 1 second and 30 seconds of a 22050Hz input signal.
- Different signals:
  - "simple signal" - 30 seconds of $\sin(100t) + \sin(300t) + \sin(500t)$.
  - "complex artificial signal" - 30 seconds of `Heartbeat_Test_Data()`. 12.5ms smoothing applied with MATLAB `smooth()` to denoise the signal.
  - "real world signal" - 30 seconds of precordial Doppler recording, taken from DCIEM training tape; more specifically the 30 seconds from 00:15-00:45 in `DCIEM-S1.mp3`. 12.5ms smoothing applied with MATLAB `smooth()`.

All other variables shall be kept fixed (i.e. default settings) except for the interpolation method, which will be set to `'PCHIP'` instead of `'Hermite'` to speed up execution of the tests. The number of sifting iterations shall be fixed at 10, and the number of IMFs extracted shall be fixed at 5.

5.4.3 Results

Streaming Data Duration	#1 Simple	#2 Artificial	#3 Real-World
1	3.52	2.25	7.46
2	8.26	9.34	15.32
3	11.25	10.16	21.61
4	14.53	23.36	29.07
5	18.49	29.36	37.59
6	23.56	34.05	45.17
7	27.26	51.66	52.95
8	31.13	70.27	59.71
9	33.94	78.78	67.08
10	37.97	87.81	76.84
15	60.73	137.90	117.94
20	78.28	175.31	152.38
25	98.98	231.03	188.29
30	120.23	280.11	228.61

Executed on desktop (Core 2 Duo E7400 @2.80Ghz.)

Table 5.1: Execution speed testing for different input lengths - results.

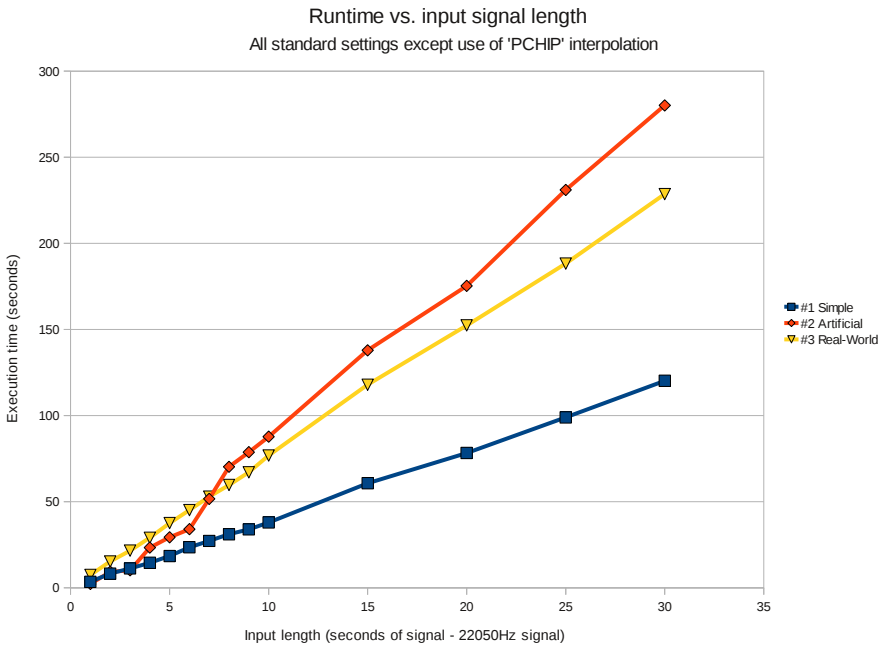


Figure 5.3: Execution speed for different input lengths.

5.4.4 Analysis

Hypothesis 1: The execution speed *does* appear to scale linearly with the input duration (Hypothesis 1.) It appears that some jitter can be expected, however. Maybe this can be explained by the spacing of peaks in the data; an iteration of block sifting will only trigger when "10 extrema worth of data" becomes available, which could be infrequent for data with widely spaced extrema.

Hypothesis 2: Apparently complex signals *do* require more work than simple signals to decompose, even when the number of IMFs and sifting iterations is fixed; this was unexpected. This may be because they have a greater density of local extrema, so the sifting code must do more work.

Performance on real data appears to be substantially slower than simple signals such as sums of sine waves. In this test, the simple signal executed in roughly 4× real-time; the artificial heartbeat data about 9× real time; and the real-world Doppler data in about 7-8× real time.

These speeds all fall well short of real-time execution, but it should be remembered that:

1. No optimisation has been attempted; the code was purposely written for clarity, not performance.
2. There are many parameters which can be used to adjust performance. For example, 10 sifting iterations may be excessive; according to Rilling, meaningful IMFs can be extracted with just four sifting iterations [28]. This could feasibly reduce run time by half.

Finally, the linear execution time ($O(n)$)may be a significant result; it shows that unlike $O(n^2)$ algorithms, the online EMD will scale well to handle large inputs.

5.5 Test Vector 2: Execution Speed vs. Number of IMFs extracted

5.5.1 Hypothesis

Each IMF requires a large amount of work (sifting) to extract. Therefore, the execution speed should increase linearly as the number of IMFs extracted.

5.5.2 Testing

Record runtimes for the EMD operating for 30 seconds on each of the three input signals, with the number of IMF's extracted set to 2, 4 and 6. All other configuration parameters shall have default values.

5.5.3 Results

Number of IMFs extracted	#1 Simple	#2 Artificial	#3 Real-World
2	54.24	200.27	160.06
4	95.92	253.40	209.07
6	146.38	314.98	254.02

Executed on desktop (Core 2 Duo E7400 @2.80Ghz.)

Table 5.2: Execution speed testing for different input lengths - results.

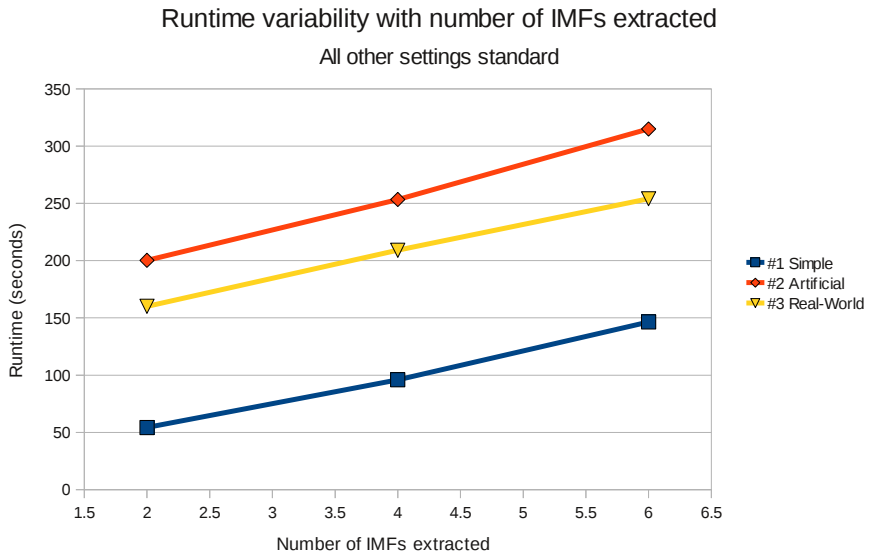


Figure 5.4: Execution speed for different input lengths.

5.5.4 Analysis

Runtime appears linear in the number of IMF's, though more data points would be required to prove this. There is also a large amount of overhead regardless of how many IMFs are extracted.

5.6 Test Vector 3 : Execution Speed vs. Number of Sifting Iterations

5.6.1 Hypothesis

Each sifting iteration requires a large amount of work - most notably repeated envelope interpolation, which is very expensive to evaluate. Therefore, we should be able to trade IMF accuracy for speed by decreasing the number of sifting iterations.

5.6.2 Experiment

Record runtimes for the EMD operating on 10 seconds on each of the three input signals, with the number of sifting iterations set to 2, 4, 6, 8 and 10. All other configuration parameters shall have default values.

5.6.3 Results

Number of sifting iterations	#1 Simple	#2 Artificial	#3 Real-World
2	35.92	83.69	71.98
4	36.21	84.19	72.02
6	36.36	84.71	73.17
8	35.62	83.90	72.76
10	36.36	84.26	73.56

Executed on desktop (Core 2 Duo E7400 @2.80Ghz.)

Table 5.3: Execution speed testing for different numbers of sifting iterations - results.

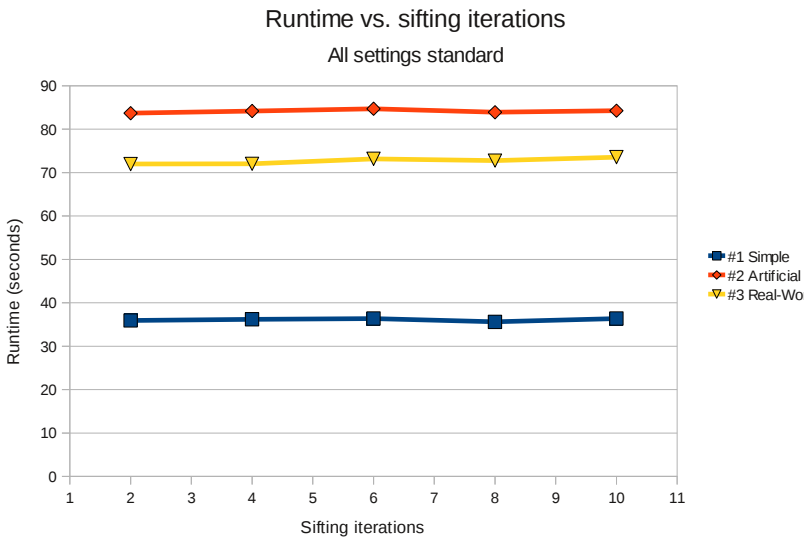


Figure 5.5: Execution speed for different numbers of sifting iterations.

5.6.4 Analysis

It appears that reducing the number of sifting iterations gives only a negligible benefit. The number of sifting iterations can therefore be left at 10, which should be assured to give high-quality IMFs with little sacrifice in speed.

5.7 Test Vector 4 : Comparison of output with other implementations

5.7.1 Hypothesis

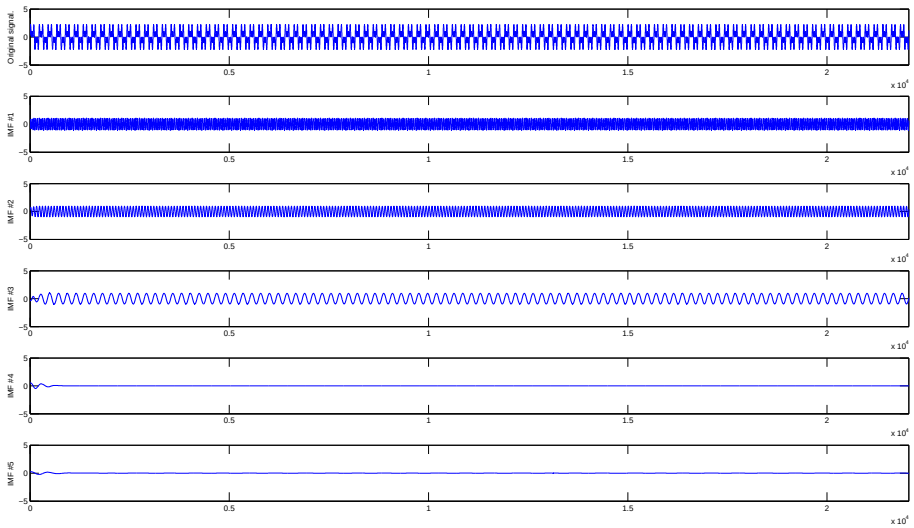
The output of the online EMD should closely resemble the output from a similar offline EMD run with the same data and same parameters.

5.7.2 Experiment

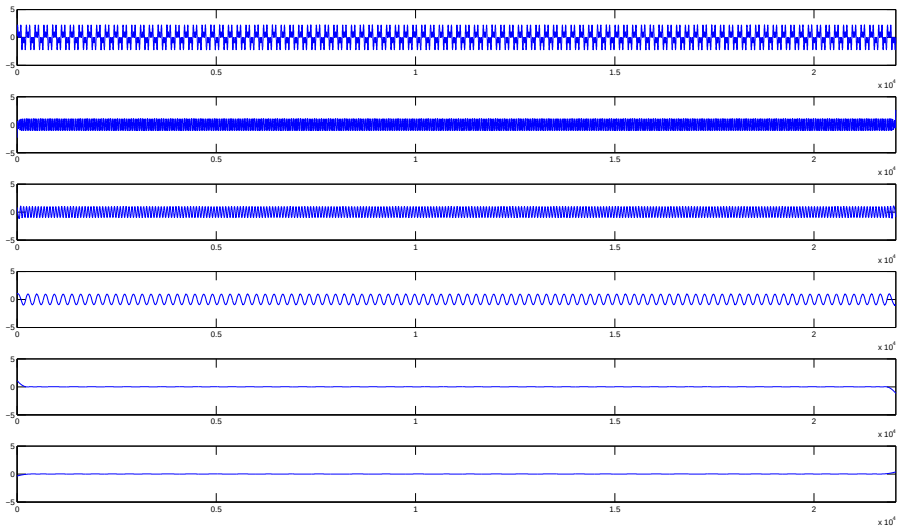
An offline EMD implementation using very similar algorithms to the online version has been developed (Appendix A). Short test signals will be applied and the output compared.

5.7.3 Results

The online and offline algorithms were run on one-second samples of signal #1 and signal #3. The first 5 IMFs extracted are shown in Figures 5.6 and 5.7.

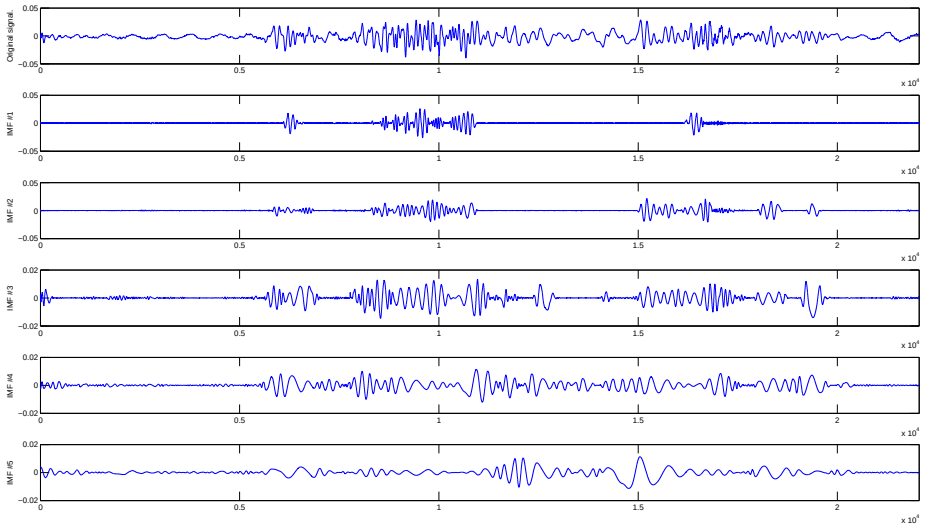


(a) Online algorithm output

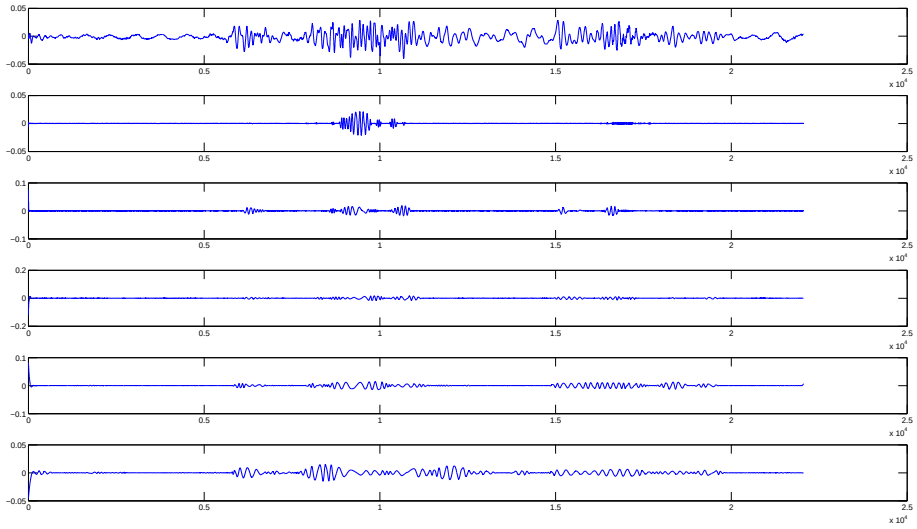


(b) Offline algorithm output

Figure 5.6: Output of online and offline implementations for Signal #1 (simple).



(a) Online algorithm output



(b) Offline algorithm output

Figure 5.7: Output of online and offline implementations for Signal #3 (real world)

5.7.4 Analysis

Both algorithms produce similar results for the simple test signal, but completely different results for the real-world signal.

This could be due to either:

- A flaw in the implementation of the online EMD
- Different (but equally valid) decompositions of the same data.

It should be noted that unlike the FFT or wavelet transforms, which have mathematical definitions, the empirical mode decomposition is defined solely by the algorithm used to calculate it. In particular, this means that the IMFs for a signal are not unique; there are as many equally valid decompositions as there are implementations of the EMD.

Further testing will be required to determine the correctness of the online implementation.

5.8 Summary

The online implementation of the EMD was tested to determine some of its performance characteristics. The testing showed:

- The running time of the algorithm ranges from 4x slower than real time to 10x slower than real time, depending on signal complexity.
- The algorithm scales well to large inputs - $O(n)$ scalability.
- The more IMFs extracted, the longer the execution time (linear variation.)
- Varying the number of sifting iterations has nearly no effect on running time.

The answer to the original question – "can an online EMD implementation be made to run in *real time*" – remains an open question. Though this implementation only runs at a fraction of real time, no optimisation of any kind was attempted. Some low-hanging fruit for optimisation, mainly the `hermite_spline()` interpolation function, were identified as potential ways to greatly accelerate the implementation.

The correctness of the algorithm could not be verified. The online and offline EMDs produced similar output when the input signal was a simple sum of sinusoids, but produced very different output for the real-world Doppler ultrasound data. This may indicate either a fault with the algorithm, which will need further testing to diagnose, or just the normal variation in decompositions produced by different implementations of the EMD.

6

Summary and Conclusions

Review

The problem of real-time, non-invasive bubble detection in the precordial region remains unsolved. The most promising work to date was due to Chappell, who used a novel signals processing technique, Huang's *empirical mode decomposition*, to analyse continuous-wave Doppler audio recordings with a great deal of success. A conversion of Chappell's method to a real-time implementation was investigated.

It was found that this task would require a real-time implementation of the empirical mode decomposition, but no suitable implementations were available. Since a real-time EMD implementation would be useful far beyond the scope of ultrasonic bubble detection, the creation of such an implementation was made the goal of this thesis.

Summary of results

A streaming-data EMD implementation was programmed in MATLAB and tested with a variety of inputs.

TIME PERFORMANCE: It was found that it could process real-world 22kHz audio data at about 10% of real-time speed. Optimisation was not attempted, but profiling revealed distinct potential for acceleration to real time. Alternately, the program could potentially be used for less demanding real-time applications as-is, by downsampling the input signal to a lower data

rate, or adjusting the configuration parameters. The functionality for doing this is already implemented.

OUTPUT QUALITY: The outputs from the online and offline implementations were very similar for simple signals, but quite different for complex signals. This is not necessarily incorrect, as the EMD is defined by an algorithm - the set of IMFs for a given signal is not unique. More testing is required to determine the correctness of the online implementation's output.

RELEASE: The MATLAB code has been placed online for public use, testing and comments. It is available at `http://www.penwatch.net/realtime_emd/`.

Directions for future work

The implementation does not yet run in real time. Profiling revealed that the majority of execution time is spent performing polynomial interpolation (`hermite_spline()`); therefore increasing the speed of this function alone would yield a large performance gain.

The MATLAB implementation has been successful as a proof-of-concept, and has been coded as an example implementation for others to learn from. The ideas from this implementation should therefore be easily transferable to a fast C implementation; if further acceleration is required, the C implementation would make it possible to use GPGPU capability (such as nVidia CUDA) for hardware acceleration.

Finally, much more testing with a large variety of input signals will be required to determine if the algorithm is correct.

References

[1] R. V. Shohet, S. Chen, Y.-T. Zhou, Z. Wang, R. S. Meidell, R. H. Unger, and P. A. Grayburn, "Echocardiographic Destruction of Albumin Microbubbles Directs Gene Delivery to the Myocardium," *Circulation*, vol. 101, no. 22, pp. 2554–2556, 2000.

[2] M. J. K. Blomley, J. C. Cooke, E. C. Unger, M. J. Monaghan, and D. O. Cosgrove, "Science, medicine, and the future: Microbubble contrast agents: a new era in ultrasound," *BMJ*, vol. 322, no. 7296, pp. 1222–1225, 2001.

[3] M. Cullinane, G. Reid, R. Dittrich, Z. Kaposzta, R. Ackerstaff, V. Babikian, D. W. Droste, D. Grossett, M. Siebler, L. Valton, and H. S. Markus, "Evaluation of new online automated embolic signal detection algorithm, including comparison with panel of international experts," *Stroke*, vol. 31, no. 6, pp. 1335–1341, 2000.

[4] C. Shilling, M. Werts, and N. Schandelmeier, *The underwater handbook: A guide to physiology and performance for the engineer*. Plenum Press, New York, 1976.

[5] C. W. Dueker, *Medical Aspects of Sport Diving*. A. S. Barnes and Co., 1970.

[6] R. Thomas and B. McKenzie, *The Diver's Medical Companion*. Sydney: Medical Diving Centre, 1979.

[7] T. Beckman, "A review of decompression sickness and arterial gas embolism," *Archives of family medicine*, vol. 6, no. 5, p. 491, 1997.

[8] R. Walker, *Diving and Subaquatic Medicine*, ch. 6, pp. 55–71. Edward Arnold, 2002.

[9] K. Kizer, "Delayed treatment of dysbarism: a retrospective review of 50 cases," *Jama*, vol. 247, no. 18, p. 2555, 1982.

[10] R. Nishi, A. Brubakk, and O. Eftedal, *Bennett and Elliott's Physiology and Medicine of Diving*, ch. 10.3, pp. 501–529. Saunders, 2003.

[11] K. Tufan, A. Ademoglu, E. Kurtaran, G. Yildiz, S. Aydin, and S. M. Egi, "Automatic detection of bubbles in the subclavian vein using doppler ultrasound signals," *AVIATION SPACE AND ENVIRONMENTAL MEDICINE*, vol. 77, pp. 957–962, SEP 2006.

[12] O. S. Eftedal, *Ultrasonic detection of decompression induced vascular microbubbles*. PhD thesis, Norwegian University of Science and Technology, 2007.

[13] N. Pollock, "Transthoracic Echocardiography (TTE)-A Tool to Monitor Unsafe Decompression Stress.," 2003.

[14] W. N. McDicken, *Diagnostic Ultrasonics: Principles and Use of Instruments*, ch. Ultrasonic wave phenomena in tissue, pp. 41–61. Crosby Lockwood Staples, 1976.

[15] F. W. Kremkau, *Diagnostic Ultrasound: Principles and Instruments (Diagnostic Ultrasound: Principles & Instruments (Kremkau))*. Saunders, 2005.

[16] J. T. Bushberg, J. A. Seibert, E. M. Leidholdt, and J. M. Boone, *The Essential Physics of Medical Imaging*. Lippincott Williams & Wilkins, 2002.

[17] J. Buckey, D. Knaus, D. Alvarenga, M. Kenton, and P. Magari, "Dual-frequency ultrasound for detecting and sizing bubbles," *Acta Astronautica*, vol. 56, no. 9-12, pp. 1041–1047, 2005.

[18] M. Chappell and S. Payne, "A method for the automated detection of venous gas bubbles in humans using empirical mode decomposition," *Annals of biomedical engineering*, vol. 33, no. 10, pp. 1411–1421, 2005.

[19] O. Eftedal and A. Brubakk, "Detecting intravascular gas bubbles in ultrasonic images," *Medical and Biological Engineering and Computing*, vol. 31, no. 6, pp. 627–633, 1993.

[20] K. Hatteland and B. Semb, "Gas bubble detection in fluid lines by means of pulsed doppler ultrasound," *Scandinavian Cardiovascular Journal*, vol. 19, no. 2, pp. 119–123, 1985.

[21] K. Kisman, "Spectral analysis of doppler ultrasonic decompression data," *Ultrasonics*, vol. 15, no. 3, pp. 105 – 110, 1977.

[22] H. Markus, M. Cullinane, and G. Reid, "Improved automated detection of embolic signals using a novel frequency filtering approach," *Stroke*, vol. 30, pp. 1610–1615, AUG 1999.

[23] C. Valens, "A really friendly guide to wavelets," 2004.

[24] P. Lui, B. Chan, F. Chan, P. Poon, H. Wang, and F. Lam, "Wavelet analysis of embolic heart sound detected by precordial Doppler ultrasound during continuous venous air embolism in dogs," *Anesthesia & Analgesia*, vol. 86, no. 2, p. 325, 1998.

[25] N. Aydin, F. Marvasti, and H. Markus, "Embolic doppler ultrasound signal detection using discrete wavelet transform," *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE*, vol. 8, pp. 182–190, JUN 2004.

[26] N. Huang, Z. Shen, S. Long, M. Wu, H. Shih, Q. Zheng, N. Yen, C. Tung, and H. Liu, "The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis," *Proceedings: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1971, pp. 903–995, 1998.

[27] M. Chappell, *Modelling and Measurement of Bubbles in Decompression Sickness.* PhD thesis, University of Oxford, 2006.

[28] G. Rilling, P. Flandrin, and P. Gonçalvès, "On empirical mode decomposition and its algorithms," in *IEEE-EURASIP workshop on nonlinear signal and image processing NSIP-03, Grado (I)*, 2003.

[29] R. Meeson, "Hht sifting and adaptive filtering," tech. rep., INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA, 2003.

[30] R. Rato, M. Ortigueira, and A. Batista, "On the HHT, its problems, and some solutions," *Mechanical Systems and Signal Processing*, vol. 22, no. 6, pp. 1374–1394, 2008.

[31] N. Huang, M. Wu, W. Qu, S. Long, and S. Shen, "Applications of Hilbert-Huang transform to non-stationary financial time series analysis," *Applied Stochastic Models in Business and Industry*, vol. 19, no. 3, pp. 245–268, 2003.

[32] L. Loudet, "Application of empirical mode decomposition to the detection of sudden ionospheric disturbances by monitoring the signal of a distant very low frequency transmitter.".

[33] C. Loh, T. Wu, and N. Huang, "Application of the empirical mode decomposition-Hilbert spectrum method to identify near-fault ground-motion characteristics and structural responses," *Bulletin of the Seismological Society of America*, vol. 91, no. 5, p. 1339, 2001.

[34] *MA2201: Numerical Mathematics lecture notes*. James Cook University, Townsville, 2007.

[35] G. D. Knott, *Interpolating Cubic Splines*. Birkhauser Boston, 2000.

[36] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*. Brooks/Cole, 4 ed., 1999.

[37] C. Moler, *Numerical Computing with MATLAB*. SIAM, Philadelphia, 2004.

[38] P. Ivanov, L. Amaral, A. Goldberger, S. Havlin, M. Rosenblum, Z. Struzik, and H. Stanley, "Multifractality in human heartbeat dynamics," *Arxiv preprint cond-mat/9905329*, 1999.

[39] J. O. Smith, *Mathematics of the Discrete Fourier Transform (DFT)*. Dept. of Music, Stanford University, 2002.

[40] C. F. Van Loan, *Introduction to Scientific Computing - A Matrix-Vector Approach using MATLAB*. Prentice-Hall, 1997.

[41] O. Eftedal, S. Lydersen, and A. Brubakk, "The relationship between venous gas bubbles and adverse effects of decompression after air dives," *UNDERSEA AND HYPERBARIC MEDICINE*, vol. 34, no. 2, p. 99, 2007.

[42] A. Erde and C. Edmonds, "Decompression sickness: a clinical series," *Journal of Occupational and Environmental Medicine*, vol. 17, no. 5, p. 324, 1975.

[43] K. Kumar, M. Powell, and J. Waligora, "Evaluation of the risk of circulating microbubbles under simulated extravehicular activities after bed rest," 1993.

[44] A. Støylen, "Basic ultrasound, echocardiography and doppler for clinicians." Website, March 2010.

[45] J. Wilbur, S. Phillips, T. Donoghue, D. Alvarenga, D. Knaus, P. Magari, and J. Buckey, "Signals consistent with microbubbles detected in legs of normal human subjects after exercise," *Journal of Applied Physiology*, vol. 108, no. 2, p. 240, 2010.

[46] N. Huang and Z. Wu, "A review on Hilbert-Huang transform: Method and its applications to geophysical studies," *Reviews of Geophysics*, vol. 46, no. 2, 2008.

[47] G. Rilling and P. Flandrin, "One or two frequencies? The empirical mode decomposition answers," *IEEE Transactions on Signal Processing*, vol. 56, no. 1, pp. 85–95, 2008.

[48] A. Boussuges, D. Carturan, P. Ambrosi, G. Habib, J. Sainty, and R. Luccioni, "Decompression induced venous gas emboli in sport diving: Detection with 2 d echocardiography and pulsed doppler," *International journal of sports medicine*, vol. 19, no. 1, pp. 7–11, 1998.

[49] H. S. Markus and C. H. Tegeler, "Experimental aspects of high-intensity transient signals in the detection of emboli," *Journal of Clinical Ultrasound*, vol. 23, no. 2, pp. 81–87, 1995.

A

MATLAB implementation of Offline EMD

As noted in Sec. 2, an actual MATLAB implementation of the EMD looks fairly close to the pseudocode. A basic implementation is given below:

```
1  function [IMFs, res] = Offline_EMD (x)
2  NUMBER_OF_IMFS = 5;
3  xlen = numel(x);
4
5  IMFs = zeros(NUMBER_OF_IMFS,xlen); % Preallocate for speed
6  res = x;
7  for n = 1:NUMBER_OF_IMFS
8      [mode,res] = Extract_Mode (res );
9      IMFs(n,:) = mode;
10 end
11 end
12
13 function [mode, res] = Extract_Mode (x)
14 NUM_SIFTING_ITERATIONS = 20;
15 xlen = numel(x);
16 t_spline = 1:xlen;
17
18 d = zeros(NUM_SIFTING_ITERATIONS+1,xlen); % Preallocate for speed
19 d(1,:) = x;
20
21 for n = 1:NUM_SIFTING_ITERATIONS
22     [imin, imax, ~] = extr(d(n,:));
23     vmin = d(n,imin);
24     vmax = d(n,imax);
25
```

```
26      env_min = hermite_spline(imin,vmin,t_spline);
27      env_max = hermite_spline(imax,vmax,t_spline);
28      env_mean = (env_min + env_max) ./ 2;
29      d(n+1,:) = d(n,:) - env_mean;
30  end
31
32  mode = d(end,:);
33  res = x - mode;
34  end
```

The `Offline_EMD()` function accepts a single input `x` – a vector of data to decompose into IMFs – and produces two outputs. The first, `IMFs`, is a matrix containing the calculcated IMFs, each in a separate row. The second, `res`, is the residue left over after IMF extraction.

The current behaviour is to extract a fixed number of IMFs, but a trivial modification to the `Offline_EMD()` function would make it "extract IMFs until the residue is very small".

Similarly, the number of sifting iterations for extracting each IMF is fixed; this could be changed to use a stopping condition instead. A suitable condition would be `max(abs(env_mean)) < epsilon`, where ϵ is some desired tolerance; this would force the extracted IMFs to be approximately zero-mean in every case.

This MATLAB code uses two non-standard MATLAB functions:

- `extr()` by Gabriel Rilling for finding local maxima, local minima, and zero crossings – available from `http://perso.ens-lyon.fr/patrick.flandrin/emd.html`, file `pack_emd.zip/package_emd/utils/extr.m`.
- `hermite_spline()`, a slight modification of the `pchip_tx()` by Cleve Moler. `pchip_tx()` is a reference implementation of MATLAB's `pchip()` "piecewise Hermite interpolation polynomial" function. You can substitute `spline()` or `pchip()` for `hermite_spline()` with only minor changes in the output.

B

User Manual: Online EMD program

Usage of the MATLAB code is fairly straightforwards. We will give a short example:

```
% Generate some test data
t = 0:0.001:100;
x = sin(2*t) + sin(3*t) + sin(5*t);

% Initialise Buffer object and EMD object
data_buffer = Buffer(100,1,5000,'No filter',0);
EMD_Object = EMD(data_buffer);

% Feed data into the system, one block at a time.
blocksize = 1000;                       % Samples/block.
num_blocks = floor(numel(x)/blocksize); % Number of blocks to be fed in.

for n = 1:num_blocks
    new_block = x( 1+(n-1)*blocksize : n*blocksize );
    data_buffer.Update(new_block); % Feed the block of data to the buffer
    EMD_Object.Update();           % Update all IMF calculations.
end

plot(EMD_Object.IMFs(1).mode);          % Plot the first IMF.
```

In this example, we create a EMD object with default options and then simulate a streaming-data application, by parcelling out a large data array in small blocks. We plot the first IMF after all the data has been streamed into the EMD; of course the data can also be extracted and used at any time, not just after the streaming is complete.

Generally, the larger the block size, the more latency in IMF extraction, but the faster the performance and the less chance of encountering IMF end effects.

B.1 Buffer object

The `Buffer` object was designed to be temporary storage for the incoming data stream.

Pushing data in: Data is pushed into the `Buffer` using the `Buffer.Update(new_data_vector)` function. To limit the amount of memory used, `Buffer` behaves like a circular buffer; the length of the buffer is finite, and the arrival of new data pushes old data out of memory.

Time offsets: Two counters, `n_earliest` and `n_latest`, keep track of the time offsets from the start of the data stream ($t = 1$) to the beginning and end of the buffer, respectively. These counters can be read with the `Buffer.Earliest()` and `Buffer.Latest()` functions.

Retrieving time slices: The implementation of Buffer as a circular buffer means that the input sample from $t = 1000$ will rarely live at index 1000 of the array. The complexity of accessing particular time slices of data from the `Buffer` is abstracted by the accessor function `Buffer.Get(n_start, n_end)`, which calculates the correct indexes into the real array.

For example, the slice of data from $t = 100$ to $t = 200$ would be accessed by calling `Buffer.Get(100,200)`. An exception is raised if the requested time slice oversteps the bounds of the `Buffer's` data array (either because the data requested has been pushed out of the buffer, or data was requested from the future.)

Retrieving the intersection between Buffers: The `Buffer.CommonSection()` function retrieves the overlapping parts of two or more `Buffers`. For example, if `B1` contains data from $t = [100 : 200]$ and `B2` contains data from $t = [150 : 250]$, then `Buffer.CommonSection([B1, B2])` will return the time slice $t = [200 : 250]$ from both `Buffers`.

Prefiltering: The `Buffer` object supports pre-filtering of the input data with a MATLAB filter object before it is stored to the `Buffer`. Any MATLAB `dfilt` or `mfilt` object can be specified when the `Buffer` is created; if no filtering is desired, specify the string `'No filter'`.

Decimation: One application of prefiltering is to decimate the incoming data to a lower sampling rate, which reduces the computation required to process the data later. The standard `mfilt.firdecim` can be used to perform lowpass anti-aliasing before decimation, but this may introduce variable group delay into the output.

If variable group delay is not acceptable, the `mfilt` object can be made to perform a rectangular moving average instead; this will delay all components by the same time. The function `NewMovingAverageFilter.m` was written to automate the creation of these filters.

Since all FIR and moving average filters will introduce group delay in the output, a `delay` parameter can be specified when the `Buffer` is created. This parameter will offset all data retrievals by the specified time in seconds, effectively cancelling out the group delay by making the Buffer lag behind the input data.

Example application: See `HeartbeatDetection/DetectHeartbeats.m` for a real-world example – a simplistic implementation of Chappell's heartbeat peak detection algorithm in real-time.

C

Testing Framework

NOTE: This document may lag behind the latest release of the code. In case of conflict between this documentation and comments in the source code, consider the source code authoritative.

C.1 EMD_Test() usage

The testing framework is used by making calls to the function `EMD_Test()`; the list of arguments defines the parameters for the test.

Argument	Type	Description
`test_name`	string	Human-readable test description - i.e. `'EMD Test with triangle-wave input'`.
`input_filename`	string	Path to a `.mat` file containing a vector of data, with variable name `raw_data`. Specifying a `.mat` filename that does not contain a vector of data called `raw_data` is flagged as an error.

Argument	Type	Description
`input_length`	integer	How many samples of `raw_data` to use as inputs to the EMD. Example: `input_length = 10000` uses the first 10,000 elements of `raw_data`. Specifying `input_length` to be larger than the number of samples of `raw_data` is flagged as an error.
`repeat_count`	integer	How many times to repeat the test.
`EMD_config`	associative array (`containers.Map`)	[optional] - configuration options for the EMD and testing process. See `EMD_Default_Config.m` (Appendix C.3) for documentation of available options. Defaults to `EMD_Default_Config()` if no argument is given.
`output_directory`	string	Directory to which the output files `log.txt`, `IMFs.pdf`, and `data.mat` will be written. If the directory exists, the directory and all contents will be deleted.
`plot_params`	struct	Struct with three fields: `plot_params.PlotP` - true to display and save plot as .pdf. `plot_params.LimX` and `plot_params.LimY` - same meaning as for normal plots. If plot_params is not provided, it is assumed that no plots are desired.

C.2 EMD_Test() example

```
1  fs = 22050;
2  t = [1/fs:1/fs:10] * 2*pi;
3  raw_data = sin(100*t) + sin(300*t) + sin(500*t);
4
5  save('sin_t_sin_3t.mat','raw_data');
6  EMD_Test(...
7      'Basic test', ...
8      './sin_t_sin_3t.mat', ...
9      numel(raw_data), ...
10     20, ...
11     EMD_Default_Config(), ...
12     '/home/lws/tmp/EMD_Test/' );
```

C.3 Standard Test Settings

The standard test settings are specified in `EMD_Default_Config.m`. This is reproduced below:

```
1  function config = EMD_Default_Config()
2  config = containers.Map;
3
4  %% Testing configuration options.
5  config('TESTING:BLOCK_SIZE') = 2000;
6
7  % See Buffer class for explanations of these options.
8  config('TESTING:BUFFER:SAMPLING_FREQUENCY') = 22050; % 22.050 kHz. No effect in
       practice.
9  config('TESTING:BUFFER:DECIMATION_RATE') = 1;        % Decimation rate. 1 = no
       decimation.
10 config('TESTING:BUFFER:HISTORY_LENGTH') = 22050*10;  % Number of samples retained in
       buffer.
11 config('TESTING:BUFFER:FILTER') = 'No filter';       % Do not apply any digital
       filtering to bufffer data.
12 config('TESTING:BUFFER:PROPAGATION_DELAY') = 0;      % No delay between buffer input
       and output.
13
14 %% EMD configuration options.
15 % Number of IMFs to extract from input data stream.
16 config('EMD:NUMBER_OF_IMFS') = 5;
17
18 %% IMF configuration options.
19 % Number of sifting iterations applied to each block of IMF.
20 config('IMF:SIFT_BLOCK:NUM_SIFTING_ITERATIONS') = 10;
21
22 % Controls amount of historical data included in each IMF block calculation.
23 config('IMF:SIFT_BLOCK:NUM_EXTREMA_PRECEDING') = 10;
24
25 % Controls how much of the right of each block is thrown away to guard against
26 % end effects.
27 config('IMF:SIFT_BLOCK:NUM_RIGHT_TRIM_EXTR') = 3;
28
29 % Turns mirrorising of extrema on or off. Default on, which theoretically
30 % reduces end effects.
31 config('IMF:SIFT_BLOCK:MIRRORISE_EXTREMA') = true;
32
33 % Valid choices are:
34 % 'Hermite' -> hermite_spline()
35 % 'PCHIP' -> MATLAB's pchip()
36 % 'Cubic' -> MATLAB's spline().
37 config('IMF:SIFT_BLOCK:ENVELOPE_INTERPOLATION_METHOD') = 'Hermite'; % Other valid
       choices: 'PCHIP', 'Cubic'
38
39 %% Debug options.
40 % Debug options which modify the behaviour of the program.
41 config('IMF:SIFT_BLOCK:DEBUG:USE_ALL_HISTORIC_DATA') = false;
42
43 % Debug output options - for displaying pretty graphs.
44 config('IMF:SIFT_BLOCK:DEBUG:DISPLAY_EACH_SIFTING_ITERATION') = false;
45 config('IMF:SIFT_BLOCK:DEBUG:DISPLAY_BLOCK_OVERLAP_DIAGNOSTIC') = false;
46 config('IMF:SIFT_BLOCK:DEBUG:DISPLAY_EACH_BLOCK') = false;
47 end
```