An Application of Fractal Analysis and Wearable Tech to the Health of the Heart

Erik S. Peterson
M.S. Applied and Computational Mathematics Candidate

Dr. Bruce Peckham
Advisor

University of Minnesota Duluth
Department of Mathematics and Statistics
August 17, 2015

Acknowledgements

The following are those who helped in some way with my paper, either by serving on my committee, proofreading my paper, helping me practice my presentation, attending my presentation, helping me refine my explanation of my project, or helping in any other way. Or being my advisor, and helping in *all* those ways. Special mention also goes to those who helped me move both times when I had a temporary job in the Twin Cities in the middle of my project, as that was an ordeal in itself. My sincerest thanks to everyone listed here (listed in no particular order):

Dr. Bruce Peckham
Dr. Guihua Fei
Dr. John Pastor
Aaron Potvien
Darren Skjoelsvold
Alicia Skjoelsvold
Beverly Peterson
Bryan Peterson
Dennis Peterson
Donna Peterson
Barb Schweda
Ted Schweda
Ian Vincent
Kristina Hill
Heather Brink
Joe McDonald
Doug Paulson
Steve Robinson
Daniel Oyinloye
Uncle Mike
Nate Pearson
Lucas Streng
Dr. Dalibor Froncek
Pamela Arola
Elee Vang
Ferry Vue
Jesse Peterson
Sarah Peterson
Joseph Crowe
Dustin Cahill
Dr. Steve Matthews
Chad Pierson
Dr. Susan Nordin
Troy Nellis
Zack Filipovich
Dr. David Worley
Dr. Harlan Stech
Sam McCurry
Aunt Diane
And many more. Thank you!

And to all my friends and family who provided moral support as I worked on this research. I wish I could list them all, but that would exceed the limits of this page (I would need a microscopic font). That said, if I forgot anyone who helped in any of the ways mentioned above (as it was a long project), then I owe them a drink (or snack, or something). If that's you, just give me a call and say "Hey, where am I?" and let me know your preferred (single-digit-priced; keep it reasonable) drink or snack.

Thank you to everyone who also provided encouragement as I worked on this project, even in small ways! Though there may not have been enough room to list everyone in that category, it meant more than words can say (even written across the entire surface of the earth at a nanoscopic font).

This paper is dedicated to the memory of my grandfather, who taught me many concepts in math well in advance of when my school did so, and set me on the path towards earning my Master’s.

Abstract

Wearable Tech is a new arrival to the marketplace, primarily in the form of smart watches and fitness trackers. But imagine if our clothes themselves were wearable computers with greater capability than current smart phones, and both felt and fit no different than ordinary clothing. Smart Fabrics are being developed by a number of different companies and labs right now, and a type that can monitor vitals is already available in some markets. With a form of fractal analysis called detrended fluctuation analysis (DFA), they could be programmed to take the heartbeat time series data they're gathering and use it to detect whether or not there could be a problem well before it becomes a crisis, and do so in a way that's clear to the wearer whether or not they need to see their doctor. The result is a single number, where 1 means their heart is healthy, and anything else means they definitely need a checkup. In this paper, I demonstrate how fractals and smart fabrics combine to make a detrended fluctuation analysis possible in real time, and how this critical number is calculated. The combination of progress in all these areas could allow for better early detection of a problem with the heart.

Table of Contents

1.0 INTRODUCTION

Fractals are simply amazing, both visually and mathematically. Fractals are a key place where the left side and right side of the brain meet, or, to be clearer, where math and art combine. It could even be said that they are the lovechild of mathematics and art, with far more of the genes of art than *Euclidean* geometry (fractals are still geometric objects, but non-Euclidean ones **[1—4]**). When rendered by computer and filled in with artificial coloring, fractals look like unique works of art and are more visually striking than the typical geometrical objects that most people recognize. Self-repeating and self-similar across the scales, if one zooms in to a part of the object, that small part will resemble the whole **[1—3]**. A spiral is the simplest example of such a fractal, so anything that is fundamentally spiral in nature can be described and understood using fractals **[2]**. Knowing that, it is easy to see that fractals do a better job of modeling items in nature than Euclidean shapes. A tree, for instance, can have branches that resemble the overall forest, and the segments of the leaves can resemble the branches. Because anything spiral is also fractal, they can help model spiral galaxies, spiral nebulae, shells, and anything else that is spiral in shape **[2]**. If we did not already understand spiral staircases (seeing as we created them), we could use fractals to model them too.

Fractals are everywhere, all around us: from the clouds in the sky, to the coastlines of each country, and to the structure of the organs and genes in our bodies **[3 – 4]**. Fractals can therefore be used to understand the systems in the body, particularly the structure of lungs, kidneys, blood vessels and the whole circulatory system **[4]**. Peter Burns of the University of Toronto has observed that the fractal structure is different between a healthy organ and one with cancer, and so sees fractals as a way to develop mathematical models that could help detect cancer earlier **[4]**. Heartbeat time series are also fractals, and the eyes even take in information in a fractal manner **[4]**. The specific focus of this paper will be on the fractal nature of heartbeat time series.

Prior to describing the application of fractals and fractal analysis to the health of the heart, more in-depth mathematical background about fractals will be provided, along with the basics about nanotechnology and how to construct smart fabrics. Smart fabrics make the fractal analysis of the heartbeat possible, because they constantly monitor the heartbeat and can be programmed to run a fractal analysis on the resulting time series data. Constructing smart fabrics involves nanotechnology, so the background of this field is provided as well, through a brief literature review of the current state of progress made with this emerging technology. I will further explain how the synthesis of these disparate fields could allow for better early detection of the health of the heart, and demonstrate the program I created that makes this form of early detection possible. Smart fabrics would provide a more convenient and comfortable way to test this early detection method on larger groups than past studies, as well. Finally, I will touch on the future potential of a more mature form of this technology, and describe some remaining questions that programmable smart fabrics would be capable of answering.

Note: The data points in some of the graphs are distorted, due to formatting incompatibilities between Mathematica, different versions of Word, and the PDF. They didn't appear until after the export-to-PDF step. If you want to view the better-quality Word version, go to http://d.umn.edu/~bpeckham/students.html#Students and look for my name.

2.0 MATH BACKGROUND

Mathematically, a fractal is a set with a fractal dimension that equals or exceeds its topological dimension (when the fractal dimension exceeds its topological dimension, it is said to have fractal geometry) **[5 – 6]**. The topological dimension is the type of dimension more commonly thought of when the term “dimension” is used, and the one more familiar to most people. It is defined inductively: *A set is of dimension zero if for any point* $p \in X$ *there are arbitrarily small neighborhoods of* p *whose boundary is empty. A set is of dimension* D *if there are arbitrarily small neighborhoods of any point* p *whose boundary is of* $\text{dimension} \leq D - 1$ **[7]**.

Unlike the topological dimension, the fractal dimension is not restricted to the more familiar integer dimensions, and may even fall between two integers **[6]**. For instance, the fractal that looks like a jagged line, that resembles either a coastline or an X-ray image of a bone fracture, can have a dimension that falls between 1 and 2 **[5]**. While a set's fractal dimension can exceed its topological dimension, it can't exceed the dimension of the ambient space that contains it.

Different forms of the fractal dimension are the box dimension, information dimension, correlation dimension, Hausdorff dimension, and packing dimension **[6] [8]**. Each is a different way of measuring the fractal's change in detail with its change in scale. The most important one to look at in detail for this paper is the box dimension.

Box dimension (alternately known as the box-counting dimension, Minkowski dimension, or Minkowski-Bouligand dimension) involves placing the fractal within an evenly-spaced k-dimensional grid, and then counting how many boxes are necessary to cover the fractal. To calculate what its box dimension is, you make the grid increasingly finer, and then take the limit of the logarithm of the number of boxes divided by the logarithm of the inverse of the length of each box, as the length of each box goes to 0 **[6]**. So, using *L* to represent the length of the box and *num* to represent the number of boxes, where *f* is the set defining the fractal, the calculation for the box dimension is:

$$dim_{box}(f) = \lim_{L \to 0} \frac{\log(num(L))}{\log(1/L)} \qquad \textbf{\{1\}}$$

Note that this implies that $(num(L)) \cong (1/L)^{dim_{box}(f)}$.

If the above limit does not exist, then the calculation must be addressed in terms of the upper box dimension (alternately known as the entropy dimension, Kolmogorov dimension, Kolmogorov capacity, or upper Minkowski dimension) and the lower box dimension (alternately known as the lower Minkowski dimension). The box dimension is not considered to be well-defined if these values are different (when the above limit does not exist) **[6]**.

Fractals exhibit a type of symmetry called self-similarity, with respect to the fact that their patterns repeat across the scales, as you zoom in or out. There are several different forms of self-similarity: exact self-similarity, quasi-self-similarity, and statistical self-similarity. Exact self-similarity is when a part of the fractal looks exactly identical to the whole, so that the same shapes repeat when you zoom in or out on it. Quasi-self-similarity is when part of the fractal looks like an approximation to the whole. Statistical self-similarity is when parts of the fractal show the same statistical properties as the whole across the scales **[6]**.

Furthermore, unlike Euclidian geometric shapes, fractals can have irregular patterns and are related to chaotic behavior when they arise as invariant sets for chaotic dynamical systems (both as sets and as behavior on those sets) **[3]**. Many of the items in nature are also more likely to have a fractal appearance and structure than Euclidian shapes. Natural objects that are fractal in structure include trees, clouds, snowflakes, seashells, river systems, mountain ranges, coastlines, other complex landscapes, biological systems, the large-scale structure of the universe, and things with spiral structure (such as spiral shells, spiral galaxies, and spiral nebulae) **[2] [4] [8 – 9]**. Further detail regarding the fractal nature of several of these objects can be found in the appendix.

Fractals arise in a variety of ways for dynamical systems, with several different generating techniques (that primarily involve set removal and random iteration). For example, there are iterated function systems (described below), strange attractors, L-systems (along with strange attractors, these involve iterating a single orbit), escape-time fractals (where the fractal is the boundary between points that escape and points that don't, such as the Julia and Mandelbrot sets in the complex plane), and random fractals **[5] [10]**. Each is a different way of using a mathematical formula or computer code to generate the same fractal. An example of how to generate an iterated function system in Mathematica is located below.

A Simple way of Generating Fractals:

Iterated function systems are ones in which the fractal is formed by a predefined geometric replacement at each step. It starts with a specific pattern, typically some compact set in R^H, which is the 0th iteration. Then for each new iteration, replace that pattern into each particular subinterval, and continue for enough steps until an approximation of how the desired pattern should look repeating itself across the scales is achieved **[6]**. An example would be how the Koch Snowflake can be built. Start with an equilateral triangle for the 0th iteration. In the next step, extend smaller equilateral triangles (without the base) from each side of the previous triangle. In the next step, extend even smaller triangles (also without the base) from each side of the previous triangles. After several steps, the result will look like a snowflake that is self-repeating across the scales. The fractal is the limit of continuing this process indefinitely **[3]**.

The code for the Sierpinski Triangle and Sierpinski Carpet that I created in Mathematica is located on the next several pages. I generated the Sierpinski Triangle first, starting by defining specific starting points, and then applying transformations to that set of points. The initial set of points I used were $\{(0,0), (1,0), (1/2, \sqrt{3}/2)\}$, so that a filled equilateral triangle would be created when I used the Polygon function on that set of points. For the Sierpinski Carpet, to create a square, I used $\{(0,0),(1,0),(1,1),(0,1)\}$.

The starting triangle and square are located below:

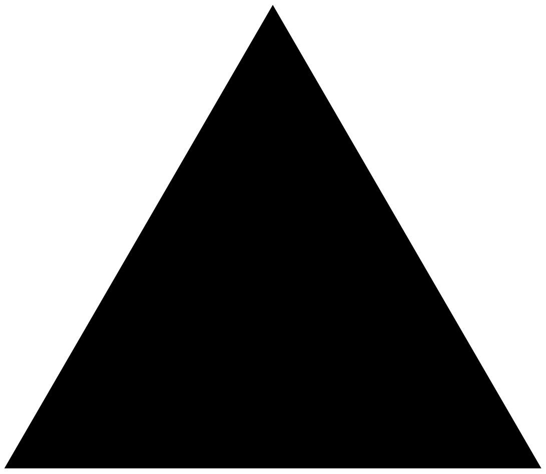


Figure 1: The 0th iteration of the Sierpinski Triangle, a Euclidean equilateral triangle.

Figure 2: The 0th iteration of the Sierpinski Carpet, a Euclidean square.

Next, I used specific transformations to generate each successive iteration's set of points from the original. For the Sierpinski Triangle, the following code shows the transformations I used:

```
T1[x_]:={1/2*x[[1]],1/2*x[[2]]}
```

(This transformation shrinks the triangle down to one with a quarter of the area of the original by reducing the side length to half of the original, and places it in the lower left corner of where the original was).

```
T2[x_]:={1/2*x[[1]]+1/2,1/2*x[[2]]}
```

(This transformation shrinks the triangle as above, and places it in the lower right corner of where the original was).

```
T3[x_]:={1/2*x[[1]]+1/4,1/2*x[[2]]+Sqrt[3]/4}
```

(This transformation shrinks the triangle as above, and places it in the upper corner of where the original was).

In the first iteration of the Sierpinski Triangle, each triangle that remains, after “removing” the middle equilateral triangle (the union of the above transformations is the geometric equivalent to removing the middle triangle), is a quarter of the area of the original triangle. So the transformations need to generate three such triangles of this smaller scale, which also share common endpoints with the original triangle. The first transformation generates the triangle that starts at (0,0), so it only needs to multiply all the points in the set by $1/2$ in order to generate the smaller triangle (whose area is a quarter of the original). The next transformation needs to shift the set by $1/2$ along the x-axis as well as the scaling, so that one of its endpoints will still be (1,0). And the last transformation needs to have the resulting triangle start halfway towards the endpoint of $(1/2, \sqrt{3}/2)$, so it must shift the triangle by $1/4$ along the x-axis and by $\sqrt{3}/4$ along the y-axis. A new set of points is generated by mapping all three of these transformations to the original set, resulting in a set that has nine points instead of three. More triangles that are even smaller get generated by applying the transformations to this new set of points. The resulting figure is closer and closer to the Sierpinski Triangle with each iteration. The complete Sierpinski Triangle is achieved as a limit to this process continuing indefinitely.

The following transformations were used to generate the Sierpinski Carpet:

```
T1[x_]:={1/3*x[[1]],1/3*x[[2]]}
```
(This transformation shrinks the square to one with a ninth of the area as the original by reducing the side length to a third of the original, and places it in the lower left corner of where the original was).
```
T2[x_]:={1/3*x[[1]]+1/3,1/3*x[[2]]}
```
(This transformation shrinks the square as above, and places it in the lower center of where the original was).
```
T3[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]}
```
(This transformation shrinks the square as above, and places it in the lower right corner of where the original was).
```
T4[x_]:={1/3*x[[1]],1/3*x[[2]]+1/3}
```
(This transformation shrinks the square as above, and places it in the center left of where the original was).
```
T5[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]+1/3}
```
(This transformation shrinks the square as above, and places it in the center right of where the original was).
```
T6[x_]:={1/3*x[[1]],1/3*x[[2]]+2/3}
```
(This transformation shrinks the square as above, and places it in the upper left corner of where the original was).
```
T7[x_]:={1/3*x[[1]]+1/3,1/3*x[[2]]+2/3}
```
(This transformation shrinks the square as above, and places it in the upper center of where the original was).
```
T8[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]+2/3}
```
(This transformation shrinks the square as above, and places it in the upper right corner of where the original was).

In the first iteration of the Sierpinski Carpet, after removing the middle square, the remaining squares are a ninth of the area of the original square. So the transformations must shrink the length and width by a factor of three, and shifted around as needed to meet up with the original endpoints. The eight transformations above generate the new set of points for the eight squares that circle the missing center square. Applying the transformations to this new set generates more squares that are even smaller yet, and the resulting figure is closer and closer to the Sierpinski Carpet with each iteration. Just as with the Sierpinski Triangle, the complete Sierpinski Carpet is achieved as a limit to this process.

The following pieces of code are the Do loops used to generate both the several iterations of the Sierpinski Triangle and Sierpinski Carpet:

```
Do[{t[j]={Map[T1,s[j-1]],Map[T2,s[j-1]],Map[T3,s[j-1]]},
s[j]=Flatten[t[j],1],l[j]=Partition[s[j],3],Print[Graphics[Polygon[l[j]]]]},{j,1,i}]

Do[{t[j]={Map[T1,s[j-1]],Map[T2,s[j-1]],Map[T3,s[j-1]],   Map[T4,s[j-1]],Map[T5,s[j-1]],
Map[T6,s[j-1]],Map[T7,s[j-1]],Map[T8,s[j-1]]},s[j]=Flatten[t[j],1],l[j]=Partition[s[j],4],
Print[Graphics[Polygon[l[j]]]]},{j,1,i}]
```

Each time a transformation is applied, it places the resulting set inside an extra set of brackets, and too many extraneous brackets prevent the Polygon function from working properly. So the Flatten function is used in the loop to remove these extra sets of brackets. Then Partition is used to separate all the points in the set into groups of three or four (depending on whether it's the Triangle or the Carpet that's being generated), so that the Polygon function will generate either multiple triangles or squares when applied to the whole set. Finally the Print function is used to display the particular iteration of the fractal being generated. Located below are the remaining iterations of the Triangle and the Carpet:

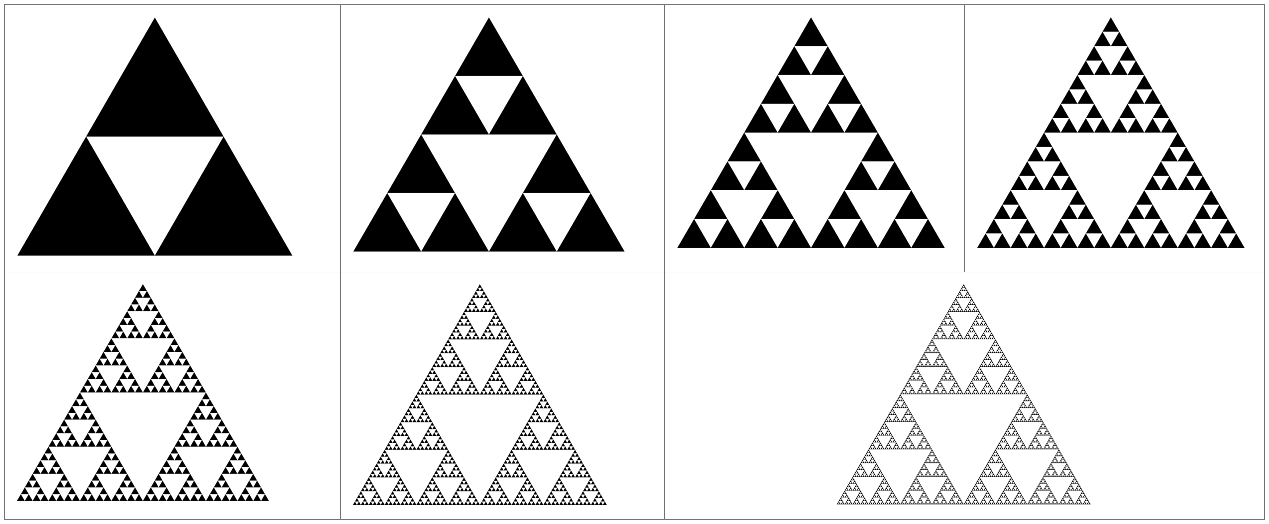


Table 1: The first 7 iterations of the Sierpinski Triangle. The fractal is the limit as the number of iterations go to infinity.

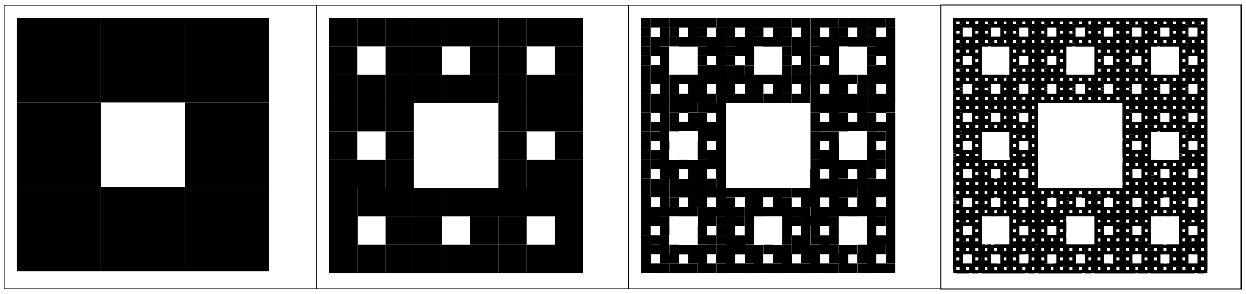


Table 2: The first 4 iterations of the Sierpinski Carpet. As above, the fractal is the limit as the number of iterations go to infinity.

(See the appendix for the full code for both the Sierpinski Triangle and Sierpinski Carpet).

Fractals from Time Series:

As indicated in the introduction, the specific type of fractal that will be analyzed in this paper is the heartbeat time series. A time series is a collection of data values taken at uniform time intervals, and can have a fractal appearance before a best-fit line is applied (but not all time series are fractal). Other than the heartbeat, long-time weather records and the charts tracking the change in stock market prices are also fractal in nature with regard to their time series **[11 – 12]**. The heartbeat time series can be measured using smart fabrics, which are themselves constructed using nanotechnology.

3.0 NANOTECHNOLOGY BACKGROUND

Nanotechnology is the field of building technological devices from single atoms and molecules. It is called “nanotechnology” because these devices are on the scale of nanometers (billionths of a meter) in size (the relative size of a nanometer to a meter is a little smaller than a dime compared to the size of the earth, based on calculating the quotient between their diameters; it can also be understood as “the amount a man’s beard grows in the time it takes him to lift a razor to his face”) **[13—17]**. The major benefit of such technology is that they take advantage of quantum mechanical effects and behavior in a way that macroscopic devices cannot. Nanotechnology also refers more broadly to the study of manipulating matter on an atomic and molecular scale, and is multidisciplinary, intersecting frequently with chemistry, molecular biology, materials science, computer science, and condensed matter physics (and any others that are concerned with either studying or manipulating anything that is nanoscopic) **[13] [18]**. Nanotechnology can cause molecular changes at precise locations in a controlled manner, and grants devices on this scale the ability to transform matter, energy and information at very small levels **[15]**.

There are numerous challenges to navigate before nanotechnology reaches its full potential. The biggest are (the physics version of) Brownian motion, quantum effects (and the need for some mechanism to provide corrections to compensate for those effects) at atomic scales, lack of sufficiently small batteries (that would need to be even smaller than the microbatteries that are in the early development stage **[19]**), the viscosity of fluids at small scales (analogous to molasses), and the difficulty in constructing and manipulating (and programming) something so small to begin with **[13] [15 – 16]**. “Brownian motion” is the name for the phenomenon where microscopic particles in a liquid or gas are constantly in motion, undergoing small, random fluctuations, due to being bombarded by the fast-moving atoms or molecules of the respective liquid or gas **[15] [20 – 21]**. This makes nanobots much more difficult to work with than macroscopic objects, since they can't be completely still or move in ways as easily predictable as a macroscopic object. Brownian motion, along with additional quantum effects and corrections, would need to be compensated for at the scale that nanotechnology operates in. There is another obstacle caused by the lack of nanoscale batteries for those nano-devices that need a current, but some researchers are having success finding a potential solution, by creating a battery out of a bacteriophage **[22]**. Other issues continually being researched are the potential toxicity levels of the materials used to create nano-devices, what the impact of nanotechnology on the environment could be (along with social and economic impact), and (in the most extreme, speculative case) what the risk level of runaway replication might be **[23]**. This latter risk has a special name called “gray goo”, for what everything could turn into if it were to happen, and some scientists have the idea to create special “blue goo” nanobots to combat the “gray goo”, just in case (physicist Michio Kaku points out that this would be akin to our immune systems’ antibodies and white bloods cells to fight off viruses and harmful bacteria, which are like an organic version of “gray goo”) **[13—15] [24]**. It should be noted that nanotech experts have pointed out that this risk is based on outdated assumptions from a time when nanotechnology was purely speculative, and that the necessary complexity for runaway replication couldn’t be achieved by the nanobots on accident; however, they do point out that the possibility of nano-weaponry being developed is a valid concern, so it is still very wise to develop safeguards for any such worst-case scenarios **[25]**. It’s a good thing that the worst anxieties about spontaneous runaway replication are unfounded, too, because self-assembling quantum dots (nanocrystals of semiconductor material) already exist **[26]**.

A Few Potential Applications:

Nanotechnology would have a remarkable impact on construction, allowing for new construction resources that are both lighter and stronger than conventional materials. This is similar to the principle already used for composite metals, to build buildings and bridges (and other structures) with minimized material and cost while maximizing strength (together the combined materials form a structure that’s stronger than the sum of its individual parts) **[27]**. Making those composites themselves out of nanomaterial-composites would entail a fractal-like structure, and nanotechnology would allow this to be done all the way down to the microscopic scale and give the overall structure greater strength (with material that is still lightweight) **[28—30]**. The result would be the possibility of stronger, taller buildings and bigger bridges **[28]**. Repairs could be initiated immediately upon detection of damage or deterioration, and both construction and repair could be automated **[31]**. Self-repairing roads could be the roads of the future, assuming *Back to the Future's* Doc Brown is wrong and we do indeed still need roads.

Nanotechnology could have the most significant impact in medicine. The introduction of smart drugs would involve nanotechnology. The microscopic machines would deliver a precise amount of medicine to the exact location in the body where it's needed, tackling any ailment directly with minimal or no side effect **[32 – 33]**. Less of the medicine would be needed as well, since it would no longer have to spread through the whole body to get where it is needed. This could lead to a much better way to combat cancer. Instead of using chemotherapy that kills healthy cells as well as cancerous cells, making the patient sick in the process, smart drugs could be programmed to attack the cancerous cells (and only the cancerous cells) directly **[33 – 34]**. Smart drugs alone would be a real breakthrough in medicine. Health-monitoring tattoos would be another. Such tattoos would use a special nano-ink that can monitor the composition of the blood. One application being considered is using them to monitor the glucose levels in people with diabetes. When detected, the tattoos would give off a glow that can only be detected with an ultraviolet light. The individual would also have a watch that could give off such a light, and include a reader that would convert the glow to the number that informs them of the amount of glucose in their blood. For many, this could very well be preferable to having to pierce their skin in order to do such checks. The tattoos would even dissolve after a few months (roughly 6), so they wouldn't get stuck with a design that they were getting more tired of with time **[35]**.

Current Uses:

There are some devices that nanotechnology already plays a role in. For instance, the smart fabrics already on the market utilize nanotechnology (a list of additional items already on the market can be found in the appendix) **[36]**. The next section will discuss smart fabrics in fuller detail.

4.0 SMART FABRICS

Smart Fabrics are a type of wearable tech that can be worn in the same way as ordinary clothes (and even be made to look like ordinary clothes), and there are several different types that are either already on the market or being developed in the laboratory: basic vitals-monitoring smart fabrics, flexible bulletproof fabrics, electricity-generating fabrics, flexible displays, instant-dry clothes and water (or other material) resistant fabrics, powered exoskeletons, metamaterial fabrics that can utilize transformation optics (this would allow for the creation of invisibility cloaks), and programmable matter materials (the latter is a hypothetical extension of nanotechnology's capability) **[15] [17] [23] [36—49]**. The focus of this section will be on the vitals-monitoring smart fabrics.

Perhaps the most immediate application for smart fabrics that monitor vitals is that they could alert emergency crews automatically if someone got into a crash or a big fall, or some other means of injury that prevented them from being able to call for help themselves. The clothing could detect that its wearer is bleeding by its ability to pick up on the presence of the protein albumin, which is common in blood, and then transmit an emergency signal with their GPS location **[46] [50]**. Many of the deaths that result from car crashes are due to emergency personnel not being aware of the victims in time to have made a difference, and this function of smart fabrics would be able to reduce the number of such fatalities substantially.

The first form of smart fabrics were smart textiles, also known as electronic textiles or e-textiles, which contained either metallic or optical wires weaved together with the textile fibers, and would be uncomfortable to wear. They also wouldn't be able to endure the same degree of bending and stretching that ordinary clothes can, because of how fragile they are. And metal fibers corrode, making such textiles problematic to wash **[46]**. The main way to make smart fabrics that aren't metallic and are thus more comfortable, durable, and easily cleaned, is to use nanotechnology. Such smart fabrics are known as organic electronics textiles **[51 – 52]**.

The specific form of nanotechnology needed to make organic electronic materials is a solution of carbon nanotubes. Cotton yarn is first dipped into this solution, and then dipped into a sticky polymer in ethanol. Nicholas Kotov and Bongsup Shim, engineers from the University of Michigan, discovered that the yarn becomes conductive after repeating these two steps several times, then letting it dry **[46]**. The yarn remained pliable and soft, though it turned black due to the carbon. If organic electronics textiles are desired in other colors, this may be achieved by combining ordinary clothing fibers with the conductive yarn, and the overall garment will still be one that is conductive. An advance beyond this, would be smart fabrics that can change color, in response to their surroundings or wearer's preference **[53]**.

When the antibody anti-albumin is added to the carbon nanotube solution, the yarn becomes even more conductive. The anti-albumin is how the smart fabrics are able to pick up the presence of albumin, which reacts with albumin, and thus detect whether the wearer is bleeding. So this gives the material its emergency aid capabilities as well as its extra conductivity **[46] [50]**.

Smart fabrics can also be used to monitor heartbeat and body temperature. The benefit to monitoring body temperature is that the innermost layer of the fabric could change its phase depending on whether you're getting too hot or cold. Fully solid if you're getting cold, and turning to liquid that will cool you down when you're getting too warm **[38] [54—56]**.

Smart fabrics are able to detect the heartbeat in a similar manner to heart rate monitors, which are already on the market. They work by picking up the electrical signal that's transmitted through the heart each time it contracts, which can be detected through the skin **[56]**. Heart rate monitors are essentially a very basic form of a smart fabric that cover only the part of the body near the heart, rather than being a regular article of clothing that has the heartbeat monitoring functionality like the more advanced smart fabrics do.

5.0 APPLICATION TO THE HEALTH OF THE HEART

Monitoring the heartbeat is where fractals come in, since a normal heartbeat time series has a particular fractal pattern, one that is multifractal **[11] [57 – 58]**. A multifractal system is a fractal system in which a single exponent (the fractal dimension) is not enough to describe its structure, and a continuous spectrum of exponents (also called the singularity spectrum, which is the collection of fractal dimensions of different subsets of points in the function) is needed to do so, because different parts of the time series exhibit different scaling behavior rather than consistent scaling behavior throughout **[11] [57 – 58]**. A time series is said to be monofractal when it has consistent scaling behavior that can be accounted for by a single scaling exponent **[11]**. As explained below, the heartbeat time series can be treated as monofractal if whether or not it's healthy is the only desired information, and so it will be treated as such in this paper An entire paper written solely for the purposes of giving a more complete and precise definition of multifractals themselves can be found in my sources **[59]**. The simplest description of multifractals which that paper gave was that they were like several "highly interwoven fractals" **[59]**.

A healthy heartbeat has a multifractality that is broader than an unhealthy heart **[57 – 58]**. An unhealthy heart, whether it's unusually erratic or unusually calm, has a very narrow range in the scaling exponents of its time series. A healthy heart can be erratic in some moments and calm in others, but the key is its singularity spectrum goes across a broad range over time **[57 – 58] [60 – 61]**. The multifractality would be reasonable for a trained doctor to analyze, but trying to tell whether the multifractality is broad enough to indicate a healthy heart would be a difficult task for many consumers who aren't trained in such complex areas of math. Fortunately, there is a way to analyze the time series and determine whether the heart is healthy or not, without having to find all of the scaling exponents, and it is a method that results in a single number that tells the wearer if they need to have their doctor take a closer look at how unhealthy their heart is (if "unhealthy" is the verdict that the analysis determines). This simpler method is called (monofractal) *detrended fluctuation analysis* (DFA), and it results in a single number whether the analysis is done on a monofractal or a multifractal **[57]**. The result of the monofractal analysis is actually the first scaling exponent of a multifractal analysis, so the result for determining the heart's health (or lack thereof) won't change based on whether it was a monofractal or multifractal DFA (or other multifractal analysis) that was performed **[57]**. The monofractal DFA can determine *whether* the heart is healthy or unhealthy, and a multifractal analysis can determine *how* an unhealthy heart is unhealthy (whether due to disease, aging, or congestive heart failure). The math is simpler for a monofractal time series, so the main focus of this section will be the monofractal detrended fluctuation analysis, but a fully robust algorithm could also run a multifractal DFA in the background and send those results to the doctor(s) to give more information if an unhealthy heart is indicated.

A quick note about multifractal DFA: DFA initially was only a monofractal analysis, but a multifractal DFA has been proposed as an extension to the monofractal DFA, and to duplicate the results of other multifractal analyses while being simpler than those other methods. The answer to "whether" the heart is healthy would be the first scaling exponent of this test, and the paper proposing it found that the remainder of the results (for determining "how" an unhealthy heart was unhealthy) were equivalent to other multifractal analyses **[11]**.

Another note about DFA in general: its usefulness is not just in analyzing the health of the heart. It was first proposed and developed for analyzing DNA sequences **[62]**, and has gone on to be utilized in numerous other areas: a small sample includes studying human gait, long-time weather records, solid state physics, cloud structure, geology, and economics time series **[11]**. And it was found empirically to be a wonderfully useful method for gauging the health of the heart. Prior to this, it was known that irregular heartbeats had chaotic rhythms, through the work of Leon Glass, Alvin Shrier and Jacques Bélair (see **[63]** in the sources), who pioneered the modeling of cardiac arrhythmias and the fractal dynamics of the heartbeat more generally; with the DFA method, the deeper multifractal nature of the heartbeat time series had been discovered.

This paper is also focusing just on a first-order (or linear) DFA, because a recent paper has demonstrated that the linear DFA is all that is necessary to distinguish a healthy heart from all the ways it can be unhealthy (and because the log-log plot, described below, that results from heartbeat time series is approximately linear; the linearity is greater for a healthy heart **[64 – 65]**).

To execute the DFA, first $N + 1$ heartbeats are recorded along with the times that each beat occurs (t_i). Then the *N* interbeat intervals are determined between each beat, by subtracting the previous beat's time from the current beat's time. For the *i*th beat, where *i* goes from 1 to *N*, that gives the following formula for the duration of the interbeat interval, *B(i)*:

$$B(i) = t_{i+1} - t_i \tag*{\{2\}}$$

After recording the data for all beats over a select period of time and calculating the *N* interbeat intervals, a discrete integration is taken of the interbeat intervals data. The discrete integration, *y(k)* (where *k* is the upper limit of the discrete integration), is taken by summing the difference between each interbeat interval and the average interbeat interval, B_{ave}, up through index *k* (the list of these differences is represented by *w*):

$$B_{ave} = \frac{1}{N}\sum_{i=1}^{N} B(i) = \frac{t_{N+1} - t_1}{N} \tag*{\{3\}}$$

$$y(k) = \sum_{i=1}^{k} [B(i) - B_{ave}] = \sum_{i=1}^{k} w(i) \tag*{\{4\}}$$

The full time series is then broken up into partitions (or boxes) of length *n* (length is from 0, even though the index of the interbeat intervals starts at 1), resulting in *N/n* separate partitions, and then a least squares fit is applied within each partition (for simplicity, each *n* chosen is a factor of *N* in the program I describe below, so the best numbers of beats to record are those that are a single value greater than numbers with several factors). The least squares fit is a line that best matches the trend of the data within the *n*th particular partition in which it is applied. This line is called the "local trend". Then at each upper limit of the discrete integration, *k*, the y-coordinate of the local trend, $y_n(k)$, is subtracted from the y-coordinate from the discrete integration of the actual time series, *y(k)* (this is the "detrending" part of DFA). This difference is squared, then the squared differences for all values of *k* are summed up, and this sum is divided by the total number of interbeat intervals. The square root of the result is the detrended fluctuation, *F(n)*, of the heart's time series, for a time series divided into partitions of length *n*:

$$F(n) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}[y(k) - y_n(k)]^2} \qquad \textbf{\{5\}}$$

For the complete analysis, it is necessary to compute this same calculation for multiple different box lengths, *n* **[62] [65 – 66]**. These partitions must have a length greater than 1. If the length is less than 1, then each partition would contain only at most one data point (some intervals could have no data points and thus would be completely useless), and there could be an infinite number of possible linear fits for a single point, so the analysis could not be done. If *n* = 1, then after the first partition (which would only have one point), the two endpoints would be the only points in each partition **(**an endpoint is part of both partitions that it's an endpoint of, except for the absolute endpoints of the time series as a whole), which the linear fit would match exactly, so that there would be no difference from the actual y-coordinate from the discrete integration of the time series and the y-coordinate of the local trend (meaning *F(n)*=0, so the logarithm could not be taken).

The calculation must be run on enough different box lengths to give enough data points for a $\log F(n)$ vs. $\log n$ plot, because the way to determine the heart's health is to take the logarithm of both *F(n)* and *n*, then plot them together. A resulting slope of 1 or very close to 1, over the long term, is evidence of a healthy heart, whereas any deviation from that slope is an indication of some problem **[57]**. The more sizes of *n* that the calculation is run on, the more data is available for both an accurate and precise long term analysis of the heart's health.

It can be shown that the DFA is related to the box dimension by solving for *F(n)* in terms of *n*. The $\log F(n)$ vs. $\log n$ plot is the graph of the equation $\log F(n) = m * \log n + b$, where *m* is the slope and *b* is the y-intercept. Raise 10 to the power of both sides of the equation to solve for *F(n)*. Due to the rules of logarithms, this results in the equation $F(n) = 10^b * n^m$ (or $F(n) = e^b * n^m$ if using natural logarithms, as Mathematica does by default). The slope m in this equation is analogous to the box dimension (equation **{1}**), and the process for solving for m is similar to calculating the box dimension:

$$m = \frac{\log F(n) - b}{\log n} \qquad \{6\}$$

The general usefulness of the DFA process is that it is one method of revealing the fractal nature of a set that looks like it is just chaos or noise, due to it having a form of self-similarity called “self-affinity”. A fractal is self-affine when its self-similarity is not apparent, because its “pieces” are scaled by different amounts along each axis. Such fractals need to be rescaled to observe their self-similarity, and the DFA process is one way of doing so **[67]**. The specific form of noise that a healthy heartbeat time series resembles is $1/f$ noise (inverse-frequency noise, also known as “pink noise”), and consistently does so for short term and long term analyses, whereas unhealthy heartbeat time series resemble either white noise or brown noise (and lack consistency when the time frame changes) **[58] [65 – 66] [68]**. Below is an image that includes a time series of these different types of noise:

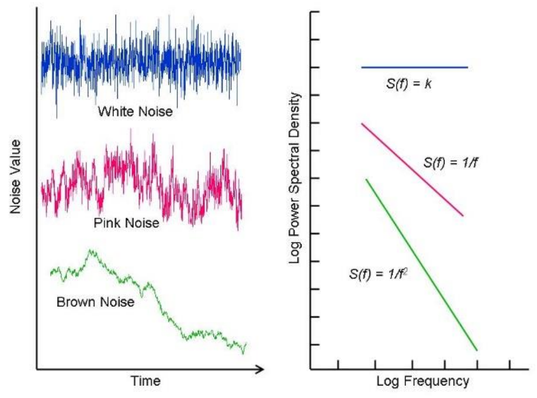


Figure 3: Time series and spectral density plots for white noise, pink noise, and brown noise.

I created a program in Mathematica to do this analysis for 1001 heartbeats, or 1000 interbeat intervals. It took several iterations to create a program that was fully automated and could run through the full DFA for any number of beats by just making a single change to the code (the number of interbeat intervals). First, I worked out the algorithm by hand on a small number of beats, then proceeded to replicate that process via computer. I experimented on non-automated code of a small number of beats (different programs of 7, 11, and 13 beats), slowly automating different chunks of the code until I figured out how to automate everything. The following subsections detail how I progressed from the by-hand analysis on a small number of beats, to the full program that automatically does the analysis on the full number of 1001 beats.

Toy Example By-Hand Analysis:

For the by-hand analysis, I first wrote down a list of plausible times for 7 heartbeats. Then I created another list of the interbeat intervals, by calculating the differences between those times. I calculated the average of the interbeat intervals, then made a list, *w*, of the differences of that average between each interbeat interval. For the discrete integration list, *y*, I calculated the partial sums of *w* from row 1 to row *k* for each k^{th} row. The following table shows these calculations (the average of the interbeat intervals was $\frac{47}{60}$):

i	Times (*t*)	Interbeat Interval (*B(i)*) **{2}**	Difference from average (*w*), **{2} – {3}**	*k*	Discrete Integration (*y(k)*), **{4}**
0	0.5	N/A	N/A	N/A	N/A
1	1.3	0.8	$\frac{1}{60}$	1	$\frac{1}{60}$
2	1.9	0.6	$-\frac{11}{60}$	2	$-\frac{1}{6}$
3	3.1	1.2	$\frac{5}{12}$	3	$\frac{1}{4}$
4	3.8	0.7	$-\frac{1}{12}$	4	$\frac{1}{6}$
5	4.7	0.9	$\frac{7}{60}$	5	$\frac{17}{60}$
6	5.2	0.5	$-\frac{17}{60}$	6	0

Table 3: The times of the ith *beat for the Toy Example, and the calculations for the interbeat intervals and the discrete integration (from* i *to* k*).*

After all these calculations, I graphed the discrete integration for each index, k (the x-axis being the index values). Then I divided that graph into boxes of different lengths (along the x-axis), 2, 3 and 6. The first box has one less data point in it than all other boxes, because we started the indexing at 1 rather than 0 (this is the default for Mathematica). On the by-hand version, I approximated the line fits in each box by eyeballing them, and used those approximate values to calculate the DFA. Here, though, I'm going to do a more accurate calculation of the linear fit of the local trend, utilizing the following calculations:

$$y_n(k) = a + b * k \qquad \textbf{\{7\}}$$

where:

$$a = \frac{(\sum_{k=u}^{v} y(k))(\sum_{k=u}^{v} k^2) - (\sum_{k=u}^{v} k)(\sum_{k=u}^{v} k * y(k))}{N_n(\sum_{k=u}^{v} k^2) - (\sum_{k=u}^{v} k)^2} \qquad \textbf{\{8\}}$$

$$b = \frac{N_n(\sum_{k=u}^{v} k * y(k)) - (\sum_{k=u}^{v} k)(\sum_{k=u}^{v} y(k))}{N_n(\sum_{k=u}^{v} k^2) - (\sum_{k=u}^{v} k)^2} \qquad \textbf{\{9\}}$$

and:

N_n represents the number of points in each partition, u is the lower multiple of n (left endpoint) in each partition after the first, and v is the higher multiple of n (right endpoint) in each partition after the first (in the first partition, $u = 1$, $v = n$, and $N_n = n$; $N_n = n + 1$ for all other partitions; $v = N$ in the final partition) **[69]**. For the by-hand analysis, the local trend values at the endpoints of each partition were the mean of the local trends determined from each neighboring partitions' respective local trend equation (see the Programming Notes subsection for more). Keeping track of these values (for $n = 2$) requires another table:

k	$y(k)$ **{4}**	$k*y(k)$	k^2	a **{8}**	b **{9}**	Local trend equation $y_2(k)$ **{7}**
1	$\frac{1}{60}$	$\frac{1}{60}$	1	$\frac{1}{5}$	$-\frac{11}{60}$	$\frac{1}{5}-\frac{11}{60}k$ or 0.2 – 0.18333k
2	$-\frac{1}{6}$	$-\frac{1}{3}$	4			
Sum = 3	$-\frac{3}{20}$	$-\frac{19}{60}$	5			
2	$-\frac{1}{6}$	$-\frac{1}{3}$	4	$-\frac{5}{12}$	$\frac{1}{6}$	$-\frac{5}{12}+\frac{1}{6}k$ or -0.41667 + 0.16667k
3	$\frac{1}{4}$	$\frac{3}{4}$	9			
4	$\frac{1}{6}$	$\frac{2}{3}$	16			
Sum = 9	$\frac{1}{4}$	$1\frac{1}{12}$	29			
4	$\frac{1}{6}$	$\frac{2}{3}$	16	$\frac{17}{30}$	$-\frac{1}{12}$	$\frac{17}{30}-\frac{1}{12}k$ or 0.56667 – 0.08333k
5	$\frac{17}{60}$	$1\frac{5}{12}$	25			
6	0	0	36			
Sum = 15	$\frac{9}{20}$	$2\frac{1}{12}$	77			

Table 4: Calculating the local trend equations for the Toy Example, based on the sums of k, y(k), k*y(k), *and* k^2.

The DFA calculations (for $n = 2$) are shown in the table below:

k	Interval of i	Local trend equation ($y_2(k)$ for general k) **{7}**	Local trend coordinate ($y_2(k)$ for particular k) **{7}** (with values of k plugged in)	Mean of coordinate values ($y_2(k)$ for endpoint values)	$y(k)$ **{4}**	Difference between discrete integration and local trend (innermost calculation of **{5}**, d)	d^2 (term contained within the summation under the root in **{5}**, s)
1	(0,2]	$0.2 - 0.18333k$	$\frac{1}{60}$	$\frac{1}{60}$	$\frac{1}{60}$	0	0
2^-	(0,2] (Endpoint)		$-\frac{1}{6}$	$-\frac{1}{8}$	$-\frac{1}{6}$	$-\frac{1}{24}$	$\frac{1}{576}$
2^+	[2,4] (Endpoint)	$-0.41667 + 0.16667k$	$-\frac{1}{12}$				
3	[2,4]		$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{1}{36}$
4^-	[2,4] (Endpoint)		$\frac{1}{4}$	$\frac{29}{120}$	$\frac{1}{6}$	$-\frac{3}{40}$	$\frac{9}{1600}$
4^+	[4,6] (Endpoint)	$0.56667 - 0.08333k$	$\frac{7}{30}$				
5	[4,6]		$\frac{3}{20}$	$\frac{3}{20}$	$\frac{17}{60}$	$\frac{2}{15}$	$\frac{4}{225}$
6	[4,6] (Endpoint)		$\frac{1}{15}$	$\frac{1}{15}$	0	$-\frac{1}{15}$	$\frac{1}{225}$

Table 5: Calculating the detrended fluctuation for the Toy Example. At endpoints, where the values were different based on which local trend equation was used, I chose the mean of the resulting values to represent $y_2(k)$*. (See the Programming Notes for more about how the endpoints were handled in the automated versions of the program). The final column is summed up, divided by* N*, and the square root of the resulting value is F(2).*

The sum of the squared differences is $\frac{413}{7200}$. From these calculations, the detrended fluctuation for $n = 2$, $F(2)$, is $\frac{\sqrt{1239}}{360} \approx 0.0977762$. The remaining detrended fluctuations for the full analysis were calculated when I converted the algorithm to the Mathematica implementation. By using either Mathematica or a similar process to the above, $F(3) \approx 0.114891$, and $F(6) \approx 0.146168$. The remaining steps for the $\log F(n)$ vs. $\log n$ plot are explained in the Mathematica Implementation and Visualization subsection.

Toy Example Mathematica Implementation and Visualization:

I started the Mathematica implementation by using the List function to create the same list of times that I used for the by-hand version. The Differences function took care of calculating the list of interbeat intervals, then I used Mean to find the average of this list. Subtracting this mean from the list of interbeat intervals gave the list *w*, then using the Accumulate function on *w* generated the list of partial sums of *w* (which is the discrete integration of the interbeat intervals list), represented by *y*. The full code for all these steps can be seen in the appendix. Below are the graphs that result from these steps:

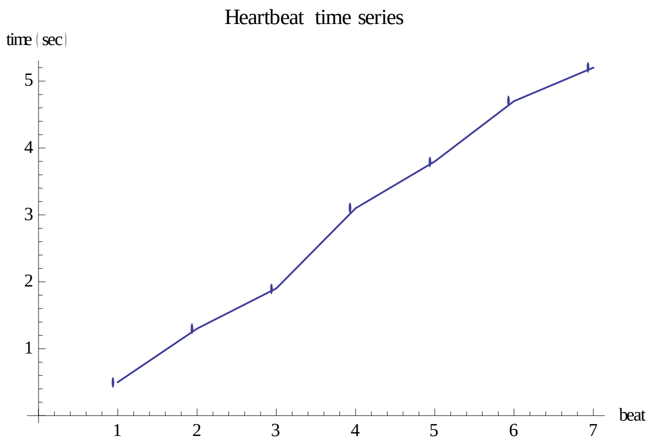


Figure 4: The heartbeat time series of the Toy Example, time *(in seconds) vs.* ith *beat.*

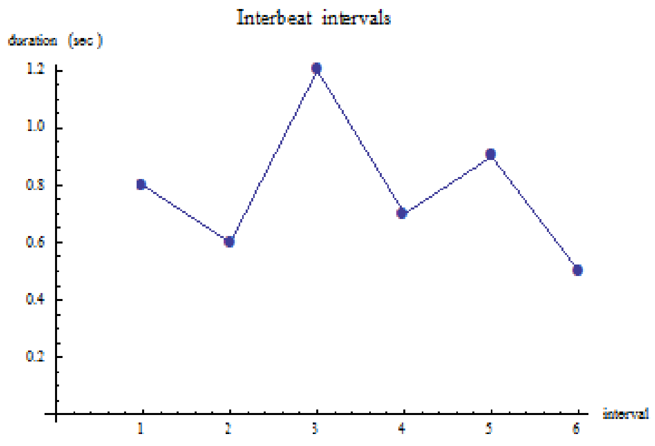


Figure 5: The interbeat intervals of the Toy Example, duration *(in seconds) vs.* ith *interval.*

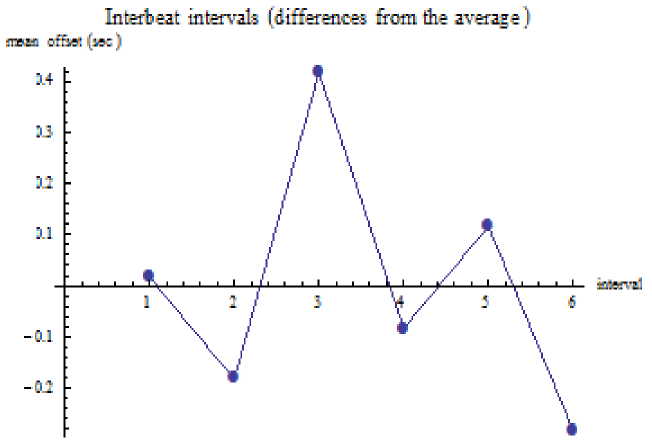


Figure 6: Interbeat intervals with the average subtracted.

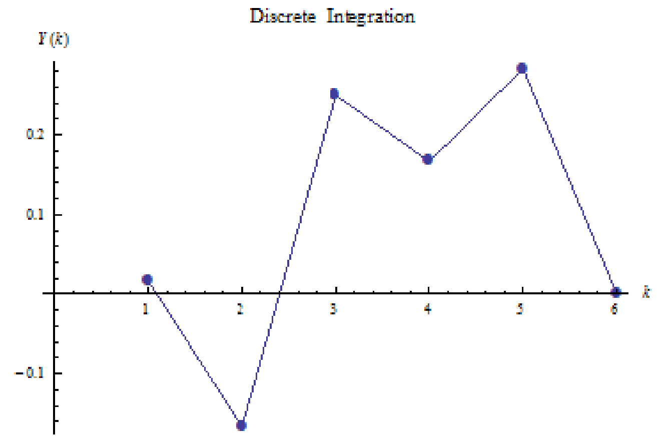


Figure 7: The discrete integration of the Toy Example, y(k) *vs.* k.

I defined several sub-lists of *y*, designated by *z*, in order to find the partial line fits (using the Fit function on these sub-lists) for the different partitions that the entire set was broken into. The following graph results for $n = 2$:

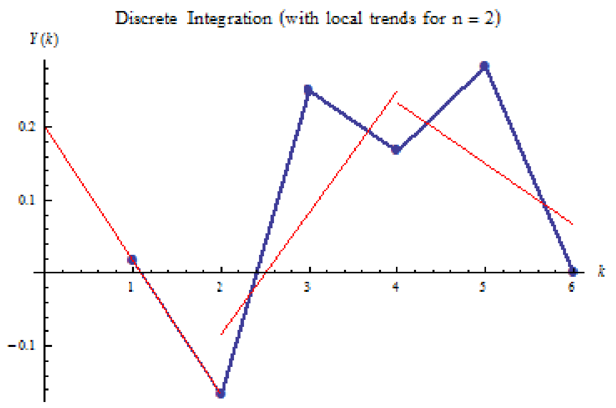


Figure 8: Discrete integration with local trends for n = 2, y(k) *vs.* k.

The following is the graph for $n = 3$:

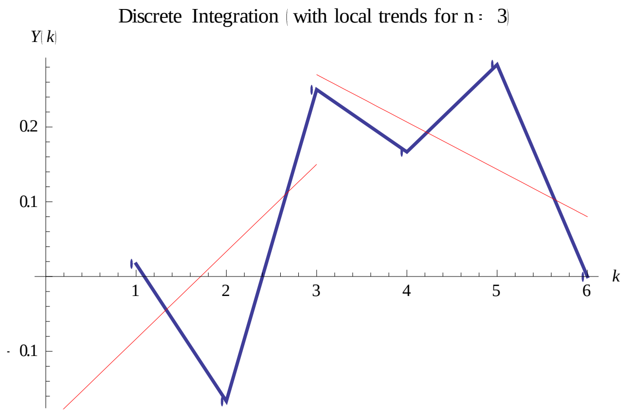


Figure 9: Discrete integration with local trends for n = 3, y(k) *vs.* k.

And the following is the graph for $n = 6$:

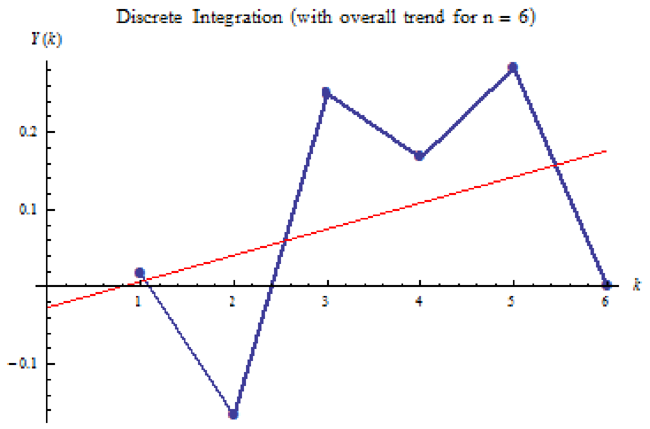


Figure 10: Discrete integration with local trends for n = 6, y(k) *vs.* k.

Next, I calculated a list (*d*) of the differences between the y-coordinates of the time series of the local trends, and a list (*s*) of the squares of these differences. To calculate the detrended fluctuation (*F*), I totaled list *s*, divided the result by *N*, then found the square root. Then I found the logarithms for *F* and *n*, and placed them in a nested list ({*P*, *p*}, where $P = \log n$ and $p = \log F(n)$) that was used to generate the $\log F(n)$ vs. $\log n$ plot located below:

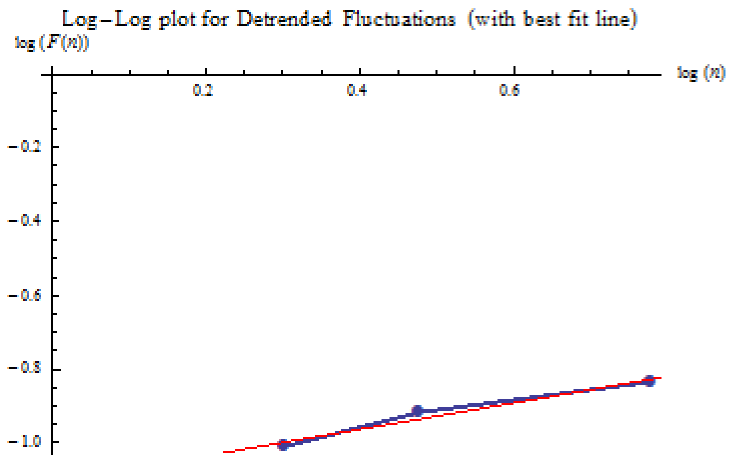


Figure 11: Log-log plot for the detrended fluctuations (with best fit line), log F(n) *vs.* log n.

The slope of the best fit line is 0.35911, which would indicate a very unhealthy heart if it were a real heartbeat time series.

Randomizing and Automating the Toy Example:

I used a Gamma Distribution to randomize the list of interbeat intervals. This is necessary to simulate a large number of beats, because inputting the data manually would take too long otherwise. And they were only simulated for the remaining programs because I was not able to get the data for an actual heartbeat time series, as that is protected information. But even with randomization, the necessary lines of code for the remaining analysis would take too long without automating the process. So I used Do loops, For loops, and the Piecewise function to automate the program (as a reminder, the full code for this can be seen in the appendix).

Realistic Scale Example:

In the final iteration of the program (viewable in the appendix), I used a nested Do loop to automate everything after the determination of the discrete integration list, *y*. I started by defining *M* to be the number of interbeat intervals, since *N* can't be used as a variable in Mathematica (it's a function that returns the numerical value of something). It is possible to use a list to define the increment change within a Do loop, so I created the list of partition sizes, *n*, by finding the factors of *M* with the Divisors function and removing 1 from the resulting list using the Length function (taking one less element than the length of the list from that list). This nested loop brought the total lines of code down to 15 (whereas earlier iterations of the code would have required several thousand lines of code to conduct the analysis on 1000 interbeat intervals) and made it so that a single change in the first line (the value of *M*) was all that was necessary to run the analysis for any number of beats. The following are the graphs of the interbeat intervals and the discrete integration:

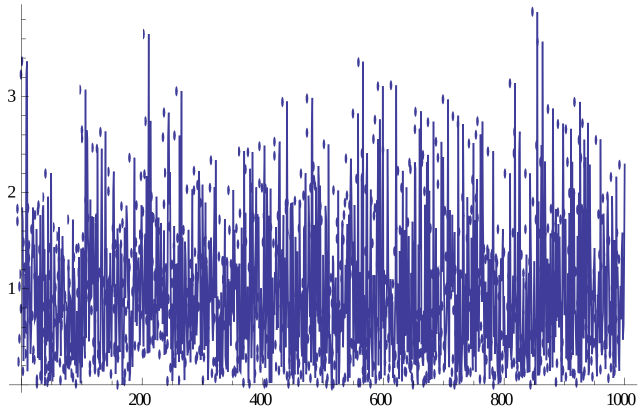


Figure 12: 1000 randomized interbeat intervals, duration *(in seconds) vs.* ith *beat.*

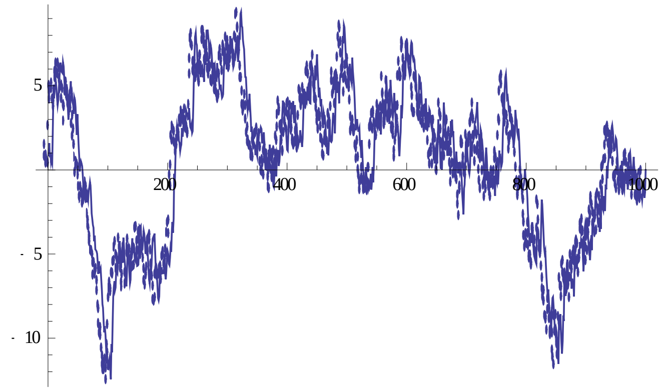
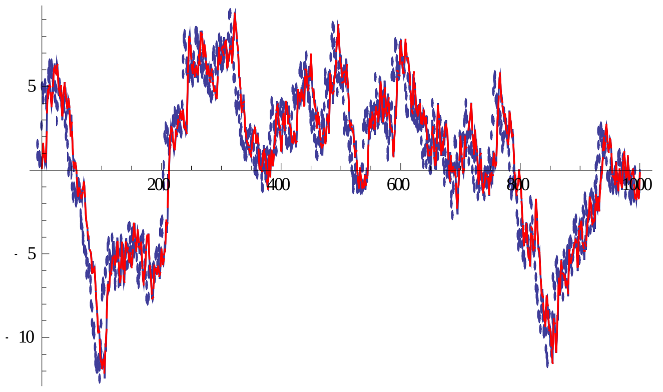
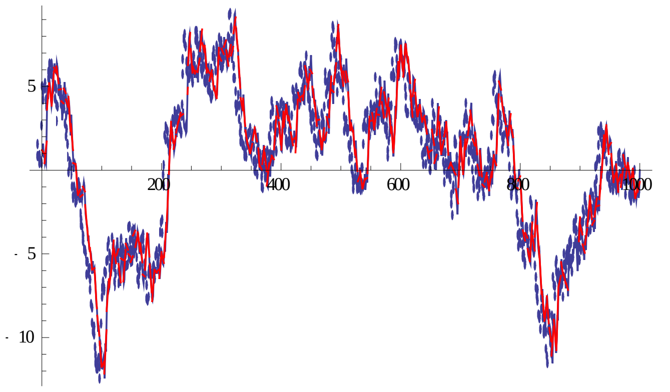


Figure 13: Discrete integration for 1000 intervals, y(k) *vs.* k.

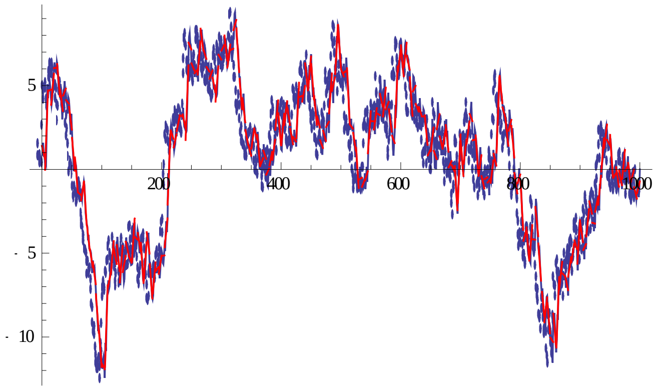
I made an empty list, *r*, into which I appended all the logarithm values ({*P,p*}) as the final step of the nested Do loop. Since *i* is already the variable that represents each iteration in the inner Do and For loops, *j* serves this purpose in the outer Do loop, and is set to the values of the partition sizes, *n*. Neither Show nor Print on their own would generate the graphs within the loop, but using them both made it possible, by defining a variable to be equal to the Show command, then using the Print function on that variable. After each graph, I printed a list that contained the values for the partition size of that iteration, the logarithm of that partition size, the detrended fluctuation, and the logarithm of that detrended fluctuation. The following graphs were the result:



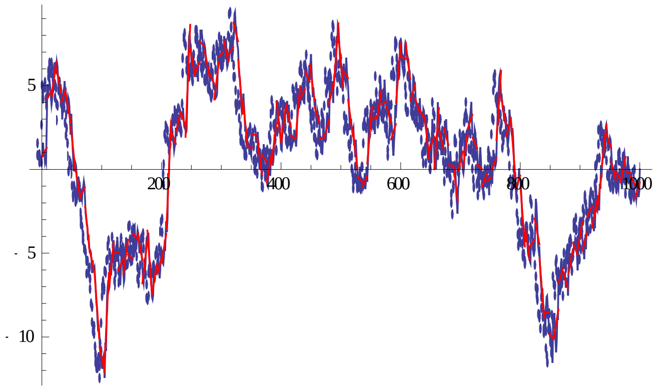
```
{{2, 0.30103},{0.278422,-0.555296}}
```



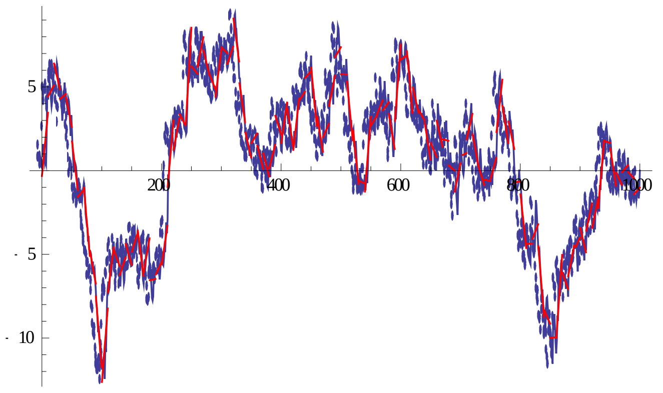
```
{{4, 0.60206},{0.402361,-0.395384}}
```



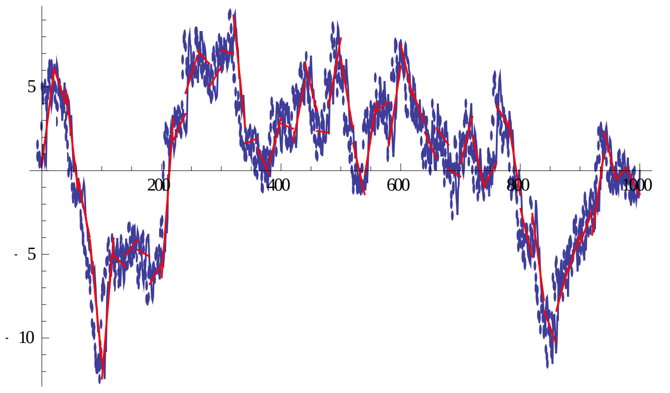
```
{{5, 0.69897},{0.438259,-0.358269}}
```



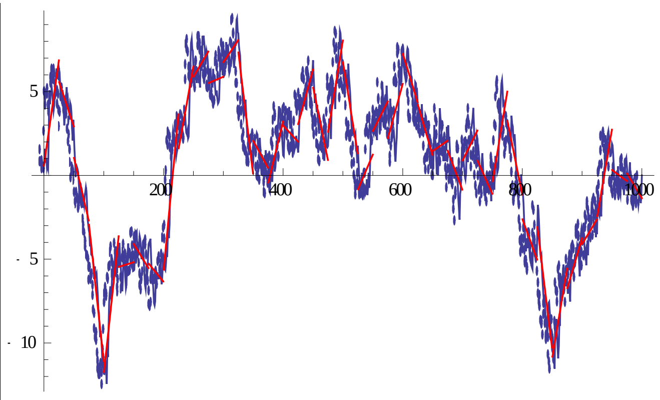
```
{{8, 0.90309},{0.553744,-0.256691}}
```



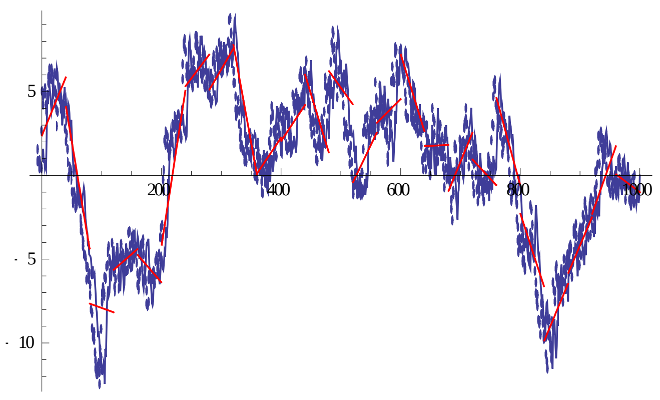
```
{{10, 1.},{0.628815,-0.201477}}
```



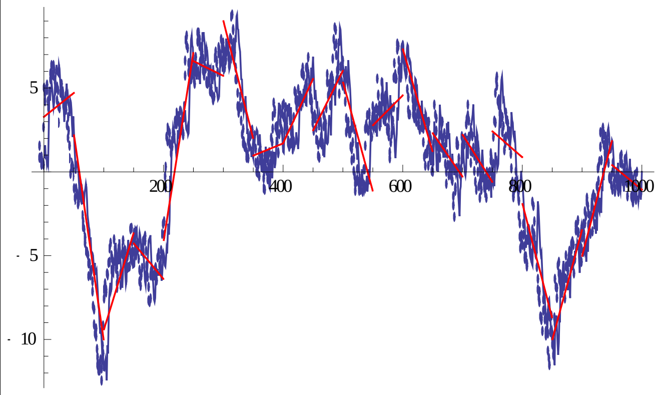
```
{{20, 1.30103},{0.821954,-0.0851527}}
```



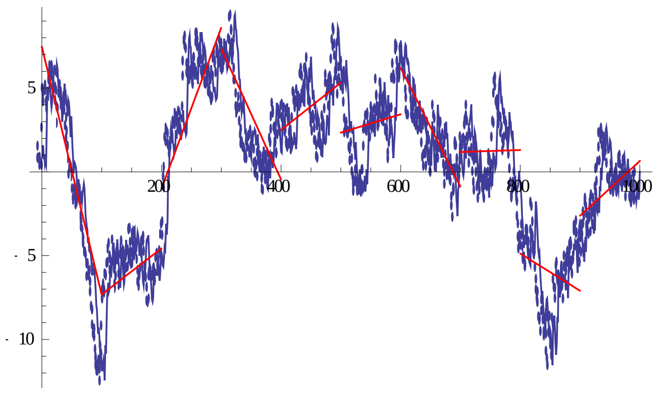
```
{{25, 1.39794},{0.892852,-0.0492207}}
```



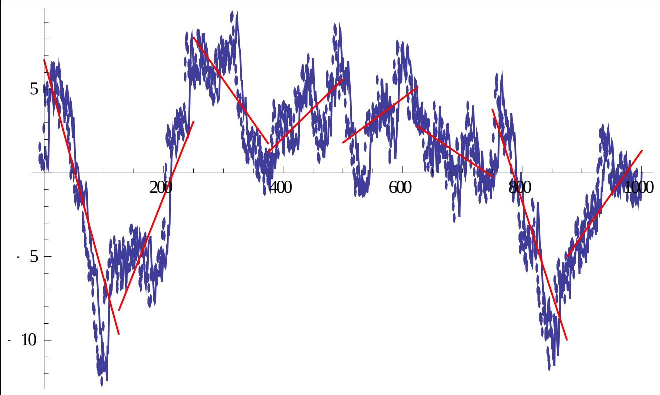
```
{{40, 1.60206},{1.20778,0.0819889}}
```



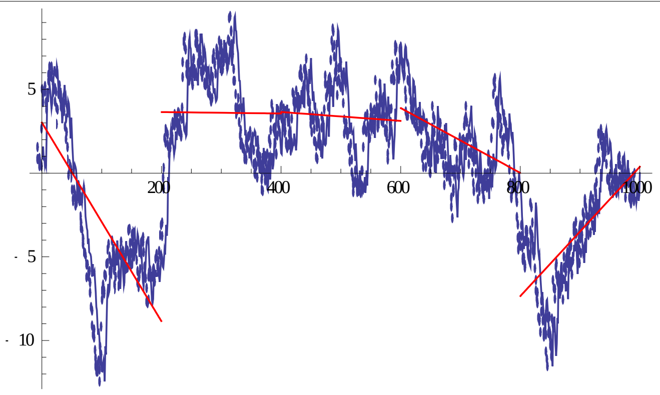
```
{{50, 1.69897},{1.27295,0.104812}}
```



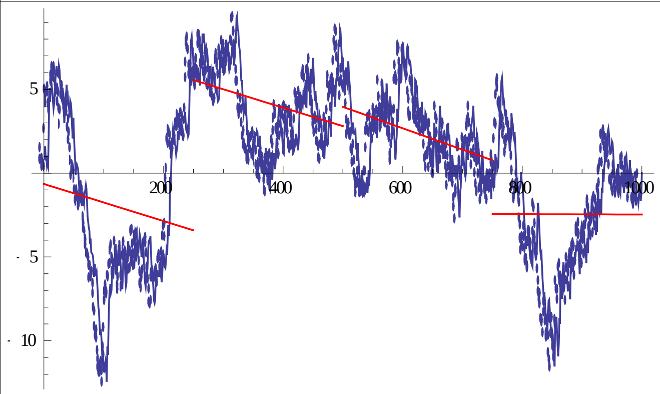
```
{{100, 2.},{1.89343,0.27725}}
```



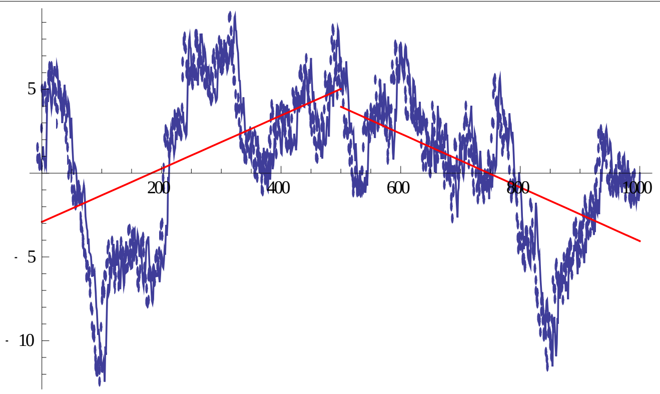
```
{{125, 2.09691},{2.014,0.304059}}
```



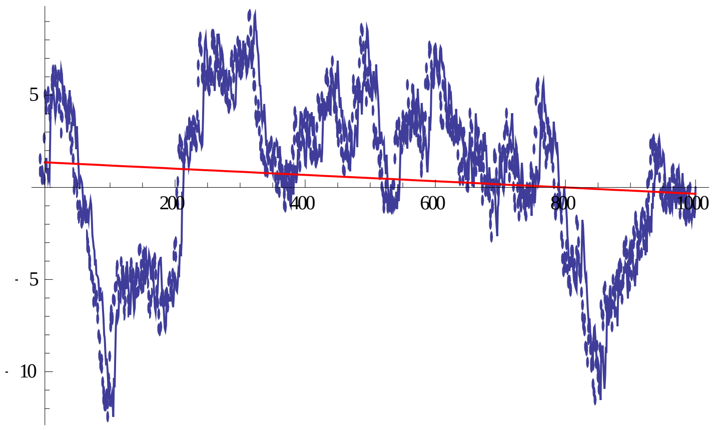
```
{{200, 2.30103},{2.61507,0.417484}}
```



```
{{250, 2.39794},{3.40491,0.532106}}
```



```
{{500, 2.69897},{3.81196,0.581149}}
```



```
{{1000, 3.},{4.45966,0.649302}}
```

Table 6: Discrete integration with local trends for 1000 intervals, where n *equals each factor of 1000 from 2 to 1000. The values in the brackets are* {{n, log n},{F(n), log F(n)}}.

And below is the log $F(n)$ vs. log n plot:

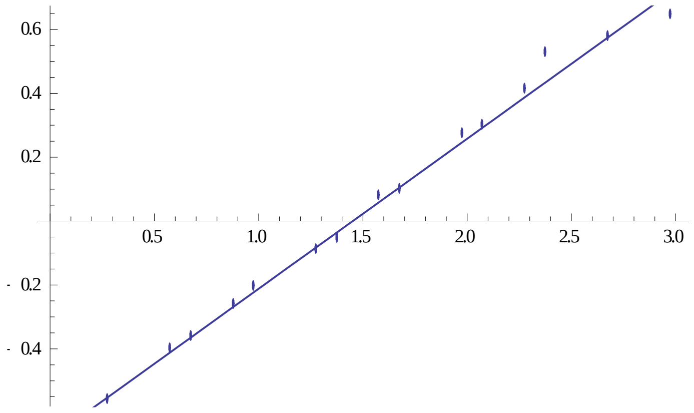


Figure 14: Log-log plot for 1000 intervals, log F(n) *vs.* log n.

The slope of the best fit line above is 0.469464, which would also indicate a very unhealthy heart, though it's closer to healthy than the Toy Example (if that's any consolation, considering how even this result is much too far from healthy).

Realistic Data Set:

Before producing the code for the Sierpinski Triangle and Carpet, there was code for a jagged line that I found online and analyzed to figure out the process of coding a fractal in Mathematica (see the Programming Notes section for more detail on this code). Below is a graph of this jagged line (representing the interbeat intervals in this scenario):

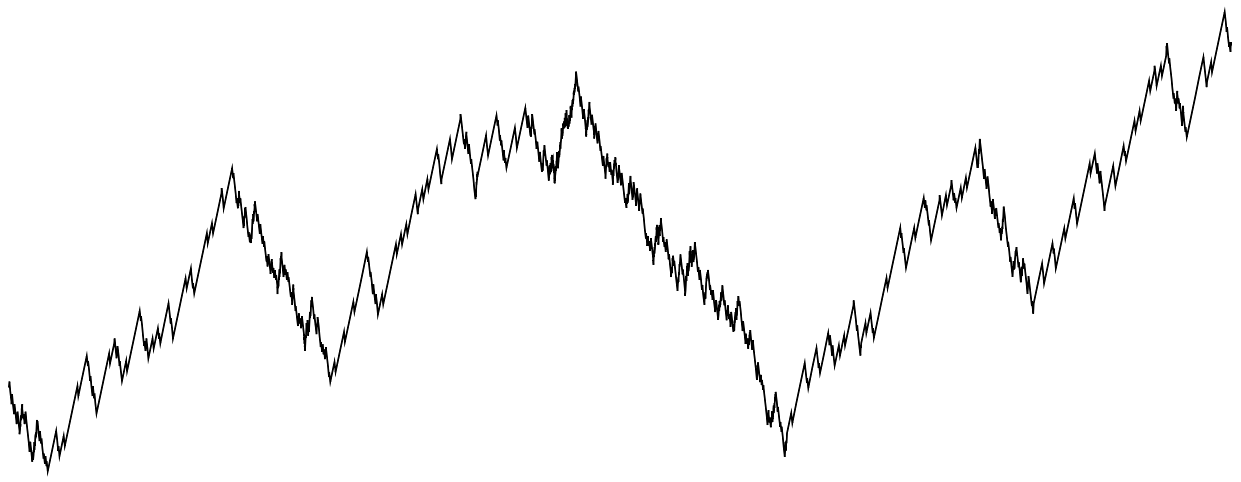


*Figure 15: Code for a "NonRandom Cartoon" that I found online **[70]**. I studied the code to get an idea of what would be involved in the process of trying to generate other iterated function systems in Mathematica.*

After I finished the DFA code from the preceding sections, I looked back at the output of this earlier code and noticed that it looked a lot closer to a graph of an actual heartbeat time series than the randomized data set did. Out of curiosity, I tried running the jagged line code through the algorithm (partly to see how close the result would be to a healthy heartbeat time series' expected result, and partly to test how well the DFA algorithm could work for any time series data), and the end result of running the algorithm on this data set was closer to the expected result for a healthy heartbeat than any instance of running the algorithm for a randomized time series (the exact value of the final slope was 0.920727).

Reverse Engineering the Algorithm:

In an effort to better understand why the DFA algorithm operates as it does, my advisor and I devised a way to reverse engineer the algorithm, starting with a discrete integration list that would give a slope of one from the analysis, and working backwards from there. We came up with a piecewise function for the discrete integration that could be manipulated and allow the final slope that results from the analysis to be controlled, based on modifying certain parameters. This was created recursively with the following algorithm: the 0th iteration is a horizontal line at 0 between the x-coordinates α and β; the 1st iteration replaces this line with a piecewise function that increases constantly from $(\alpha, 0)$ to $(\alpha + \frac{\beta-\alpha}{4}, 1)$, then decreases constantly from $(\alpha + \frac{\beta-\alpha}{4}, 1)$ to $(\alpha + \frac{\beta-\alpha}{2}, -2)$, increases constantly from $(\alpha + \frac{\beta-\alpha}{2}, -2)$ to $(\alpha + \frac{3(\beta-\alpha)}{4}, 1)$, and then finally decreases from $(\alpha + \frac{3(\beta-\alpha)}{4}, 1)$ to $(\beta, 0)$, making the slope of the 0th iteration its mean; for the *(h+1)th* iteration, each line segment in the *hth* iteration is replaced with itself plus a new multiple of the basic shape, where the *hth* iteration's slope (of the replaced line segment) serves as the mean in each partition. This process can be repeated indefinitely for greater precision, but we stopped after the 3rd iteration using just 3 adjustable parameters. The following code shows how this was created in Mathmatica (with both the general formula and a special case):

```
f[x_]:=Piecewise[{{4x, 0≤x<1/4}, {1-12 (x-1/4), 1/4≤x<1/2}, {-2+12(x-1/2), 1/2≤x<3/4}, {1-4(x-
3/4), 3/4≤x≤1}},0]
A1[x_]:=f[x/64]
A2[x_]:=f[Mod[x/16, 1]]
A3[x_]:=f[Mod[x/4, 1]]

General function = a1*A1[t]+a2*A2[t]+a3*A3[t]
Special case : a1 = 1, a2 = 1/4, a3 = 1/16
```

This results in the graphs located below, the initial version of the piecewise function and the special case of the composite function:

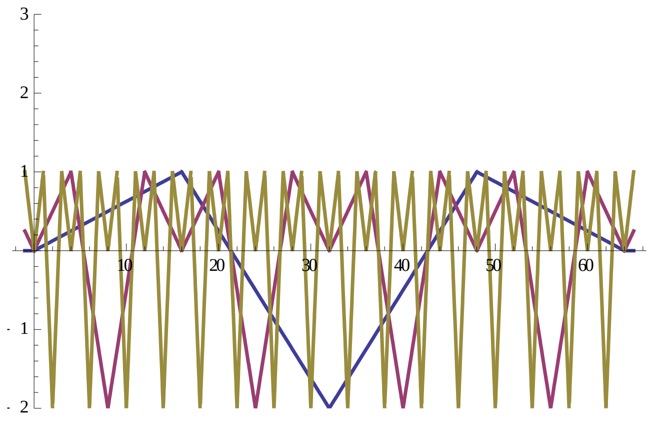


Figure 16: Each A(h) *subfunction plotted together, without being added together or multiplied by the* a(h) *parameters.* y(k) *vs.* k.

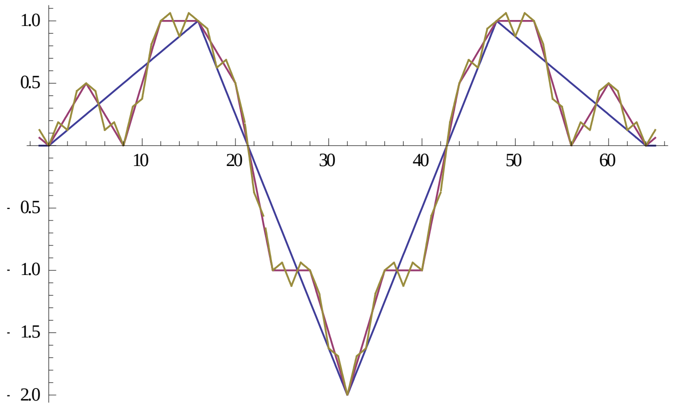


Figure 17: The first 3 iterations of this adjustable function representing a discrete integration. Each new iteration adds an A(h) *subfunction multiplied by a new* a(h) *parameter.* y(k) *vs.* k.

The graphs below are the slope I got for the DFA plot after manipulating these parameters, and what the discrete integration graph looks like when it's run through the usual DFA process:

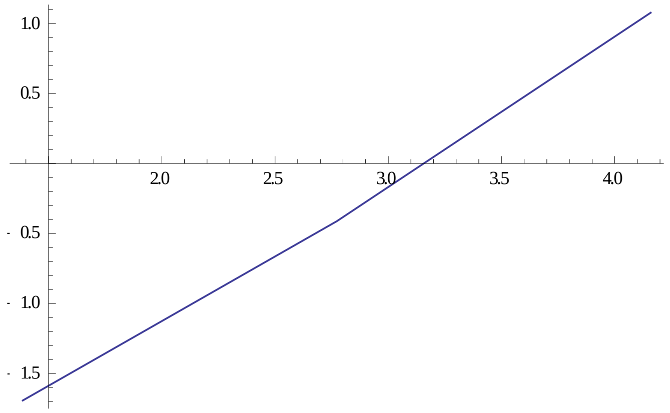


Figure 18: The line that results for the log-log plot after adjusting the parameters to force it to have a slope of 1. log F(n) *vs.* log n.

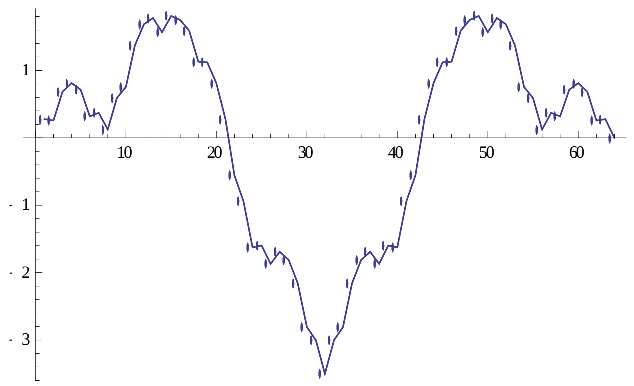


Figure 19: The discrete integration that results in the log-log plot from Figure 18, y(k) *vs.* k.

I adjusted the parameters until I found the necessary values that would result in a slope of one within a realistic time frame (64 heartbeats occurring in roughly a minute). The values I used for the parameters were $a1 = 1.75, a2 = 0.375$, and $a3 = 0.075$, which results in a precise slope of 1.0001. Then, using a loop, the discrete integration list was determined based on these parameters, and the *w* list was determined from that via the difference of consecutive discrete integration values (being the reverse of the accumulated sum). Because the average interbeat interval value drops out during the analysis, it could be chosen to be anything, so I chose the minimum necessary value to make all the resulting interbeat intervals positive (as negative durations would be nonsense). The resulting interbeat interval graph looked almost identical to the previous ones (that indicated a healthy heart) in papers on the DFA process (the graph that represents actual times is located to the right):

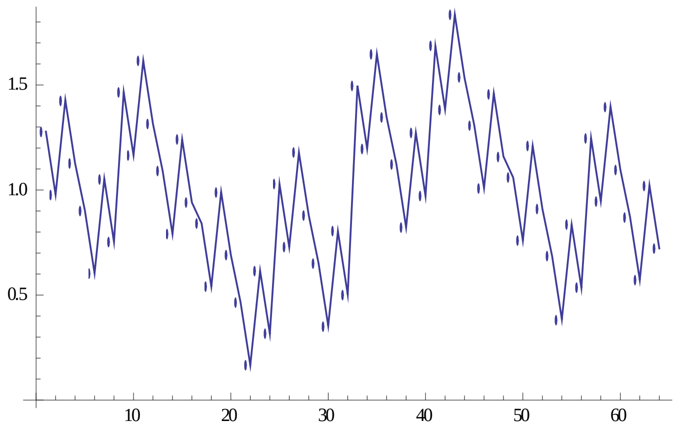


Figure 20: The interbeat intervals that result from doing the DFA calculations in reverse, duration *(in seconds) vs.* ith *beat. As expected, this has a resemblance to the Pink Noise image from Figure 3.*

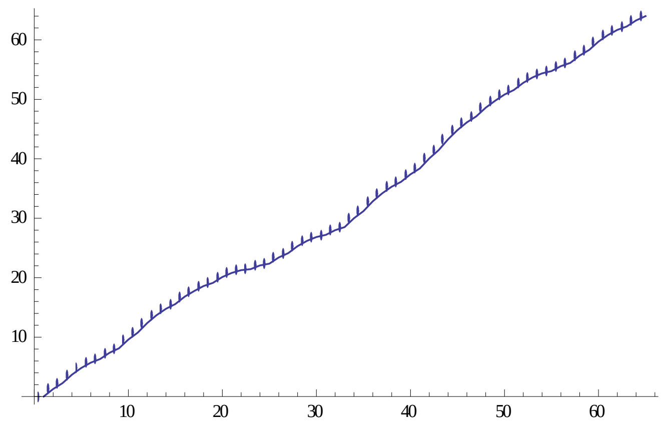


Figure 21: These are the times that result from the final step of the reverse DFA, time *(in seconds) vs.* ith *beat.*

To further illustrate how these parameters can be adjusted to achieve any positive slope, I used the parameters $a1 = 0.7, a2 = 0.4$, and $a3 = 0.25$ to get a slope less than 1 (approximately 0.314758). And I used the parameters $a1 = 3.25, a2 = 0.35$, and $a3 = 0.03$ to get a slope greater than 1 (approximately 1.54674). I also calculated these cases in reverse, and the graphs for the discrete integration, interbeat intervals, and times can be found below:

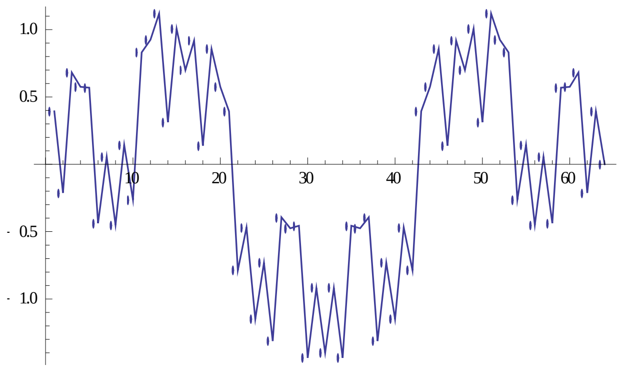


Figure 22: The discrete integration that results when the slope is adjusted to be less than 1. y(k) *vs.* k.

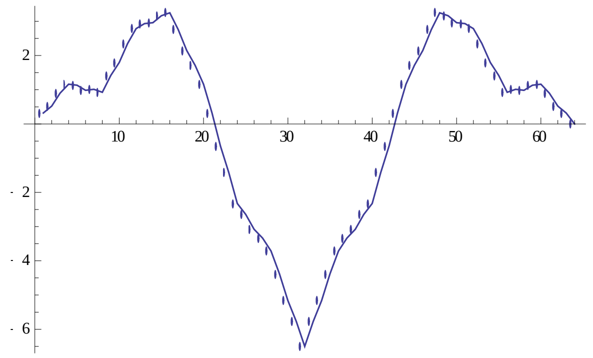


Figure 23: The discrete integration that results when the slope is Adjusted to be greater than 1. y(k) *vs.* k.

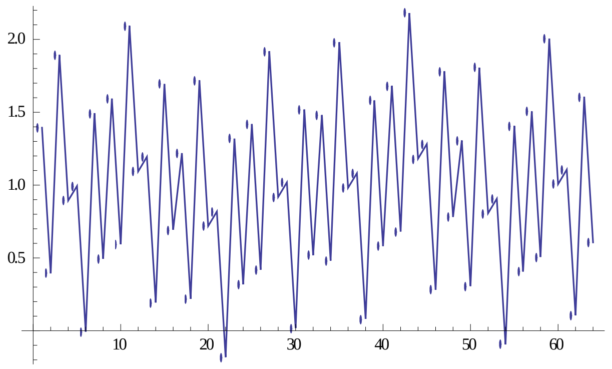


Figure 24: The interbeat intervals that result from the reverse DFA calculation, when the slope is less than 1, duration *(in seconds) vs.* ith *interval. This is closer to the white noise than the pink noise, from Figure 3, but not as random as expected. The resemblance may be greater if the number of intervals had been more than 64.*

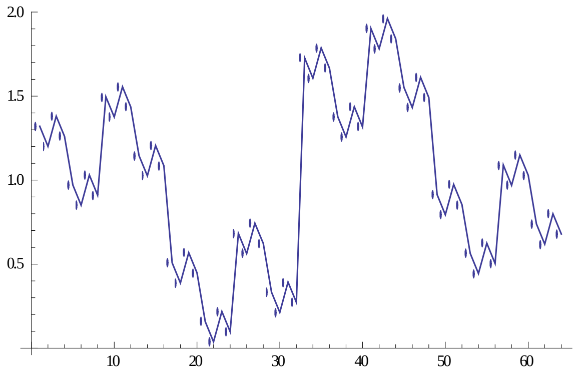


Figure 25: The interbeat intervals that result from the reverse DFA calculation, when the slope is greater than 1, duration *(in seconds) vs.* ith *interval. This is closer to the brown noise than pink noise from Figure 3. Here too, more intervals should make the resemblance more obvious.*

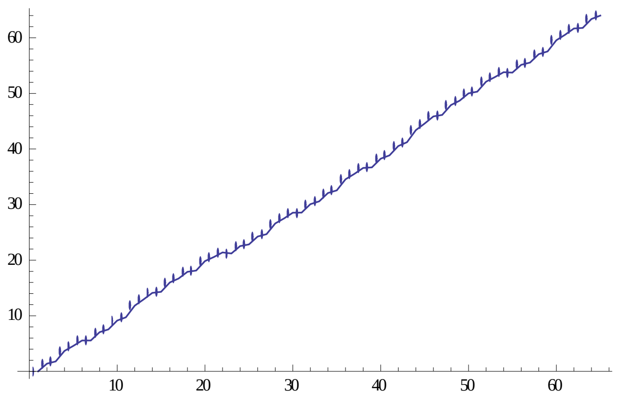


Figure 26: The times that result when the slope is adjusted to be less than 1, time *(in seconds) vs.* ith *beat.*

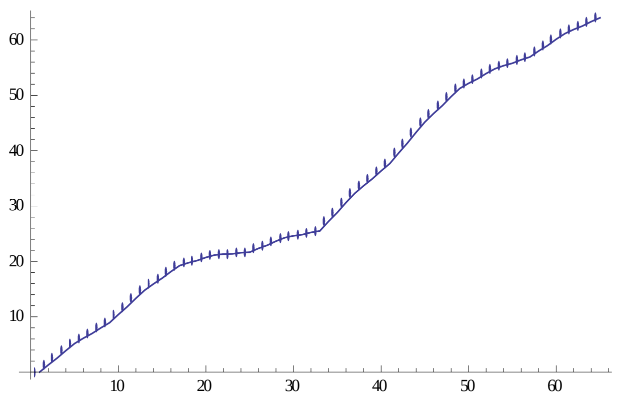


Figure 27: The times that result when the slope is adjusted to be greater than 1, time *(in seconds) vs.* ith *beat.*

Further exploration of this algorithm (in a future experiment) would clearly have great value in determining how and why the DFA algorithm works the way it does. The full DFA Reverse Engineer code can be found in the appendix.

Programming Notes:

- To learn how to create the code for the Sierpinski Triangle and Carpet, I started by analyzing the code for a (nonrandom) jagged line fractal that I found online **[70]**, to determine what was the specific code that was used to result in that fractal. (To find the specific code, go to row 13, click on "Mathematica notebooks", then click "NonRandom Cartoon"). I used a similar process, but with different functions and transformations, to generate both the Sierpinski Triangle and Carpet.
- I learned that the best approach to create a fractal in Mathematica was to start by defining specific starting points, then apply transformations to that set of points for several iterations.
- The first function I tried using to show the image of the Sierpinski Triangle was the Line function, but it never worked right because it resulted in either missing segments or erroneous segments; the Polygon function worked far better.
- The display of the Sierpinski Triangle broke down after a few iterations, because applying the transformations led to more brackets being placed around the set of points and the function

couldn't read the set anymore after enough iterations. So I used the Flatten function to get rid of the extraneous brackets.

- A side effect of the Flatten function was that it removed the way that the transformed points were grouped into triangles, so I added the Partition function to prevent this from happening.
- I learned that only the Print function could be used to show all iterations of the Sierpinski Triangle and Sierpinski Carpet at once, so I used that instead of the Show function.
- Very early on, when I was first going through the DFA by hand, I didn't yet understand that it was necessary to do the analysis for multiple different partition sizes of the set of interbeat intervals, but I learned this as I was performing the analysis and gained a deeper understanding of how the analysis worked than I had before. I also learned that it was better to start calculating the interbeat intervals from the first beat recorded, because that's more accurate than assuming a beat occurred at time 0.
- The first part of the DFA code that I automated was the list of local trend coordinates. I started with an empty list and used a For loop and the Append function to add each new calculated coordinate to the list after each iteration.
- Next, I automated the partial line fits, and found that the For loop wouldn't work to do this, because it only allowed an increase by 1 after each iteration. I learned the Do loop worked for this, because the size of the increase after each iteration can be controlled using the Do loop, so I could set that increase to be higher than 1. I started by creating a list with the first linear fit, then appending all remaining linear fits to that list using the Do loop.
- I chose to use just factors of N for the lengths of n, because each partition needs to be an equal length, and the programming logic was clearer to understand for n based on factors of N. For N with a large enough amount of different factors, the result should be accurate based on the linearity of a heartbeat time series DFA slope, but more partition lengths would certainly give greater precision. However, this modification is not a necessity if it proves too computationally

complex, because one of my sources reported 98% accuracy when using 20 scales over the range *$n = 10$* to *$n = 100$* and 40 scales over the range *$n = 100$* to *$n = 500$*, so analyzing all incremental values of *n* from 2 to *N* might just be over-engineering the code (perhaps providing only a psychological benefit to the wearer, rather than a genuine mathematical or medical one) **[64]**.

- It's necessary to translate all linear fits after the first one, because each sub-list has its initial value's index reset to "1" (and so it wouldn't line up on the graph at the correct place).
- When randomizing the list of times (instead of interbeat intervals directly), it's necessary to use the Sort function in order to ensure that the resulting list of times is in a logical, ascending sequence.
- When I automated the code, I first chose a Gamma distribution, because I needed a randomization method in which all the values could be forced to be positive (as there can't be negative durations). Later, I switched to a Half-Normal distribution, and experimented with a Chi-Square distribution with different degrees of freedom, and in all cases I found the results to be similar. For randomized intervals using each of these distributions, the final slope remains at a value very near 0.5. (This is based on running the algorithm for 1000 intervals using these distributions multiple times, but there may be distributions I didn't check in which the slope is consistently a much different value).
- The first way I handled the endpoints of the partitions was to find the mean of the local trend values of both partial fits that aligned with each respective endpoint. I experimented with different ways to handle the endpoints, and learned that the difference in the final value after the analysis was negligible for a large number of beats. In later iterations of the program, I just plugged the appropriate values into a piecewise function to determine all the y-coordinates of the local trends (where the local trend equation in the left-hand partition is used to determine the y-coordinate at the endpoints of each partition).

6.0 CONCLUSION

Using smart fabrics, someone's heartbeat could be monitored as they go about the course of their daily life, and a number of different programs could be run on the resulting time series to determine the health of the heart well before any health problem occurs. The detrended fluctuation analysis algorithm, as described in the previous section, returns the simplest result for the layperson to distinguish between a healthy or unhealthy heartbeat, but a multifractal extension of that algorithm could be programmed and run on the time series at the same time for the further benefit of a more in-depth assessment by their doctor. Another advantage to running two (or more, if programming and running some of the other multifractal analyses were desired) different algorithms at the same time is that it could compensate for any error that could occur in the run of any single algorithm on its own.

The combination of progress in nanotechnology, smart fabrics, and fractal research could allow for much greater methods of early detection of heart problems than we have now. But before this method of early detection could become the norm, there is some additional testing that is still necessary.

Future Experiments:

There are a number of further experiments that could be done with this code from here. A significant one would be to run the program on a number of different actual heartbeat time series data sets, and even try seeing to duplicate the results of any of the papers that performed the analysis directly on real patients' heartbeat time series, if they could access the same data. If they could get actual smart fabrics, they could try programming them to run the DFA program, and see how it compares to the current methods we have of monitoring the health of the heart. The program could be made more robust by modifying it to work for all values of n less than N (or simply additional values, if all possible values proves too complex), rather than just factors of N (and thus be able to be run on any number of beats, rather than just those values that are one greater than a number with multiple factors). This should increase the precision and accuracy of the result, especially for those values of N with a small number of factors. And for maximal analysis, another potential project could be to modify the code so it does a multifractal DFA, and then compare those results to the above experiments.

Before fabrics loaded with this program could go on the market, it is definitely needed to test the DFA method on larger groups than previous studies. Previous studies have been done on very small groups, largely due to the difficulty in finding data for significantly large groups over a long enough period of time for the analysis (and wearing the current devices to gather the data over enough time is often uncomfortable for the average volunteer). One of my sources has proposed a solution to this by using a method to distinguish the DFA results between healthy and unhealthy hearts on very short time scales (around 20 seconds) **[63]**. Multiple sources have indicated discrepancies from the longer time scale results during such a short time frame (the result is no longer 1 for a healthy heart in such a short period of time, and one hypothesis is that the dynamics of respiration is what dominates on this scale) **[64—66]**. My own proposal is to use the smart fabrics as a more comfortable way to gather the necessary data for larger groups over a larger period of time. But with different ways to modify the process to allow for testing on larger group sizes (as well as different types of multifractal analyses that already exist), it may be the case that different companies would be using different analyses by the time such smart fabrics were ready for the market. And if the numbers don't match for the very short time frame and longer ones, then it would be even better for the final step of the health test to be a color that part of the fabric changes to, based on the result of the specific analysis used (green or blue for healthy, and red for unhealthy, or even a possibility of orange or yellow for those multifractal analyses that can determine degree of health).

Another possibility would be to investigate whether the program could be loaded onto a fitness tracker and give results that match other methods, and get data from larger groups in the near term that way. However the studies are done on larger groups, it is very important that they are done, in order to answer questions regarding the likelihood of false positives or negatives, and determine how much the DFA slope can deviate from 1 and still indicate a healthy heart.

Future Potential:

Smart fabrics will be an immense help for reducing the incidence of heart attacks, thanks to the early detection they will make possible. And around the time the technology is reaching maturity, new technologies will emerge that will make early detection even easier. One of those will be pocket-sized MRIs (Magnetic Resonance Imaging devices) **[15]**.

Currently, the smallest MRI is called the MRI-MOUSE ("MOUSE" stands for "mobile universal surface explorer"), and it's about a foot tall in size. It is able to be so much smaller than a standard MRI thanks to its ability to function on non-uniform magnetic fields **[15]**. Standard MRIs require a uniform magnetic field, or else the resulting image will be distorted and therefore useless, and so very large magnets are needed to create this uniform magnetic field (which is also 20,000 times more powerful than the Earth's magnetic field **[15]**). The MRI-MOUSE compensates for the distortion from non-uniform fields, by sending multiple radio pulses into whatever it's scanning and detecting the resulting echoes. Computers then analyze the echoes and make up for the distortion. These weaker magnetic fields also make it possible to use the MRI-MOUSE on objects that contain metal, unlike conventional MRI machines (thus they would pose no problem to anyone with a metal implant). The MRI-MOUSE also consumes far less energy than a standard MRI, only requiring as much electricity as a light bulb **[15]**.

As the technique is improved, such mini-MRIs can be made smaller still, and eventually may even reach the size of a cell phone, a size that would make them useful to a consumer once their cost to create is low enough to also make them affordable. Mini-MRIs of this size would be extremely thin too, as thin as a dime and definitely unobtrusive **[15]**. When it's possible to reduce MRIs to such a size and put them on the market, anyone will be able to get a direct look at how healthy their organs are from their home. If the consumer doesn't have the expertise to determine whether what they're seeing is healthy or not, mini-MRIs can be loaded with software that can determine that for them (such as for those organs in which their health can be determined from their fractal dimension).

An advancement beyond this would be the ability to use ordinary visible light to see inside the body. MIT's Ramesh Raskar believes this could eventually be possible thanks to a highly advanced virtual slow-motion camera created by MIT's Camera Culture Group, which can capture action at a trillion exposures a second and thus make is possible to see the movement of individual photons **[71 – 72]**. Professor Raskar believes this will allow us to see inside the body because it will make it possible to see how light will scatter volumetrically inside the body. Once that is determined, a method of taking images of the interior of the body, using just visible light, can be developed **[71 – 72]**. This will make it even easier to determine early whether anyone has any potential health issues that can be dealt with well before they become a crisis. Eventually, we may even be able to grow new organs in the lab from a patient's very own individual cells **[15]**, and that ability combined with the smart fabrics will greatly reduce several of the top natural killers. A current obstacle to growing viable organs is developing the blood vessels necessary to supply them with nutrients, but researchers are beginning to have success in that goal, through studying their fractal structure and using 3D printers and lasers that stimulate the molecules on the nanoscale **[73]**.

Even beyond organ health, nanotechnology and fractals could unite to reduce the fatalness of viruses. Ji-Huan He (the author of the paper that discovered the relation between a virus' fractal geometry and its fatalness) has proposed using nanotechnology to reduce the complexity of a virus' surface (through medicine or temperature manipulation), thereby reducing the virus' fatalness. If successful, this could even prevent many organs and tissues from getting to the point that they need to be replaced (due to disease, as they'd still age).

Final Word:

It is clear, now, that not only are fractals an interesting area of math, but they may even save our lives one day.

7.0 Appendix

A. Full Mathematica Code for Sierpinski Triangle

Number of iterations:

```
i=7;
```

Below are the transformations used to generate the sets for successive iterations:

```
T1[x_]:={1/2*x[[1]],1/2*x[[2]]}
T2[x_]:={1/2*x[[1]]+1/2,1/2*x[[2]]}
T3[x_]:={1/2*x[[1]]+1/4,1/2*x[[2]]+Sqrt[3]/4}
```

Below is the starting set for the 0th iteration (representing a Euclidean equilateral triangle):

```
t[0]={{{0,0},{1,0},{1/2,Sqrt[3]/2}}};
```

Need to include a Flatten step in the loop, to keep later sets from being so nested that the transformations won't work on them:

```
s[0]=Flatten[t[0],1];
```

Need to group the resulting set as a nested set of triangles, using the Partition function:

```
l[0]=Partition[s[0],3];
```

Use Polygon in the Print function to draw the set as a triangle:

```
Print[Graphics[Polygon[l[0]]]];
```

The Do loop below creates all 7 iterations of the Sierpinski Triangle:

```
Do[{t[j]={Map[T1,s[j-1]],Map[T2,s[j-1]],Map[T3,s[j-1]]},s[j]=Flatten[t[j],1],l[j]=Partition[s[j],3], Print[Graphics[Polygon[l[j]]]]},{j,1,i}]
```

(The graphs can be seen in the main body of the paper).

B. Full Mathematica Code for Sierpinski Carpet

```
i=4;
```

More transformations are needed in order to create the sets representing the new squares, for each successive iteration:

```
T1[x_]:={1/3*x[[1]],1/3*x[[2]]}
T2[x_]:={1/3*x[[1]]+1/3,1/3*x[[2]]}
T3[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]}
T4[x_]:={1/3*x[[1]],1/3*x[[2]]+1/3}
T5[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]+1/3}
T6[x_]:={1/3*x[[1]],1/3*x[[2]]+2/3}
T7[x_]:={1/3*x[[1]]+1/3,1/3*x[[2]]+2/3}
T8[x_]:={1/3*x[[1]]+2/3,1/3*x[[2]]+2/3}
```

The following set represents a regular Euclidean square, for the 0th iteration:

```
t[0]={{{0,0},{1,0},{1,1},{0,1}}};

s[0]=Flatten[t[0],1];

l[0]=Partition[s[0],4];
```

Still use Polygon, but this time it draws the set as several squares:

```
Print[Graphics[Polygon[l[0]]]]
```

This Do loop creates all 4 iterations of the Sierpinski Carpet:

```
Do[{t[j]={Map[T1,s[j-1]],Map[T2,s[j-1]],Map[T3,s[j-1]],Map[T4,s[j-1]],Map[T5,s[j-1]],Map[T6,s[j-1]], Map[T7,s[j-1]],Map[T8,s[j-
1]]},s[j]=Flatten[t[j],1],l[j]=Partition[s[j],4],Print[Graphics[Polygon[l[j]]]]},{j,1,i}]
```

(The graphs can be seen in the main body of the paper).

C. Full Mathematica Implementation of the Toy Example and Large Data Set (with Code)

I started the Mathematica implementation by using the List function to create the same list of times that I used for the by-hand version. The Differences function took care of calculating the list of interbeat intervals, then I used Mean to find the average of this list. Subtracting this mean from the list of interbeat intervals gave the list *w*, then using the Accumulate function on *w* generated the list of partial sums of *w* (which is the discrete integration of the interbeat intervals list), represented by *y*. The code for all these steps can be seen below:

```
ListLinePlot[{t},AxesOrigin->{0,0},AxesLabel->{beat,time[sec]},PlotMarkers->Automatic,PlotLabel->"Heartbeat time series"]
```

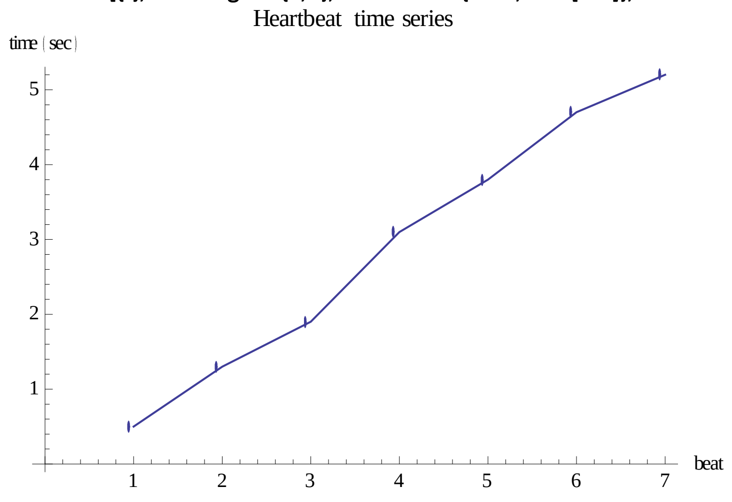


Figure 28: The heartbeat time series of the Toy Example, time *(in seconds) vs.* ith *beat.*

```
B=Differences[t]
{0.8,0.6,1.2,0.7,0.9,0.5}
```

In Mathematica, the default indexing for lists begins with 1, so the ListLinePlot function perfectly reproduces the results of the by-hand algorithm:

```
ListLinePlot[{B},AxesOrigin->{0,0},AxesLabel->{interval,duration[sec]},PlotMarkers->Automatic,PlotLabel->"Interbeat intervals"]
```

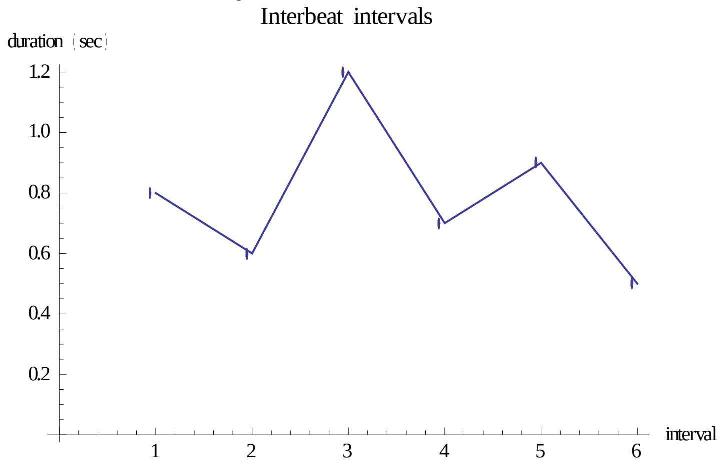


Figure 29: The interbeat intervals of the Toy Example, duration *(in seconds) vs.* ith *interval.*

```
Bave=Mean[B]
0.783333
w=B-Bave
{0.0166667,-0.183333,0.416667,-0.0833333,0.116667,-0.283333}
ListLinePlot[{w},AxesOrigin->{0,0},AxesLabel->{interval,mean offset[sec]},PlotMarkers->Automatic,PlotLabel->"Interbeat intervals (differences from the average)"]
```

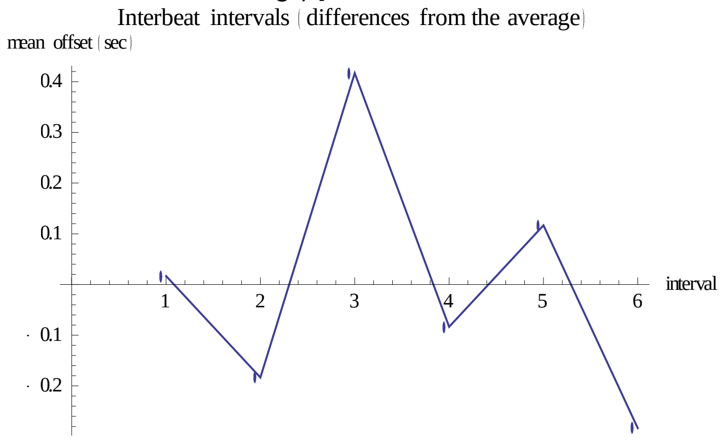


Figure 30: Interbeat intervals with the average subtracted.

```
y=Chop[Accumulate[w]]
{0.0166667,-0.166667,0.25,0.166667,0.283333,0}
ListLinePlot[{y},AxesOrigin->{0,0},AxesLabel->{k,Y[k]},PlotStyle->Thick,PlotMarkers->Automatic,PlotLabel->"Discrete Integration"]
```

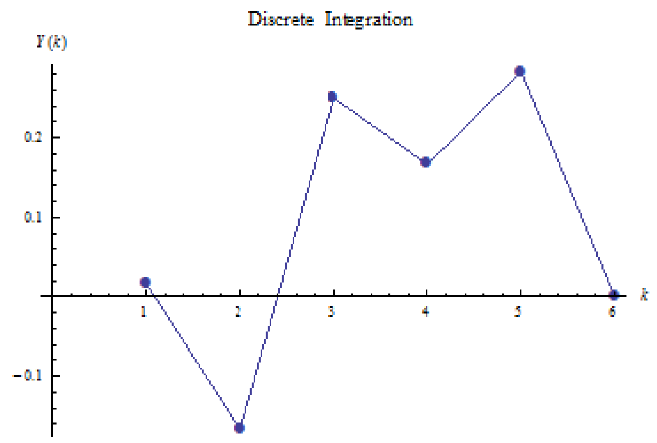


Figure 31: The discrete integration of the Toy Example, y(k) *vs.* k.

I defined several sublists of y, designated by z, in order to find the partial line fits (using the Fit function on these sublists) for the different partitions that the entire set was broken into. The following code is for partitions of length $n = 2$:

```
z1=y[[1;;2]]
{0.0166667,-0.166667}
z2=y[[2;;4]]
{-0.166667,0.25,0.166667}
z2=y[[4;;6]]
{0.166667,0.283333,2.22045×10^-16}
l1=Fit[z1,{1,x},x]
0.2 -0.183333 x
L1=l1;
l2=Fit[z2,{1,x},x]
-0.25+0.166667 x
L2=l2/.x->(x-1)
-0.416667+0.166667 x
l3=Fit[z3,{1,x},x]
0.316667 -0.0833333 x
L3=l3/.x->(x-3)
0.566667 -0.0833333 x
y21=L1/.x->1
0.0166667
y22a=L1/.x->2
-0.166667
y22b=L2/.x->2
-0.0833333
y22=Mean[{y22a,y22b}]
-0.125
y23=L2/.x->3
0.0833333
y24a=L2/.x->4
0.25
y24b=L3/.x->4
0.233333
y24=Mean[{y24a,y24b}]
0.241667
y25=L3/.x->5
0.15
```

```
y26=L3/.x->6
0.0666667
y2={y21,y22,y23,y24,y25,y26}
{0.0166667,-0.125,0.0833333,0.241667,0.15,0.0666667}
```

(See the main paper for the piecewise version of the resulting graph).

Now something similar for *n* = 3:

```
Z1=y[[1;;3]]
{0.0166667,-0.166667,0.25}
Z2=y[[3;;6]]
{0.25,0.166667,0.283333,2.22045×10^-16}
m1=Fit[Z1,{1,x},x]
-0.2+0.116667 x
M1=m1;
m2=Fit[Z2,{1,x},x]
0.333333 -0.0633333 x
M2=m2/.x->(x-2)
0.46 -0.0633333 x
y31=M1/.x->1
-0.0833333
y32=M1/.x->2
0.0333333
y33a=M1/.x->3
0.15
y33b=M2/.x->3
0.27
y33=Mean[{y33a,y33b}]
0.21
y34=M2/.x->4
0.206667
y35=M2/.x->5
0.143333
y36=M2/.x->6
0.08
y3={y31,y32,y33,y34,y35,y36}
{-0.0833333,0.0333333,0.21,0.206667,0.143333,0.08}
```

(See the main paper for the piecewise version of the resulting graph).

Now something similar for *n* = 6:

```
V1=y;
n1=Fit[V1,{1,x},x]
-0.0266667+0.0338095 x
Show[ListLinePlot[{y},AxesOrigin->{0,0}],Plot[{n1},{x,0,10},AxesOrigin->{0,0}]]
```

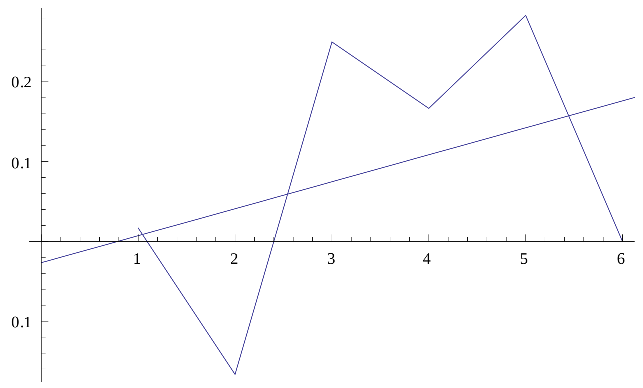


Figure 32: Discrete integration with local trends for n = 6, y(k) *vs.* k.

```
y61=n1/.x->1
 0.00714286
 y62=n1/.x->2
 0.0409524
 y63=n1/.x->3
 0.0747619
 y64=n1/.x->4
 0.108571
 y65=n1/.x->5
 0.142381
 y66=n1/.x->6
 0.17619
 y6={y61,y62,y63,y64,y65,y66}
 {0.00714286,0.0409524,0.0747619,0.108571,0.142381,0.17619}
```

Below, *d* represents the differences between the y-coordinates of the time series and the local trends, *s* represents the squares of these differences, *F* is the detrended fluctuation as previously defined, *p* is the logarithm of the detrended fluctuation, and *P* is the logarithm of the partition length, *n*. The Total function helps in determining *F* because it sums all the elements in each *s* list:

```
 d2=y-y2
 {-2.77556×10^-17,-8.32667×10^-17,0.166667,-0.0833333,0.133333,-0.0666667}
 s2=d2^2
 {7.70372×10^-34,6.93335×10^-33,0.0277778,0.00694444,0.0177778,0.00444444}
 F2=Sqrt[Total[s2]/6]
 0.0974204
 p2=Log10[F2]
-1.01135
 P2=N[Log10[2]]
 0.30103
 d3=y-y3
 {0.1,-0.2,0.1,-0.04,0.14,-0.08}
 s3=d3^2
 {0.01,0.04,0.01,0.0016,0.0196,0.0064}
 F3=Sqrt[Total[s3]/6]
 0.12083
 p3=Log10[F3]
-0.917824
 P3=N[Log10[3]]
```

```
0.477121
d6=y-y6
{0.00952381,-0.207619,0.175238,0.0580952,0.140952,-0.17619}
s6=d6^2
{0.0000907029,0.0431057,0.0307084,0.00337506,0.0198676,0.0310431}
F6=Sqrt[Total[s6]/6]
0.146168
p6=Log10[F6]
-0.835148
P6=N[Log10[6]]
0.778151
```

The nested list below collects together all the logarithm values to produce a log *F(n)* vs. log *n* plot:

```
p={{P2,p2},{P3,p3},{P6,p6}}
{{0.30103,-1.01135},{0.477121,-0.917824},{0.778151,-0.835148}}
P=ListPlot[p,PlotMarkers->Automatic]
q=Fit[p,{1,x},x]
-1.10773+0.35911 x
Show[P,Plot[{q},{x,0,1}]]
```

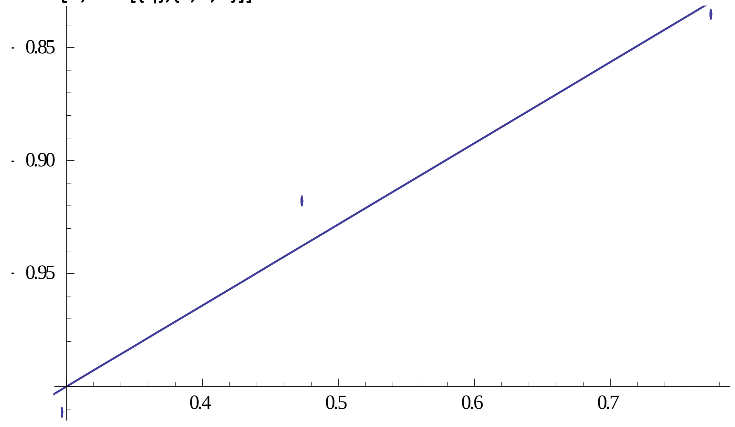


Figure 33: Log-log plot for the detrended fluctuations (with best fit line), log F(n) *vs.* log n *(the origin is absent in this graph).*

Randomizing the Toy Example:

Before I discovered how to automate the toy example, I randomized the list of interbeat intervals using the following code:

```
B=RandomReal[GammaDistribution[1,1],6]
{4.17923,0.641174,0.365345,0.122367,1.10855,2.65316}
ListLinePlot[B,AxesOrigin->{0,0}]
```

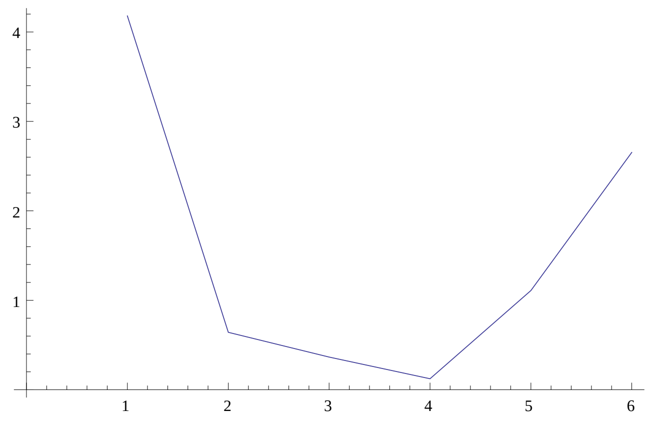


Figure 34: Randomized interbeat intervals for the Toy Example, duration (in seconds) vs. ith interval.

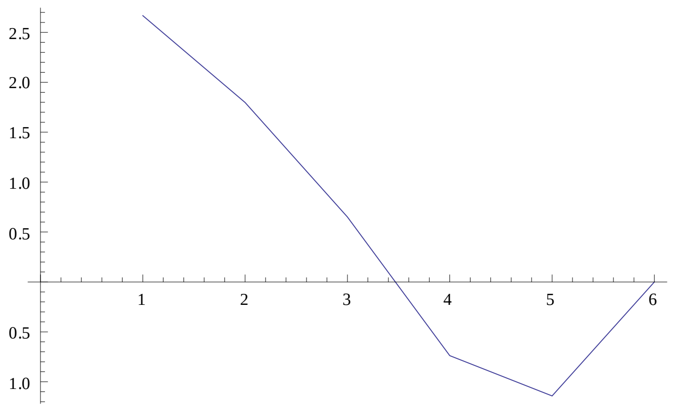


Figure 35: The discrete integration that results for the randomized Toy Example, y(k) vs. k.

(The remaining code was the same for this version of the program).

This is necessary to simulate a large number of beats, because inputting the data manually would take too long otherwise. And they were only simulated for the remaining programs because I was not able to get the data for an actual heartbeat time series, as that is protected information.

Automated Toy Example:

The following code is the automated version of the linear fits. Since the indexing starts at 1 rather than 0, the first linear fit is defined manually, and the interval is included as part of the first element of the linear fits list (for the sake of creating the Piecewise function later). A Do loop is used to find the linear fits for the remaining partitions and append them to the original linear fits list:

```
l2={{Fit[y[[1;;2]],{1,x},x],0<x≤2}}
{{0.2 -0.183333 x,0<x≤2}}
Do[l2=AppendTo[l2,{Fit[y[[i;;i+2]],{1,x},x]/.x->x-i+1(*to account for the indexing starting with 1*),i<x≤i+2}],{i,2,4,2}]
l2
{{0.2 -0.183333 x,0<x≤2},{-0.416667+0.166667 x,2<x≤4},{0.566667 -0.0833333 x,4<x≤6}}
l3={{Fit[y[[1;;3]],{1,x},x],0<x≤3}}
{{-0.2+0.116667 x,0<x≤3}}
Do[l3=AppendTo[l3,{Fit[y[[i;;i+3]],{1,x},x]/.x->x-i+1,i<x≤i+3}],
{i,3,3,3}]
l3
{{-0.2+0.116667 x,0<x≤3},{0.46 -0.0633333 x,3<x≤6}}
l6={{Fit[y[[1;;6]],{1,x},x],0<x≤6}}
{{-0.0266667+0.0338095 x,0<x≤6}}
```

I used the Piecewise function on the list of linear fits and intervals, to ensure the local trend line would appear on the graph only along the relevant interval, and I gave the fits a different color so that it'd be easier to differentiate them from the main graph of the discrete integration of the interbeat intervals:

```
G2=Piecewise[L2]
 {
 {{
   {0.2-0.183333 x, 0<x≤2},
   {-0.416667+0.166667 x, 2<x≤4},
   {0.566667-0.0833333 x, 4<x≤6},
   {0, True}
  }}
}
 g2=Plot[G2,{x,0,6},AxesOrigin->{0,0},PlotStyle->RGBColor[1,0,0]]
 Show[g,g2]
```

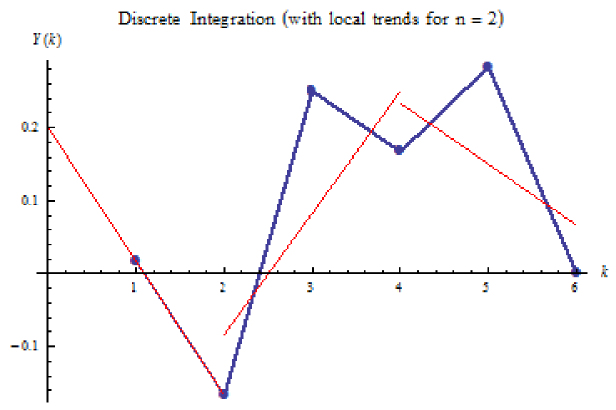


Figure 36: Discrete integration with local trends for n = 2, y(k) *vs.* k.

```
G3=Piecewise[L3]
 {
 {{
   {-0.2+0.116667 x, 0<x≤3},
   {0.46-0.0633333 x, 3<x≤6},
   {0, True}
  }}
}
 g3=Plot[G3,{x,0,6},AxesOrigin->{0,0},PlotStyle->RGBColor[1,0,0]]
 Show[g,g3]
```

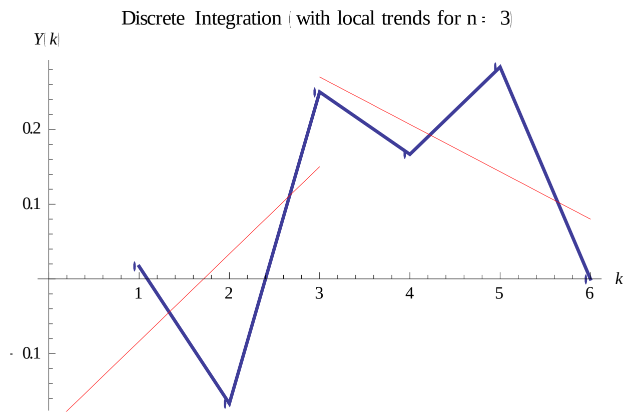


Figure 37: Discrete integration with local trends for n = 3, y(k) *vs.* k.

```
G6=Piecewise[L6]
{
{{
 {-0.0266667+0.0338095 x, 0<x≤6},
 {0, True}
}}
}
Show[g,Plot[G6,{x,0,6},AxesOrigin->{0,0},PlotStyle->RGBColor[1,0,0]]]
```

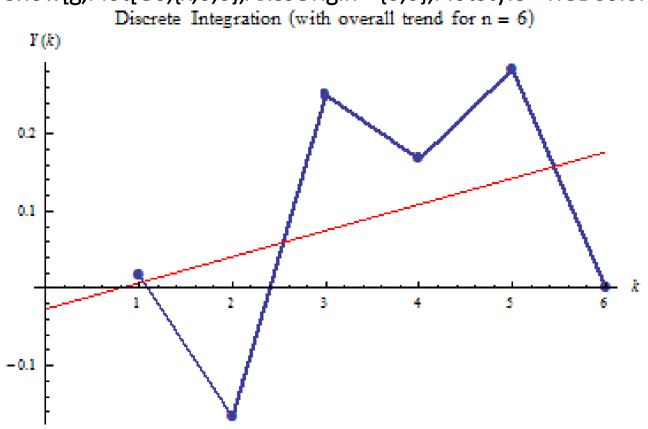


Figure 38: Discrete integration with local trends for n = 6, y(k) *vs.* k.

The Piecewise function makes it possible to automate the lists of the coordinates of the local trends, and here I used a For loop to build the lists:

```
y2={};
For[i=1,i≤6,i++,y2=AppendTo[y2,G2/.x->i]]
y2
{0.0166667,-0.166667,0.0833333,0.25,0.15,0.0666667}
y3={};
For[i=1,i≤6,i++,y3=AppendTo[y3,G3/.x->i]]
y3
{-0.0833333,0.0333333,0.15,0.206667,0.143333,0.08}
y6={};
For[i=1,i≤6,i++,y6=AppendTo[y6,G6/.x->i]]
y6
{0.00714286,0.0409524,0.0747619,0.108571,0.142381,0.17619}
```

Realistic Scale Example:

In the final iteration of the program, I used a nested Do loop to automate everything after the determination of the discrete integration list, *y*. I started by defining *M* to be the number of interbeat intervals, since *N* can't be used as a variable in Mathematica (it's a function that returns the numerical value of something). It is possible to use a list to define the increment change within a Do loop, so I created the list of partition sizes, *n*, by finding the factors of *M* with the Divisors function and removing 1 from the resulting list using the Length function (taking one less element than the length of the list from that list). This nested loop brought the total lines of code down to 15 (whereas earlier iterations of the code would have required several thousand lines of code to conduct the analysis on 1000 interbeat intervals) and made it so that a single change in the first line (the value of *M*) was all that was necessary to run the analysis for any number of beats:

```
M=1000;
div=Divisors[M]
 {1,2,4,5,8,10,20,25,40,50,100,125,200,250,500,1000}
n=Take[div,-(Length[div]-1)]
 {2,4,5,8,10,20,25,40,50,100,125,200,250,500,1000}
B=RandomReal[HalfNormalDistribution[1],M];
ListLinePlot[B,AxesOrigin->{0,0},PlotMarkers->Automatic]
```

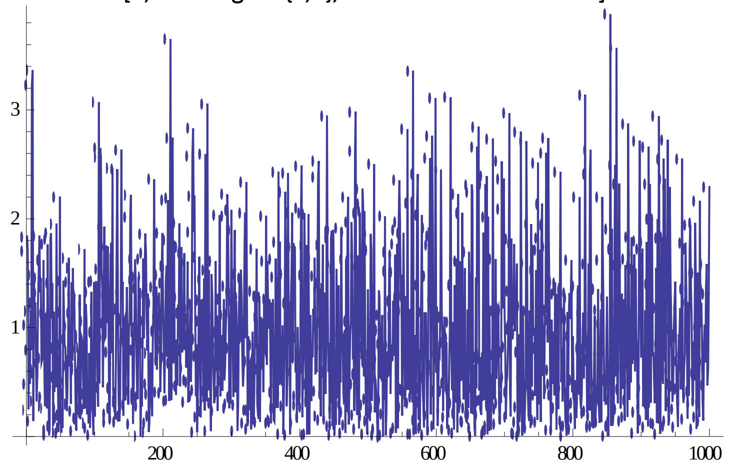


Figure 39: 1000 randomized interbeat intervals, duration *(in seconds) vs.* ith *interval.*

```
Bave=Mean[B]
0.974379
w=B-Bave;
y=Accumulate[w];
g=ListLinePlot[y,AxesOrigin->{0,0},PlotMarkers->Automatic]
```

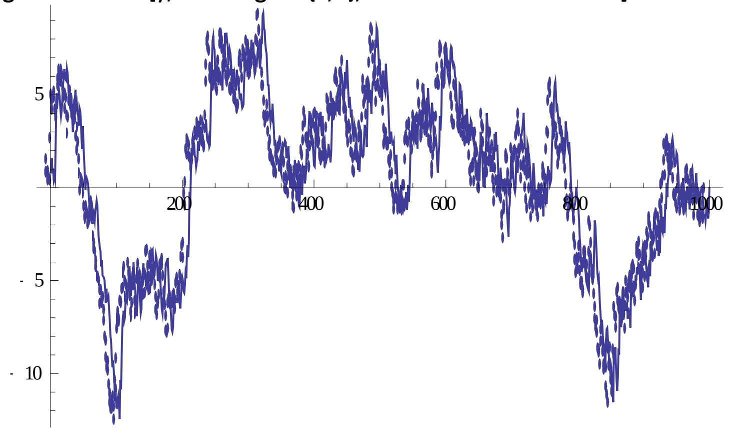


Figure 40: Discrete integration for 1000 intervals, y(k) *vs.* k.

I made an empty list, *r*, into which I appended all the logarithm values ({*P*,*p*}) as the final step of the nested Do loop. Since *i* is already the variable that represents each iteration in the inner Do and For loops, *j* serves this purpose in the outer Do loop, and is set to the values of the partition sizes, *n*. Neither Show nor Print on their own would generate the graphs within the loop, but using them both made it possible, by defining a variable to be equal to the Show command, then using the Print function on that variable. After each graph, I printed a list that contained the values for the partition size of that iteration, the logarithm of that partition size, the detrended fluctuation, and the logarithm of that detrended fluctuation:

```
r={};

Do[l[k]={{Fit[y[[1;;k]],{1,x},x],0<x≤k}};Do[l[k]=AppendTo[l[k],{Fit[y[[i;;i+k]],{1,x},x]/.x->x-i+1,i<x≤i+k}],{i,k,M-k,k}];
G[k]=Piecewise[l[k]];A[k]=Show[g,Plot[G[k],{x,0,M},AxesOrigin->{0,0},PlotStyle->RGBColor[1,0,0]]];Print[A[k]];Y[k]={};
For[i=1,i≤M,i++,Y[k]=AppendTo[Y[k],G[k]/.x->i]];d[k]=y-Y[k];s[k]=d[k]^2;F[k]=Sqrt[Total[s[k]]/M];p[k]=Log10[F[k]];
P[k]=N[Log10[k]];Print[{{k,P[k]},{F[k],p[k]}}];r=AppendTo[r,{P[k],p[k]}],{k,n}]
```

(The resulting graphs can be seen in Table 6 in the main body of the paper).

```
r
 {{0.30103,-0.555296},{0.60206,-0.395384},{0.69897,-0.358269},{0.90309,-0.256691},{1.,-0.201477},{1.30103,-
0.0851527},{1.39794,-
0.0492207},{1.60206,0.0819889},{1.69897,0.104812},{2.,0.27725},{2.09691,0.304059},{2.30103,0.417484},{2.39794,0.532106}
,{2.69897,0.581149},{3.,0.649302}}
R=ListPlot[r,PlotMarkers->Automatic]
q=Fit[r,{1,x},x]
Show[R,Plot[q,{x,0,3}]]
```

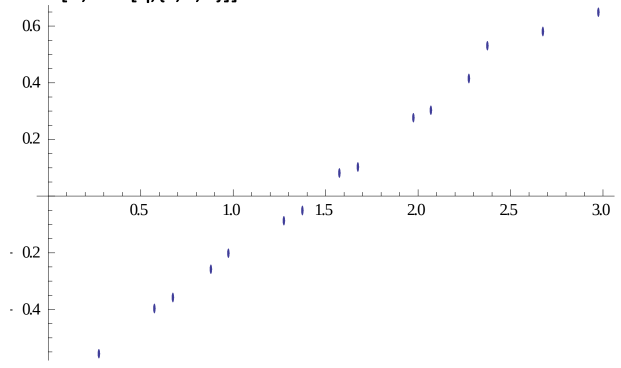


Figure 41: Log-log plot for 1000 intervals (without best fit line), log F(n) *vs.* log n.

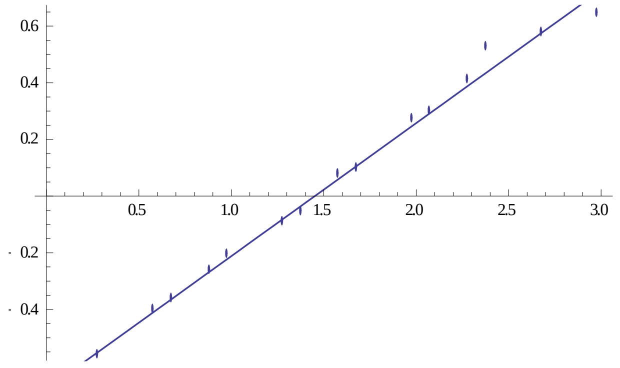


Figure 42: Log-log plot for 1000 intervals (with best fit line), log F(n) *vs.* log n.

```
-0.681365+0.469464 x
```

The slope above would also indicate an unhealthy heart, though it's closer to healthy than the Toy Example (not that this would be much solace if it were an actual result for some wearer, considering how it’s still very far from healthy).

D. Full Mathematica Code for Reverse Engineering the DFA Algorithm

```
f[x_]:=Piecewise[{{4x, 0≤x<1/4}, {1-12 (x-1/4), 1/4≤x<1/2}, {-2+12(x-1/2), 1/2≤x<3/4}, {1-4(x-3/4), 3/4≤x≤1}},0]
A1[x_]:=f[x/64]
A2[x_]:=f[Mod[x/16, 1]]
A3[x_]:=f[Mod[x/4, 1]]
Plot[{A1[t], A2[t], A3[t]}, {t,-1,65}, PlotRange->{-2,3}, PlotStyle->Thick]
```

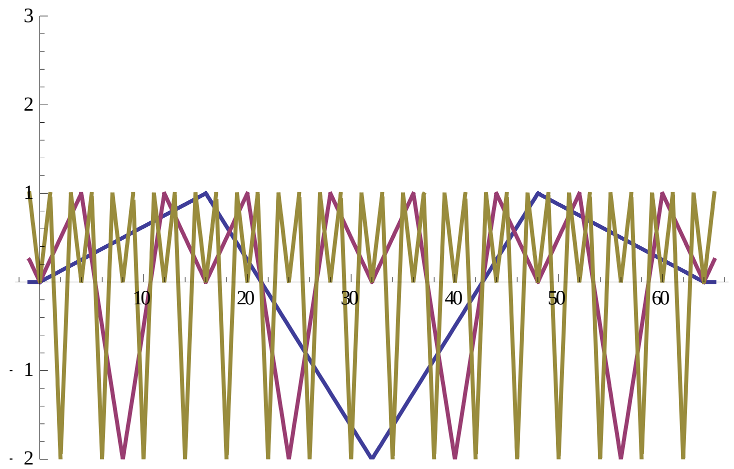


Figure 43: Each A(h) *subfunction plotted together, without being added together or multiplied by the* a(h) *parameters.* y(k) *vs.* k.

General function = a1*A1[t]+a2*A2[t]+a3*A3[t]
Special case : a1 = 1, a2 = 1/4, a3 = 1/16

Plot[{A1[t], A1[t]+A2[t]/4,A1[t]+A2[t]/4+A3[t]/16}, {t, -1, 65}]

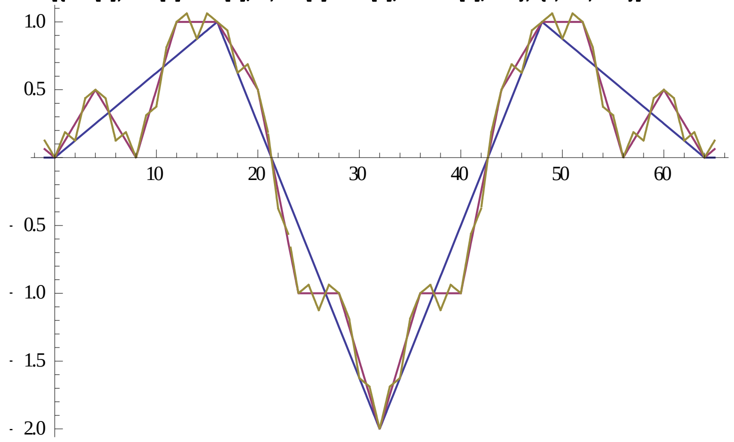


Figure 44: The first 3 iterations of this adjustable function representing a discrete integration. Each new iteration adds an A(h) *subfunction multiplied by a new* a(h) *parameter.* y(k) *vs.* k.

Back to general case with coefficients a1, a2, a3.
For n = 4 (intervals of length 4), with full function a1*A1[t] + a2*A2[t] + a3*A3[t], the local trend is a1*A1[t] + a2*A2[t], and the difference is a3*A3[t]. The sums of these squares gives F4:

F4=Sqrt[Sum[a3*A3[t]*a3*A3[t],{t,0,64}]/64]

$$\sqrt{\frac{3}{2}}\sqrt{\mathrm{a3}^2}$$

For n = 16 (intervals of length 16), with full function a1*A1[t] + a2*A2[t] + a3*A3[t], the local trend is a1*A1[t] and the difference is a2*A2[t] + a3*A3[t]. The sums of these squares gives F16 :

F16=Sqrt[Sum[(a2*A2[t]+a3*A3[t])*(a2*A2[t]+a3*A3[t]),{t,0,64}]/64]

$$\frac{1}{8}\sqrt{24\mathrm{a2}^2+8(-\frac{\mathrm{a2}}{2}-2\mathrm{a3})^2+8(\frac{\mathrm{a2}}{2}-2\mathrm{a3})^2+8(-\frac{5\mathrm{a2}}{4}+\mathrm{a3})^2+16(\frac{\mathrm{a2}}{4}+\mathrm{a3})^2+8(\frac{3\mathrm{a2}}{4}+\mathrm{a3})^2}$$

For n = 64 (intervals of length 64), with full function a1*A1[t] + a2*A2[t] + a3*A3[t], the local trend is 0 and the difference is a2*A2[t] + a3*A3[t]. The sums of these squares gives F64:

F64=Sqrt[Sum[(a1*A1[t]+a2*A2[t]+a3*A3[t])*(a1*A1[t]+a2*A2[t]+a3*A3[t]),{t,0,64}]/64]

$$\frac{1}{8}\sqrt{}(6a1^2 + 2(-\frac{a1}{2} - 2a2)^2 + 2(\frac{a1}{2} - 2a2)^2 + 2(-\frac{5a1}{4} + a2)^2 + 4(\frac{a1}{4} + a2)^2 + 2(\frac{3a1}{4} + a2)^2 + 2(-\frac{7a1}{8} - \frac{a2}{2} - 2a3)^2$$
$$+ 2(-\frac{a1}{8} - \frac{a2}{2} - 2a3)^2 + 2(\frac{3a1}{8} - \frac{a2}{2} - 2a3)^2 + 2(\frac{5a1}{8} - \frac{a2}{2} - 2a3)^2 + 2(-\frac{13a1}{8} + \frac{a2}{2} - 2a3)^2$$
$$+ 2(\frac{a1}{8} + \frac{a2}{2} - 2a3)^2 + 2(\frac{5a1}{8} + \frac{a2}{2} - 2a3)^2 + 2(\frac{7a1}{8} + \frac{a2}{2} - 2a3)^2 + 2(-\frac{11a1}{16} - \frac{5a2}{4} + a3)^2$$
$$+ 2(-\frac{5a1}{16} - \frac{5a2}{4} + a3)^2 + 2(\frac{7a1}{16} - \frac{5a2}{4} + a3)^2 + 2(\frac{9a1}{16} - \frac{5a2}{4} + a3)^2 + 2(-\frac{29a1}{16} + \frac{a2}{4} + a3)^2$$
$$+ 2(-\frac{17a1}{16} + \frac{a2}{4} + a3)^2 + 4(\frac{a1}{16} + \frac{a2}{4} + a3)^2 + 2(\frac{5a1}{16} + \frac{a2}{4} + a3)^2 + 2(\frac{11a1}{16} + \frac{a2}{4} + a3)^2$$
$$+ 2(\frac{13a1}{16} + \frac{a2}{4} + a3)^2 + 2(\frac{15a1}{16} + \frac{a2}{4} + a3)^2 + 2(-\frac{23a1}{16} + \frac{3a2}{4} + a3)^2 + 2(\frac{3a1}{16} + \frac{3a2}{4} + a3)^2$$
$$+ 2(\frac{7a1}{16} + \frac{3a2}{4} + a3)^2 + 2(\frac{13a1}{16} + \frac{3a2}{4} + a3)^2)$$

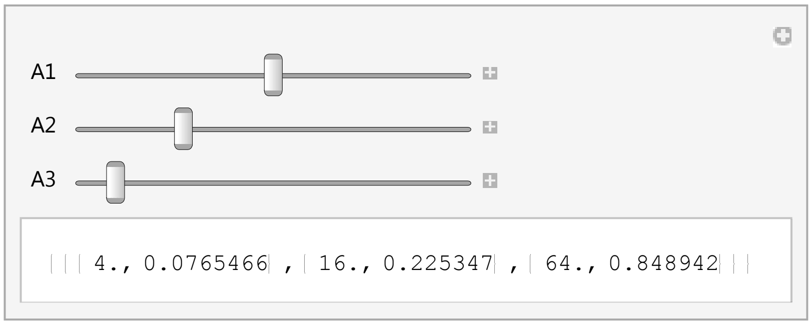
Simplify[F64]

$$\frac{1}{16}\sqrt{\frac{343a1^2}{2} + 184a2^2 + 384a3^2}$$

```
N[F16/.{a1->1, a2->.25, a3->1/16}]
N[F64/.{a1->1, a2->.25, a3->1/16}]
N[F4/.{a1->1, a2->.25, a3->1/16}]
 0.225347
 0.848942
 0.0765466
```

List coordinates for (n, F[n]) :

```
Manipulate[N[List[{{4, F4}, {16, F16}, {64, F64}}]/.{a1->A1,a2->A2, a3->A3}],{{A1,1}, 0, 2, .05},{{A2, .25},0, 1, .05},{{A3,1/16}, 0,1, .05}]
```



```
||| 4., 0.0765466| , | 16., 0.225347| , | 64., 0.848942|||
```

Figure 45: Sliders that allow for the direct manipulation of the detrended fluctuation values. The brackets aren't showing up due to the aforementioned formatting issues.

Plot (n, F[n]) :

```
Manipulate[ListLinePlot[{{4, F4},{16, F16},{64, F64}}/.{a1->A1,a2->A2, a3->A3},PlotRange->{{0,70}, {0,10}},
PlotMarkers->Automatic],{{A1,1}, 0, 2, .05},{{A2, .25},0, 1, .05},{{A3,1/16}, 0,1, .05}]
```

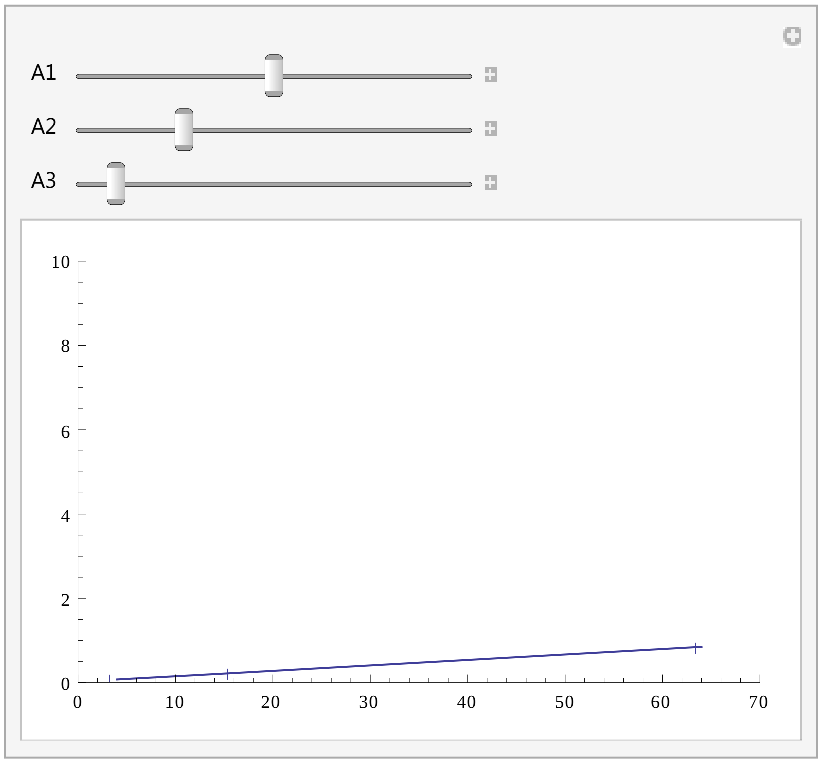
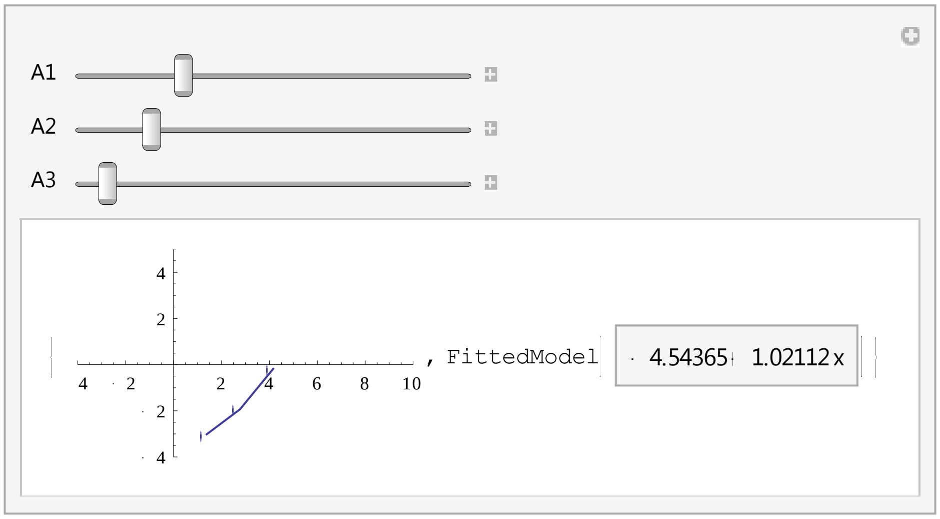


Figure 46: Plot that results from directly manipulating the detrended fluctuation values, F(n) *vs.* n.

Plot (n, F[n]) and compute the best linear fit:. Note that multiplying all coefficients by a constant k does not change the slope of the linear fit. Translates the graph up by Log(k).

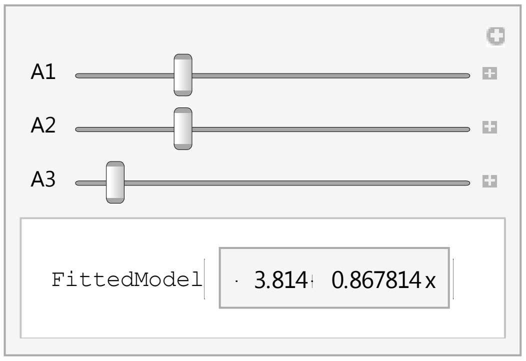
Manipulate[{ListLinePlot[{{Log[4], Log[F4]},{Log[16], Log[F16]}, {Log[64], Log[F64]}}/.{a1->A1,a2->A2, a3->A3},PlotRange->{{-4,10}, {-4,5}},PlotMarkers->Automatic],LinearModelFit[{{Log[4], Log[F4]},{Log[16], Log[F16]}, {Log[64], Log[F64]}}/.{a1->A1,a2->A2, a3->A3}, x,x]}, {{A1,1}, 0, 4, .05},{{A2, .16},0, 1,.01}, {{A3,.04}, 0,1, .01}]



, FittedModel[4.54365 1.02112 x]

Figure 47: Equation and plot for the log-log plot taht results from directly manipulating the detrended fluctuation values, log F(n) *vs.* log n. *The equation is distorted due to the aforementioned formatting issues. It should read* -4.54365+1.02112x.

Manipulate[LinearModelFit[{{Log[4], Log[F4]},{Log[16], Log[F16]}, {Log[64], Log[F64]}}/.{a1->A1,a2->A2, a3->A3}, x,x], {{A1,1}, 0, 4, .05},{{A2, .25},0, 1, .01},{{A3,1/16}, 0,1, .01}]



```
FittedModel  · 3.814· 0.867814 x
```

*Figure 48: Manipulable equation of the log-log plot, by itself (*log F(n) *vs.* log n*). The equation is distorted due to the aforementioned formatting issues. It should read* -3.814+0.867814x.

```
r=List[{{Log[4],Log[F4]},{Log[16],Log[F16]},{Log[64],Log[F64]}}/.{a1->1.75,a2->0.375,a3->0.075}]
{{{Log[4],-2.38753},{Log[16],-1.10586},{Log[64],0.38532}}}
```

```
q=ListLinePlot[r]
```

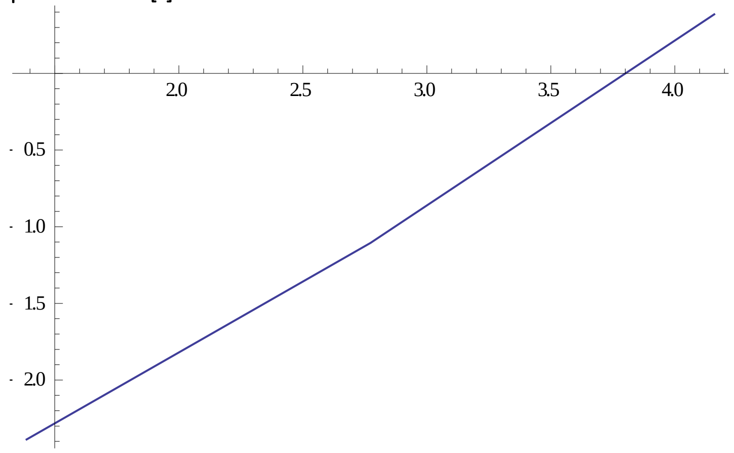


Figure 49: The line that results for the log-log plot after adjusting the parameters to force it to have a slope of 1. log F(n) *vs.* log n.

```
Fit[Flatten[r,1],{1,x},x]
 -3.80888+1.0001 x
s[t_]:=1.75*A1[t]+0.375*A2[t]+0.075*A3[t]
y={};
For[i=1,i<65,i++,y=AppendTo[y,s[i]]]
y

{0.278125,0.25625,0.684375,0.8125,0.715625,0.31875,0.371875,0.125,0.590625,0.75625,1.37188,1.6875,1.77813,1.56875,1.80938,1.75,1.59063,1.13125,1.12188,0.8125,0.278125,-0.55625,-0.940625,-1.625,-1.59688,-1.86875,-1.69063,-1.8125,-2.15938,-2.80625,-3.00313,-3.5,-3.00313,-2.80625,-2.15938,-1.8125,-1.69063,-1.86875,-1.59688,-1.625,-0.940625,-0.55625,0.278125,0.8125,1.12188,1.13125,1.59063,1.75,1.80938,1.56875,1.77813,1.6875,1.37188,0.75625,0.590625,0.125,0.371875,0.31875,0.715625,0.8125,0.684375,0.25625,0.278125,0.}

ListLinePlot[y,AxesOrigin->{0,0},PlotMarkers->Automatic]
```

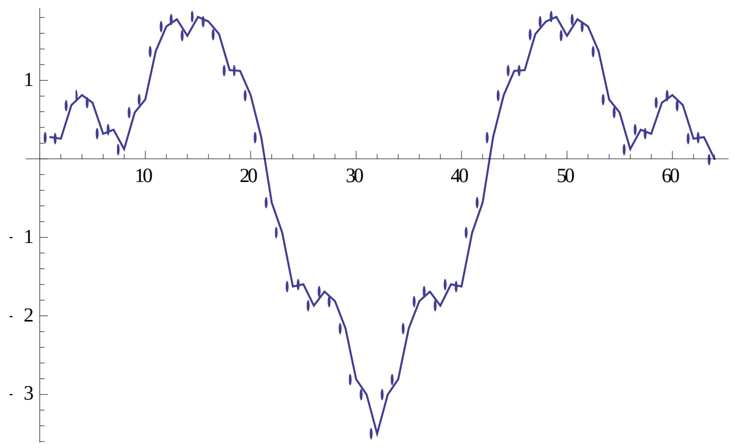


Figure 50: The discrete integration that results in the log-log plot from Figure 42, y(k) *vs.* k.

```
w={y[[1]]}
 {0.278125}
For[i=2,i<65,i++,w=AppendTo[w,y[[i]]-y[[i-1]]]]
w
 {0.278125,-0.021875,0.428125,0.128125,-0.096875,-0.396875,0.053125,-
0.246875,0.465625,0.165625,0.615625,0.315625,0.090625,-0.209375,0.240625,-0.059375,-0.159375,-0.459375,-0.009375,-
0.309375,-0.534375,-0.834375,-0.384375,-0.684375,0.028125,-0.271875,0.178125,-0.121875,-0.346875,-0.646875,-
0.196875,-0.496875,0.496875,0.196875,0.646875,0.346875,0.121875,-0.178125,0.271875,-
0.028125,0.684375,0.384375,0.834375,0.534375,0.309375,0.009375,0.459375,0.159375,0.059375,-0.240625,0.209375,-
0.090625,-0.315625,-0.615625,-0.165625,-0.465625,0.246875,-0.053125,0.396875,0.096875,-0.128125,-0.428125,0.021875,-
0.278125}

z=Accumulate[w]
{0.278125,0.25625,0.684375,0.8125,0.715625,0.31875,0.371875,0.125,0.590625,0.75625,1.37188,1.6875,1.77813,1.56875,1.
80938,1.75,1.59063,1.13125,1.12188,0.8125,0.278125,-0.55625,-0.940625,-1.625,-1.59688,-1.86875,-1.69063,-1.8125,-
2.15938,-2.80625,-3.00313,-3.5,-3.00313,-2.80625,-2.15938,-1.8125,-1.69063,-1.86875,-1.59688,-1.625,-0.940625,-
0.55625,0.278125,0.8125,1.12188,1.13125,1.59063,1.75,1.80938,1.56875,1.77813,1.6875,1.37188,0.75625,0.590625,0.125,0
.371875,0.31875,0.715625,0.8125,0.684375,0.25625,0.278125,2.22045×10^-16}
Chop[%]
{0.278125,0.25625,0.684375,0.8125,0.715625,0.31875,0.371875,0.125,0.590625,0.75625,1.37188,1.6875,1.77813,1.56875,1.
80938,1.75,1.59063,1.13125,1.12188,0.8125,0.278125,-0.55625,-0.940625,-1.625,-1.59688,-1.86875,-1.69063,-1.8125,-
2.15938,-2.80625,-3.00313,-3.5,-3.00313,-2.80625,-2.15938,-1.8125,-1.69063,-1.86875,-1.59688,-1.625,-0.940625,-
0.55625,0.278125,0.8125,1.12188,1.13125,1.59063,1.75,1.80938,1.56875,1.77813,1.6875,1.37188,0.75625,0.590625,0.125,0
.371875,0.31875,0.715625,0.8125,0.684375,0.25625,0.278125,0}

B=w+1

{1.27813,0.978125,1.42813,1.12813,0.903125,0.603125,1.05313,0.753125,1.46563,1.16563,1.61563,1.31563,1.09063,0.7906
25,1.24062,0.940625,0.840625,0.540625,0.990625,0.690625,0.465625,0.165625,0.615625,0.315625,1.02813,0.728125,1.178
12,0.878125,0.653125,0.353125,0.803125,0.503125,1.49688,1.19688,1.64688,1.34687,1.12188,0.821875,1.27187,0.971875,
1.68438,1.38438,1.83438,1.53438,1.30938,1.00938,1.45937,1.15938,1.05938,0.759375,1.20937,0.909375,0.684375,0.38437
5,0.834375,0.534375,1.24688,0.946875,1.39688,1.09688,0.871875,0.571875,1.02188,0.721875}

ListLinePlot[B,AxesOrigin->{0,0},PlotMarkers->Automatic]
```

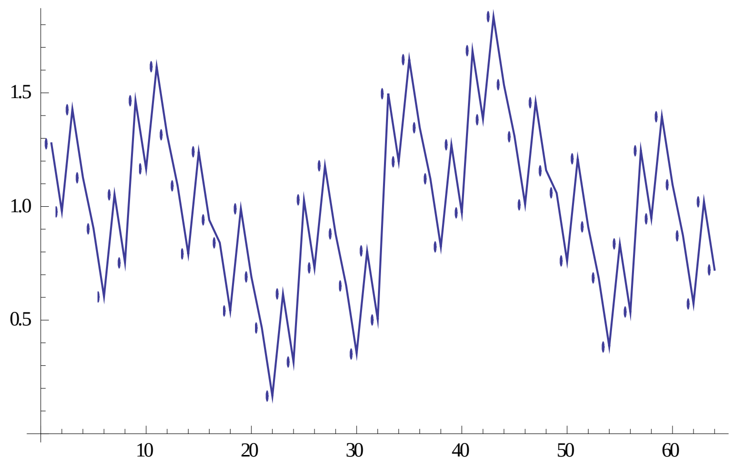


Figure 51: Figure 19: The interbeat intervals that result from doing the DFA calculations in reverse, duration *(in seconds) vs.* ith *beat. As expected, this has a resemblance to the Pink Noise image from Figure 3.*

```
Total[B]/64
 1.
t=Flatten[{0,Accumulate[B]}]
```

{0,1.27813,2.25625,3.68437,4.8125,5.71563,6.31875,7.37188,8.125,9.59063,10.7563,12.3719,13.6875,14.7781,15.5688,16.8094,17.75,18.5906,19.1313,20.1219,20.8125,21.2781,21.4438,22.0594,22.375,23.4031,24.1313,25.3094,26.1875,26.8406,27.1938,27.9969,28.5,29.9969,31.1938,32.8406,34.1875,35.3094,36.1313,37.4031,38.375,40.0594,41.4438,43.2781,44.8125,46.1219,47.1313,48.5906,49.75,50.8094,51.5688,52.7781,53.6875,54.3719,54.7563,55.5906,56.125,57.3719,58.3188,59.7156,60.8125,61.6844,62.2563,63.2781,64.}

```
ListLinePlot[t,AxesOrigin->{0,0},PlotMarkers->Automatic]
```

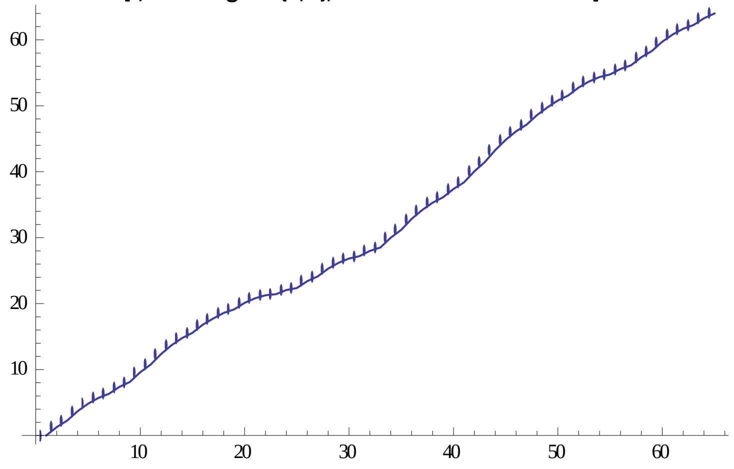


Figure 52: These are the times that result from the final step of the reverse DFA, time *(in seconds) vs.* ith *beat.*

E. Applications of Fractals to Art and Entertainment (Just a Small Sample)

1. Computerized generation of new types of music **[74]**
2. Computerized generation of new forms of art **[74]**
   a. Abstract paintings
   b. Abstract sculptures
3. Developing more efficient real-world simulations **[3] [8] [74—76]**
   a. More realistic terrain
   b. Designing environments in video games, especially organic ones
   c. Being able to generate such environments instantly, when needed, rather than having to store them long-term
4. New designs for T-shirts and new camouflage patterns **[4]**
5. Image and signal compression **[77]**
6. Communications
   a. Fractal antennas gave rise to our mobile devices **[78]**
   b. Innovating communications with fractals could lead to innovating energy as well: the same optoelectronic devices made using layers with fractal nanopatterns may prove useful in reducing the cost of solar energy **[79]**
7. Further applications are continually emerging

F. What else can fractals be used to model or understand better?

1. The full and precise structure of trees, clouds, snowflakes, seashells, complex landscapes, coastlines, river systems, mountain ranges, etc. **[2—4] [8]**

Many structures in the natural world have a fractal appearance. Trees, for instance, have large branches, which themselves have smaller branches, and smaller sub-branches, and even the leaves have similar subdivisions. The forest as a whole has a fractal appearance as well. Other parts of the natural world that are fractal in their structure include clouds, snowflakes, seashells, river systems, mountain ranges, and coastlines and other complex landscapes.

2. The weather and climate **[2 – 3] [8]**

The weather is one area in which fractals would be of immense help to better understand it. The current methods used in meteorology fail to be small-scale enough to accurately pick up the precise wind variations of hurricanes and other storms, and simply trying to just use a smaller scale to gain the necessary precision would require too much computing power while adapting the method to calmer weather. Fractals would be a great help, because they allow for a grid system of variable scale using far less computer power.

3. Genes

Fractals could advance our understanding of genes, because the field of Genetics involves a lot of information compression due to the fact that genes are very small molecules with enough complex information encoded to create an entire organism. Plus, the fatalness of a virus has been found to depend on its cell fractal geometry **[80]**, and the DFA algorithm was first developed to study DNA **[66]**.

4. Large-scale structure of the universe **[9]**

The universal structure out past the scale of galactic superclusters is not well understood, because it is not uniform and has large gaps, or voids among and between the superclusters. Structures at this scale are known as walls (along with a few even-larger great walls) of galaxies, and their dimensions are on the order of tens of millions of light years (along each axis) or larger. Walls and their respective voids are too large for current theories to deal with, but fractals could be used to describe such structures very well. Plus, since fractals can describe objects with spiral structure very well, they could be used to model and understand individual spiral galaxies and even spiral nebulae within those galaxies.

Recent reports have found that the universe itself is not fractal, but at “small” scales (“small” meaning on the scale of superclusters and walls, up to regions of 330 million light years wide) it is fractal-like. The latest study tested the composition of the universe on the scale of 3 billion light years (cubed), and found matter to be distributed uniformly (as the standard model of physics predicts, so it would’ve had to have been overhauled if the result were otherwise). The team that did the study acknowledges it could be fractal-like again at even larger scales, but the telescope meant to do such an even-larger-scale test has not been completed yet **[81]**.

5. The subatomic realm

The dimensions of spacetime may have fractal properties at quantum scales **[82]**.

6. Surface physics **[83]**

G. Current applications of nanotechnology [36]

1. Stain-repellant khakis (or other clothing eventually)
2. Better filters for removing toxic materials and viruses in water
3. Better car bodies
4. Antimicrobial dressings
5. Light harvester in photovoltaic devices
6. Increasing bioavailability of nutrients in foods
7. Cosmetics and sunscreens with improved UV protection
8. Better polymer dispersions for paints, coatings, adhesives, paper, textiles and leather
9. Rocket propellants that burn at double the rate
10. Better automotive lubricant
11. Synthetic bone
12. Producing an alloy that's as hard as diamond (for better cutting tools, drill bits, armor, and jet engine parts)
13. Thinner, lighter, more flexible, and less-power-consuming displays for cameras, phones, tablets, laptops, and TVs (and soon better-than-HDTV displays)
14. Lighter, thinner packaging
15. Tennis balls that bounce twice as long, and better tennis rackets
16. Reducing the cost of efforts to liquefy coal and turn it into gas (can also reduce pollution by removing sulfur from the coal)
17. More efficient catalytic converters
18. Stronger, lighter golf clubs
19. Smart Fabrics **[17] [23] [37—49]**

Additional emerging applications on the market can be tracked here: http://www.nanotechproject.org/cpi/

H. A Small Sample of the Future Potential of Nanotechnology

1. Space elevator **[84]**
    a. Make access to space more affordable and safer
    b. Constructed from carbon nanotubes
    c. The functionality of such a device has already been tested on a smaller scale, with a robot climber scaling a ribbon that was attached to a high altitude balloon
2. Pollution control using nanotechnology
    a. Advanced nanofilters **[36]** could be used to separate the toxic and nontoxic materials at polluted sites
    b. The large factories and manufacturing plants responsible for much of the pollution would be obsolete once all the mass production is being done by nanomachines **[15]**
    d. Nanotech paper that cleans up oil spills **[85]**
        i. Mesh of nanowires made of potassium manganese oxide
        ii. Wires have a coating that repels water, but oil can seep into the pores and thus remain in the material
        iii. Can hold 20 times its weight in oil
        iv. Potassium manganese oxide is stable at high temperatures, so paper can be cleaned by heating it to the boiling point of oil, and then the paper can be reused
3. Studying the microscopic
    a. Photographing the microscopic, such a molecule's electrical charge in action **[86]**
    b. Hearing bacteria and viruses **[87]**
4. Paper that's stronger than metal **[88]**

8.0 Sources

1. Lauwerier, Hans, and Sophia Gill-Hoffstadt (translator). *Fractals: Endlessly Repeated Geometrical Figures*. 1st ed. Princeton: University Press, 1991. Print.

2. Hastings, Harold M., and George Sugihara. *Fractals: A User's Guide for the Natural Sciences*. 1st ed. Oxford: University Press, 1994. Print.

3. Schroeder, Manfred. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. Dover ed. Mineola, N.Y.: Dover Publications, 2009. Print.

4. *Nova: Hunting the Hidden Dimension*. PBS Video, 2008. Film.

5. Mandelbrot, Benoit B. *Fractals and Chaos: The Mandelbrot Set and Beyond*. 2004 ed. New York: Springer, 2004. Print.

6. Falconer, Kenneth. *Fractal Geometry: Mathematical Foundations and Applications*. 2nd ed. Chichester: Wiley, 2003. Print.

7. Steinhurst, Benjamin A. "Notions of Dimension." *Cornell University Department of Mathematics*. 9 June 2010. Web. July 2012. <http://www.math.cornell.edu/~steinhurst/docs/dimension.pdf>.

8. Mandelbrot, Benoit B. *The Fractal Geometry of Nature*. 1st ed. New York: W.H. Freeman, 1982. Print.

9. Battersby, Stephen. "Largest Cosmic Structures 'too big' for Theories." *New Scientist*. Reed Business Information Ltd., 21 June 2011. Web. May 2012. <https://www.newscientist.com/article/dn20597-largest-cosmic-structures-too-big-for-theories/>.

10. Pickover, Clifford A. *Chaos and Fractals: A Computer Graphical Journey*. Amsterdam: Elsevier, 1998. Print.

11. Kantelhardt, Jan W., Stephan A. Zschiegner, Eva Koscielny-Bunde, Shlomo Havlin, Armin Bunde, and H. Eugene Stanley. "Multifractal Detrended Fluctuation Analysis of Nonstationary Time Series." *Physica A: Statistical Mechanics and Its Applications* 316 (2002): 87-114. Science Direct. Elsevier. Web. May 2012. <http://ac.els-cdn.com/S0378437102013833/1-s2.0-S0378437102013833-main.pdf?_tid=ec48b37a-3fc6-11e5-b8d4-00000aab0f26&acdnat=1439256110_600adf4c0cf1661a7a165fe72746bfbe>.

12. Mandelbrot, Benoit B. "How Fractals Can Explain What's Wrong with Wall Street." *Scientific American*. Scientific American (republished), 15 July 2008. Web. Aug. 2012. <http://www.scientificamerican.com/article/multifractals-explain-wall-street/>.

13. Milner, Robin, and Susan Stepney. "Nanotechnology: Computer Science Opportunities and Challenges." *University of York Department of Computer Science*. 1 Aug. 2003. Web. Oct. 2012. <http://www-users.cs.york.ac.uk/~susan/bib/ss/nonstd/rsrae03.htm>.

14. Stepney, Susan. "Nanotechnology and Complexity: Consequences for Computing." *University of York Department of Computer Science*. 1996. Web. Nov. 2011. <http://www-users.cs.york.ac.uk/susan/complex/nanotalk.htm>.

15. Kaku, Michio. *Physics of the Future: How Science Will Shape Human Destiny and Our Daily Lives by the Year 2100*. 1st ed. New York: Doubleday, 2011. Print.

16. Mallouk, Thomas E., and Ayusman Sen. "How to Build Nanotech Motors." *Scientific American* May 2009. Print. (Accessible online: <http://www.scientificamerican.com/article.cfm?id=how-to-build-nanotech-motors>).

17. Kahn, Jennifer. "Nano's Big Future: Welcome to the World of Nanotechnology." *National Geographic* June 2006. Print. (Accessible online: <http://ngm.nationalgeographic.com/2006/06/nanotechnology/kahn-text>).

18. Porter, Alan L., and Jan Youtie. "Where Does Nanotechnology Belong in the Map of Science?" *Nature Nanotechnology* 4 (2009): 534-36. *Nature Nanotechnology*. Web. May 2013. <http://www.nature.com/nnano/journal/v4/n9/full/nnano.2009.207.html>.

19. Ahlberg, Liz. "Small in Size, Big on Power: New Microbatteries the Most Powerful Yet." *Phys.org*. Science X Network,16 Apr. 2013. Web. June 2013. <http://phys.org/news/2013-04-small-size-big-power-microbatteries.html>.

20. "Kinetic Particle Theory and State Changes: Brownian Motion." *BBC News Bitesize*. BBC. Web. 13 June 2015. <http://www.bbc.co.uk/education/guides/zgr2pv4/revision/5>.

21. "Brownian Motion (Physics)." *Encyclopædia Britannica*. Online ed. 2013. Web. December 2013. <http://www.britannica.com/science/Brownian-motion>.

22. Hadhazy, Adam. "Sick Power: Viral Batteries Closer to Energizing Hybrid Cars, Cell Phones." *Scientific American Observations Blog*. 2 Apr. 2009. Web. Nov. 2012. <http://blogs.scientificamerican.com/news-blog/virus-battery-sick-power-2009-04-02/>.

23. The Royal Society & The Royal Academy of Engineering. "Nanoscience and Nanotechnologies: Opportunities and Uncertainties." Latimer Trend Ltd, 29 July 2004. Web. 31 July 2015. <http://www.nanotec.org.uk/finalReport.htm>.

24. McKibben, Bill. *Enough: Staying Human in an Engineered Age*. 1st ed. New York: Owl Books, 2004. Print.

25. Phoenix, Chris, and Eric Drexler. "Safe Exponential Manufacturing." *Nanotechnology* 15.August (2004): 869-72. *Center for Responsible Nanotechnology*. Institute of Physics. Web. 17 June 2015. <http://www.crnano.org/IOP - Safe Exp Mfg.pdf>.

26. "Team Demonstrates Quantum Dots That Assemble Themselves." *Phys.org*. Science X Network, 16 Apr. 2013. Web. July 2013. <http://phys.org/news/2013-04-team-quantum-dots.html>.

27. Williams, Jim. "The Science and Technology of Composite Materials." *Nova*. Australian Academy of Science, 18 June 2015. Web. 7 July 2015. <http://www.nova.org.au/technology-future/science-and-technology-composite-materials>.

28. Sampson, Mark, and Adam Dylewski. "Future Cities: Nanotechnology Promises More Sustainable Buildings, Bridges, and Other Structures." *Global Challenges/Chemistry Solutions*. American Chemical Society, 7 Sept. 2010. Web. May 2012. <http://www.acs.org/content/acs/en/pressroom/podcasts/globalchallenges/future/future-cities-nanotechnology-promises-more-sustainable-buildings-bridges-and-other-structures.html>.

29. Choi, Charles. "World's Lightest Solid Takes Inspiration From Eiffel Tower." *Live Science*. Yahoo! News, 18 Nov. 2011. Web. Feb. 2012. <http://news.yahoo.com/worlds-lightest-solid-takes-inspiration-eiffel-tower-134809070.html>.

30. Franzen, Carl. "How The World's Lightest Material Was Made." *Talking Points Memo*. TPM Media LLC, 21 Nov. 2011. Web. Jan. 2012. <http://talkingpointsmemo.com/idealab/how-the-world-s-lightest-material-was-made>.

31. Gardner, Bill. "The End of Potholes? UK Scientists Invent 'self-healing Concrete'" *The Telegraph*. Telegraph Media Group, 2 Dec. 2014. Web. 15 Mar. 2015. <http://www.telegraph.co.uk/news/uknews/road-and-rail-transport/11268310/The-end-of-potholes-UK-scientists-invent-self-healing-concrete.html>.

32. Greenemeier, Larry. "Could Nanotech Particles Help Treat STDs?" *Scientific American Observations Blog*. 4 May 2009. Web. June 2012. <http://blogs.scientificamerican.com/news-blog/could-nanotech-particles-help-treat-2009-05-04/>.

33. Stanford University Medical Center. "Nanoparticles home in on brain tumors, boost accuracy of surgical removal." *ScienceDaily*. ScienceDaily, 15 April 2012. Web. June 2012. <www.sciencedaily.com/releases/2012/04/120415151334.htm>.

34. Greenemeier, Larry. "Nanoparticles Enable Surgical Strikes against Cancer." *Scientific American*. Scientific American, 27 Nov. 2007. Web. June 2012. <http://www.scientificamerican.com/article/nanoparticles-nanotech-cancer-tumor/>.

35. Dillow, Clay. "Nano-ink Tattoos Could Continuously Monitor Glucose in Diabetics." *Popular Science*. Bonnier Corporation, 1 June 2010. Web. Aug. 2012. <http://www.popsci.com/science/article/2010-06/nano-ink-tattoos-could-continuously-monitor-glucose-diabetics>.

36. "Current Nanotechnology Applications." Nanotechnology Now, 20 Apr. 2015 (updated). Web. 28 May 2015. <http://www.nanotech-now.com/current-uses.htm>.

37. "Transformation Optics May Usher in a Host of Radical Advances." *AZoNano.com*. AZoNetwork, 17 Oct. 2008. Web. May 2010. <http://www.azonano.com/news.aspx?newsID=8164>.

38. Warren, Susan. "'Smart' Fabrics Function Like Appliances: They Keep Track of Vital Signs, Hide Odor." *The Wall Street Journal*. Dow Jones & Company, Inc., 10 Aug. 2001. Web. Dec. 2012. <http://www.wsj.com/articles/SB997391399491586938>.

39. "Power Suit, Meet the Power Shirt." *Science Buzz*. Science Museum of Minnesota, 15 Feb. 2008. Web. Jan. 2013. <http://www.sciencebuzz.org/blog/power_suit_meet_the_power_shirt>.

40. Pittman, David. "Power from Motion." *Chemical & Engineering News* 2 Aug. 2010: 50-51. Print. (Accessible online: <http://cen.acs.org/articles/88/i31/Power-Motion.html>).

41. Department of Energy Ames Laboratory. "Metamaterials Found to Work for Visible Light." *EurekAlert!* American Association for the Advancement of Science, 4 Jan. 2007. Web. May 2013. <http://www.eurekalert.org/pub_releases/2007-01/dl-mft010407.php>.

42. Kaku, Michio. *Physics of the Impossible*. 1st ed. New York: Doubleday, 2008. Print.

43. Franzen, Carl. "'Invisibility Cloak' Uses Transparent Carbon Nanotube Sheets." *Talking Points Memo*. TPM Media LLC, 6 Oct. 2011. Web. May 2013. <http://talkingpointsmemo.com/idealab/invisibility-cloak-uses-transparent-carbon-nanotube-sheets>.

44. "'Mirage-effect' Helps Researchers Hide Objects." *IOP*. Institute of Physics, 4 Oct. 2011. Web. May 2013. <http://www.iop.org/news/11/oct/page_52313.html>.

45. Lee, Jaeah. "Bendable Electronics Will Change Everything From Surgery To Energy." *Talking Points Memo*. TPM Media LLC, 15 July 2011. Web. Oct. 2012. <http://talkingpointsmemo.com/idealab/bendable-electronics-will-change-everything-from-surgery-to-energy>.

46. University of Michigan. "Clothing With A Brain: 'Smart Fabrics' That Monitor Health." *ScienceDaily*. ScienceDaily, 10 December 2008. Web. May 2012. <www.sciencedaily.com/releases/2008/12/081208085058.htm>.

47. Yang, Sarah. "UC Berkeley Researchers Developing Robotic Exoskeleton That Can Enhance Human Strength and Endurance." *UC Berkeley News*. UC Regents, 3 Mar. 2004. Web. Sept. 2013. <http://www.berkeley.edu/news/media/releases/2004/03/03_exo.shtml>.

48. Bland, Eric. "How to Make a Bulletproof T-shirt." *Discovery News*. Discovery Communications, LLC, 1 Apr. 2010. Web. July 2013. <http://news.discovery.com/tech/gear-and-gadgets/t-shirt-body-armor-tank.htm>.

49. Minkel, JR. "Sea-Urchin-Inspired Nanotech Gel May Make Rain Slickers Slicker." *Scientific American*. Scientific American, 25 Jan. 2007. Web. May 2013. <http://www.scientificamerican.com/article.cfm?id=sea-urchin-inspired-nanot>.

50. Greenemeier, Larry. "Carbon Nanotube Clothing Could Take Charge in an Emergency." *Scientific American*. Scientific American, 12 Dec. 2008. Web. May 2012. <http://www.scientificamerican.com/article/carbon-nanotube-clothing/>.

51. Ferris, Robert. "Super Thin And Flexible Circuits Clear The Way For Truly Wearable Computers." *Business Insider*. Business Insider, Inc, 29 July 2013. Web. Sept. 2013. <http://www.businessinsider.com/flexible-thin-electronics-breakthrough-2013-7>.

52. Yang, Sarah. "Organic Electronics Could Lead to Cheap, Wearable Medical Sensors." *Berkeley News*. UC Berkeley, 10 Dec. 2014. Web. 6 June 2015. <http://news.berkeley.edu/2014/12/10/organic-electronics-cheap-wearable-medical-sensors/>.

53. Coxworth, Ben. "Squid-inspired Tech Could Lead to Color-changing Smart Materials." *Gizmag*. Gizmag, 2 May 2012. Web. June 2013. <http://www.gizmag.com/squid-inspired-color-changing-clothes/22383/>.

54. Martin, Rebecca. "Smart Fabrics." *Catapult*. Australian Broadcasting Corporation, 2005. Web. Nov. 2012. <http://www.abc.net.au/catapult/indepth/s1435357.htm>.

55. Weiss, C. C. "Next-generation Clothing Monitors Your Heart, Tracks Your Posture and Gives You a Hug." *Gizmag*. Gizmag, 13 Mar. 2013. Web. Dec. 2013. <http://www.gizmag.com/wearable-technologies-round-up/26599/>.

56. Wood, Robert. "Heart Rate Monitors - How Do They Work?" *Topend Sports*. Topend Sports Network, 2010. Web. Oct. 2013. <http://www.topendsports.com/fitness/equip-monitors.htm>.

57. Cipra, Barry A. "A Healthy Heart Is a Fractal Heart." *SIAM News* Sept. 2003. Print. (Accessible online: <https://www.siam.org/pdf/news/353.pdf>).

58. Goldberger, A. L., Luis A. N. Amaral, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, C.-K. Peng, and H. Eugene Stanley. "Fractal Dynamics in Physiology: Alterations with Disease and Aging." *Proceedings of the National Academy of Sciences* 99 (Supplement 1, 2002): 2466-472. Print. (Accessible online: <http://www.pnas.org/content/99/suppl_1/2466.full.pdf>).

59. Riedi, Rudolf H. "Introduction to Multifractals." *Rice University Statistics Department*. 26 Oct. 1999. Web. Feb. 2013. <http://www.stat.rice.edu/~riedi/Publ/PDF/intro.pdf>.

60. Goldberger, A. L. "Non-linear Dynamics for Clinicians: Chaos Theory, Fractals, and Complexity at the Bedside." *The Lancet* 347.May (1996): 1312-314. Print. (Accessible online: <http://reylab.bidmc.harvard.edu/pubs/1996/lancet-1996-347-1312.pdf>).

61. Losa, Gabriele A. "The Fractal Geometry of Life." *Rivista Di Biologia / Biology Forum* 102 (2009): 29-60. Print. (Accessible online: <http://www.researchgate.net/publication/26778380_The_fractal_geometry_of_life>).

62. Peng, C.-K., S. V. Buldyrev, S. Havlin, M. Simons, H. E. Stanley, and A. L. Goldberger. "Mosaic Organization of DNA Nucleotides." *Physical Review E* 49.2 (1994): 1685-689. Print. (Accessible online: <http://journals.aps.org/pre/pdf/10.1103/PhysRevE.49.1685>).

63. Glass, Leon, Alvin Shrier, and Jacques Bélair. "Chaotic Cardiac Rhythms." *Chaos*. Ed. Arun V. Holden. Princeton: University Press, 1986. 237-256. Print.

64. Kamath, Chandrakar. "A New Approach to Detect Congestive Heart Failure Using Detrended Fluctuation Analysis of Electrocardiogram Signals." *Journal of Engineering Science and Technology* 10.2 (2015): 145-59. Web. 8 July 2015. <http://jestec.taylors.edu.my/Vol 10 issue 2 February 2015/Volume (10) Issue (2) 145-159.pdf>.

65. Peng, C.-K., S. Havlin, J. M. Hausdorff, J. E. Mietus, H. E. Stanley, and A. L. Goldberger. "Fractal Mechanisms and Heart Rate Dynamics: Long-range Correlations and Their Breakdown with Disease." *Journal of Electrocardiology* 28 (Supplement, 1995): 59-65. Print. (Accessible online: <http://ac.els-cdn.com/S0022073695800174/1-s2.0-S0022073695800174-main.pdf?_tid=e3c5acac-37e8-11e5-a8e4-00000aacb35e&acdnat=1438391089_85cb3050dc909518174053ffcd57c80f>).

66. Peng, C.-K., Shlomo Havlin, H. Eugene Stanley, and Ary L. Goldberger. "Quantification of Scaling Exponents and Crossover Phenomena in Nonstationary Heartbeat Time Series." *Chaos* 5.1 (1995): 82-87. Print.

67. Mandelbrot, Benoit B. "Self-affinity and fractal dimension." *Physica Scripta* 32.4 (1985): 257-260. Print.

68. Ward, Lawrence M., and Dr. Priscilla E. Greenwood. "1/f Noise." *Scholarpedia*. 29 Oct. 2007. Web. 1 Aug. 2015. <http://www.scholarpedia.org/article/1/f_noise>.

69. "How to Find a Linear Regression Equation: Overview." *Statistics How To*. Statistics How To, 2009. Web. Oct. 2012. <http://www.statisticshowto.com/how-to-find-a-linear-regression-equation/>.

70. Frame, Michael, Benoit Mandelbrot, and Nial Neger. "Fractal Geometry." *Yale University*. 2004. Web. May 2012. <https://classes.yale.edu/fractals/>.

71. Franzen, Carl. "MIT's Light Speed Camera Captures Photons Moving." *Talking Points Memo*. TPM Media LLC, 14 Dec. 2011. Web. Sept. 2013. <http://talkingpointsmemo.com/idealab/mit-s-light-speed-camera-captures-photons-moving>.

72. Hardesty, Larry. "Trillion-frame-per-second Video." *MIT News*. Massachusetts Institute of Technology, 13 Dec. 2011. Web. Sept. 2013. <http://newsoffice.mit.edu/2011/trillion-fps-camera-1213>.

73. Moskvitch, Katia. "Artificial Blood Vessels Created on a 3D Printer." *BBC News*. BBC, 16 Sept. 2011. Web. July 2013. <http://www.bbc.co.uk/news/mobile/technology-14946808>.

74. Wilson, Stephen. *Information Arts: Intersections of Art, Science, and Technology*. 1st ed. Cambridge, Mass.: MIT, 2002. Print.

75. Martz, Paul. "Generating Random Fractal Terrain." *GameProgrammer.com*. 1996. Web. Apr. 2010. <http://gameprogrammer.com/fractal.html>.

76. "Fractals and Artificial Intelligence." *Herself's Artificial Intelligence*. TimesToCome, 2008. Web. July 2010. <http://www.herselfsai.com/2008/01/fractals-and-artificial-intelligence.html>.

77. Davis, Frederic E. "My Main Squeeze: Fractal Compression." *Wired* 1.05 Nov. 1993. Print. (Accessible online: <http://archive.wired.com/wired/archive/1.05/fractal.html>).

78. Felber, Philip. "Fractal Antennas." *Illinois Institute of Technology*. 16 Jan. 2001 (revised). Web. Sept. 2011. <http://www.ece.iit.edu/~pfelber/fractalantennas.pdf>.

79. Murphy, Helene. "Fractals on New Mountbatten Building Inspired by Nanotechnology Research." *Information Technology: Forum for Science, Industry and Business*. Innovations-report, 12 Nov. 2007. Web. Feb. 2013. <http://www.innovations-report.com//html/reports/information-technology/report-99872.html>.

80. He, Ji-Huan. "Fatalness of Virus Depends upon Its Cell Fractal Geometry." Chaos, Solitons and Fractals 38.5 (2008): 1390-1393. Print.

81. Slezak, Michael. "Giant Fractals Are out – the Universe Is a Big Smoothie." *New Scientist*. Reed Business Information Ltd. (expanded), 24 Aug. 2012. Web. 7 Apr. 2015. <https://www.newscientist.com/article/dn22214-giant-fractals-are-out--the-universe-is-a-big-smoothie/>.

82. Zyga, Lisa. "Spacetime May Have Fractal Properties on a Quantum Scale." *Phys.org*. Science X Network, 25 Mar. 2009. Web. Jan. 2011. <http://phys.org/news/2009-03-spacetime-fractal-properties-quantum-scale.html>.

83. Wautelet, Philippe. "Applications of Fractals." *FractalZone*. 2007 (revised). Web. Oct. 2010. <http://www.fractalzone.be/applications.php>.

84. David, Leonard. "Space Elevator Concept Undergoes 'Reel' World Testing." *Space.com*. Purch, Inc., 23 Sept. 2005. Web. Feb. 2012. <http://www.space.com/businesstechnology/050923_spaceelevator_test.html>.

85. Bland, Eric. "Nanopaper Soaks up Oily Spills." *ABC Science*. Australian Broadcasting Corporation, 2 June 2008. Web. May 2012. <http://www.abc.net.au/science/articles/2008/06/02/2262281.htm>.

86. Franzen, Carl. "IBM Produces First-Ever Image Of A Molecule's Electrical Charge Distribution." *Talking Points Memo*. TPM Media LLC, 27 Feb. 2012. Web. Nov. 2012. <http://talkingpointsmemo.com/idealab/ibm-produces-first-ever-image-of-a-molecule-s-electrical-charge-distribution>.

87. Wilkins, Alasdair. "Ultra-tiny 'nano-ear' Can Hear Bacteria and Viruses." *io9*. Gawker Media, 11 Mar. 2012. Web. June 2013. <http://io9.com/5876669/ultra-tiny-nano-ear-can-hear-bacteria-and-viruses>.

88. "Could Stronger, Tougher Paper Replace Metal?" *Phys.org*. Science X Network, 24 July 2015. Web. 30 July 2015. <http://phys.org/news/2015-07-stronger-tougher-paper-metal.html#nRlv>.