4-20-2022

A Fractal Geometry for Hydrodynamics

Jonah Mears
Ouachita Baptist University

A Fractal Geometry for Hydrodynamics

Jonah Mears

An Honors Thesis for Ouachita Baptist University's Carl Goodson Honors Program

Table of Contents

Experiments have shown that objects with uneven surfaces, such as golf balls, can have less drag than those with smooth surfaces. Since fractal surfaces appear naturally in other areas, it must be asked if they can produce less drag than a traditional surface and save energy. Little or no research has been conducted so far on this question. The purpose of this project is to see if fractal geometry can improve boat hull design by producing a hull with low friction.

The term *fractal* refers to a type of geometric figure with self-similarity and fractal dimension. Self-similarity refers to the fact that no matter how much a fractal is magnified, it will still look the same (from statistically similar for completely random fractals to exactly the same for the more organized fractals). Consider the Sierpinski gasket (Figure 1). It can be regarded as a solid triangle with infinitely many triangular holes cut out. The central hole divides it into three smaller copies of itself. Each of the copies has its own central hole, dividing each copy into still smaller copies.

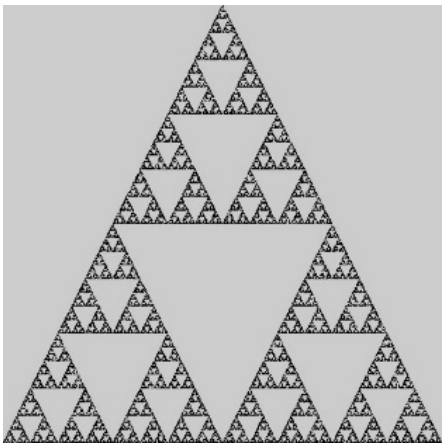

Figure 1

The term *fractal dimension* refers to the non-integer dimensionality of fractals. For example, a fractal in two-dimensional space can be a 1.75-dimensional object. This is because fractals are made of solid shapes with infinitely many holes cut out or infinitely many points and, therefore, do not have as much volume (for three-dimensional fractals) or area (for two-dimensional fractals) as Euclidean shapes and solids with integer dimensionality have.

Hydrodynamics is the study of the flow of water. If an object has the proper geometry, water can flow around it with low resistance. The goal of this thesis is to determine if fractal geometry has a positive impact on the flow of water around a boat hull.

Fractal surfaces were chosen because they appear often in nature. For example, tree branches are approximate copies of the entire tree, the smaller branches on the main branches are further copies, and so on. Clouds and mountains have near-infinite levels of detail and are self-similar in such a way that pictures of small portions of clouds are similar to large portions (as long as the pictures are not too large or too small). Even the romanesco (the vegetable in Figure2) has self-similarity.

Figure 2

In this project, a system for 3D printing and testing hull models was implemented. The test apparatus was a long tank of water with a counterweight system for pulling the models through the water (Figure 9 on page 8). Several hulls were designed with different modifications to a basic design. This work focused on designing and testing boat hulls with the application of fractal geometry and hydrodynamics and analyzing data found from testing the hulls under similar conditions.

The boat hull designs were graphs of elliptic paraboloids (Figure 3) or solids formed by revolving spline curves. They were written as STL files and constructed by a 3D printer. Two boat hulls were solids of revolution formed by rotating spline curves in Autodesk Inventor, but the rest were elliptic paraboloids. Since fractals have infinite complexity and any approximations must be suitably detailed, normal boat modeling methods were ruled out. Since there is a 3D printer available at the Computer Science lab at OBU, the models were 3D printed. Since Autodesk Inventor, the available modeling system, was not programmable, Python programs were used to write the STL files (see Appendix I). The system used had to be programmable to avoid manual input of the high levels of detail.

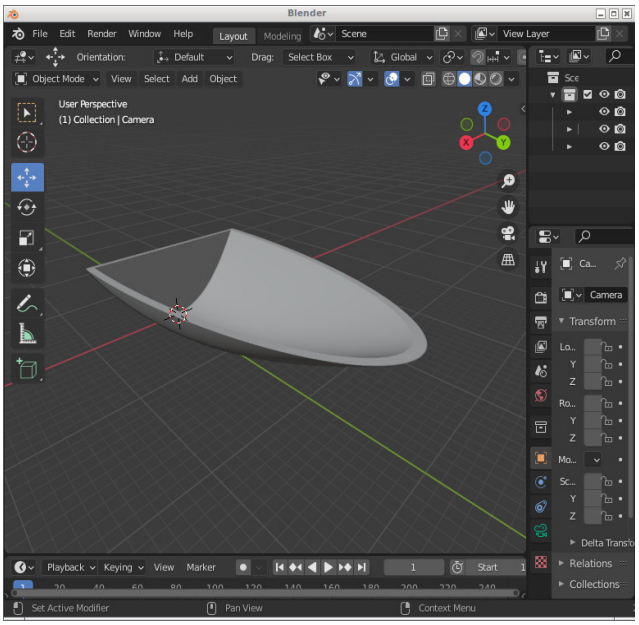

Figure 3

The Python programs used to write the STL files took the x and y coordinates of points within the hull, used the equation of an elliptic paraboloid to find the z coordinate, and stored the coordinates in the output file. The points along the edge of the hull were problematic since they did not coincide exactly with the step size used and had to be interpolated. A potentially better approach would be to take a series of layers: take a two-dimensional slice of the paraboloid and use the x and z coordinates from the edge of the slice to find the y coordinate. This method

would have eliminated the need to interpolate the points on the top layer. Each hull was stored as a list of points in the form of individual triangles along with their normal vectors (since some computer rendering programs' shading algorithms require the normal vectors). The triangles connect together to form the surface.

The Chaos Game algorithm is the method used to approximate the fractals for the boat hulls. To play the game (or run the algorithm), one needs a system of at least three linear equations, a starting point, and part of a two-dimensional surface. Let one of the equations be expressed by

$$f(x,y) = (ax + by + c, dx + gy + h), \quad (1)$$

where a and b are both less than 0.5 and all the constants are real numbers. Choose the starting point, pick one of the equations randomly, and substitute in the starting point. Then pick another equation and substitute in the last function's output. After enough of these iterations, enough points on the surface will be found for a graph of a fractal. The Sierpinski gasket illustration (Figure 1) was drawn using this algorithm in Dr. Jeff Matocha's Programming I lab at OBU. For a good reference on the algorithm (and an old Basic version of it), see Barnsley's *The Desktop Fractal Design Handbook*.

Since fractals are self-similar, if a fractal is copied and shrunk, moved, or rotated in the right way, it will be unchanged. The parts are copies of the whole, so they can be substituted for the whole. Because linear equations can be found to perform this process, it is possible to start with a single point and start iterating it with the equations. If the point is substituted into all the equations and the results are substituted into the equations many times, the final result will be a

set of points that, when plotted, will form a picture of the fractal. Choosing a starting point and picking a random function for every iteration provides the same effect if done long enough. (Peitgen, 1992, 258-342).

The first test hull, a spline curve rotated about an axis, was designed with Autodesk Inventor. The second was from the same program, but it had holes modeled through it and had the curve re-extruded with a smaller radius to close them. See Figure 4 for those hulls. The next hull was an elliptic paraboloid from a Python program, as described before (see Figures 3, 5, and 9). It and the smooth spline hull were used as a control group. For the noise hull with the hole in the bow (right hull in Figure 6; the string is tied through the hole), points on the hull's exterior were moved outward by

Figure 4

Figure 5

Figure 6

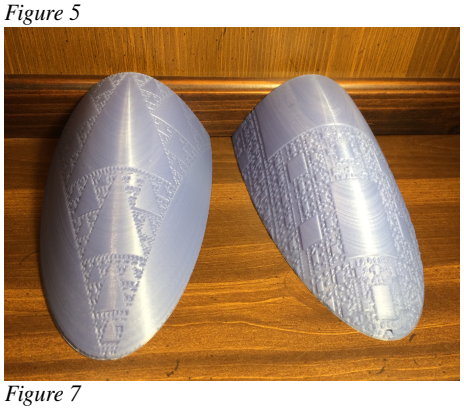

Figure 7

scaled random amounts. For the hull on the left, each point was randomly moved inward by a set amount or left as it was on the original. The resulting pattern on the hulls in Figure 6 is called random noise. The left hull in Figure 7 has the Sierpinski gasket. Its program used a graph of the gasket from the Chaos Game algorithm to decide which points on the hull to move outwards. The right hull followed a similar method with the Sierpinski carpet (Figure 8). All methods, however, left the stern flat. The spline hulls had a stern radius of 5.3 centimeters, while the other hulls had stern widths ranging from 12.3 to 13.2 centimeters and heights ranging from 6.2 to 6.6 centimeters.

Figure 9

To test the boat hulls, an 8-foot (about 2.4 meters) towing tank was built out of plastic sheeting, PVC pipes, staples, painter's tape, and a used airplane cargo strap (Figure 9). The pipes, not being glued together, could be disassembled for storage. The tank was half filled with water since that amount gave the hulls free movement without bursting the tank. The hulls were attached to a piece of kite string which ran through a pulley, up to another pulley on a

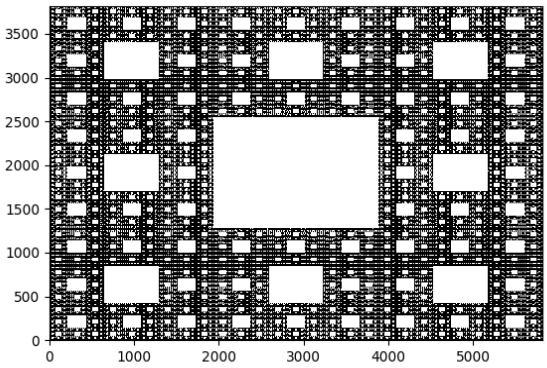

Figure 8

ladder, and down to a 50-gram Beck hanger (a hanger for weights used in many undergraduate physics lab activities).

To make sure that the boats were deep enough in the water, each was ballasted with the same bag of marbles. When the hanger was dropped from the pulley on the ladder, it pulled the boat through the water in the tank. The pulley closer to the ground had a photogate, which was attached to a laptop with PASCO Capstone, a data-taking computer program, installed. The program monitored the motion of the pulley to record the speed and acceleration of each boat hull.

The speeds were recorded by PASCO Capstone in a CSV file, a kind of text-based spreadsheet file, and analyzed with the help of simple Python programs (see Appendix II for data). The speeds and accelerations were plotted with respect to time, as shown in Figure 10, for the first trial with the parabolic hull. The speed was averaged from when the boat reached top

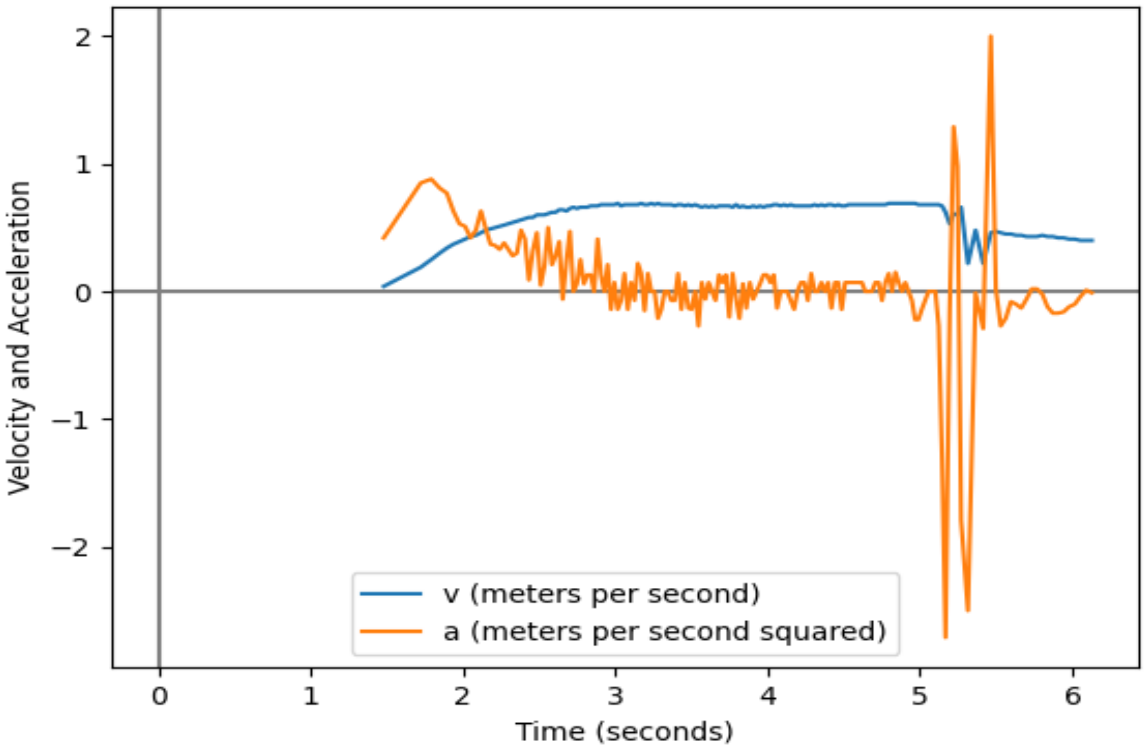

Figure 10

speed until just before the acceleration started spiking (due to the boat hitting the pulley). Of course, sometimes the boat hit the pulley after the program stopped taking the data. In those cases, the data was averaged from the time the boat reached top speed to before the data capture stopped (to make certain the data from after the collision was left out). The speed data for the hulls is displayed in Table I. Each entry shows the average speed for a trial (in meters per second).

Table 1: Average speeds (in meters per second) for the various boat hulls.

Simple Parabola	Simple Spline	Spline With Holes	Noise Hull (With Hole)	Noise Hull (No Hole)	Sierpinski Carpet	Sierpinski Gasket
0.6761	0.6996	0.6967	0.6783	0.6671	0.6811	0.6746
0.6754	0.7046	0.7063	0.6720	0.6547	0.6856	0.6736
0.3493	0.6978	0.6864	0.6652	0.6625	0.6631	0.6732
0.6762	0.7015	0.6948	0.6824	0.6560	0.6927	0.6925
0.6831	0.6885	0.6843	0.6609	0.6589	0.6751	0.6876
0.6718	0.7101					
0.5975	0.7283					
0.6998						
Average: 0.6804	Average: 0.7044	Average: 0.6937	Average: 0.6717	Average: 0.6598	Average: 0.6795	Average: 0.6803

Judging by this data, the simple spline hull went the fastest. This is to be expected since it was the most streamlined. The Sierpinski gasket hull seems to have performed better than the rest of the rough hulls and approximately as well as the smooth parabolic hull.

However, due to inconsistencies in the 3D printer's settings, the boat hulls differ in volume, making it hard to compare them based on speed alone. To avoid this problem, it is helpful to look at the drag coefficients for the hulls. The equation for the drag coefficient is, according to Bruce R. Munson, Donald F. Young, and Theodore H. Okiishi's *Fundamentals of Fluid Mechanics,*

$$C_n = \frac{2\mathcal{D}}{\rho U^2 A}, \tag{2}$$

where $\mathcal{D}$ is the drag force (the weight of the Beck hanger since the drag balances it due to the constant speed), ρ is the density of water (999 kg/m^3 at 15.6°C) (according to pages 22 and 14 of *Fundamentals of Fluid Mechanics*), U is the velocity of the boat relative to the water, and A is the area of the stern. (Other areas of the hull can be used, but the stern's area was chosen since it was the easiest to measure.) The computed drag coefficients are as follows:

Table 2: Drag coefficients for the boat hulls.

Simple Parabola	Simple Spline	Spline With Holes	Noise Hull (With Hole)	Noise Hull (No Hole)	Sierpinski Carpet	Sierpinski Gasket
0.4825	0.4665	0.4809	0.4573	0.5067	0.4817	0.4806

The noise hull with the hole has the lowest drag coefficient, indicating that it has the least drag. The Sierpinski gasket and carpet hulls slightly outperform the simple parabolic hull, but not the simple spline hull. The difference between the gasket hull's drag coefficient and that of the simple parabolic hull is 0.019, however, indicating that this may be due to errors in measurements. The noise hull that has the hole seems to have the least drag, but its vertices have been extended outward by random increments instead of a fixed step, making it smoother on average than the other rough hulls. Additionally, unlike the noise hulls, the fractal hulls have large smooth areas. The data may mean that fractal hulls are not as good or that these are the wrong fractals, but the sources of error are large enough that no valid conclusion can be drawn. Maybe chaos theory can help generate a better fractal for drag reduction.

The friction in the pulleys and the weight of the string were neglected. This extra force might have been enough that the drag force was not exactly equal to the weight of the hanger. Additionally, when the boats hit the pulley, it sometimes became crooked, possibly misleading the photogate and adding extra friction. The PASCO Capstone program displayed the data to three decimal places, indicating possible uncertainty in the velocity of ±0.005 m/s, but it may display more than one uncertain digit and the error itself may be greater. The areas of the hulls' sterns were found with equations for the areas of circles and parabolas. The dimensions were measured with a ruler with ±0.5mm accuracy, but the error may be larger since the hulls, due to errors in the printing process, might be somewhat distorted and therefore not perfectly parabolic or semicircular in the stern. Additionally, each boat's area is probably off by about 10% since the boat hulls were not completely submerged in the tank. This problem, however, might be

prevented in future experiments by measuring the height of the stern starting at its waterline, and it might not be much of a problem in these experiments since some hulls, due to their lift and the pull of the string, tilted enough that the stern was almost submerged.

This project studied the impact that different surface textures have on the amount of friction produced between a boat hull and the water. Data was collected on the performance of various hulls in a towing tank. Initial results indicate that fractal geometry may improve boat hulls' performance since the fractal hulls had lower drag coefficients than the smooth hulls. The hulls with noise over the entire surface (except the stern) had even lower drag coefficients. These results seem to show that fractal geometry and random noise have a positive impact on boat hull design, but the sources of error were large enough to prevent a definite decision. Additional work can be done to minimize the sources of error to better determine how fractal geometry will reduce boat hull friction.

Sources Cited

Michael F. Barnsley. *The Desktop Fractal Design System.* San Diego: Academic Press, 1989.

Heinz-Otto Peitgen and Dietmar Saupe, ed. *The Science of Fractal Images.* New York: Springer-Verlag New York Inc., 1988.

Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. *Fractals for the Classroom.* New York: Springer-Verlag New York, Inc., 1992.

Bruce R. Munson, Donald F. Young, and Theodore H. Okiishi. *Fundamentals of Fluid Mechanics.* New York: John Wiley and Sons, Inc. 1990.

T. E. Faber. *Fluid Dynamics for Physicists.* Cambridge: Cambridge University Press, 1995.

Steven H. Strogatz. Nonlinear Dynamics and Chaos. Boca Raton: CRC Press, an imprint of the Taylor and Francis Group. 2015.

Jamis Buck. *The Ray Tracer Challenge.* Raleigh: The Pragmatic Programmers, LLC. 2019.

It tells how to write some of the functions in rtfunctions.py, originally intended for a ray tracer.

The graph and the Sierpinski carpet illustration were plotted in matplotlib, a Python library, found at matplotlib.org.

Romanesco picture: By Ivar Leidus - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=100133434

Appendix I: Computer Code

rt_functions.py

(A directory the boat hull programs import)

```
"""Functions and other code that may come in handy for ray tracing"""

#Import the sine and cosine functions
from math import cos
from math import sin

import random

def point(x, y, z):
	"""Takes in xyz coordinates, puts them into a tuple, and adds 1 to the end."""
	pt = (x, y, z, 1)
	return pt

def vector(x, y, z):
	"""Takes in xyz components, puts them into a tuple, and adds 0 to the end."""
	vector = (x, y, z, 0)
	return vector

def aboutEqual(a, b, epsilon):
	"""Determines if two lists or tuples are almost the same"""
	#The text suggests that one input 0.00001 for epsilon.
	if len(a) != len(b):
		check = False
	else:
		i = 0
		for item in a:
			if abs(item - b[i]) <= epsilon:
				check = True
			else:
				check = False
				break
```

```
                        i = i + 1
        return check

def add(a, b):
        """Adds two tuples together"""
        i = 0
        c = []
        for item in a:
                c.append(a[i]+b[i])
                i = i + 1
        c = tuple(c)
        return c

def sbtct(a, b):
        """Subtracts tuples"""
        i = 0
        c = []
        for item in a:
                c.append(a[i]-b[i])
                i = i + 1
        c = tuple(c)
        return c

def negate(a):
        """Reverses the signs of each member of a tuple"""
        i = 0
        b = []
        for item in a:
                b.append(0 - a[i])
                i = i + 1
        b = tuple(b)
        return b

def multiply(a, b):
        """Multiplies each element in a tuple by the number b."""
        c = []
```

```
        for item in a:
                c.append(item*b)
        c = tuple(c)
        return(c)

def div(a, b):
        """Divides each element in a tuple by the number b."""
        c = []
        for item in a:
                c.append(item/b)
        c = tuple(c)
        return(c)

def v_mag(a):
        """Finds the magnitude of a vector"""
        sum = 0
        for item in a:
                sum = sum + item**2
        m = sum**0.5
        return(m)

def norm(a):
        """Normalizes vectors (finds their unit vectors)."""
        m = v_mag(a)

        n = []
        for item in a:
                if m == 0:
                        print("I don't like this vector:")
                        print(a);
                        n.append(-2000000)
                else:
                        n.append(item/m)
        n = tuple(n)
        return n
```

```
def dot(tuple_a, tuple_b):
	"""Takes the dot product of two vectors."""
	product = 0
	count = 0
	for a in tuple_a:
		product = product + a * tuple_b[count]
		count = count + 1
	return product

def cross(a, b):
	"""Takes the cross product of two vectors."""
	#If v = vector(x, y, z), where the variables are really numbers, then
	#x = v[0], y = v[1], and z = v[2].
	return(vector(a[1]*b[2]-a[2]*b[1], a[2]*b[0]-a[0]*b[2], a[0]*b[1]-a[1]*b[0]))

def color(r, g, b):
	return((r, g, b))

def hdm(a, b):
	"""Takes the Hadamard or Schur product of two color tuples"""
	c = []
	count = 0
	for item in a:
		c.append(item * b[count])
		count += 1
	return tuple(c)

class Canvas():
	"""A method of storing an image"""
	def __init__(self, width, height):
		self.width = width
		self.height = height
		#A list as long as the height with each member set to None.
		self.table = [None] * height
		#Fits lists into the previous list.
		for y in range(height):
```

```
                self.table[y] = [None] * width
        for y in range(height):
                for x in range(width):
                        self.table[y][x] = (0, 0, 0)

def write_pixel(self, x, y, shade):
        """Sets the color of a certain pixel on the canvas."""
        if x < self.width and x >= 0 and y < self.height and y >= 0:
                self.table[y][x] = shade
                #x and y go up to width - 1 and height - 1, respectively.

def read_pixel(self, x, y):
        """Returns the color of the specified pixel."""
        return(self.table[y][x])

def ppm_write(self):
        """Returns the canvas as text that can be written to a PPM file."""
        ppm = "P3\n" + str(self.width) + " " + str(self.height) + "\n255\n"
        #A list of color tuple indices to loop through.
        indices = [0, 1, 2]
        for y in range(self.height):
                i = 0
                for x in range(self.width):
                        for z in indices:
                                n = self.table[y][x][z]
                                if n < 0:
                                        n = 0
                                elif n > 1:
                                        n = 255
                                else:
                                        n = int(n*255)

                                #Ensure that the number of characters in a line
                                #does not exceed 70.
                                length = len(str(n))
                                if i + length >= 70:
```

```
                                        ppm = ppm.rstrip() + "\n" + str(n) + " "
                                        i = len(str(n)) + 1
                                else:
                                        ppm = ppm + str(n) + " "
                                        i += len(str(n)) + 1
                        if ppm[len(ppm)-1] != "\n" or ppm[len(ppm)-1] == " \n":
                                ppm = ppm.rstrip() + "\n"
                filename = "image.txt"
                with open(filename, 'w') as file_object:
                        file_object.write(ppm)

#Notes on matrices
#Just use a 2D list as a matrix.  For example:
#[[0, 0], [1, 1]] for
#|0 0|
#|1 1|
#matrix[row number][column number] or matrix[y][x]

def matrix_write(height, width, elements):
        matrix = []
        for y in range(height):
                matrix.append([])
        i = 0
        for y in range(height):
                for x in range(width):
                        matrix[y].append(elements[i])
                        i += 1
        return matrix

def matrix_equal(m1, m2):
        y = 0
        l1 = len(m1) - 1
        w1 = len(m1[0]) - 1
        l2 = len(m2) - 1
        w2 = len(m2[0]) - 1
        if l1 == l2 and w1 == w2:
```

```
                answer = True
                while y < l1:
                        if aboutEqual(m1[y], m2[y], 0.00001) == False:
                                answer = False
                                break
                        y += 1


        else:
                answer = False
        return answer


def matrix_multiply(a, b):
        """This works for two 4x4 matrixes or a matrix and a tuple."""
        #See if both are matrixes.
        if isinstance(a, list) and isinstance(b, list):
                m = [[], [], [], []]
                for y in range(4):
                        for x in range(4):
                                m[y].append(a[y][0]*b[0][x]
                                +a[y][1]*b[1][x]
                                +a[y][2]*b[2][x]
                                +a[y][3]*b[3][x])


        #See if a is a tuple.
        elif isinstance(a, tuple):
                m = []
                for y in range(4):
                        m.append(b[y][0]*a[0]+b[y][1]*a[1]+b[y][2]*a[2]+b[y][3]*a[3])
                m = tuple(m)
        #Assume b is a tuple.
        else:
                m = []
                for y in range(4):
                        m.append(a[y][0]*b[0]+a[y][1]*b[1]+a[y][2]*b[2]+a[y][3]*b[3])
                m = tuple(m)
        return m
```

```
id_matrix = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]

def matrix_transpose(m):
	"""Transposes matrices.  Is this comment even necessary?"""
	height = len(m)
	width = len(m[0])
	m1 = []
	for x in range(width):
		m1.append([])
	for x in range(width):
		for y in range(height):
			m1[x].append(m[y][x])
	return m1

def determinant(m):
	"""Finds the determinant of matrices."""
	height = len(m)
	width = len(m[0])
	if height == 2 and width == 2:
		d = m[0][0]*m[1][1] - m[0][1]*m[1][0]
	else:
		d = 0
		for y in range(height):
			d += m[y][0] * cofactor(m, y, 0)
	return d

def submatrix(a, row, column):
	h1 = len(a)
	w1 = len(a[0])
	b = []
	for y in range(h1 - 1):
		b.append([])
	#i = 0
	j = 0
	for y in range(h1):
```

```
                if y != row:
                        for x in range(w1):
                                if x != column:
                                        b[j].append(a[y][x])
                                        #i += 1
                        j += 1
        return b

def minor(a, row, column):
        return determinant(submatrix(a, row, column))

def cofactor(a, row, column):
        m = determinant(submatrix(a, row, column))
        if (row + column)%2 == 1:
                m = -1 * m
        return m

def is_invertible(m):
        if determinant(m) != 0:
                verdict = True
        else:
                verdict = False
        return verdict

def invert(a):
        if is_invertible:
                height = len(a)
                width = len(a[0])
                dtrmnt = determinant(a)
                b = [None] * height
                for y in range(height):
                        b[y] = [None] * width
                for y in range(height):
                        for x in range(width):
                                c = cofactor(a, y, x)
                                b[x][y] = c/dtrmnt
```

```
        else:
                b = False
        return b

def transform(x, y, z):
        """Converts a tuple to a transformation matrix"""
        transform = [[1, 0, 0, x], [0, 1, 0, y], [0, 0, 1, z], [0, 0, 0, 1]]
        return transform

def scaling(x, y, z):
        return [[x, 0, 0, 0], [0, y, 0, 0], [0, 0, z, 0], [0, 0, 0, 1]]

def rotation_x(theta):
        return [[1, 0, 0, 0],
                [0, cos(theta), -1*sin(theta), 0],
                [0, sin(theta), cos(theta), 0],
                [0, 0, 0, 1]]

def rotation_y(theta):
        return [[cos(theta), 0, sin(theta), 0],
                [0, 1, 0, 0],
                [-1*sin(theta), 0, cos(theta), 0],
                [0, 0, 0, 1]]

def rotation_z(theta):
        return [[cos(theta), -1*sin(theta), 0, 0],
                [sin(theta), cos(theta), 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1]]

def find_normal(triangle):
        """Finds the normal vector of a triangle"""
        n = norm(cross(sbtct(triangle[0], triangle[1]), sbtct(triangle[2], triangle[1])))
        if n[3] != 0:
                print(triangle)
        return n
```

```
class Model3D:
	"""A class to keep up with 3D models made up of triangles"""

	def __init__(self):
		self.triangles = []
		self.vertices = []

	def add_triangle(self, a, b, c):
		"""Adds a triangle.  Be careful not to add duplicate triangles."""
		self.triangles.append([a, b, c])
		#a, b, and c are points in 3D space
		#include the fourth component, 1

	def add_vertex(self, a):
		"""Adds a vertex.  Be careful not to add duplicates."""
		self.vertices.append(a)
		#a should be an index in the triangles array

	def write(self, filename):
		"""Writes the model to an STL file."""
		with open(filename, 'w') as file_object:
			file_object.write("solid CATIA STL\n")

			i = 0;
			for triangle in self.triangles:
				file_object.write("  facet normal")
				norm = find_normal(triangle);
				if norm[3] != 0:
					print("Position in array:")
					print(i)
				file_object.write("  "+str(norm[0])+"  "+str(norm[1])+"
"+str(norm[2])+"\n")
				file_object.write("    outer loop\n")
				#for vertex in triangle:
				for vertex in range(0, 3):
```

```
                                        file_object.write("      vertex  "+str(triangle[vertex][0])
+"  "+str(triangle[vertex][1])+"  "+str(triangle[vertex][2])+"\n")


                                file_object.write("    endloop\n  endfacet\n")
                                i += 1
                        file_object.write("endsolid CATIA STL\n")


        def write_0(self, filename):
                """Writes the model to an STL file.
                Written to deal with extra counters (usually zeros) in the facets array.
                """
                with open(filename, 'w') as file_object:
                        file_object.write("solid CATIA STL\n")


                        i = 0;
                        for triangle in self.triangles:
                                #Remove the counters
                                triangle = (triangle[0], triangle[2], triangle[4])
                                file_object.write("  facet normal")
                                norm = find_normal(triangle);
                                if norm[3] != 0:
                                        print("Position in array:")
                                        print(i)
                                file_object.write("  "+str(norm[0])+"  "+str(norm[1])+"
"+str(norm[2])+"\n")
                                file_object.write("    outer loop\n")
                                #for vertex in triangle:
                                for vertex in range(0, 3):
                                        file_object.write("      vertex  "+str(triangle[vertex][0])
+"  "+str(triangle[vertex][1])+"  "+str(triangle[vertex][2])+"\n")


                                file_object.write("    endloop\n  endfacet\n")
                                i += 1
                        file_object.write("endsolid CATIA STL\n")


def ifs(code, prob, width, height):
        """IFS algorithm taken from The Desktop Fractal Design Handbook by
```

```
Michael F. Barnsley"""

#number of iterations
n = 10000000
#width = 1000
#height = 1000
x = width
y = 0

def round(x):
        """Rounds the input to the nearest integer."""
        x1 = int(x)
        if x > 0:
                if ((x1 + 0.5) <= x):
                        x1 = x1 + 1
        elif x < 0:
                if ((x1 + 0.5) >= x):
                        x1 = x1 - 1
        return x1

attractor = [None]*(height + 1)
#len(attractor) is now equal to height + 1
#and the last member has index height
for j in range(height + 1):
        attractor[j] = [1]*(width + 1)

#Find the pixels that are part of the fractal, and color them black
for i in range(n):
        r = random.randint(0, len(code) - 1)
        x = code[r][0]*x + code[r][1]*y + code[r][4]
        y = code[r][2]*x + code[r][3]*y + code[r][5]
        x = round(x)
        y = round(y)
        attractor[y][x] = 0

return attractor
```

keelBoat2.py

(Simple Parabolic Hull)

```
from rt_functions import *
#This is called keelBoat since a keel was originally planned.

upper_bound = 8
step = 0.1

def increment(a):
	"""
	Find the next point for the vertex.
	Add step to the x-component and find the new z-component.
	"""
	return (a[0] + step, a[1], .1*(a[0] + step)**2 + .01*(a[1])**2, 1)

def increment2(a):
	"""Increment function for the inner hull"""
	return (a[0] + step, a[1], m*(a[0] + step)**2 + n*(a[1])**2 + w, 1)

def interpolate(p):
	"""
	Finds the proper x-coordinate or y-coordinate for point p.
	It assumes y is correct and finds x.
	If y is not on the surface for the maximum z-value,
	it assumes x is correct and finds a new value for y.
	"""

	y = p[1]
	x = (80 - 0.1*y**2)**0.5
	if p[0] < 0:
		x = x*(-1)

	#If that gives a complex number, try this:
	if type(x) == complex:
```

```
                x = p[0]
                y = (800 - 10*x**2)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def interpolate2(p):
        """Interpolate function for the inner hull"""

        y = p[1]
        x = ((upper_bound - w - n*y**2)/m)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = ((upper_bound - w - m*x**2)/n)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def transform(increment, interpolate):
        """Finds the xyz coordinates for the next two triangles."""
        global a
        global b
        global c
        global d

        global c1

        aUsed = True
        bUsed = True
        cUsed = True
```

```
        dUsed = True


        a = increment(a)
        b = increment(b)
        c = increment(c)
        d = increment(d)


        #Check  to see if all the z-coordinates are out of bounds
        if a[2] > upper_bound and b[2] > upper_bound and c[2] > upper_bound and d[2] >
upper_bound:
                #Since this pair of triangles is out of bounds,
                #do not add it to the hull.
                aUsed = False
                bUsed = False
                cUsed = False
                dUsed = False


        #See if a and c are both out of bounds
        elif a[2] > upper_bound and c[2] > upper_bound:
                #Fix a
                a1 = interpolate(a)


                #See if b and d need to be fixed
                if b[2] > upper_bound:
                        b1 = interpolate(b)
                else:
                        b1 = b
                if d[2] > upper_bound:
                        d1 = interpolate(d)
                else:
                        d1 = d


                facets.add_triangle(a1, b1, c1)


                #See if it is far enough to the stern to need extra triangles
                c1 = interpolate(c)
```

```
        if a1[0] != c1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, c1, d1)
        else:
                dUsed = False

#See if b and d are both out of bounds
elif b[2] > upper_bound and d[2] > upper_bound:

        b1 = interpolate(b)

        #Fix a and c if they need it
        if a[2] > upper_bound:
                a1 = interpolate(a)
        else:
                a1 = a

        if c[2] > upper_bound:
                c1 = interpolate(c)
        else:
                c1 = c

        #Check for "whiskers"
        if b1[1] < a1[1]:
                #If b1 goes too far when being interpolated,
                #move it to the other side of the square.
                b1 = (b[0] + step, b[1], b[2], 1)
                b1 = interpolate(b1)
        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the bow to need extra triangles
        d1 = interpolate(d)
        if b1[0] != d1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, c1, d1)
        else:
                dUsed = False
```

```
else:
        a1 = a
        b1 = b
        c1 = c
        d1 = d

        if a[2] > upper_bound:
                #Fix a, but store the coordinates separately
                a1 = interpolate(a)

        if b[2] > upper_bound:
                b1 = interpolate(b)

        if c[2] > upper_bound:
                c1 = interpolate(c)

        if d[2] > upper_bound:
                d1 = interpolate(d)

        facets.add_triangle(a1, b1, c1)
        facets.add_triangle(b1, c1, d1)
if cUsed and c1[1] == stern:
        stern_vertices.append(c1)

#Get the lower right vertex
if cUsed and c1[1] == stern and c1[2] == upper_bound:
        gvRight.append(c1)
if bUsed and b1[2] == upper_bound:
        if b1[0] <= 0 and b1 != gvLeft[-1]:
                gvLeft.append(b1)
        elif gvRight:
                if b1 != gvRight[-1] and b1[0] > 0:
                        gvRight.append(b1)
if dUsed and d1[2] == upper_bound and d1 != gvRight[-1]:
        if d1[0] > 0:
                gvRight.append(d1)
```

```
                else:
                        gvLeft.append(d1)
        if cUsed and c1[2] == upper_bound and c1 <= d1:
                if c1[0] > 0:
                        gvRight.append(c1)
                else:
                        gvLeft.append(c1)

facets = Model3D()
#Declare vertices for two starter triangles.

a = (-10, 0, 0, 1)
b = (-10, 0.1, 0, 1)
c = increment(a)
d = (-9.9, 0.1, 0, 1)

#The y-coordinate of the stern
stern = 0
#An array to hold the vertices of the stern
#transform() should add to it
stern_vertices = []
c7 = interpolate(c)
stern_vertices.append(c7)



#Arrays to hold the gunwales' vertices (gv)
gvLeft = []
gvRight = []
a7 = interpolate(a)
gvLeft.append(a7)
b7 = interpolate(b)
gvLeft.append(b7)
start = 0
i = 0
j = 0
while j < 29.1:
```

```
        start = b[0]
        while i < 19:
                transform(increment, interpolate)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0

#Add the stern
#This may not work if there is an even number of vertices across the back
stoppingPoint = int((len(stern_vertices))/2) - 1
count1 = 0
count2 = len(stern_vertices) - 1
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1

#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])
numOuter = len(stern_vertices) - 1

gvRight_length = len(gvRight)
gvLeft_length = len(gvLeft)

#Add the interior hull surface
m = 0.11
n = 0.01
```

```
w = 0.5

a = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.5, 0, 1)
b = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.6, 0, 1)
c = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.5, 0, 1)
d = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.6, 0, 1)

c7 = interpolate2(c)
stern_vertices.append(c7)

a7 = interpolate2(a)
gvLeft.append(a7)
stern = 0.5

start = 0
i = 0
j = 0
while j < 30:
        start = b[0]
        while i < 18.1:
                transform(increment2, interpolate2)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0

#Add the interior stern
#This may not work if there is an even number of vertices across the back
stoppingPoint = int((len(stern_vertices) - numOuter)/2) - 1 + numOuter
count1 = numOuter + 1
count2 = len(stern_vertices) - 1
```

```
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1


#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#The counter for the outer hull values
count3 = int(stern/step)
#The counter for the inner hull values
count4 = gvLeft_length


#Add the top stern
index = 0
while index < count3:
        facets.add_triangle(gvRight[index], gvRight[index + 1], gvRight[gvRight_length])
        facets.add_triangle(gvLeft[index], gvLeft[index + 1], gvLeft[gvLeft_length])
        index += 1


facets.add_triangle(gvRight[0], gvLeft[0], gvRight[gvRight_length])
facets.add_triangle(gvLeft[0], gvRight[gvRight_length], gvLeft[gvLeft_length])


#Add the left section of the top edge
while count4 < len(gvLeft) - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        facets.add_triangle(gvLeft[count3 + 1], gvLeft[count4], gvLeft[count4 + 1])
        count3 += 1
        count4 += 1


#outer hull
count5 = int(stern/step)
#inner hull
```

```
count6 = gvRight_length

backwards = []
#Reorganize the vertices from the inner hull's right bow
yBound = gvRight[-1][1] - step
while gvRight[-1][1] > yBound:
        backwards.append(gvRight.pop())
#They should now be in the proper order
del backwards[-1]
#The last one is a duplicate

for vertex in backwards:
        gvRight.append(vertex)

while count6 < len(gvRight) - 1:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        facets.add_triangle(gvRight[count5 + 1], gvRight[count6], gvRight[count6 + 1])
        count5 += 1
        count6 += 1

#Print the last few vertices in the right outer hull
count7 = count5
while count7 <= gvRight_length:
        count7 += 1

#Reorganize the vertices from the outer hull's right bow
backwards = []
counter = 0
yBound = gvRight[gvRight_length - 1][1] - step
while gvRight[gvRight_length - 1 - counter][1] > yBound:
        backwards.append(gvRight.pop(gvRight_length - 1 - counter))
        counter += 1
#end of backwards order

count7 = count5
while count7 <= gvRight_length:
```

```
        count7 += 1
#The vertices in backwards added to gvRight

counter -= 1
count7 = len(backwards) - 1
while count7 >= 0:
        gvRight.insert(gvRight_length - 1 - counter, backwards[count7])
        count7 -= 1

#Add the starboard bow
while count5 < gvRight_length - 2:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        count5 += 1

#Add the port bow
while count3 < gvLeft_length - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        count3 += 1

#Add the last triangle in the bow
facets.add_triangle(gvLeft[count3], gvLeft[count4], gvRight[count6])
facets.add_triangle(gvLeft[count3], gvRight[count5], gvRight[count6])

facets.write("keelBoat2.stl")
```

hull11.py

(Noise Hull Without Hole)

```
from rt_functions import *
from random import random
from random import choice
import math
import copy
#Multiplies the normal vector by a random number

upper_bound = 8
step = 0.1

def increment(a):
        """
        Find the next point for the vertex.
        Add step to the x-component and find the new z-component.
        """
        return (a[0] + step, a[1], .1*(a[0] + step)**2 + .01*(a[1])**2, 1)

def increment2(a):
        """Increment function for the inner hull"""
        return (a[0] + step, a[1], m*(a[0] + step)**2 + n*(a[1])**2 + w, 1)

def interpolate(p):
        """
        Finds the proper x-coordinate or y-coordinate for point p.
        It assumes y is correct and finds x.
        If y is not on the surface for the maximum z-value,
        it assumes x is correct and finds a new value for y.
        """


        y = p[1]
        x = (80 - 0.1*y**2)**0.5
        if p[0] < 0:
```

```
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = (800 - 10*x**2)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def interpolate2(p):
        """Interpolate function for the inner hull"""

        y = p[1]
        x = ((upper_bound - w - n*y**2)/m)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = ((upper_bound - w - m*x**2)/n)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def listCheck(member, start, address, point):
        """
        See if the member is in the part of the vertex list past or at start.
        If it is, add an address.
        If it is not, add it and its address.

        One should input only the part of the list in which duplicates will be.
        """
```

```
        notFound = True
        index = len(facets.vertices) - 2*start
        if index < 0:
                index = 0
        while index < len(facets.vertices) and notFound:
                if (member[0] == facets.vertices[index][0]
                and member[1] == facets.vertices[index][1]
                and member[2] == facets.vertices[index][2]):
                        facets.vertices[index].append(address)
                        facets.vertices[index].append(point)
                        notFound = False
                index += 1

        if notFound:
                facets.vertices.append([member[0], member[1], member[2], address, point])

def randomNumber():
        r = random()
        r = r*(choice([-1, 1]))*step
        return r

def domainConvert(theta):
        """Get theta within the domain of arcsin(theta)"""
        if theta < -1 or theta > 1:
                #Take just the decimal component
                phi = int(theta)
                theta = theta - phi
        return theta

address = 0
#transform() uses this

def transform(increment, interpolate, xBound, yBound):
        """Finds the xyz coordinates for the next two triangles."""
        global a
        global b
```

```
        global c
        global d

        global c1
        global address

        aUsed = True
        bUsed = True
        cUsed = True
        dUsed = True

        a = increment(a)
        b = increment(b)
        c = increment(c)
        d = increment(d)

        #Check  to see if all the z-coordinates are out of bounds
        if a[2] > upper_bound and b[2] > upper_bound and c[2] > upper_bound and d[2] >
upper_bound:
                #Since this pair of triangles is out of bounds,
                #do not add it to the hull.
                aUsed = False
                bUsed = False
                cUsed = False
                dUsed = False

        #See if a and c are both out of bounds
        elif a[2] > upper_bound and c[2] > upper_bound:

                #Fix a
                a1 = interpolate(a)

                #See if b and d need to be fixed
                if b[2] > upper_bound:
                        b1 = interpolate(b)
                else:
```

```
                b1 = b
        if d[2] > upper_bound:
                d1 = interpolate(d)
        else:
                d1 = d

        facets.add_triangle(a1, b1, c1)
        #if c1 == c8:

        #See if it is far enough to the stern to need extra triangles
        c1 = interpolate(c)
        if a1[0] != c1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

#See if b and d are both out of bounds
elif b[2] > upper_bound and d[2] > upper_bound:
        b1 = interpolate(b)

        #Fix a and c if they need it
        if a[2] > upper_bound:
                a1 = interpolate(a)
        else:
                a1 = a

        if c[2] > upper_bound:
                c1 = interpolate(c)
        else:
                c1 = c

        #Check for "whiskers" (triangles that protrude unnecessarily from the edge)
        if b1[1] < a1[1]:
                #If b1 goes too far when being interpolated,
                #move it to the other side of the square.
                b1 = (b[0] + step, b[1], b[2], 1)
```

```
                b1 = interpolate(b1)
        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the bow to need extra triangles
        d1 = interpolate(d)
        if b1[0] != d1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

else:
        a1 = a
        b1 = b
        c1 = c
        d1 = d

        if a[2] > upper_bound:
                #Fix a, but store the coordinates separately
                a1 = interpolate(a)

        if b[2] > upper_bound:
                b1 = interpolate(b)

        if c[2] > upper_bound:
                c1 = interpolate(c)

        if d[2] > upper_bound:
                d1 = interpolate(d)

        facets.add_triangle(a1, b1, c1)
        facets.add_triangle(b1, d1, c1)

if cUsed and c1[1] == stern:
        stern_vertices.append(c1)

#Get the lower right vertex
```

```
if cUsed and c1[1] == stern and c1[2] == upper_bound:
        gvRight.append(c1)
if bUsed and b1[2] == upper_bound:
        if b1[0] <= 0 and b1 != gvLeft[-1]:
                gvLeft.append(b1)
        elif gvRight:
                if b1 != gvRight[-1] and b1[0] > 0:
                        gvRight.append(b1)
if dUsed and d1[2] == upper_bound and d1 != gvRight[-1]:
        if d1[0] > 0:
                gvRight.append(d1)
        else:
                gvLeft.append(d1)
if cUsed and c1[2] == upper_bound and c1 <= d1:
        if c1[0] > 0:
                gvRight.append(c1)
        else:
                gvLeft.append(c1)

#Add new vertices to the array of vertices
start = int(yBound/step) + 1
if aUsed:
        listCheck(a1, start, address, 0)
        if bUsed:
                listCheck(b1, start, address, 1)
                #address += 1
        if cUsed:
                listCheck(c1, start, address, 2)
                #address += 1
        address += 1

if dUsed:
        listCheck(d1, start, address, 1)
        if bUsed:
                listCheck(b1, start, address, 0)
        if cUsed:
```

```
                    listCheck(c1, start, address, 2)
            address += 1

facets = Model3D()
#Declare vertices for two starter triangles.

a = (-10, 0, 0, 1)
b = (-10, 0.1, 0, 1)
c = increment(a)
d = (-9.9, 0.1, 0, 1)

#The y-coordinate of the stern
stern = 0
#An array to hold the vertices of the stern
#transform() should add to it
stern_vertices = []
c7 = interpolate(c)
stern_vertices.append(c7)



#Arrays to hold the gunwales' vertices (gv)
gvLeft = []
gvRight = []
a7 = interpolate(a)
gvLeft.append(a7)
b7 = interpolate(b)
gvLeft.append(b7)

start = 0
i = 0
j = 0
yBound = 29.1
while j < yBound:
        start = b[0]
        xBound = 19
        while i < xBound:
```

```
                transform(increment, interpolate, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0


stoppingPoint = int((len(stern_vertices))/2) - 1
count1 = 0
count2 = len(stern_vertices) - 1

numOuter = len(stern_vertices) - 1

gvRight_length = len(gvRight)
gvLeft_length = len(gvLeft)

spareTriangles = copy.deepcopy(facets.triangles)
#Perform a deep copy to keep the original points.
#They will be needed when the normals are computed.

#Add texture to the outer hull
pos = 3
for point in facets.vertices:
        if point[2] < upper_bound - step and point[1] > 0:
                pos = 3
                normal = find_normal(spareTriangles[point[pos]])

                normal = multiply(normal, randomNumber())

                r1 = normal[0]
                r2 = normal[1]
```

```
                r3 = normal[2]

                while pos < len(point):

                        facets.triangles[point[pos]][point[pos + 1]] =
(facets.triangles[point[pos]][point[pos + 1]][0] + r1, facets.triangles[point[pos]][point[pos +
1]][1] + r2, facets.triangles[point[pos]][point[pos + 1]][2] + r3, 1)

                        pos += 2


#Add the interior hull surface

m = 0.11

n = 0.01

w = 0.5


a = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.5, 0, 1)

b = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.6, 0, 1)

c = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.5, 0, 1)

d = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.6, 0, 1)


c7 = interpolate2(c)

stern_vertices.append(c7)


a7 = interpolate2(a)

gvLeft.append(a7)

stern = 0.5


start = 0

i = 0

j = 0

yBound = 30

while j < yBound:

        start = b[0]

        xBound = 18.1

        while i < xBound:

                transform(increment2, interpolate2, xBound, yBound)

                i += step


        a = (start, a[1] + step, a[2], 1)

        b = (start, b[1] + step, b[2], 1)
```

```
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)


        j += step
        i = 0


#Add the exterior stern
#This may not work if there is an even number of vertices across the back
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1


#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#Add the interior stern
#This may not work if there is an even number of vertices across the stern
stoppingPoint = int((len(stern_vertices) - numOuter)/2) - 1 + numOuter
count1 = numOuter + 1
count2 = len(stern_vertices) - 1
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1


#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#The counter for the outer hull values
count3 = int(stern/step)
```

```
#The counter for the inner hull values
count4 = gvLeft_length

#Add the top stern
index = 0
while index < count3:
        facets.add_triangle(gvRight[index], gvRight[index + 1], gvRight[gvRight_length])
        facets.add_triangle(gvLeft[index], gvLeft[index + 1], gvLeft[gvLeft_length])
        index += 1

facets.add_triangle(gvRight[0], gvLeft[0], gvRight[gvRight_length])
facets.add_triangle(gvLeft[0], gvRight[gvRight_length], gvLeft[gvLeft_length])

#Add the left section of the top edge
while count4 < len(gvLeft) - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        facets.add_triangle(gvLeft[count3 + 1], gvLeft[count4], gvLeft[count4 + 1])
        count3 += 1
        count4 += 1

#outer hull
count5 = int(stern/step)
#inner hull
count6 = gvRight_length

backwards = []
#Reorganize the vertices from the inner hull's right bow
yBound = gvRight[-1][1] - step
while gvRight[-1][1] > yBound:
        backwards.append(gvRight.pop())
#They should now be in the proper order

#end of backwards order
del backwards[-1]
#The last one is a duplicate
```

```
for vertex in backwards:
        gvRight.append(vertex)

while count6 < len(gvRight) - 1:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        facets.add_triangle(gvRight[count5 + 1], gvRight[count6], gvRight[count6 + 1])
        count5 += 1
        count6 += 1

#Print the last few vertices in the right outer hull

#Reorganize the vertices from the outer hull's right bow
backwards = []
counter = 0
yBound = gvRight[gvRight_length - 1][1] - step
while gvRight[gvRight_length - 1 - counter][1] > yBound:
        backwards.append(gvRight.pop(gvRight_length - 1 - counter))
        counter += 1

count7 = count5
while count7 <= gvRight_length:
        count7 += 1

#The vertices in backwards added to gvRight

counter -= 1
count7 = len(backwards) - 1
while count7 >= 0:
        gvRight.insert(gvRight_length - 1 - counter, backwards[count7])
        count7 -= 1

#Add the starboard bow
while count5 < gvRight_length - 2:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        count5 += 1
```

```
#Add the port bow
while count3 < gvLeft_length - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        count3 += 1

#Add the last triangle in the bow
facets.add_triangle(gvLeft[count3], gvLeft[count4], gvRight[count6])
facets.add_triangle(gvLeft[count3], gvRight[count5], gvRight[count6])

facets.write("hull11.stl")
```

hull12.py

(Noise Hull With Hole)

(The hole for the string was added with Blender)

```
from rt_functions import *
from random import random
from random import choice
import math
import copy
#Multiplies the x, y, and z components of the normal vector by random numbers

upper_bound = 8
step = 0.1

def increment(a):
        """
        Find the next point for the vertex.
        Add step to the x-component and find the new z-component.
        """
        return (a[0] + step, a[1], .1*(a[0] + step)**2 + .01*(a[1])**2, 1)

def increment2(a):
        """Increment function for the inner hull"""
        return (a[0] + step, a[1], m*(a[0] + step)**2 + n*(a[1])**2 + w, 1)

def interpolate(p):
        """
        Finds the proper x-coordinate or y-coordinate for point p.
        It assumes y is correct and finds x.
        If y is not on the surface for the maximum z-value,
        it assumes x is correct and finds a new value for y.
        """

        y = p[1]
        x = (80 - 0.1*y**2)**0.5
```

```
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = (800 - 10*x**2)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def interpolate2(p):
        """Interpolate function for the inner hull"""

        y = p[1]
        x = ((upper_bound - w - n*y**2)/m)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = ((upper_bound - w - m*x**2)/n)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def listCheck(member, start, address, point):
        """
        See if the member is in the part of the vertex list past or at start.
        If it is, add an address.
        If it is not, add it and its address.

        One should input only the part of the list in which duplicates will be.
```

```
        """
        notFound = True
        index = len(facets.vertices) - 2*start
        if index < 0:
                index = 0
        while index < len(facets.vertices) and notFound:
                if (member[0] == facets.vertices[index][0]
                and member[1] == facets.vertices[index][1]
                and member[2] == facets.vertices[index][2]):
                        facets.vertices[index].append(address)
                        facets.vertices[index].append(point)
                        notFound = False
                index += 1

        if notFound:
                facets.vertices.append([member[0], member[1], member[2], address, point])

def randomNumber():
        #Get a random decimal
        r = random()
        #Decide if it is positive or negative, and scale it properly
        r = r*(choice([-1, 1]))*step
        return r

def domainConvert(theta):
        """Get theta within the domain of arcsin(theta)"""
        if theta < -1 or theta > 1:
                #Take just the decimal component
                phi = int(theta)
                theta = theta - phi
        return theta

address = 0
#transform() uses this

def transform(increment, interpolate, xBound, yBound):
```

```
        """Finds the xyz coordinates for the next two triangles."""
        global a
        global b
        global c
        global d

        global c1
        global address

        aUsed = True
        bUsed = True
        cUsed = True
        dUsed = True

        a = increment(a)
        b = increment(b)
        c = increment(c)
        d = increment(d)

        #Check  to see if all the z-coordinates are out of bounds
        if a[2] > upper_bound and b[2] > upper_bound and c[2] > upper_bound and d[2] >
upper_bound:
                #Since this pair of triangles is out of bounds,
                #do not add it to the hull.
                aUsed = False
                bUsed = False
                cUsed = False
                dUsed = False

        #See if a and c are both out of bounds
        elif a[2] > upper_bound and c[2] > upper_bound:

                #Fix a
                a1 = interpolate(a)

                #See if b and d need to be fixed
```

```
        if b[2] > upper_bound:
                b1 = interpolate(b)
        else:
                b1 = b
        if d[2] > upper_bound:
                d1 = interpolate(d)
        else:
                d1 = d

        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the stern to need extra triangles
        c1 = interpolate(c)
        if a1[0] != c1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

#See if b and d are both out of bounds
elif b[2] > upper_bound and d[2] > upper_bound:

        b1 = interpolate(b)

        #Fix a and c if they need it
        if a[2] > upper_bound:
                a1 = interpolate(a)
        else:
                a1 = a

        if c[2] > upper_bound:
                c1 = interpolate(c)
        else:
                c1 = c

        #Check for "whiskers"
        if b1[1] < a1[1]:
```

```
                #If b1 goes too far when being interpolated,
                #move it to the other side of the square.
                b1 = (b[0] + step, b[1], b[2], 1)
                b1 = interpolate(b1)
        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the bow to need extra triangles
        d1 = interpolate(d)
        if b1[0] != d1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

else:
        a1 = a
        b1 = b
        c1 = c
        d1 = d

        if a[2] > upper_bound:
                #Fix a, but store the coordinates separately
                a1 = interpolate(a)

        if b[2] > upper_bound:
                b1 = interpolate(b)

        if c[2] > upper_bound:
                c1 = interpolate(c)

        if d[2] > upper_bound:
                d1 = interpolate(d)

        facets.add_triangle(a1, b1, c1)
        facets.add_triangle(b1, d1, c1)

if cUsed and c1[1] == stern:
```

```
        stern_vertices.append(c1)

#Get the lower right vertex
if cUsed and c1[1] == stern and c1[2] == upper_bound:
        gvRight.append(c1)
if bUsed and b1[2] == upper_bound:
        if b1[0] <= 0 and b1 != gvLeft[-1]:
                gvLeft.append(b1)
        elif gvRight:
                if b1 != gvRight[-1] and b1[0] > 0:
                        gvRight.append(b1)
if dUsed and d1[2] == upper_bound and d1 != gvRight[-1]:
        if d1[0] > 0:
                gvRight.append(d1)
        else:
                gvLeft.append(d1)
if cUsed and c1[2] == upper_bound and c1 <= d1:
        if c1[0] > 0:
                gvRight.append(c1)
        else:
                gvLeft.append(c1)

#Add new vertices to the array of vertices
start = int(yBound/step) + 1
if aUsed:
        listCheck(a1, start, address, 0)
        if bUsed:
                listCheck(b1, start, address, 1)
        if cUsed:
                listCheck(c1, start, address, 2)
        address += 1

if dUsed:
        listCheck(d1, start, address, 1)
        if bUsed:
                listCheck(b1, start, address, 0)
```

```
                if cUsed:
                        listCheck(c1, start, address, 2)
                address += 1

facets = Model3D()
#Declare vertices for two starter triangles.

a = (-10, 0, 0, 1)
b = (-10, 0.1, 0, 1)
c = increment(a)
d = (-9.9, 0.1, 0, 1)

#The y-coordinate of the stern
stern = 0
#An array to hold the vertices of the stern
#transform() should add to it
stern_vertices = []
c7 = interpolate(c)
stern_vertices.append(c7)



#Arrays to hold the gunwales' vertices (gv)
gvLeft = []
gvRight = []
a7 = interpolate(a)
gvLeft.append(a7)
b7 = interpolate(b)
gvLeft.append(b7)

#The outer hull
start = 0
i = 0
j = 0
yBound = 29.1
while j < yBound:
        start = b[0]
```

```
        xBound = 19
        while i < xBound:
                transform(increment, interpolate, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)
        j += step
        i = 0
stoppingPoint = int((len(stern_vertices))/2) - 1
count1 = 0
count2 = len(stern_vertices) - 1

numOuter = len(stern_vertices) - 1

gvRight_length = len(gvRight)
gvLeft_length = len(gvLeft)

spareTriangles = copy.deepcopy(facets.triangles)
#Perform a deep copy to keep the original points.
#They will be needed when the normals are computed.

#Add texture to the outer hull
pos = 3
for point in facets.vertices:
        if point[2] < upper_bound - step and point[1] > 0:
                pos = 3
                normal = find_normal(spareTriangles[point[pos]])

                r1 = normal[0]*randomNumber()
                r2 = normal[1]*randomNumber()
                r3 = normal[2]*randomNumber()
                while pos < len(point):
```

```
                        facets.triangles[point[pos]][point[pos + 1]] =
(facets.triangles[point[pos]][point[pos + 1]][0] + r1, facets.triangles[point[pos]][point[pos +
1]][1] + r2, facets.triangles[point[pos]][point[pos + 1]][2] + r3, 1)
                        pos += 2


#Add the interior hull surface
m = 0.11
n = 0.01
w = 0.5


a = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.5, 0, 1)
b = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.6, 0, 1)
c = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.5, 0, 1)
d = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.6, 0, 1)


c7 = interpolate2(c)
stern_vertices.append(c7)


a7 = interpolate2(a)
gvLeft.append(a7)
stern = 0.5


start = 0
i = 0
j = 0
yBound = 30
while j < yBound:
        start = b[0]
        xBound = 18.1
        while i < xBound:
                transform(increment2, interpolate2, xBound, yBound)
                i += step


        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)
```

```
        j += step
        i = 0


#Add the exterior stern
#This may not work if there is an even number of vertices across the back
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1


#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#Add the interior stern
#This may not work if there is an even number of vertices across the stern
stoppingPoint = int((len(stern_vertices) - numOuter)/2) - 1 + numOuter
count1 = numOuter + 1
count2 = len(stern_vertices) - 1
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])
        count1 += 1
        count2 -= 1


#Add the bottom triangle
facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#The counter for the outer hull values
count3 = int(stern/step)
#The counter for the inner hull values
count4 = gvLeft_length
```

```
#Add the top stern
index = 0
while index < count3:
        facets.add_triangle(gvRight[index], gvRight[index + 1], gvRight[gvRight_length])
        facets.add_triangle(gvLeft[index], gvLeft[index + 1], gvLeft[gvLeft_length])
        index += 1

facets.add_triangle(gvRight[0], gvLeft[0], gvRight[gvRight_length])
facets.add_triangle(gvLeft[0], gvRight[gvRight_length], gvLeft[gvLeft_length])

#Add the left section of the top edge
while count4 < len(gvLeft) - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        facets.add_triangle(gvLeft[count3 + 1], gvLeft[count4], gvLeft[count4 + 1])
        count3 += 1
        count4 += 1

#outer hull
count5 = int(stern/step)
#inner hull
count6 = gvRight_length

backwards = []
#Reorganize the vertices from the inner hull's right bow
yBound = gvRight[-1][1] - step
while gvRight[-1][1] > yBound:
        backwards.append(gvRight.pop())
#They should now be in the proper order

#end of backwards order
del backwards[-1]
#The last one is a duplicate

for vertex in backwards:
        gvRight.append(vertex)
```

```
while count6 < len(gvRight) - 1:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        facets.add_triangle(gvRight[count5 + 1], gvRight[count6], gvRight[count6 + 1])
        count5 += 1
        count6 += 1

#Reorganize the vertices from the outer hull's right bow
backwards = []
counter = 0
yBound = gvRight[gvRight_length - 1][1] - step
while gvRight[gvRight_length - 1 - counter][1] > yBound:
        backwards.append(gvRight.pop(gvRight_length - 1 - counter))
        counter += 1

count7 = count5
while count7 <= gvRight_length:
        count7 += 1

counter -= 1
count7 = len(backwards) - 1
while count7 >= 0:
        gvRight.insert(gvRight_length - 1 - counter, backwards[count7])
        count7 -= 1

while count5 < gvRight_length - 2:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        count5 += 1

#Add the port bow
while count3 < gvLeft_length - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        count3 += 1

#Add the last triangle in the bow
facets.add_triangle(gvLeft[count3], gvLeft[count4], gvRight[count6])
```

```
facets.add_triangle(gvLeft[count3], gvRight[count5], gvRight[count6])


facets.write("hull12.stl")
```

hull16.py

(Sierpinski gasket hull)

```
"""A boat hull with the Sierpinski gasket"""
#Note:  Make sure to run ifs2.py to get the Sierpinski gasket's JSON file.
from rt_functions import *
from random import random
from random import choice
import math
import copy
import json

upper_bound = 8
step = 0.1

filename = 'ifs2.json'
#Get the fractal from the input file
with open(filename) as f_obj:
        attractor = json.load(f_obj)

def increment(a):
        """
        Find the next point for the vertex.
        Add step to the x-component and find the new z-component.
        """
        return (a[0] + step, a[1], .1*(a[0] + step)**2 + .01*(a[1])**2, 1)

def increment2(a):
        """Increment function for the inner hull"""
        return (a[0] + step, a[1], m*(a[0] + step)**2 + n*(a[1])**2 + w, 1)

def interpolate(p):
        """
        Finds the proper x-coordinate or y-coordinate for point p.
        It assumes y is correct and finds x.
```

```
        If y is not on the surface for the maximum z-value,
        it assumes x is correct and finds a new value for y.
        """

        y = p[1]
        x = (80 - 0.1*y**2)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = (800 - 10*x**2)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def interpolate2(p):
        """Interpolate function for the inner hull"""

        y = p[1]
        x = ((upper_bound - w - n*y**2)/m)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = ((upper_bound - w - m*x**2)/n)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def listCheck(member, start, address, point):
```

```
        """
        See if the member is in the part of the vertex list past or at start.
        If it is, add an address.
        If it is not, add it and its address.

        One should input only the part of the list in which duplicates will be.
        """
        notFound = True
        index = len(facets.vertices) - 2*start
        if index < 0:
                index = 0
        while index < len(facets.vertices) and notFound:
                if (member[0] == facets.vertices[index][0]
                and member[1] == facets.vertices[index][1]
                and member[2] == facets.vertices[index][2]):
                        facets.vertices[index].append(address)
                        facets.vertices[index].append(point)
                        notFound = False
                index += 1

        if notFound:
                facets.vertices.append([member[0], member[1], member[2], address, point])

def randomNumber():
        #Get a random decimal
        r = random()
        #Decide if it is positive or negative, and scale it properly
        r = r*(choice([-1, 1]))*step
        return r

def domainConvert(theta):
        """Get theta within the domain of arcsin(theta)"""
        if theta < -1 or theta > 1:
                #Take just the decimal component
                phi = int(theta)
                theta = theta - phi
```

```
        return theta


address = 0
#transform() uses this


def transform(increment, interpolate, xBound, yBound):
        """Finds the xyz coordinates for the next two triangles."""
        global a
        global b
        global c
        global d

        global c1
        global address

        aUsed = True
        bUsed = True
        cUsed = True
        dUsed = True

        a = increment(a)
        b = increment(b)
        c = increment(c)
        d = increment(d)

        #Check  to see if all the z-coordinates are out of bounds
        if a[2] > upper_bound and b[2] > upper_bound and c[2] > upper_bound and d[2] >
upper_bound:
                #Since this pair of triangles is out of bounds,
                #do not add it to the hull.
                aUsed = False
                bUsed = False
                cUsed = False
                dUsed = False

        #See if a and c are both out of bounds
```

```
elif a[2] > upper_bound and c[2] > upper_bound:
#This if statement may not have been used for a boat that does not extend past 0.

        #Fix a
        a1 = interpolate(a)

        #See if b and d need to be fixed
        if b[2] > upper_bound:
                b1 = interpolate(b)
        else:
                b1 = b
        if d[2] > upper_bound:
                d1 = interpolate(d)
        else:
                d1 = d

        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the stern to need extra triangles
        c1 = interpolate(c)
        if a1[0] != c1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

#See if b and d are both out of bounds
elif b[2] > upper_bound and d[2] > upper_bound:

        b1 = interpolate(b)

        #Fix a and c if they need it
        if a[2] > upper_bound:
                a1 = interpolate(a)
        else:
                a1 = a
```

```
        if c[2] > upper_bound:
                c1 = interpolate(c)
        else:
                c1 = c

        #Check for "whiskers"
        if b1[1] < a1[1]:
                b1 = (b[0] + step, b[1], b[2], 1)
                b1 = interpolate(b1)
        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the bow to need extra triangles
        d1 = interpolate(d)
        if b1[0] != d1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

else:
        a1 = a
        b1 = b
        c1 = c
        d1 = d

        if a[2] > upper_bound:
                #Fix a, but store the coordinates separately
                a1 = interpolate(a)

        if b[2] > upper_bound:
                b1 = interpolate(b)

        if c[2] > upper_bound:
                c1 = interpolate(c)

        if d[2] > upper_bound:
                d1 = interpolate(d)
```

```
        facets.add_triangle(a1, b1, c1)
        facets.add_triangle(b1, d1, c1)

if cUsed and c1[1] == stern:
        stern_vertices.append(c1)

#Get the lower right vertex
if cUsed and c1[1] == stern and c1[2] == upper_bound:
        gvRight.append(c1)
if bUsed and b1[2] == upper_bound:
        if b1[0] <= 0 and b1 != gvLeft[-1]:
                gvLeft.append(b1)
        elif gvRight:
                if b1 != gvRight[-1] and b1[0] > 0:
                        gvRight.append(b1)
if dUsed and d1[2] == upper_bound and d1 != gvRight[-1]:
        if d1[0] > 0:
                gvRight.append(d1)
        else:
                gvLeft.append(d1)
if cUsed and c1[2] == upper_bound and c1 <= d1:
        if c1[0] > 0:
                gvRight.append(c1)
        else:
                gvLeft.append(c1)

#Add new vertices to the array of vertices
start = int(yBound/step) + 1
if aUsed:
        listCheck(a1, start, address, 0)
        if bUsed:
                listCheck(b1, start, address, 1)
        if cUsed:
                listCheck(c1, start, address, 2)
        address += 1
```

```
        if dUsed:
                listCheck(d1, start, address, 1)
                if bUsed:
                        listCheck(b1, start, address, 0)
                if cUsed:
                        listCheck(c1, start, address, 2)
                address += 1

def round(x):
        """Rounds the input to the nearest integer."""
        x1 = int(x)
        if x > 0:
                if ((x1 + 0.5) <= x):
                        x1 = x1 + 1
        elif x < 0:
                if ((x1 + 0.5) >= x):
                        x1 = x1 - 1
        return x1

facets = Model3D()
#Declare vertices for two starter triangles.

a = (-10, 0, 0, 1)
b = (-10, 0.1, 0, 1)
c = increment(a)
d = (-9.9, 0.1, 0, 1)

#The y-coordinate of the stern
stern = 0
#An array to hold the vertices of the stern
#transform() should add to it
stern_vertices = []
c7 = interpolate(c)
stern_vertices.append(c7)
```

```
#Arrays to hold the gunwales' vertices (gv)
gvLeft = []
gvRight = []
a7 = interpolate(a)
gvLeft.append(a7)
b7 = interpolate(b)
gvLeft.append(b7)

#The outer hull
start = 0
i = 0
j = 0
yBound = 29.1
while j < yBound:
        start = b[0]
        xBound = 19
        while i < xBound:
                transform(increment, interpolate, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0


stoppingPoint = int((len(stern_vertices))/2) - 1
count1 = 0
count2 = len(stern_vertices) - 1

numOuter = len(stern_vertices) - 1
```

```
gvRight_length = len(gvRight)
gvLeft_length = len(gvLeft)


spareTriangles = copy.deepcopy(facets.triangles)
#Perform a deep copy to keep the original points.
#They will be needed when the normals are computed.


#Add texture to the outer hull
pos = 3
for point in facets.vertices:
        if point[2] < upper_bound - step and point[1] > 0:
                pos = 3
                normal = find_normal(spareTriangles[point[pos]])


                xA = round(point[0]*100 + 1910)
                yA = round(point[1]*100 + 2910)


                if attractor[yA][xA] == 1:
                        normal = multiply(normal, -1*step)
                        r1 = normal[0]
                        r2 = normal[1]
                        r3 = normal[2]
                        while pos < len(point):
                                facets.triangles[point[pos]][point[pos + 1]] =
(facets.triangles[point[pos]][point[pos + 1]][0] + r1, facets.triangles[point[pos]][point[pos +
1]][1] + r2, facets.triangles[point[pos]][point[pos + 1]][2] + r3, 1)
                                pos += 2


#Add the interior hull surface
m = 0.11
n = 0.01
w = 0.5


a = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.5, 0, 1)
b = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.6, 0, 1)
c = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.5, 0, 1)
d = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.6, 0, 1)
```

```
c7 = interpolate2(c)
stern_vertices.append(c7)

a7 = interpolate2(a)
gvLeft.append(a7)
stern = 0.5

start = 0
i = 0
j = 0
yBound = 30
while j < yBound:
        start = b[0]
        xBound = 18.1
        while i < xBound:

                transform(increment2, interpolate2, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0

#Add the exterior stern
#This may not work if there is an even number of vertices across the back
while count1 < stoppingPoint:
        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])
        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])

        count1 += 1
```

```
        count2 -= 1


#Add the bottom triangle

facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#Add the interior stern

#This may not work if there is an even number of vertices across the stern

stoppingPoint = int((len(stern_vertices) - numOuter)/2) - 1 + numOuter

count1 = numOuter + 1

count2 = len(stern_vertices) - 1

while count1 < stoppingPoint:

        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])

        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])

        count1 += 1

        count2 -= 1


#Add the bottom triangle

facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])


#The counter for the outer hull values

count3 = int(stern/step)

#The counter for the inner hull values

count4 = gvLeft_length


#Add the top stern

index = 0

while index < count3:

        facets.add_triangle(gvRight[index], gvRight[index + 1], gvRight[gvRight_length])

        facets.add_triangle(gvLeft[index], gvLeft[index + 1], gvLeft[gvLeft_length])

        index += 1


facets.add_triangle(gvRight[0], gvLeft[0], gvRight[gvRight_length])

facets.add_triangle(gvLeft[0], gvRight[gvRight_length], gvLeft[gvLeft_length])
```

```
#Add the left section of the top edge
while count4 < len(gvLeft) - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        facets.add_triangle(gvLeft[count3 + 1], gvLeft[count4], gvLeft[count4 + 1])
        count3 += 1
        count4 += 1

#outer hull
count5 = int(stern/step)
#inner hull
count6 = gvRight_length

backwards = []
#Reorganize the vertices from the inner hull's right bow
yBound = gvRight[-1][1] - step
while gvRight[-1][1] > yBound:
        backwards.append(gvRight.pop())
#They should now be in the proper order

#end of backwards order
del backwards[-1]
#The last one is a duplicate

for vertex in backwards:
        gvRight.append(vertex)

while count6 < len(gvRight) - 1:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        facets.add_triangle(gvRight[count5 + 1], gvRight[count6], gvRight[count6 + 1])
        count5 += 1
        count6 += 1

#Reorganize the vertices from the outer hull's right bow
backwards = []
counter = 0
yBound = gvRight[gvRight_length - 1][1] - step
```

```
while gvRight[gvRight_length - 1 - counter][1] > yBound:
        backwards.append(gvRight.pop(gvRight_length - 1 - counter))
        counter += 1

count7 = count5
while count7 <= gvRight_length:
        count7 += 1

#The vertices in backwards added to gvRight
counter -= 1
count7 = len(backwards) - 1
while count7 >= 0:
        gvRight.insert(gvRight_length - 1 - counter, backwards[count7])
        count7 -= 1

#Add the starboard bow
while count5 < gvRight_length - 2:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        count5 += 1

#Add the port bow
while count3 < gvLeft_length - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        count3 += 1

#Add the last triangle in the bow
facets.add_triangle(gvLeft[count3], gvLeft[count4], gvRight[count6])
facets.add_triangle(gvLeft[count3], gvRight[count5], gvRight[count6])

facets.write("hull16.stl")
```

hull17.py

(Sierpinski Carpet)

```
"""A boat hull with the Sierpinski carpet"""
#Remember to run ifs4.py to get the Sierpinski carpet JSON file
from rt_functions import *
from random import random
from random import choice
import math
import copy
import json

upper_bound = 8
step = 0.1

filename = 'ifs4.json'
#Get the fractal from the input file
with open(filename) as f_obj:
        attractor = json.load(f_obj)

def increment(a):
        """
        Find the next point for the vertex.
        Add step to the x-component and find the new z-component.
        """
        return (a[0] + step, a[1], .1*(a[0] + step)**2 + .01*(a[1])**2, 1)

def increment2(a):
        """Increment function for the inner hull"""
        return (a[0] + step, a[1], m*(a[0] + step)**2 + n*(a[1])**2 + w, 1)

def interpolate(p):
        """
        Finds the proper x-coordinate or y-coordinate for point p.
        It assumes y is correct and finds x.
```

```
        If y is not on the surface for the maximum z-value,
        it assumes x is correct and finds a new value for y.
        """

        y = p[1]
        x = (80 - 0.1*y**2)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = (800 - 10*x**2)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def interpolate2(p):
        """Interpolate function for the inner hull"""

        y = p[1]
        x = ((upper_bound - w - n*y**2)/m)**0.5
        if p[0] < 0:
                x = x*(-1)

        #If that gives a complex number, try this:
        if type(x) == complex:
                x = p[0]
                y = ((upper_bound - w - m*x**2)/n)**0.5
                if p[1] < 0:
                        y = y*(-1)

        return (x, y, upper_bound, 1)

def listCheck(member, start, address, point):
```

```
        """
        See if the member is in the part of the vertex list past or at start.
        If it is, add an address.
        If it is not, add it and its address.

        One should input only the part of the list in which duplicates will be.
        """
        notFound = True
        index = len(facets.vertices) - 2*start
        if index < 0:
                index = 0
        while index < len(facets.vertices) and notFound:
                if (member[0] == facets.vertices[index][0]
                and member[1] == facets.vertices[index][1]
                and member[2] == facets.vertices[index][2]):
                        facets.vertices[index].append(address)
                        facets.vertices[index].append(point)
                        notFound = False
                index += 1

        if notFound:
                facets.vertices.append([member[0], member[1], member[2], address, point])

def randomNumber():
        #Get a random decimal
        r = random()
        #Decide if it is positive or negative, and scale it properly
        r = r*(choice([-1, 1]))*step
        return r

def domainConvert(theta):
        """Get theta within the domain of arcsin(theta)"""
        if theta < -1 or theta > 1:
                #Take just the decimal component
                phi = int(theta)
                theta = theta - phi
```

```
        return theta


address = 0
#transform() uses this


def transform(increment, interpolate, xBound, yBound):
        """Finds the xyz coordinates for the next two triangles."""
        global a
        global b
        global c
        global d

        global c1
        global address

        aUsed = True
        bUsed = True
        cUsed = True
        dUsed = True

        a = increment(a)
        b = increment(b)
        c = increment(c)
        d = increment(d)

        #Check  to see if all the z-coordinates are out of bounds
        if a[2] > upper_bound and b[2] > upper_bound and c[2] > upper_bound and d[2] >
upper_bound:
                #Since this pair of triangles is out of bounds,
                #do not add it to the hull.
                aUsed = False
                bUsed = False
                cUsed = False
                dUsed = False

        #See if a and c are both out of bounds
```

```
elif a[2] > upper_bound and c[2] > upper_bound:

        #Fix a
        a1 = interpolate(a)

        #See if b and d need to be fixed
        if b[2] > upper_bound:
                b1 = interpolate(b)
        else:
                b1 = b
        if d[2] > upper_bound:
                d1 = interpolate(d)
        else:
                d1 = d

        facets.add_triangle(a1, b1, c1)
        #if c1 == c8:
                #print("copy?")

        #See if it is far enough to the stern to need extra triangles
        c1 = interpolate(c)
        if a1[0] != c1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

#See if b and d are both out of bounds
elif b[2] > upper_bound and d[2] > upper_bound:
        b1 = interpolate(b)

        #Fix a and c if they need it
        if a[2] > upper_bound:
                a1 = interpolate(a)
        else:
                a1 = a
```

```
        if c[2] > upper_bound:
                c1 = interpolate(c)
        else:
                c1 = c

        #Check for "whiskers"
        if b1[1] < a1[1]:
                #If b1 goes too far when being interpolated,
                #move it to the other side of the square.
                b1 = (b[0] + step, b[1], b[2], 1)
                b1 = interpolate(b1)
        facets.add_triangle(a1, b1, c1)

        #See if it is far enough to the bow to need extra triangles
        d1 = interpolate(d)
        if b1[0] != d1[0] and d1[1] >= c1[1] and b1[1] >= a1[1]:
                facets.add_triangle(b1, d1, c1)
        else:
                dUsed = False

else:
        a1 = a
        b1 = b
        c1 = c
        d1 = d

        if a[2] > upper_bound:
                #Fix a, but store the coordinates separately
                a1 = interpolate(a)

        if b[2] > upper_bound:
                b1 = interpolate(b)

        if c[2] > upper_bound:
                c1 = interpolate(c)
```

```
        if d[2] > upper_bound:
                d1 = interpolate(d)


        facets.add_triangle(a1, b1, c1)
        facets.add_triangle(b1, d1, c1)
if cUsed and c1[1] == stern:
        stern_vertices.append(c1)


#Get the lower right vertex
if cUsed and c1[1] == stern and c1[2] == upper_bound:
        gvRight.append(c1)
if bUsed and b1[2] == upper_bound:
        if b1[0] <= 0 and b1 != gvLeft[-1]:
                gvLeft.append(b1)
        elif gvRight:
                if b1 != gvRight[-1] and b1[0] > 0:
                        gvRight.append(b1)
if dUsed and d1[2] == upper_bound and d1 != gvRight[-1]:
        if d1[0] > 0:
                gvRight.append(d1)
        else:
                gvLeft.append(d1)
if cUsed and c1[2] == upper_bound and c1 <= d1:
        if c1[0] > 0:
                gvRight.append(c1)
        else:
                gvLeft.append(c1)


#Add new vertices to the array of vertices
start = int(yBound/step) + 1
if aUsed:
        listCheck(a1, start, address, 0)
        if bUsed:
                listCheck(b1, start, address, 1)
        if cUsed:
                listCheck(c1, start, address, 2)
```

```
                address += 1

        if dUsed:
                listCheck(d1, start, address, 1)
                if bUsed:
                        listCheck(b1, start, address, 0)
                if cUsed:
                        listCheck(c1, start, address, 2)
                address += 1

def round(x):
        """Rounds the input to the nearest integer."""
        x1 = int(x)
        if x > 0:
                if ((x1 + 0.5) <= x):
                        x1 = x1 + 1
        elif x < 0:
                if ((x1 + 0.5) >= x):
                        x1 = x1 - 1
        return x1

facets = Model3D()
#Declare vertices for two starter triangles.

a = (-10, 0, 0, 1)
b = (-10, 0.1, 0, 1)
c = increment(a)
d = (-9.9, 0.1, 0, 1)

#The y-coordinate of the stern
stern = 0
#An array to hold the vertices of the stern
#transform() should add to it
stern_vertices = []
c7 = interpolate(c)
stern_vertices.append(c7)
```

```
#Arrays to hold the gunwales' vertices (gv)
gvLeft = []
gvRight = []
a7 = interpolate(a)
gvLeft.append(a7)
b7 = interpolate(b)
gvLeft.append(b7)

#The outer hull
start = 0
i = 0
j = 0
yBound = 29.1
while j < yBound:
        start = b[0]
        xBound = 19
        while i < xBound:
                transform(increment, interpolate, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0

stoppingPoint = int((len(stern_vertices))/2) - 1
count1 = 0
count2 = len(stern_vertices) - 1

numOuter = len(stern_vertices) - 1

gvRight_length = len(gvRight)
```

```
gvLeft_length = len(gvLeft)


spareTriangles = copy.deepcopy(facets.triangles)
#Perform a deep copy to keep the original points.
#They will be needed when the normals are computed.


#Add texture to the outer hull
pos = 3
print(len(attractor))
print(len(attractor[0]))
for point in facets.vertices:
        print(point)
        if point[2] < upper_bound - step and point[1] > 0:
                pos = 3
                normal = find_normal(spareTriangles[point[pos]])


                print(point[0])
                print(point[1])


                xA = round(point[0]*100 + 1910)
                yA = round(point[1]*100 + 2910)


                print(xA)
                print(yA)
                if attractor[yA][xA] == 1:
                        normal = multiply(normal, -1*step)
                        r1 = normal[0]
                        r2 = normal[1]
                        r3 = normal[2]
                        while pos < len(point):
                                facets.triangles[point[pos]][point[pos + 1]] =
(facets.triangles[point[pos]][point[pos + 1]][0] + r1, facets.triangles[point[pos]][point[pos +
1]][1] + r2, facets.triangles[point[pos]][point[pos + 1]][2] + r3, 1)
                                pos += 2


#Add the interior hull surface
m = 0.11
```

```
n = 0.01
w = 0.5

a = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.5, 0, 1)
b = (-((upper_bound - w - n*(0.1**2))/m)**0.5 - step, 0.6, 0, 1)
c = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.5, 0, 1)
d = (-((upper_bound - w - n*(0.1**2))/m)**0.5, 0.6, 0, 1)

c7 = interpolate2(c)
stern_vertices.append(c7)

a7 = interpolate2(a)
gvLeft.append(a7)
stern = 0.5

start = 0
i = 0
j = 0
yBound = 30
while j < yBound:
        start = b[0]
        xBound = 18.1
        while i < xBound:

                transform(increment2, interpolate2, xBound, yBound)
                i += step

        a = (start, a[1] + step, a[2], 1)
        b = (start, b[1] + step, b[2], 1)
        c = (start + step, c[1] + step, c[2], 1)
        d = (start + step, d[1] + step, d[2], 1)

        j += step
        i = 0

#Add the exterior stern
```

```
#This may not work if there is an even number of vertices across the back

while count1 < stoppingPoint:

        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])

        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])



        count1 += 1

        count2 -= 1



#Add the bottom triangle

facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])



#Add the interior stern

#This may not work if there is an even number of vertices across the stern

stoppingPoint = int((len(stern_vertices) - numOuter)/2) - 1 + numOuter

count1 = numOuter + 1

count2 = len(stern_vertices) - 1

while count1 < stoppingPoint:

        facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[count2
- 1])

        facets.add_triangle(stern_vertices[count1], stern_vertices[count1 + 1],
stern_vertices[count2 - 1])



        count1 += 1

        count2 -= 1



#Add the bottom triangle

facets.add_triangle(stern_vertices[count1], stern_vertices[count2], stern_vertices[stoppingPoint
+ 1])



#The counter for the outer hull values

count3 = int(stern/step)

#The counter for the inner hull values

count4 = gvLeft_length



#Add the top stern

index = 0
```

```
while index < count3:
        facets.add_triangle(gvRight[index], gvRight[index + 1], gvRight[gvRight_length])
        facets.add_triangle(gvLeft[index], gvLeft[index + 1], gvLeft[gvLeft_length])
        index += 1

facets.add_triangle(gvRight[0], gvLeft[0], gvRight[gvRight_length])
facets.add_triangle(gvLeft[0], gvRight[gvRight_length], gvLeft[gvLeft_length])

#Add the left section of the top edge
while count4 < len(gvLeft) - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        facets.add_triangle(gvLeft[count3 + 1], gvLeft[count4], gvLeft[count4 + 1])
        count3 += 1
        count4 += 1

#outer hull
count5 = int(stern/step)
#inner hull
count6 = gvRight_length

backwards = []
#Reorganize the vertices from the inner hull's right bow
yBound = gvRight[-1][1] - step
while gvRight[-1][1] > yBound:
        backwards.append(gvRight.pop())
#They should now be in the proper order
#end of backwards order
del backwards[-1]
#The last one is a duplicate

for vertex in backwards:
        gvRight.append(vertex)
while count6 < len(gvRight) - 1:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        facets.add_triangle(gvRight[count5 + 1], gvRight[count6], gvRight[count6 + 1])
        count5 += 1
```

```
        count6 += 1

#Reorganize the vertices from the outer hull's right bow
backwards = []
counter = 0
yBound = gvRight[gvRight_length - 1][1] - step
while gvRight[gvRight_length - 1 - counter][1] > yBound:
        backwards.append(gvRight.pop(gvRight_length - 1 - counter))
        counter += 1
#end of backwards order

count7 = count5
while count7 <= gvRight_length:
        count7 += 1
#The vertices in backwards added to gvRight

counter -= 1
count7 = len(backwards) - 1
while count7 >= 0:
        gvRight.insert(gvRight_length - 1 - counter, backwards[count7])
        count7 -= 1

#Add the starboard bow
while count5 < gvRight_length - 2:
        facets.add_triangle(gvRight[count5], gvRight[count5 + 1], gvRight[count6])
        count5 += 1

#Add the port bow
while count3 < gvLeft_length - 1:
        facets.add_triangle(gvLeft[count3], gvLeft[count3 + 1], gvLeft[count4])
        count3 += 1

#Add the last triangle in the bow
facets.add_triangle(gvLeft[count3], gvLeft[count4], gvRight[count6])
facets.add_triangle(gvLeft[count3], gvRight[count5], gvRight[count6])
facets.write("hull17.stl")
```

ifs2.py

(A program to write the Sierpinski gasket file for hull16.py)

```
"""IFS algorithm taken from The Desktop Fractal Design Handbook by Michael F. Barnsley"""
#Working Python IFS

import json
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import random

#number of iterations
n = 50000000

width = 3820
height = 5820

#Define the IFS code
#This code draws the Sierpinski gasket.
code = [[.5, 0, 0, .5, 0, 0],[.5, 0, 0, .5, width/2, 0],[.5, 0, 0, .5, width/4, height/2]]

#x = width
x = 0
y = 0

def round(x):
        """Rounds the input to the nearest integer."""
        x1 = int(x)
        if x > 0:
                if ((x1 + 0.5) <= x):
                        x1 = x1 + 1
        elif x < 0:
                if ((x1 + 0.5) >= x):
                        x1 = x1 - 1
        return x1
```

```
attractor = [None]*(height + 1)
#len(attractor) is now equal to height + 1 and the last member has index height
for j in range(height + 1):
        attractor[j] = [1]*(width + 1)


print("Attractor defined")


#Find the pixels that are part of the fractal, and color them black
for i in range(n):
        r = random.randint(0, len(code) - 1)
        x = code[r][0]*x + code[r][1]*y + code[r][4]
        y = code[r][2]*x + code[r][3]*y + code[r][5]
        x = round(x)
        y = round(y)
        attractor[y][x] = 0


print("Attractor filled")


with open('ifs2.json', 'w') as f_obj:
        json.dump(attractor, f_obj)


print("Attractor saved")


#plt.imshow(attractor, origin = 'lower', extent = (0, width, 0, height), cmap = #cm.Greys_r,
interpolation='nearest')
#plt.colorbar()
#plt.show()
```

ifs4.py

(Writes the Sierpinski carpet JSON file)

```
"""IFS algorithm taken from The Desktop Fractal Design Handbook by Michael F. Barnsley"""
#Working Python IFS (Iterated Function System)
#Draws the Sierpinski carpet

import json
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import random

#number of iterations
n = 100000000

width = 7640
height = 11640
w2 = 3820
h2 = 5820
f = 1/3

#Define the IFS code
code = [[f, 0, 0, f, 0, 0],
[f, 0, 0, f, 0, height*f],
[f, 0, 0, f, width*f, height*f],
[f, 0, 0, f, width*f, 0],
[f, 0, 0, f, 0, height*f/2],
[f, 0, 0, f, width*f/2, height*f],
[f, 0, 0, f, width*f, height*f/2],
[f, 0, 0, f, width*f/2, 0]]

x = 0
y = 0

def round(x):
```

```
        """Rounds the input to the nearest integer."""
        x1 = int(x)
        if x > 0:
                if ((x1 + 0.5) <= x):
                        x1 = x1 + 1
        elif x < 0:
                if ((x1 + 0.5) >= x):
                        x1 = x1 - 1
        return x1

attractor = [None]*(h2 + 1)
#len(attractor) is now equal to height + 1 and the last member has index height
for j in range(h2 + 1):
        attractor[j] = [1]*(w2 + 1)

print("Attractor defined")

#Find the pixels that are part of the fractal, and color them black
for i in range(n):
        r = random.randint(0, len(code) - 1)
        x = code[r][0]*x + code[r][1]*y + code[r][4]
        y = code[r][2]*x + code[r][3]*y + code[r][5]
        x = round(x)
        y = round(y)
        attractor[y][x] = 0

print("Attractor filled")

with open('ifs4.json', 'w') as f_obj:
        json.dump(attractor, f_obj)

print("Attractor saved")
```

ifsPrint.py

(A program for displaying the fractals from the ifs programs)

```
"""A display program for the ifs series"""

import json

import matplotlib.pyplot as plt

import matplotlib.cm as cm


filename = input("The filename, please!\n")

#Get the fractal from the input file

with open(filename) as f_obj:

        attractor = json.load(f_obj)


print("Loaded")


#Draw the fractal

plt.imshow(attractor, origin = 'lower', extent = (0, len(attractor) - 1, 0, len(attractor[0]) -
1), cmap = cm.Greys_r, interpolation='nearest')

plt.show()
```

Appendix II: Data

The tables are organized in the following order:

Trials 1-6	Trials 7-13	Trials 14-18	Trials 19-23	Trials 24-28	Trials 29-33	Trials 34-38
Simple Parabola	Simple Spline	Spline With Holes	Noise Hull (With Hole)	Sierpinski Carpet	Noise Hull (No Hole)	Sierpinski Gasket

Trial 1a			Trial 1b			Trial 1c			Trial 2			Trial 3		
t (s)	v (m/s)	a (m/s2)	t	v	a	t	v	a	t	v	a	t	v	a
1.48	0.04	0.42	6.492	0.01	0	2.683	0.07	0.57	0.035	0.42	0.73	0.03	0.61	-0.5
1.723	0.19	0.85	9.501	0	0.000431	2.835	0.17	0.77	0.07	0.45	0.67	0.054	0.6	0.1
1.792	0.25	0.88	12.26	0.01	0.08	2.911	0.23	0.83	0.103	0.47	0.64	0.079	0.61	0.36
1.847	0.3	0.81	13.21	0.12	0.64	2.97	0.28	0.81	0.134	0.49	0.42	0.103	0.62	0.05
1.894	0.34	0.77	13.314	0.19	0.7	3.019	0.32	0.71	0.164	0.5	0.33	0.127	0.62	0.11
1.937	0.37	0.63	13.385	0.24	0.69	3.064	0.35	0.66	0.194	0.51	0.28	0.151	0.63	0.27
1.977	0.39	0.53	13.444	0.28	0.69	3.105	0.38	0.62	0.224	0.51	0.34	0.175	0.63	-0.16
2.014	0.41	0.51	13.495	0.31	0.66	3.144	0.4	0.5	0.253	0.53	0.48	0.199	0.62	0.23
2.05	0.43	0.42	13.54	0.34	0.66	3.181	0.42	0.47	0.281	0.54	0.38	0.223	0.64	0.27
2.085	0.44	0.48	13.583	0.37	0.67	3.216	0.43	0.49	0.308	0.55	0.49	0.247	0.63	-0.12
2.118	0.46	0.63	13.622	0.4	0.58	3.25	0.45	0.51	0.335	0.57	0.47	0.27	0.64	0.35
2.15	0.48	0.49	13.659	0.42	0.51	3.283	0.47	0.47	0.361	0.57	0.08	0.294	0.65	0.17
2.181	0.49	0.37	13.694	0.43	0.5	3.314	0.48	0.36	0.388	0.57	-2.12E-14	0.317	0.64	-2.63E-14
2.211	0.5	0.36	13.728	0.45	0.51	3.345	0.49	0.4	0.414	0.57	0.35	0.34	0.65	0.24
2.241	0.51	0.33	13.761	0.47	0.52	3.375	0.51	0.33	0.44	0.59	0.35	0.363	0.66	2.74E-14
2.27	0.52	0.38	13.792	0.48	0.34	3.405	0.51	0.3	0.465	0.59	0.28	0.386	0.65	0.13
2.298	0.53	0.33	13.823	0.49	0.38	3.434	0.52	0.38	0.49	0.6	0.24	0.409	0.66	0.19
2.326	0.54	0.28	13.853	0.51	0.45	3.462	0.53	0.37	0.515	0.6	0.15	0.432	0.66	-0.13
2.354	0.55	0.3	13.882	0.52	0.44	3.49	0.54	0.28	0.54	0.61	0.31	0.455	0.66	-0.06
2.381	0.56	0.48	13.911	0.53	0.42	3.518	0.55	0.11	0.564	0.62	0.26	0.477	0.66	0.19
2.407	0.57	0.41	13.939	0.54	0.21	3.545	0.55	0.27	0.588	0.62	0.11	0.5	0.66	0.06
2.433	0.58	0.09	13.967	0.54	0.14	3.572	0.56	0.47	0.612	0.62	0.11	0.523	0.66	5.32E-14
2.459	0.58	0.32	13.994	0.55	0.38	3.598	0.57	0.16	0.636	0.63	0.05	0.546	0.66	0.13
2.485	0.6	0.46	14.021	0.56	0.42	3.625	0.57	0.17	0.66	0.63	6.04E-14	0.568	0.66	1.03E-27
2.51	0.6	0.05	14.048	0.57	0.12	3.651	0.58	0.48	0.684	0.63	0.17	0.591	0.66	0.07
2.535	0.6	0.2	14.074	0.57	0.21	3.676	0.6	0.13	0.708	0.64	0.22	0.613	0.67	0.07
2.559	0.61	0.5	14.1	0.58	0.44	3.701	0.59	0.05	0.731	0.64	1.13E-13	0.636	0.67	0.13

2.584	0.62	0.21	14.126	0.59	0.13	3.727	0.6	0.33	0.755	0.64	0.3	0.658	0.67	-0.13
2.608	0.62	0.28	14.151	0.59	0	3.752	0.61	0.29	0.778	0.65	0.06	0.681	0.66	-1.33E-15
2.632	0.64	0.39	14.176	0.59	0.48	3.776	0.61	0.1	0.801	0.64	0.13	0.703	0.67	0.2
2.655	0.64	-0.06	14.201	0.61	0.19	3.801	0.61	0.16	0.825	0.66	0.42	0.726	0.67	0.07
2.679	0.63	0.24	14.226	0.6	-0.05	3.825	0.62	0.37	0.847	0.66	-1.03E-27	0.748	0.68	0.07
2.702	0.65	0.47	14.251	0.61	0.36	3.849	0.63	0	0.87	0.66	0.13	0.77	0.67	-0.07
2.725	0.66	0	14.275	0.62	0.1	3.873	0.62	0.11	0.893	0.66	-0.06	0.792	0.67	0.07
2.748	0.65	0.06	14.3	0.61	-0.05	3.897	0.64	0.39	0.916	0.66	-3.38E-14	0.815	0.68	-0.07
2.771	0.66	0.25	14.324	0.62	0.21	3.92	0.64	1.08E-15	0.938	0.66	0.19	0.837	0.67	-0.07
2.794	0.66	0.06	14.348	0.63	0.16	3.944	0.64	0.12	0.961	0.66	-0.06	0.859	0.67	0.07
2.816	0.66	0.13	14.372	0.62	0	3.967	0.64	0.23	0.984	0.66	0.2	0.882	0.67	7.72E-14
2.839	0.67	0.13	14.396	0.63	0.17	3.991	0.65	0.06	1.006	0.67	-0.06	0.904	0.67	0
2.861	0.67	2.94E-13	14.42	0.63	0.22	4.014	0.65	-0.06	1.029	0.66	0.21	0.926	0.67	-0.07
2.884	0.67	0.41	14.444	0.64	0.11	4.037	0.64	0.18	1.051	0.68	0.26	0.949	0.67	0.35
2.906	0.68	0.13	14.467	0.64	0.17	4.06	0.66	0.12	1.073	0.67	-0.07	0.971	0.69	0.13
2.928	0.68	0	14.49	0.64	0.06	4.083	0.65	-0.18	1.095	0.68	0.2	0.993	0.68	1.44E-15
2.95	0.68	0.21	14.514	0.64	0.06	4.106	0.65	0.25	1.118	0.68	0.07	1.015	0.69	0.21
2.972	0.68	-0.14	14.537	0.65	0.18	4.129	0.66	0.18	1.14	0.68	0.14	1.037	0.68	1.62E-13
2.994	0.68	0.07	14.56	0.65	0.12	4.152	0.66	0.06	1.162	0.68	-0.21	1.058	0.69	0.07
3.016	0.69	-0.14	14.583	0.65	-1.08E-12	4.175	0.66	0.13	1.184	0.67	-0.14	1.08	0.69	-0.07
3.038	0.67	-0.07	14.606	0.65	-1.42E-14	4.198	0.66	5.68E-13	1.206	0.68	-0.26	1.102	0.68	0
3.06	0.68	0.14	14.629	0.65	0.12	4.22	0.66	0.06	1.228	0.66	-1.11E-16	1.124	0.69	-0.21
3.082	0.68	-0.14	14.652	0.66	0.06	4.243	0.66	-0.06	1.251	0.68	0.06	1.146	0.68	1.14E-14
3.104	0.68	0.07	14.675	0.66	0.06	4.265	0.66	0.07	1.273	0.66	-0.27	1.168	0.69	-1.02E-13
3.126	0.68	-0.07	14.698	0.66	0.06	4.288	0.67	0.13	1.296	0.67	0.27	1.19	0.68	1.13E-14
3.148	0.68	0.22	14.721	0.66	0.13	4.311	0.67	1.09E-13	1.318	0.68	-0.06	1.212	0.69	0.07
3.17	0.69	0.14	14.744	0.66	-1E-14	4.333	0.67	0.07	1.34	0.66	0.07	1.234	0.68	-0.07
3.192	0.68	-0.15	14.766	0.66	0.13	4.355	0.67	0.2	1.363	0.68	-1.34E-13	1.256	0.68	0.21
3.214	0.68	0.14	14.789	0.67	0.13	4.378	0.68	0.13	1.385	0.66	0	1.278	0.69	2.19E-15
3.236	0.69	-5.83E-15	14.811	0.66	-1.04E-14	4.4	0.68	-0.07	1.407	0.68	0.06	1.3	0.68	0.07
3.258	0.68	1.99E-13	14.834	0.67	0.07	4.422	0.67	0.07	1.43	0.67	-0.21	1.321	0.69	-0.07
3.28	0.69	-0.21	14.856	0.67	-0.07	4.444	0.68	-0.07	1.452	0.67	0.07	1.343	0.68	-0.15
3.302	0.68	-0.14	14.879	0.67	0.2	4.467	0.67	0.07	1.474	0.67	-0.26	1.365	0.68	0.07
3.324	0.68	3.06E-13	14.901	0.68	0.07	4.489	0.68	-0.07	1.497	0.66	1.02E-14	1.387	0.68	0.07
3.346	0.68	-3.11E-13	14.924	0.67	4.31E-14	4.511	0.67	-0.21	1.52	0.67	-1.32E-13	1.409	0.69	-1.62E-13
3.368	0.68	-0.07	14.946	0.68	0.07	4.534	0.67	0	1.542	0.66	0.07	1.431	0.68	-0.07
3.39	0.67	-0.07	14.968	0.67	-0.14	4.556	0.67	-0.07	1.565	0.67	0.13	1.453	0.68	0
3.412	0.68	0.14	14.99	0.67	0.21	4.578	0.67	0.13	1.587	0.66	-0.07	1.475	0.68	-0.07
3.434	0.68	-0.14	15.013	0.68	1.98E-13	4.601	0.67	-0.13	1.61	0.67	0.27	1.497	0.68	1.18E-13
3.456	0.67	0.07	15.035	0.67	-0.07	4.623	0.66	0.07	1.632	0.68	-0.13	1.518	0.68	1.57E-13
3.479	0.68	-0.13	15.057	0.68	0.13	4.646	0.68	0	1.655	0.66	0.21	1.54	0.68	-0.07

3.501	0.67	-0.14	15.079	0.68	-0.07	4.668	0.66	-0.07	1.677	0.68	0.06	1.562	0.68	-0.07
3.523	0.68	2.83E-13	15.101	0.68	0.07	4.69	0.67	0.13	1.699	0.67	-0.07	1.584	0.68	0
3.546	0.67	-0.27	15.124	0.68	0.07	4.713	0.67	-0.13	1.721	0.68	-1.01E-14	1.607	0.68	0.14
3.568	0.66	0.07	15.146	0.68	0.07	4.735	0.67	0.21	1.743	0.67	-0.14	1.628	0.68	-2.3E-13
3.591	0.67	-0.06	15.168	0.68	-0.07	4.758	0.68	2.22E-13	1.766	0.68	6.61E-14	1.65	0.68	0.07
3.613	0.66	0.07	15.19	0.68	0.14	4.78	0.67	-3.23E-14	1.788	0.67	-0.07	1.672	0.69	0
3.636	0.67	-2.86E-13	15.212	0.69	1.21E-12	4.802	0.68	0.07	1.811	0.67	0.13	1.694	0.68	-0.07
3.658	0.66	-0.07	15.234	0.68	-1.37E-12	4.824	0.67	-0.07	1.833	0.67	0.07	1.716	0.68	0
3.681	0.67	0.13	15.256	0.69	0.07	4.847	0.68	0	1.855	0.68	0.14	1.738	0.68	-0.07
3.703	0.67	0.07	15.278	0.68	-0.14	4.869	0.67	-0.2	1.877	0.68	0.07	1.76	0.68	0
3.726	0.67	0.13	15.3	0.68	0.07	4.891	0.67	-0.07	1.899	0.68	0.14	1.782	0.68	-0.14
3.748	0.67	-0.2	15.322	0.68	1.48E-12	4.914	0.67	-0.19	1.921	0.68	0.14	1.804	0.68	-0.07
3.77	0.66	0.14	15.344	0.68	0	4.937	0.66	-0.07	1.943	0.68	-1.57E-13	1.826	0.68	0.14
3.793	0.68	-0.06	15.366	0.68	-0.07	4.959	0.66	-0.06	1.965	0.68	0.07	1.848	0.68	7.22E-16
3.815	0.66	-0.21	15.388	0.68	0	4.982	0.66	-0.06	1.987	0.69	-0.07	1.87	0.68	0
3.838	0.67	0.06	15.41	0.68	-0.07	5.005	0.66	0.13	2.009	0.68	-0.07	1.892	0.68	-6.94E-16
3.86	0.66	-0.13	15.432	0.68	-0.07	5.028	0.66	3.05E-26	2.031	0.68	-0.14	1.915	0.68	0
3.883	0.66	0.07	15.454	0.68	0.14	5.05	0.66	0.2	2.053	0.68	-2.29E-13	1.937	0.68	-2.26E-13
3.905	0.67	-0.06	15.476	0.68	-0.07	5.073	0.67	0.13	2.075	0.68	0.07	1.959	0.68	-0.07
3.928	0.66	7.66E-15	15.498	0.68	0.14	5.095	0.67	0.07	2.097	0.68	-0.07	1.981	0.68	-5.05E-29
3.95	0.67	0.06	15.52	0.69	9.1E-14	5.118	0.67	2.12E-14	2.119	0.68	0.07	2.003	0.68	5.05E-29
3.973	0.66	0.13	15.542	0.68	0.07	5.14	0.67	0.14	2.141	0.68	-0.14	2.025	0.68	0.07
3.995	0.67	0.13	15.564	0.69	0.21	5.162	0.68	-0.06	2.163	0.68	-0.14	2.047	0.68	0.14
4.018	0.67	0.07	15.586	0.68	-0.07	5.185	0.66	-0.14	2.185	0.68	-0.07	2.069	0.68	-3.15E-13
4.04	0.68	0.13	15.608	0.69	-0.07	5.207	0.67	-3.15E-14	2.208	0.67	3.1E-13	2.091	0.68	0.07
4.062	0.68	-0.13	15.63	0.68	-0.21	5.23	0.66	-0.07	2.23	0.68	0.07	2.113	0.69	0.07
4.085	0.67	-2.24E-13	15.652	0.68	-2.29E-13	5.252	0.67	0.2	2.252	0.68	-0.07	2.135	0.68	-5.6E-13
4.107	0.68	2.38E-13	15.674	0.68	-0.14	5.274	0.67	2.21E-13	2.274	0.67	5.34E-13	2.157	0.69	0.07
4.129	0.67	0	15.696	0.67	0	5.297	0.67	0.07	2.297	0.68	0	2.178	0.69	3.21E-13
4.152	0.68	-0.07	15.718	0.68	0.07	5.319	0.68	-0.07	2.319	0.67	0.07	2.2	0.69	0.07
4.174	0.67	-0.14	15.74	0.68	0.07	5.341	0.67	-0.07	2.341	0.68	0.07	2.222	0.69	-0.07
4.196	0.67	-5.86E-13	15.762	0.68	0.14	5.364	0.67	2.12E-14	2.363	0.68	-0.07	2.244	0.68	-0.07
4.219	0.67	0.07	15.784	0.68	0	5.386	0.67	-0.07	2.385	0.68	0.14	2.266	0.69	0.07
4.241	0.67	0.07	15.806	0.68	-1.24E-12	5.409	0.67	1.05E-14	2.407	0.68	0.07	2.287	0.69	-0.07
4.264	0.67	-0.07	15.828	0.68	-0.14	5.431	0.67	-0.13	2.429	0.68	0.07	2.309	0.68	-4.36E-15
4.286	0.67	0.14	15.85	0.68	-0.07	5.454	0.66	0.13	2.451	0.68	-2.3E-13	2.331	0.69	-2.31E-13
4.308	0.68	2.22E-13	15.872	0.68	-0.07	5.476	0.67	0.07	2.473	0.68	-0.07	2.353	0.68	0.07
4.331	0.67	1.62E-14	15.895	0.68	0	5.499	0.67	-0.07	2.495	0.68	1.39E-15	2.375	0.69	0.07
4.353	0.68	0.07	15.917	0.68	-0.07	5.521	0.67	-5.86E-13	2.518	0.68	-0.07	2.396	0.69	-0.07
4.375	0.67	-6.06E-13	15.939	0.67	-0.14	5.543	0.67	0	2.54	0.68	0.07	2.418	0.69	-0.07
4.397	0.68	0.07	15.961	0.67	0	5.566	0.67	0.07	2.562	0.68	0.07	2.44	0.68	-0.14

4.419	0.68	-0.13	15.983	0.67	0.14	5.588	0.67	0.07	2.584	0.68	3.13E-13	2.462	0.68	-0.07
4.442	0.67	0.07	16.006	0.68	0.14	5.611	0.67	3.75E-13	2.606	0.68	0.07	2.484	0.68	-2.3E-13
4.464	0.68	0.07	16.028	0.68	-1.01E-12	5.633	0.67	-0.07	2.628	0.68	0.07	2.506	0.68	-0.07
4.486	0.67	-0.14	16.05	0.68	0.07	5.655	0.67	-0.07	2.65	0.68	1.87E-26	2.528	0.68	-0.07
4.509	0.67	0.07	16.072	0.68	0	5.678	0.67	-1.05E-14	2.671	0.68	0.07	2.55	0.68	-0.27
4.531	0.68	0.07	16.094	0.68	-0.14	5.7	0.67	-0.07	2.693	0.69	5.83E-15	2.572	0.67	-0.59
4.553	0.68	0.07	16.116	0.68	-0.07	5.723	0.66	-0.07	2.715	0.68	3.26E-13	2.595	0.65	-1.96
4.575	0.68	0.07	16.138	0.68	-0.07	5.745	0.67	-5.86E-13	2.737	0.69	0	2.62	0.57	-1.03
4.597	0.68	0.07	16.161	0.67	-0.07	5.768	0.66	-0.07	2.759	0.68	0.15	2.646	0.6	-1.2
4.619	0.68	0.07	16.183	0.67	-0.07	5.79	0.66	-0.06	2.781	0.69	0.07	2.673	0.49	0.67
4.641	0.68	6.26E-13	16.205	0.67	0.07	5.813	0.66	-0.19	2.802	0.69	-0.15	2.7	0.64	4.4
4.663	0.68	4.8E-26	16.227	0.68	0.13	5.836	0.66	-0.13	2.824	0.69	3.21E-13	2.722	0.72	-1.46
4.685	0.68	-6.26E-13	16.25	0.68	0	5.859	0.66	5.55E-13	2.846	0.69	-0.07	2.747	0.55	-1.76
4.707	0.68	0.07	16.272	0.68	4.26E-25	5.882	0.66	-5.05E-29	2.868	0.68	-0.07	2.772	0.64	0.79
4.729	0.68	0.07	16.294	0.68	-0.13	5.905	0.66	5.05E-29	2.89	0.68	-0.07	2.796	0.59	-0.62
4.751	0.68	-0.07	16.316	0.67	-0.14	5.928	0.66	0.06	2.912	0.68	-0.07	2.821	0.61	0.14
4.773	0.68	0.07	16.339	0.67	-0.13	5.95	0.66	0	2.934	0.68	-1.11E-26	2.846	0.6	-0.4
4.795	0.69	0.14	16.361	0.66	-0.13	5.973	0.66	-2.14E-13	2.956	0.68	-0.07	2.871	0.6	-0.05
4.817	0.69	8.76E-13	16.384	0.66	0.07	5.996	0.66	-0.06	2.978	0.68	5.45E-13	2.896	0.6	-0.27
4.839	0.69	0.15	16.406	0.67	0.07	6.019	0.65	-0.06	3	0.68	1.39E-15	2.921	0.58	1.68E-14
4.86	0.69	0.07	16.429	0.67	0.07	6.042	0.66	0	3.022	0.68	-0.14	2.947	0.6	0.04
4.882	0.69	6.5E-13	16.451	0.67	0.13	6.065	0.65	-0.06	3.044	0.68	-0.14	2.972	0.58	-0.05
4.904	0.69	0.07	16.474	0.67	-2.6E-12	6.088	0.65	0.06	3.066	0.67	-0.14	2.998	0.6	0.09
4.925	0.69	-4.24E-13	16.496	0.67	2.61E-12	6.111	0.66	7.55E-13	3.088	0.67	-0.2	3.023	0.59	-0.23
4.947	0.69	-0.07	16.518	0.67	1.78E-13	6.134	0.65	0.13	3.111	0.66	-0.44	3.049	0.58	-0.05
4.968	0.69	-0.22	16.541	0.67	-0.2	6.157	0.66	0.19	3.134	0.65	-0.98	3.074	0.59	-0.17
4.99	0.68	-0.22	16.563	0.66	-0.13	6.179	0.66	-0.06	3.158	0.62	-2.61	3.1	0.57	-0.18
5.012	0.68	-0.14	16.586	0.66	4.35E-13	6.202	0.66	-0.06	3.184	0.51	-0.56	3.126	0.58	-0.12
5.034	0.68	-0.07	16.608	0.66	2.31E-12	6.225	0.66	-0.06	3.212	0.59	1.26	3.152	0.57	-0.25
5.056	0.68	-4.55E-13	16.631	0.66	-4.35E-13	6.248	0.66	-0.36	3.237	0.58	-1.23	3.179	0.56	-0.04
5.078	0.68	0	16.654	0.66	0.07	6.271	0.64	-0.9	3.265	0.52	-2.83	3.205	0.57	1.76E-13
5.101	0.68	4.55E-13	16.676	0.67	0.13	6.295	0.61	-1.99	3.349	0.11	0.31	3.232	0.56	0.04
5.123	0.68	-0.27	16.699	0.67	0.07	6.321	0.54	-2.51	3.432	0.56		3.258	0.57	0.04
5.145	0.67	-1.31	16.721	0.67	-4.41E-13	6.351	0.47	-2.68				3.285	0.57	-0.16
5.168	0.62	-2.71	16.743	0.67	-0.26	6.388	0.35	-0.32				3.312	0.56	-0.12
5.195	0.53	-0.16	16.766	0.66	-1.01	6.426	0.45	2.27				3.338	0.56	-0.11
5.221	0.61	1.29	16.79	0.62	-3.34	6.457	0.51	0.77				3.365	0.55	-0.23
5.246	0.6	0.98	16.818	0.47	-0.11	6.583	0.07	-0.56				3.393	0.55	-0.15
5.27	0.66	-1.79	16.846	0.62	3.64	6.713	0.4	0.49				3.42	0.55	-0.11
5.315	0.22	-2.5	16.869	0.67	1.36	6.751	0.39	0				3.448	0.54	-0.14
5.365	0.48	-0.01	17.016	0.06	-0.95	6.789	0.4	0.11				3.475	0.54	-4.46E-13

5.415	0.22	-0.29	17.169	0.43	0.11	6.826	0.4	0.06				3.503	0.54	0.07
5.465	0.46	2	17.204	0.41	-0.26	6.864	0.4	-0.03				3.531	0.54	-0.04
5.498	0.47	0.02	17.241	0.41	0.03	6.902	0.4	-0.11				3.559	0.54	-0.1
5.53	0.46	-0.27	17.278	0.41	0.11	6.94	0.39	-0.16				3.587	0.54	-0.14
5.563	0.45	-0.21	17.314	0.42	0.05	6.978	0.39	-0.04				3.615	0.53	-0.13
5.597	0.45	-0.08	17.35	0.42	-0.05	7.017	0.39	-0.16				3.643	0.53	-0.16
5.631	0.44	-0.1	17.386	0.41	-0.14	7.056	0.37	-0.28				3.672	0.52	
5.665	0.44	-0.13	17.422	0.41	-0.14	7.097	0.37	-0.15						
5.699	0.43	-0.06	17.459	0.4	-0.14	7.138	0.36	-0.04						
5.734	0.43	0.02	17.497	0.4	-0.17	7.18	0.36	0.05						
5.768	0.43	0.02	17.535	0.39	-0.12	7.221	0.36	-0.01						
5.803	0.44	-0.02	17.573	0.39	-0.09	7.262	0.36	-0.04						
5.837	0.43	-0.12	17.612	0.38	-0.03	7.303	0.36	-0.07						
5.872	0.43	-0.17	17.651	0.39	-5.9E-13	7.345	0.36	-0.12						
5.907	0.42	-0.17	17.69	0.38	-0.01	7.388	0.35	-0.15						
5.943	0.42	-0.16	17.729	0.38	1.4E-13	7.431	0.34	-0.17						
5.979	0.41	-0.12	17.768	0.38	-0.06	7.475	0.34	-0.1						
6.016	0.41	-0.1	17.808	0.38	-0.12	7.52	0.33	-0.07						
6.053	0.4	-0.04	17.847	0.37	-0.16	7.565	0.33	-0.02						
6.091	0.4	0.01	17.888	0.37	-0.18	7.61	0.33	0.02						
6.128	0.4	-0.01	17.929	0.36	-0.13	7.655	0.33	-0.02						
6.165	0.4		17.971	0.36	-0.1	7.701	0.33	-0.03						
			18.014	0.35	-0.03	7.746	0.33	-0.07						
			18.056	0.35	0.02	7.792	0.32	-0.12						
			18.099	0.35	3.52E-13	7.839	0.32	-0.14						
			18.141	0.35	-0.02	7.887	0.31	-0.13						
			18.184	0.35	-0.08	7.935	0.3	-0.11						
			18.227	0.35	-0.11	7.985	0.3	-0.08						
			18.27	0.34	-0.15	8.035	0.3	-0.01						
			18.314	0.33	-0.17	8.086	0.3	0.01						
			18.36	0.33	-0.13	8.136	0.3	-0.02						
			18.406	0.32	-0.08	8.186	0.3	-0.03						
			18.453	0.32	-0.01	8.237	0.29	-0.09						
			18.5	0.32	0.03	8.288	0.29	-0.13						
			18.547	0.32	0.01	8.341	0.28	-0.14						
			18.593	0.32	-0.01	8.395	0.27	-0.14						
			18.64	0.32	-0.06	8.451	0.27	-0.11						
			18.687	0.32	-0.11	8.508	0.26	-0.07						
			18.735	0.31	-0.15	8.566	0.26	-0.01						
			18.784	0.3	-0.16	8.624	0.26	0.01						
			18.835	0.29	-0.13	8.681	0.26	-0.02						

			18.886	0.29	-0.09	8.739	0.26	2.28						
			18.939	0.28	-0.02	8.785	0.45	684.57						
			18.991	0.29	0.02	8.803	16.67	632.27						
			19.044	0.29	0	8.855	0.14	-282.72						
			19.096	0.29	-0.02	9.36	0.02							
			19.148	0.29	-0.06									
			19.202	0.28	-0.12									
			19.256	0.27	-0.15									
			19.312	0.26	-0.16									
			19.37	0.26	-0.24									
			19.681	0.03										

Trial 4			Trial 5			Trial 6			Trial 8a			Trial 8b		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.022	0.45	0.18	0.038	0.51	-0.02	0.033	0.44	0.92	0.033	0.44	0.92	0.264	0.13	0.98
0.055	0.46	0.33	0.067	0.51	0.38	0.066	0.47	0.65	0.066	0.47	0.65	0.357	0.22	1.05
0.087	0.47	0.46	0.096	0.53	0.45	0.098	0.48	0.52	0.098	0.48	0.52	0.417	0.29	1.12
0.118	0.49	0.47	0.124	0.54	0.24	0.128	0.5	0.37	0.128	0.5	0.37	0.465	0.34	1.03
0.148	0.5	0.49	0.152	0.54	0.33	0.158	0.51	0.45	0.158	0.51	0.45	0.507	0.38	0.8
0.177	0.52	0.36	0.179	0.56	0.5	0.187	0.52	0.39	0.187	0.52	0.39	0.545	0.41	0.73
0.206	0.52	0.29	0.205	0.57	0.41	0.216	0.53	0.41	0.216	0.53	0.41	0.581	0.43	0.64
0.234	0.54	0.37	0.231	0.58	0.3	0.244	0.55	0.65	0.244	0.55	0.65	0.615	0.45	0.67
0.262	0.55	0.32	0.257	0.59	0.32	0.271	0.56	0.51	0.271	0.56	0.51	0.647	0.48	0.73
0.289	0.55	0.26	0.283	0.6	0.28	0.297	0.57	0.33	0.297	0.57	0.33	0.678	0.5	0.66
0.316	0.56	0.27	0.308	0.6	0.14	0.323	0.58	0.26	0.323	0.58	0.26	0.707	0.52	0.54
0.343	0.57	0.37	0.333	0.6	0.1	0.349	0.59	0.27	0.349	0.59	0.27	0.736	0.53	0.46
0.369	0.58	0.34	0.358	0.6	0.3	0.374	0.6	0.23	0.374	0.6	0.23	0.764	0.54	0.46
0.395	0.59	0.18	0.382	0.62	0.36	0.399	0.6	0.29	0.399	0.6	0.29	0.791	0.56	0.42
0.42	0.59	0.18	0.406	0.62	0.21	0.424	0.61	0.4	0.424	0.61	0.4	0.818	0.57	0.45
0.446	0.6	0.18	0.43	0.63	0.22	0.448	0.62	0.42	0.448	0.62	0.42	0.844	0.58	0.47
0.471	0.6	0.14	0.454	0.63	0.11	0.472	0.63	0.39	0.472	0.63	0.39	0.87	0.59	0.56
0.496	0.6	0.4	0.478	0.63	0.11	0.496	0.64	0.17	0.496	0.64	0.17	0.895	0.61	0.43
0.52	0.62	0.3	0.501	0.64	0.17	0.52	0.64	0.18	0.52	0.64	0.18	0.92	0.61	0.2
0.545	0.62	0.05	0.525	0.64	0.06	0.543	0.65	0.18	0.543	0.65	0.18	0.944	0.62	0.37
0.569	0.62	0.21	0.548	0.64	0.06	0.566	0.65	0.06	0.566	0.65	0.06	0.968	0.63	0.44
0.593	0.63	0.16	0.572	0.64	0.12	0.589	0.65	0.12	0.589	0.65	0.12	0.992	0.64	0.17
0.617	0.63	0.06	0.595	0.65	0.12	0.612	0.65	0.32	0.612	0.65	0.32	1.015	0.64	0.12

0.641	0.63	-0.05	0.618	0.65	0	0.635	0.66	0.45	0.635	0.66	0.45	1.039	0.64	0.3
0.665	0.62	0.06	0.641	0.65	-0.06	0.658	0.67	0.13	0.658	0.67	0.13	1.062	0.65	0.12
0.689	0.63	0.17	0.664	0.65	-6.46E-14	0.68	0.67	0.07	0.68	0.67	0.07	1.085	0.65	0.19
0.712	0.63	-0.06	0.687	0.65	5.02E-14	0.702	0.68	0.07	0.702	0.68	0.07	1.108	0.66	0.38
0.736	0.63	0.11	0.711	0.65	0.12	0.724	0.67	0.14	0.724	0.67	0.14	1.13	0.67	-5.4E-14
0.76	0.64	0.17	0.734	0.65	0.06	0.747	0.68	0.14	0.747	0.68	0.14	1.153	0.66	0.2
0.783	0.64	4.96E-14	0.757	0.65	0.06	0.769	0.68	0.07	0.769	0.68	0.07	1.175	0.68	0.19
0.807	0.64	0.06	0.78	0.66	0.12	0.791	0.68	0.21	0.791	0.68	0.21	1.198	0.67	0.07
0.83	0.64	0.12	0.803	0.66	-0.06	0.812	0.69	0.07	0.812	0.69	0.07	1.22	0.68	0.34
0.853	0.64	0.06	0.826	0.65	7.1E-14	0.834	0.69	0.22	0.834	0.69	0.22	1.242	0.68	4.27E-15
0.877	0.64	0.12	0.849	0.66	1.39E-17	0.856	0.7	0.07	0.856	0.7	0.07	1.264	0.68	0.07
0.9	0.65	0.06	0.872	0.65	-1.75E-13	0.878	0.69	-1.19E-13	0.878	0.69	-1.19E-13	1.286	0.69	-1.2E-13
0.923	0.65	-0.12	0.895	0.66	0.06	0.899	0.7	0.07	0.899	0.7	0.07	1.308	0.68	-0.07
0.946	0.64	0.12	0.917	0.66	1.04E-13	0.921	0.69	0.08	0.921	0.69	0.08	1.33	0.68	0.14
0.969	0.65	0.06	0.94	0.66	0.06	0.942	0.7	0.15	0.942	0.7	0.15	1.352	0.68	0
0.993	0.65	6.73E-14	0.963	0.66	0.19	0.964	0.7	-0.08	0.964	0.7	-0.08	1.374	0.68	0.29
1.016	0.65	0.12	0.986	0.66	0.06	0.985	0.7	0.24	0.985	0.7	0.24	1.395	0.7	0.29
1.039	0.65	-0.12	1.009	0.66	-0.06	1.006	0.71	0.23	1.006	0.71	0.23	1.417	0.7	0.08
1.062	0.65	0.13	1.031	0.66	0.2	1.027	0.71	0.08	1.027	0.71	0.08	1.438	0.7	0.23
1.085	0.66	0.25	1.054	0.67	-0.06	1.049	0.71	0	1.049	0.71	0	1.459	0.71	0.15
1.108	0.66	2.48E-13	1.076	0.66	1.08E-13	1.07	0.71	0.08	1.07	0.71	0.08	1.481	0.71	-0.08
1.13	0.66	-1.41E-13	1.099	0.67	1.32E-13	1.091	0.72	0.16	1.091	0.72	0.16	1.502	0.7	-0.08
1.153	0.66	-0.06	1.122	0.66	-0.13	1.112	0.71	0	1.112	0.71	0	1.523	0.7	-0.08
1.176	0.66	0.06	1.144	0.66	0.13	1.133	0.72	1.8E-13	1.133	0.72	1.8E-13	1.545	0.7	0
1.199	0.66	0.19	1.167	0.66	-0.13	1.154	0.71	-0.16	1.154	0.71	-0.16	1.566	0.7	0.16
1.221	0.66	0.06	1.19	0.66	-0.13	1.175	0.71	0.25	1.175	0.71	0.25	1.587	0.71	-0.08
1.244	0.66	-0.06	1.212	0.66	-0.06	1.196	0.72	0.08	1.196	0.72	0.08	1.608	0.7	0
1.267	0.66	0.13	1.235	0.66	-2.5E-13	1.216	0.71	-0.08	1.216	0.71	-0.08	1.63	0.71	0.08
1.289	0.67	0.06	1.258	0.66	-0.06	1.237	0.72	1.81E-13	1.237	0.72	1.81E-13	1.651	0.7	0.16
1.312	0.66	-0.07	1.281	0.65	0.13	1.258	0.71	-6.02E-14	1.258	0.71	-6.02E-14	1.672	0.71	1.23E-13
1.334	0.67	0.07	1.304	0.66	0.19	1.279	0.72	-1.78E-15	1.279	0.72	-1.78E-15	1.693	0.7	-0.16
1.357	0.67	-0.13	1.326	0.66	0	1.3	0.71	-0.08	1.3	0.71	-0.08	1.715	0.71	0.08
1.379	0.66	1.45E-13	1.349	0.66	0.2	1.321	0.72	0.08	1.321	0.72	0.08	1.736	0.71	-0.23
1.402	0.67	0.06	1.372	0.67	0	1.342	0.72	-0.08	1.342	0.72	-0.08	1.757	0.7	0.08
1.425	0.66	0.13	1.394	0.66	-0.07	1.363	0.71	0.08	1.363	0.71	0.08	1.778	0.71	-0.07
1.447	0.67	0.13	1.417	0.67	0	1.384	0.72	0.08	1.384	0.72	0.08	1.8	0.69	0.08
1.469	0.67	0.07	1.439	0.66	-0.13	1.405	0.72	0.17	1.405	0.72	0.17	1.821	0.71	0.07
1.492	0.68	0.07	1.462	0.66	0	1.425	0.73	-0.08	1.425	0.73	-0.08	1.842	0.7	-0.08
1.514	0.67	-0.07	1.485	0.66	-0.06	1.446	0.71	-0.09	1.446	0.71	-0.09	1.864	0.71	0.15
1.536	0.67	0.07	1.507	0.66	0.07	1.467	0.72	0.16	1.467	0.72	0.16	1.885	0.7	-0.16
1.558	0.68	-0.13	1.53	0.67	0.13	1.488	0.72	-0.08	1.488	0.72	-0.08	1.906	0.7	0.08

1.581	0.67	0.07	1.552	0.66	0.2	1.508	0.72	-3.15E-13	1.508	0.72	-3.15E-13	1.927	0.71	-0.23
1.603	0.68	-0.06	1.575	0.68	0.13	1.529	0.72	-0.08	1.529	0.72	-0.08	1.949	0.69	0.08
1.626	0.66	-0.07	1.597	0.67	0.07	1.55	0.72	-0.08	1.55	0.72	-0.08	1.97	0.71	-0.07
1.648	0.68	0.27	1.619	0.68	0.07	1.571	0.72	-0.08	1.571	0.72	-0.08	1.992	0.69	-0.16
1.67	0.68	0.07	1.642	0.67	0.07	1.592	0.71	0	1.592	0.71	0	2.013	0.7	0.07
1.692	0.68	0.14	1.664	0.68	0.07	1.613	0.72	0	1.613	0.72	0	2.035	0.69	-0.3
1.714	0.68	0.14	1.686	0.68	-0.14	1.634	0.71	-0.08	1.634	0.71	-0.08	2.056	0.69	0.08
1.736	0.68	0.14	1.708	0.68	0.07	1.655	0.71	-7.14E-14	1.655	0.71	-7.14E-14	2.078	0.7	-2.36E-13
1.758	0.69	-0.07	1.73	0.68	-0.13	1.676	0.71	7.14E-14	1.676	0.71	7.14E-14	2.099	0.69	-9.51E-14
1.78	0.68	0.07	1.752	0.67	6.16E-15	1.697	0.71	0.08	1.697	0.71	0.08	2.121	0.7	-0.07
1.802	0.69	0	1.775	0.68	0	1.718	0.72	-0.08	1.718	0.72	-0.08	2.143	0.69	3.33E-13
1.824	0.68	-0.07	1.797	0.67	-0.07	1.739	0.71	-0.08	1.739	0.71	-0.08	2.164	0.7	0.07
1.846	0.69	0	1.819	0.68	0	1.76	0.71	-0.08	1.76	0.71	-0.08	2.186	0.69	-0.15
1.868	0.68	-0.14	1.841	0.67	-0.07	1.781	0.71	-0.31	1.781	0.71	-0.31	2.208	0.69	0.15
1.889	0.68	0.07	1.864	0.67	0.2	1.802	0.7	-2.45E-13	1.802	0.7	-2.45E-13	2.229	0.7	-0.07
1.911	0.68	-0.14	1.886	0.68	-0.07	1.824	0.71	-0.08	1.824	0.71	-0.08	2.251	0.69	2.38E-13
1.933	0.68	0.07	1.908	0.67	-0.07	1.845	0.7	-0.16	1.845	0.7	-0.16	2.272	0.7	0.07
1.955	0.69	0.14	1.931	0.68	0	1.866	0.7	-0.07	1.866	0.7	-0.07	2.294	0.69	0.08
1.977	0.68	0.07	1.953	0.67	-0.07	1.888	0.69	-0.08	1.888	0.69	-0.08	2.316	0.7	0.07
1.999	0.69	0.14	1.975	0.67	0.07	1.909	0.7	0.07	1.909	0.7	0.07	2.337	0.69	0.08
2.021	0.69	0	1.998	0.67	-0.07	1.931	0.7	-0.15	1.931	0.7	-0.15	2.359	0.7	3.27E-13
2.043	0.69	0.07	2.02	0.67	-0.07	1.953	0.69	-0.08	1.953	0.69	-0.08	2.38	0.69	-0.15
2.064	0.69	-0.14	2.042	0.67	2.21E-13	1.974	0.69	3.97E-13	1.974	0.69	3.97E-13	2.402	0.7	0.07
2.086	0.68	-8.83E-15	2.065	0.67	-3.07E-26	1.996	0.69	-0.07	1.996	0.69	-0.07	2.423	0.7	-0.22
2.108	0.69	0.07	2.087	0.67	0.07	2.018	0.69	-0.07	2.018	0.69	-0.07	2.445	0.69	-0.08
2.13	0.69	-0.14	2.109	0.67	0.14	2.039	0.69	-0.14	2.039	0.69	-0.14	2.466	0.69	0.07
2.152	0.68	-0.14	2.132	0.68	0	2.061	0.68	-0.14	2.061	0.68	-0.14	2.488	0.69	0.08
2.174	0.68	0	2.154	0.67	-0.07	2.083	0.68	-0.07	2.083	0.68	-0.07	2.51	0.7	0.3
2.196	0.68	3.15E-13	2.176	0.67	-2.99E-13	2.105	0.68	0.07	2.105	0.68	0.07	2.531	0.7	0.08
2.218	0.68	-0.07	2.199	0.67	0	2.127	0.68	0.07	2.127	0.68	0.07	2.552	0.7	0
2.24	0.68	-0.14	2.221	0.67	-0.07	2.149	0.68	-2.31E-13	2.149	0.68	-2.31E-13	2.574	0.7	0.08
2.262	0.68	-0.14	2.243	0.67	-0.13	2.171	0.68	2.31E-13	2.171	0.68	2.31E-13	2.595	0.7	0.08
2.284	0.68	1.59E-27	2.266	0.67	-2.97E-13	2.193	0.68	0.07	2.193	0.68	0.07	2.616	0.71	0.08
2.306	0.68	0	2.288	0.67	0	2.215	0.69	0	2.215	0.69	0	2.638	0.71	-0.08
2.328	0.68	-1.59E-27	2.311	0.67	0.07	2.236	0.68	-0.14	2.236	0.68	-0.14	2.659	0.7	0.08
2.35	0.68	0.07	2.333	0.67	0.07	2.258	0.68	0	2.258	0.68	0	2.68	0.71	0.08
2.373	0.68	-0.07	2.355	0.67	-0.07	2.28	0.68	3.08E-13	2.28	0.68	3.08E-13	2.701	0.71	0.08
2.395	0.67	-0.07	2.378	0.67	-2.97E-13	2.302	0.68	-0.07	2.302	0.68	-0.07	2.722	0.71	0.16
2.417	0.68	-0.07	2.4	0.67	0.07	2.324	0.68	0	2.324	0.68	0	2.743	0.71	0.08
2.439	0.67	-0.14	2.422	0.67	0.07	2.346	0.68	0.07	2.346	0.68	0.07	2.764	0.72	-2.52E-13
2.462	0.67	2.21E-13	2.445	0.67	-0.07	2.368	0.68	2.36E-13	2.368	0.68	2.36E-13	2.785	0.71	2.52E-13

2.484	0.67	-0.07	2.467	0.67	-0.07	2.39	0.68	-0.07	2.39	0.68	-0.07	2.806	0.72	0
2.507	0.67	0.07	2.49	0.67	-0.07	2.412	0.68	0	2.412	0.68	0	2.827	0.71	0
2.529	0.67	0.13	2.512	0.67	-0.13	2.434	0.68	1.09E-26	2.434	0.68	1.09E-26	2.848	0.72	0.08
2.551	0.67	0.07	2.535	0.66	-0.07	2.456	0.68	0.07	2.456	0.68	0.07	2.869	0.72	3.67E-13
2.574	0.68	-5.44E-15	2.557	0.66	-2.9E-13	2.478	0.68	0.07	2.478	0.68	0.07	2.89	0.72	-3.94E-26
2.596	0.67	-2.3E-13	2.58	0.66	7.12E-14	2.5	0.68	-2.31E-13	2.5	0.68	-2.31E-13	2.911	0.72	-6.19E-13
2.618	0.68	2.17E-13	2.602	0.66	2.19E-13	2.522	0.68	0.07	2.522	0.68	0.07	2.932	0.72	-0.08
2.64	0.67	3.03E-13	2.625	0.66	0.07	2.544	0.69	0	2.544	0.69	0	2.953	0.71	-0.08
2.663	0.68	-0.07	2.647	0.67	0.13	2.566	0.68	0.07	2.566	0.68	0.07	2.974	0.71	0
2.685	0.67	-0.2	2.67	0.67	-3.97E-15	2.587	0.69	0.14	2.587	0.69	0.14	2.995	0.71	0
2.707	0.67	-0.07	2.692	0.67	-0.07	2.609	0.69	0	2.609	0.69	0	3.016	0.71	-3.6E-13
2.73	0.67	-0.13	2.715	0.67	-0.07	2.631	0.69	0.07	2.631	0.69	0.07	3.037	0.71	6.89E-27
2.752	0.66	-0.13	2.737	0.66	-0.07	2.653	0.69	0.07	2.653	0.69	0.07	3.058	0.71	-0.08
2.775	0.66	-0.06	2.76	0.66	2.16E-13	2.674	0.69	-9.25E-14	2.674	0.69	-9.25E-14	3.079	0.71	-0.16
2.798	0.66	-0.06	2.783	0.66	0	2.696	0.69	0.08	2.696	0.69	0.08	3.1	0.71	-0.08
2.821	0.66	1.34E-27	2.805	0.66	-2.87E-13	2.717	0.7	0.07	2.717	0.7	0.07	3.121	0.71	-2.1E-26
2.843	0.66	-2.82E-13	2.828	0.66	-0.06	2.739	0.7	0	2.739	0.7	0	3.142	0.71	-3.48E-13
2.866	0.66	0.06	2.85	0.66	2.92E-13	2.76	0.7	0.08	2.76	0.7	0.08	3.163	0.71	0
2.889	0.66	2.78E-13	2.873	0.66	0	2.782	0.7	0.15	2.782	0.7	0.15	3.185	0.71	0.08
2.912	0.66	-2.88E-13	2.896	0.66	-2.92E-13	2.803	0.7	0.16	2.803	0.7	0.16	3.206	0.71	0.08
2.934	0.66	-0.06	2.918	0.66	-5.08E-15	2.824	0.71	0.16	2.824	0.71	0.16	3.227	0.71	0.08
2.957	0.66	-0.19	2.941	0.66	-0.06	2.845	0.71	-2.47E-13	2.845	0.71	-2.47E-13	3.248	0.71	0.08
2.98	0.65	-0.12	2.964	0.66	-0.13	2.867	0.71	-0.08	2.867	0.71	-0.08	3.269	0.71	-0.08
3.003	0.65	-0.06	2.987	0.66	-0.31	2.888	0.71	3.48E-13	2.888	0.71	3.48E-13	3.29	0.71	-0.08
3.026	0.65	-0.06	3.01	0.65	-0.58	2.909	0.71	-0.08	2.909	0.71	-0.08	3.311	0.71	-0.08
3.05	0.65	-0.18	3.033	0.63	-1.6	2.93	0.7	-0.16	2.93	0.7	-0.16	3.332	0.71	-6.66E-15
3.073	0.64	-0.35	3.058	0.57	-2	2.952	0.7	-0.3	2.952	0.7	-0.3	3.353	0.71	0.24
3.096	0.63	-1.24	3.086	0.52	0.26	2.973	0.69	-0.71	2.973	0.69	-0.71	3.374	0.72	0.16
3.121	0.58	-1.74	3.113	0.58	-0.89	2.995	0.67	-1.97	2.995	0.67	-1.97	3.395	0.72	6.19E-13
3.148	0.54	-1.39	3.142	0.46	6.94	3.019	0.6	-3.85	3.019	0.6	-3.85	3.416	0.72	-5.07E-13
3.177	0.5	0.91	3.167	0.87	0.12	3.048	0.46	0.2	3.048	0.46	0.2	3.437	0.72	-0.08
3.205	0.59	8.26	3.19	0.54	-6.78	3.077	0.61	2.79	3.077	0.61	2.79	3.458	0.71	-0.08
3.226	0.85	-0.34	3.216	0.6	0.79	3.101	0.63	10.12	3.101	0.63	10.12	3.479	0.71	-0.16
3.249	0.55	-10.87	3.241	0.58	-0.1	3.121	0.97	0.26	3.121	0.97	0.26	3.5	0.71	-0.16
3.287	0.3	1.07	3.267	0.6	-0.41	3.14	0.65	-8.66	3.14	0.65	-8.66	3.521	0.71	0.5
3.324	0.61	3.16	3.293	0.56	-0.14	3.163	0.67	0.87	3.163	0.67	0.87	3.542	0.73	-3.97
3.349	0.6	-0.69	3.319	0.59	0.2	3.185	0.69	0.14	3.185	0.69	0.14	3.569	0.45	-4.59
3.374	0.58	-0.1	3.345	0.57	-0.18	3.206	0.68	0.07	3.206	0.68	0.07	3.601	0.51	0.23
3.399	0.6	0.22	3.371	0.58	0.17	3.228	0.69	-0.07	3.228	0.69	-0.07	3.631	0.46	
3.425	0.59	-0.32	3.397	0.58	-0.13	3.25	0.68	4.69E-13	3.25	0.68	4.69E-13			
3.45	0.58	0.23	3.423	0.57	-0.09	3.272	0.69	-0.07	3.272	0.69	-0.07			

3.476	0.6	-0.09	3.449	0.58	-0.12	3.294	0.67	-0.22	3.294	0.67	-0.22			
3.501	0.58	-0.05	3.475	0.57	-0.21	3.316	0.68	-0.07	3.316	0.68	-0.07			
3.527	0.6	-0.04	3.502	0.57	-0.08	3.338	0.67	-0.28	3.338	0.67	-0.28			
3.552	0.58	-0.23	3.528	0.56	-0.2	3.361	0.67	-0.13	3.361	0.67	-0.13			
3.578	0.59	-0.04	3.555	0.56	-0.08	3.383	0.66	-0.07	3.383	0.66	-0.07			
3.604	0.57	-0.34	3.582	0.56	-0.04	3.406	0.67	-0.06	3.406	0.67	-0.06			
3.63	0.57		3.609	0.55	-0.08	3.428	0.66	-0.07	3.428	0.66	-0.07			
			3.636	0.55	0.04	3.451	0.67	0.13	3.451	0.67	0.13			
			3.663	0.56	-0.04	3.473	0.67	-0.13	3.473	0.67	-0.13			
			3.69	0.55	-0.08	3.496	0.66	-0.13	3.496	0.66	-0.13			
			3.717	0.55	-0.11	3.519	0.66	-0.13	3.519	0.66	-0.13			
			3.745	0.55	-0.22	3.541	0.66	-0.25	3.541	0.66	-0.25			
			3.772	0.54	-0.11	3.564	0.65	-0.18	3.564	0.65	-0.18			
			3.8	0.54	-0.1	3.588	0.65	-0.06	3.588	0.65	-0.06			
			3.828	0.53	-0.14	3.611	0.65	-0.18	3.611	0.65	-0.18			
			3.856	0.53	-0.03	3.634	0.64	-0.06	3.634	0.64	-0.06			
			3.885	0.53	0	3.657	0.64	-2.52E-13	3.657	0.64	-2.52E-13			
			3.913	0.53	-2.8E-13	3.681	0.64	-4.08E-13	3.681	0.64	-4.08E-13			
			3.941	0.53	-0.03	3.704	0.64	2.52E-13	3.704	0.64	2.52E-13			
			3.969	0.53	-0.1	3.728	0.64	-0.17	3.728	0.64	-0.17			
			3.998	0.53	-0.13	3.751	0.64	-0.17	3.751	0.64	-0.17			
			4.026	0.52	-0.16	3.775	0.63		3.775	0.63				
			4.055	0.52	-0.15									
			4.084	0.51	-0.12									
			4.113	0.51	-0.06									
			4.143	0.51	-0.06									
			4.172	0.51	-0.03									
			4.202	0.51	0.03									
			4.231	0.51	-0.09									
			4.261	0.5	-0.11									
			4.291	0.5	-0.11									
			4.321	0.5	-0.16									
			4.351	0.49	-0.13									
			4.382	0.49	-0.13									
			4.413	0.48	-0.08									
			4.444	0.48	-0.03									
			4.475	0.48	-0.03									
			4.506	0.48	0.03									
			4.537	0.48	-0.02									
			4.568	0.48	-0.1									
			4.599	0.48	-0.14									

			4.631	0.47	-0.14									
			4.663	0.47	-0.11									
			4.695	0.46	-0.13									
			4.727	0.46	-0.09									
			4.76	0.46	-0.02									
			4.793	0.46	0									
			4.825	0.46	2.07E-13									
			4.858	0.46	-0.04									
			4.891	0.46	-0.1									
			4.924	0.45	-0.12									
			4.957	0.45	-0.14									
			4.991	0.44	-0.14									
			5.025	0.44	-0.13									
			5.059	0.43	-0.07									
			5.094	0.43	-0.02									
			5.129	0.43	-0.02									
			5.163	0.43	-1.88E-26									
			5.198	0.43	-0.05									
			5.233	0.43	-0.1									
			5.268	0.42	-0.1									
			5.304	0.42	-0.13									
			5.339	0.42	-0.13									
			5.376	0.41	-0.11									
			5.412	0.41	-0.08									
			5.449	0.41	-0.03									
			5.486	0.41	1.43E-15									
			5.523	0.41	2.77									
			5.555	0.57	30.22									
			5.573	1.4	7.13									
			5.597	0.4	-25.46									
			5.635	0.4										

Trial 9			Trial 10			Trial 11			Trial 12			Trial 13		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.489	0.21	0.99	0.668	0.19	0.71	0.019	0.61	-0.17	0.03	0.55	0.3	0.052	0.35	1.05
0.551	0.28	1	0.737	0.25	0.9	0.043	0.61	0.37	0.057	0.56	0.32	0.093	0.39	1
0.601	0.33	1.01	0.792	0.3	0.96	0.068	0.62	0.2	0.084	0.57	0.37	0.129	0.43	0.85
0.644	0.37	0.81	0.838	0.35	0.88	0.092	0.62	0.17	0.11	0.58	0.48	0.163	0.45	0.67

0.684	0.39	0.63	0.879	0.38	0.84	0.116	0.63	0.5	0.135	0.6	0.46	0.196	0.47	0.78
0.721	0.42	0.71	0.917	0.41	0.75	0.139	0.64	0.06	0.16	0.6	0.34	0.227	0.5	0.69
0.756	0.45	0.66	0.952	0.44	0.77	0.163	0.64	0.18	0.185	0.61	0.41	0.256	0.52	0.66
0.789	0.46	0.6	0.986	0.47	0.81	0.186	0.65	0.42	0.209	0.63	0.21	0.285	0.54	0.71
0.82	0.48	0.71	1.017	0.49	0.75	0.209	0.66	-0.06	0.233	0.62	0.11	0.312	0.56	0.57
0.851	0.51	0.68	1.047	0.51	0.64	0.232	0.65	0.13	0.257	0.63	0.33	0.339	0.57	0.49
0.88	0.52	0.61	1.076	0.53	0.48	0.255	0.66	0.31	0.281	0.64	0.23	0.365	0.58	0.39
0.908	0.54	0.45	1.104	0.54	0.38	0.277	0.66	0.06	0.304	0.64	0.24	0.39	0.59	0.22
0.935	0.55	0.37	1.132	0.55	0.49	0.3	0.66	0.13	0.327	0.65	0.37	0.416	0.59	0.39
0.962	0.56	0.39	1.159	0.57	0.51	0.322	0.67	0.07	0.35	0.66	0.25	0.441	0.61	0.6
0.989	0.57	0.55	1.185	0.57	0.38	0.345	0.67	-0.13	0.373	0.66	0.06	0.465	0.62	0.42
1.015	0.59	0.58	1.211	0.59	0.31	0.367	0.66	0.2	0.396	0.66	0.26	0.489	0.63	0.33
1.04	0.6	0.19	1.236	0.59	0.53	0.39	0.68	0.13	0.418	0.67	0.13	0.513	0.64	0.47
1.065	0.6	0.3	1.261	0.61	0.48	0.412	0.67	0.07	0.441	0.67	0.07	0.536	0.65	0.49
1.09	0.61	0.4	1.286	0.61	0.16	0.434	0.68	0.2	0.463	0.68	0.34	0.559	0.66	0.19
1.114	0.62	0.27	1.31	0.62	0.38	0.456	0.68	0.07	0.485	0.68	0.07	0.581	0.66	0.2
1.138	0.63	0.39	1.334	0.63	0.33	0.479	0.68	0.21	0.507	0.68	0.22	0.604	0.67	0.33
1.162	0.64	0.22	1.358	0.64	0.23	0.5	0.69	-0.07	0.529	0.69	0.07	0.626	0.68	0.35
1.185	0.64	0.18	1.381	0.64	0.36	0.522	0.68	-0.14	0.551	0.68	8.13E-14	0.648	0.68	0.21
1.208	0.65	0.42	1.404	0.65	0.12	0.544	0.68	-0.07	0.573	0.69	0.07	0.67	0.68	0.15
1.231	0.66	-1.93E-13	1.427	0.65	0.13	0.566	0.68	-0.21	0.595	0.68	-5.86E-14	0.692	0.69	0.29
1.254	0.65	0	1.45	0.66	0.45	0.589	0.67	0.07	0.616	0.69	0.29	0.714	0.7	0.15
1.277	0.66	0.44	1.473	0.67	0.19	0.611	0.68	-0.13	0.638	0.7	5.91E-14	0.735	0.7	0.31
1.3	0.67	0.06	1.495	0.67	0.21	0.633	0.67	0.22	0.66	0.69	0.08	0.756	0.71	0.31
1.323	0.66	0.13	1.518	0.68	0.34	0.655	0.69	0.2	0.681	0.7	0.22	0.777	0.71	0.16
1.345	0.67	0.4	1.54	0.68	0.29	0.677	0.68	-0.07	0.703	0.7	5.97E-14	0.798	0.72	0.16
1.367	0.68	0.21	1.562	0.69	0.29	0.699	0.68	0.28	0.724	0.7	0.08	0.819	0.72	-9.03E-14
1.389	0.68	0.29	1.583	0.69	0.07	0.721	0.69	-0.14	0.745	0.7	0	0.84	0.72	0.25
1.411	0.69	0.07	1.605	0.69	0.23	0.743	0.68	-0.07	0.767	0.7	0.08	0.861	0.73	-0.08
1.433	0.68	0.07	1.626	0.7	0.15	0.765	0.68	-0.14	0.788	0.71	-3.22E-15	0.882	0.71	0.18
1.455	0.69	0.07	1.648	0.7	4.94E-14	0.787	0.67	-0.07	0.809	0.7	0.16	0.902	0.74	0.25
1.477	0.69	0.15	1.669	0.7	0.08	0.809	0.68	0.07	0.83	0.71	0.15	0.923	0.72	-0.17
1.498	0.7	0.22	1.69	0.7	-0.15	0.832	0.68	-0.14	0.852	0.71	0.08	0.944	0.73	0.26
1.52	0.7	-0.08	1.712	0.7	0.24	0.854	0.68	0.21	0.873	0.72	0.16	0.964	0.74	2.83E-15
1.541	0.7	0.15	1.733	0.71	0.08	0.876	0.68	0	0.894	0.71	0	0.985	0.73	5.55E-17
1.563	0.7	0.23	1.754	0.7	0.08	0.898	0.68	0.15	0.915	0.72	0.16	1.005	0.74	-4.77E-15
1.584	0.71	0.16	1.775	0.72	0.39	0.92	0.69	0.14	0.935	0.72	0	1.026	0.73	-0.09
1.605	0.71	0.16	1.796	0.72	0.08	0.942	0.68	-0.15	0.956	0.72	-2.21E-13	1.046	0.73	0.17
1.626	0.71	3.41E-15	1.817	0.72	0.25	0.964	0.68	0.07	0.977	0.72	8.92E-14	1.067	0.74	0.27
1.647	0.71	-0.08	1.838	0.73	-0.16	0.986	0.68	-0.14	0.998	0.72	2.21E-13	1.087	0.74	0.09
1.668	0.71	0.08	1.859	0.71	1.9E-13	1.008	0.68	-7.22E-16	1.019	0.72	-1.83E-13	1.107	0.74	9.99E-16

1.689	0.71	0	1.879	0.73	-0.08	1.03	0.68	-0.07	1.04	0.72	0.08	1.127	0.74	0.09
1.71	0.71	-0.08	1.9	0.71	0.09	1.052	0.68	0.07	1.06	0.72	0.08	1.148	0.74	-0.09
1.731	0.71	0.16	1.921	0.73	0.16	1.074	0.69	1.13E-14	1.081	0.72	-0.08	1.168	0.74	0.09
1.752	0.72	-0.16	1.942	0.72	-0.09	1.096	0.68	0.07	1.102	0.72	-0.08	1.188	0.75	0
1.773	0.7	0	1.962	0.73	0.17	1.118	0.69	0.28	1.123	0.72	-0.08	1.208	0.74	0.09
1.795	0.72	0.16	1.983	0.72	-0.17	1.139	0.69	0.15	1.144	0.72	-3.08E-13	1.228	0.75	-1.35E-13
1.816	0.71	0	2.004	0.72	-0.08	1.161	0.7	0.3	1.165	0.72	-0.16	1.249	0.74	0.1
1.837	0.72	0.08	2.025	0.72	-0.08	1.182	0.7	-0.07	1.186	0.71	0.08	1.269	0.75	0.27
1.857	0.71	-0.24	2.046	0.72	0.17	1.204	0.69	0	1.207	0.72	0.08	1.289	0.75	-0.19
1.879	0.71	-0.08	2.066	0.73	0	1.225	0.7	-0.07	1.227	0.71	1.88E-13	1.309	0.75	0.09
1.9	0.71	-0.15	2.087	0.72	0.09	1.247	0.69	-0.15	1.248	0.72	0.16	1.329	0.75	0.09
1.921	0.7	0.08	2.108	0.73	-0.08	1.269	0.69	-0.14	1.269	0.72	-0.08	1.349	0.75	-0.09
1.942	0.71	-0.15	2.129	0.71	-0.34	1.29	0.68	-0.07	1.29	0.72	0.08	1.369	0.75	-0.18
1.964	0.69	-0.32	2.149	0.72	0.08	1.312	0.69	-1.59E-13	1.311	0.72	-1.79E-13	1.389	0.74	0
1.985	0.7	0.15	2.17	0.72	-0.24	1.334	0.68	1.17E-13	1.332	0.72	1.31E-13	1.409	0.75	0.09
2.006	0.7	0.08	2.191	0.71	-1.53E-14	1.356	0.69	0.22	1.352	0.72	-0.08	1.429	0.75	-3.11E-15
2.028	0.7	0.16	2.212	0.72	5.55E-17	1.377	0.69	1.61E-13	1.373	0.71	-0.08	1.449	0.75	0.09
2.049	0.71	-0.08	2.234	0.71	1.53E-14	1.399	0.69	0.08	1.394	0.72	-0.08	1.469	0.75	0
2.07	0.7	0.08	2.255	0.72	0.08	1.421	0.7	-0.07	1.415	0.71	-0.25	1.489	0.75	2.08E-13
2.092	0.71	-0.08	2.276	0.71	0.08	1.442	0.69	-0.08	1.436	0.71	0.08	1.509	0.75	-0.18
2.113	0.7	-0.08	2.297	0.72	0.16	1.464	0.69	-0.14	1.457	0.71	-0.08	1.529	0.74	-6.91E-14
2.134	0.71	-0.15	2.317	0.72	-0.32	1.486	0.68	-0.22	1.478	0.71	-0.16	1.549	0.75	0.09
2.156	0.69	-0.31	2.338	0.71	0	1.508	0.68	0.07	1.5	0.71	0	1.569	0.75	-0.09
2.177	0.69	0.15	2.359	0.72	-0.15	1.53	0.68	-0.07	1.521	0.71	2.3E-27	1.589	0.75	-1.2E-26
2.199	0.7	-0.07	2.381	0.7	-0.16	1.552	0.68	0.07	1.542	0.71	0	1.609	0.75	-0.09
2.22	0.69	0.15	2.402	0.71	-0.08	1.574	0.69	0.07	1.563	0.71	-0.08	1.63	0.74	-0.09
2.242	0.7	0.07	2.423	0.7	-0.24	1.595	0.68	0.07	1.584	0.7	-2.49E-13	1.65	0.74	-2.72E-13
2.263	0.69	-0.15	2.445	0.7	0.08	1.617	0.69	0.14	1.606	0.71	2.42E-13	1.67	0.74	2.72E-13
2.285	0.7	-0.07	2.466	0.7	1.97E-27	1.639	0.69	-1.64E-13	1.627	0.7	-0.16	1.69	0.74	0
2.307	0.69	-0.22	2.487	0.7	3.4E-13	1.661	0.69	0	1.648	0.7	0.08	1.71	0.74	-0.18
2.328	0.69	-0.07	2.509	0.7	-0.15	1.682	0.69	0	1.67	0.71	0.08	1.731	0.74	-0.18
2.35	0.69	-0.21	2.53	0.69	-0.08	1.704	0.69	-0.07	1.691	0.7	0	1.751	0.74	1.51E-26
2.372	0.68	0.07	2.552	0.7	3.34E-13	1.726	0.69	-0.14	1.712	0.71	0	1.772	0.74	1.96E-13
2.394	0.69	-0.07	2.573	0.69	-0.08	1.748	0.68	-0.14	1.733	0.7	-0.08	1.792	0.74	-2.67E-13
2.416	0.68	0.07	2.595	0.69	-0.15	1.769	0.68	2.28E-13	1.755	0.7	-0.08	1.812	0.74	2.67E-13
2.438	0.69	0.28	2.617	0.69	-0.07	1.791	0.68	0.07	1.776	0.7	-0.08	1.833	0.74	-0.09
2.46	0.69	-0.07	2.638	0.69	0.07	1.813	0.68	0.07	1.797	0.7	-0.08	1.853	0.73	-0.09
2.481	0.69	3.22E-13	2.66	0.69	-0.14	1.835	0.69	0.22	1.819	0.7	0.08	1.874	0.73	-1.92E-13
2.503	0.69	-0.22	2.682	0.68	0.07	1.857	0.69	0	1.84	0.7	-0.07	1.894	0.73	-0.09
2.525	0.68	-3.31E-13	2.704	0.69	0.14	1.879	0.69	-0.07	1.862	0.69	-0.08	1.915	0.73	1.95E-13
2.547	0.69	-2.42E-13	2.725	0.69	-0.15	1.9	0.69	-2.35E-13	1.883	0.7	-2.4E-13	1.935	0.73	-2.61E-13

2.569	0.68	-0.07	2.747	0.69	2.35E-13	1.922	0.69	-0.07	1.905	0.69	-0.23	1.956	0.73	-0.09
2.591	0.68	-3.15E-13	2.769	0.69	-0.07	1.944	0.69	0.07	1.926	0.69	0	1.976	0.73	0.09
2.613	0.68	-0.14	2.791	0.68	-0.07	1.966	0.69	0	1.948	0.69	0.07	1.997	0.73	0
2.635	0.68	-0.07	2.813	0.69	-0.07	1.987	0.69	-0.07	1.97	0.69	0.08	2.018	0.73	3.86E-13
2.657	0.68	0.07	2.834	0.68	-0.07	2.009	0.69	0.07	1.991	0.7	0.07	2.038	0.73	-0.17
2.679	0.68	0.14	2.856	0.68	3.15E-13	2.031	0.69	-3.22E-13	2.013	0.7	0.08	2.059	0.72	-0.09
2.701	0.68	-5.75E-15	2.878	0.68	0.07	2.053	0.69	-4.47E-15	2.034	0.7	0	2.079	0.73	0.08
2.723	0.68	-5.75E-15	2.9	0.69	-3.06E-13	2.074	0.69	3.2E-13	2.056	0.7	-0.08	2.1	0.72	-0.17
2.745	0.68	0.07	2.922	0.68	-0.07	2.096	0.69	-2.35E-13	2.077	0.7	0.08	2.121	0.72	0.09
2.767	0.68	2.31E-13	2.944	0.68	0.14	2.118	0.69	0.07	2.099	0.7	0.08	2.142	0.73	6.3E-13
2.788	0.68	-1.87E-26	2.966	0.69	0	2.14	0.69	0	2.12	0.7	0.08	2.162	0.72	-0.09
2.81	0.68	-0.21	2.988	0.68	0.15	2.161	0.69	0.15	2.141	0.7	0.16	2.183	0.72	-3.7E-13
2.832	0.68	-0.07	3.01	0.69	0.14	2.183	0.7	-0.07	2.162	0.71	0.08	2.204	0.72	-0.08
2.855	0.68	0.14	3.031	0.69	4.52E-15	2.205	0.69	-0.08	2.184	0.71	0.08	2.225	0.72	-0.08
2.877	0.68	0.22	3.053	0.69	0.07	2.226	0.69	0.07	2.205	0.71	0	2.245	0.72	-0.08
2.898	0.69	0.21	3.074	0.69	-0.07	2.248	0.69	-3.32E-13	2.226	0.71	-0.16	2.266	0.72	-3.67E-13
2.92	0.69	0.15	3.096	0.69	0.08	2.27	0.69	3.27E-13	2.247	0.7	1.64E-15	2.287	0.72	3.64E-13
2.942	0.7	0.15	3.118	0.7	-2.46E-13	2.291	0.69	0.08	2.269	0.71	-2.46E-13	2.308	0.72	-2.52E-13
2.963	0.7	2.4E-13	3.139	0.69	0.08	2.313	0.7	0.15	2.29	0.7	0.08	2.329	0.72	-1.12E-13
2.985	0.7	0.08	3.161	0.7	0.15	2.334	0.7	1.98E-26	2.311	0.71	0.24	2.35	0.72	0
3.006	0.7	0	3.182	0.7	-0.08	2.356	0.7	-0.15	2.332	0.71	0.16	2.371	0.72	-0.08
3.028	0.7	-0.08	3.204	0.7	-1.98E-26	2.377	0.69	-0.15	2.353	0.72	0.08	2.392	0.71	-0.08
3.049	0.7	-0.07	3.225	0.7	0.08	2.399	0.69	-0.07	2.374	0.72	3.67E-13	2.413	0.71	-0.08
3.071	0.69	0.08	3.247	0.7	0.08	2.421	0.69	0	2.395	0.72	-0.08	2.434	0.71	-0.08
3.092	0.7	0.15	3.268	0.7	-2.44E-13	2.443	0.69	0.15	2.416	0.71	-0.08	2.455	0.71	-2.5E-13
3.114	0.7	0.08	3.29	0.7	0.08	2.464	0.69	0.15	2.437	0.71	-0.08	2.476	0.71	0.08
3.135	0.7	3.41E-13	3.311	0.7	0.08	2.486	0.7	0.07	2.458	0.71	-5.08E-15	2.497	0.71	0.08
3.156	0.7	0	3.332	0.7	0.08	2.507	0.7	-0.07	2.479	0.71	0	2.518	0.71	-3.88E-26
3.178	0.7	0	3.353	0.71	0.24	2.529	0.69	-0.08	2.5	0.71	0.08	2.539	0.71	3.6E-13
3.199	0.7	0	3.375	0.71	0.08	2.551	0.69	-0.15	2.521	0.72	0.08	2.56	0.71	0.08
3.22	0.7	0	3.396	0.71	5.11E-15	2.572	0.69	-0.07	2.542	0.71	-0.08	2.581	0.72	-0.08
3.242	0.7	-0.08	3.417	0.71	0.08	2.594	0.69	0.07	2.563	0.71	-2.51E-13	2.602	0.71	-0.08
3.263	0.7	-9.6E-14	3.438	0.71	-0.08	2.616	0.69	0.07	2.584	0.71	0.08	2.623	0.71	-0.08
3.284	0.7	-2.02E-27	3.459	0.71	-0.16	2.637	0.69	-0.48	2.605	0.72	0.08	2.644	0.71	3.48E-13
3.306	0.7	3.4E-13	3.48	0.71	-0.16	2.659	0.67	-4.35	2.626	0.72	-7.12E-27	2.665	0.71	0.08
3.327	0.7	0.16	3.501	0.7	-0.08	2.688	0.42	-3.2	2.647	0.72	0.08	2.686	0.71	0.08
3.349	0.71	0.15	3.522	0.7	-0.08	2.721	0.51	2.45	2.667	0.72	0.17	2.707	0.72	0.16
3.37	0.71	3.48E-13	3.544	0.7	0	2.749	0.57	3.37	2.688	0.72	0.08	2.728	0.72	-3.64E-13
3.391	0.71	0.16	3.565	0.7	0.16	2.773	0.68		2.709	0.72	0	2.749	0.72	0
3.412	0.71	0.16	3.586	0.71	0.08				2.73	0.72	-0.08	2.77	0.72	3.64E-13
3.433	0.71	-6.11E-13	3.608	0.71	1.02E-13				2.75	0.72	-2.52E-13	2.791	0.72	-0.16

3.454	0.71	-0.08	3.629	0.71	-0.08				2.771	0.72	0.08	2.812	0.71	-0.08
3.475	0.71	-0.08	3.65	0.7	-0.23				2.792	0.72	-0.63	2.833	0.71	0.08
3.496	0.71	-3.55E-13	3.671	0.7	-0.15				2.813	0.7	-4.86	2.854	0.71	0.08
3.517	0.71	0.08	3.693	0.7	-0.07				2.842	0.42	-3.64	2.875	0.72	0.25
3.538	0.71	0.08	3.714	0.69	-5.74E-13				2.874	0.52	1.18	2.896	0.72	0.25
3.559	0.71	-0.08	3.736	0.7	0.07				2.903	0.5		2.916	0.73	0.09
3.58	0.71	-0.08	3.757	0.7	0.08							2.937	0.73	-0.09
3.601	0.71	-5E-13	3.779	0.7	0.15							2.958	0.72	-0.17
3.623	0.71	3.55E-13	3.8	0.7	0							2.978	0.72	-0.08
3.644	0.71	0.08	3.822	0.7	-0.15							2.999	0.72	-0.08
3.665	0.71	-3.56E-13	3.843	0.7	-0.08							3.02	0.72	0.62
3.686	0.71	-0.08	3.865	0.7	-0.07							3.04	0.75	-3.97
3.707	0.71	2.36	3.886	0.69	-0.08							3.067	0.46	-5.22
3.727	0.8	-3.28	3.908	0.69	-1.27							3.098	0.51	3.3
3.753	0.45	-6.41	3.93	0.64	-4.47							3.125	0.64	1.48
3.784	0.52	2.02	3.959	0.44	-1.31							3.149	0.6	
3.812	0.57	1.42E-13	3.99	0.57	0.01									
3.84	0.52		4.02	0.44	-4.16									
			4.343	0.02										

Trial 14			Trial 15			Trial 16			Trial 17			Trial 18		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.047	0.32	0.88	0.06	0.37	0.65	0.057	0.36	0.82	0.03	0.4	0.43	0.028	0.4	0.62
0.092	0.36	0.93	0.099	0.4	0.75	0.097	0.39	0.67	0.066	0.42	0.63	0.064	0.43	0.77
0.132	0.4	0.8	0.135	0.43	0.7	0.134	0.41	0.6	0.101	0.45	0.71	0.098	0.46	0.76
0.168	0.42	0.61	0.169	0.45	0.61	0.17	0.43	0.57	0.134	0.47	0.64	0.131	0.48	0.59
0.203	0.44	0.61	0.201	0.47	0.68	0.204	0.45	0.7	0.165	0.49	0.59	0.161	0.49	0.54
0.237	0.46	0.53	0.233	0.49	0.62	0.236	0.48	0.65	0.195	0.51	0.57	0.191	0.51	0.53
0.269	0.47	0.56	0.263	0.51	0.53	0.267	0.49	0.51	0.224	0.52	0.54	0.22	0.52	0.41
0.3	0.5	0.77	0.292	0.52	0.51	0.297	0.51	0.46	0.253	0.54	0.4	0.249	0.53	0.41
0.329	0.52	0.7	0.32	0.54	0.6	0.326	0.52	0.45	0.28	0.55	0.37	0.276	0.55	0.47
0.357	0.54	0.47	0.347	0.56	0.52	0.354	0.54	0.4	0.307	0.56	0.38	0.304	0.56	0.47
0.385	0.55	0.29	0.374	0.57	0.28	0.382	0.54	0.44	0.334	0.57	0.41	0.33	0.57	0.55
0.412	0.55	0.47	0.4	0.57	0.43	0.409	0.56	0.46	0.36	0.58	0.43	0.356	0.59	0.39
0.439	0.57	0.48	0.426	0.59	0.49	0.436	0.57	0.37	0.386	0.59	0.31	0.381	0.59	0.19
0.465	0.58	0.35	0.451	0.6	0.43	0.462	0.58	0.43	0.411	0.6	0.33	0.406	0.6	0.44
0.491	0.59	0.46	0.476	0.61	0.19	0.487	0.59	0.27	0.436	0.6	0.5	0.431	0.61	0.45
0.516	0.6	0.38	0.501	0.61	0.32	0.513	0.59	0.14	0.461	0.62	0.3	0.456	0.62	0.21

0.54	0.61	0.41	0.525	0.62	0.64	0.538	0.6	0.39	0.485	0.62	0.27	0.48	0.63	0.27
0.565	0.62	0.26	0.549	0.64	0.16	0.563	0.61	0.45	0.509	0.63	0.33	0.503	0.63	0.28
0.589	0.62	0.16	0.573	0.63	0.18	0.587	0.62	0.21	0.533	0.64	0.17	0.527	0.64	0.23
0.613	0.63	0.28	0.596	0.65	0.41	0.611	0.62	0.27	0.556	0.64	0.42	0.55	0.64	0.12
0.637	0.64	0.11	0.619	0.65	0.18	0.635	0.63	0.33	0.579	0.66	0.18	0.574	0.64	0.18
0.66	0.64	0.42	0.642	0.66	0.32	0.659	0.64	0.23	0.602	0.65	4.83E-14	0.597	0.65	0.24
0.683	0.66	0.41	0.665	0.67	0.26	0.682	0.64	0.12	0.625	0.66	0.38	0.62	0.66	0.19
0.706	0.66	0.13	0.687	0.67	0.07	0.706	0.64	0.06	0.648	0.67	0.26	0.643	0.66	0.06
0.729	0.66	0.32	0.71	0.67	0.27	0.729	0.65	0.18	0.671	0.67	-0.07	0.665	0.66	0.33
0.752	0.67	0.2	0.732	0.68	0.13	0.752	0.65	0.18	0.693	0.66	0.27	0.688	0.68	0.39
0.774	0.67	-0.07	0.754	0.68	-0.07	0.775	0.66	-0.12	0.715	0.68	0.4	0.71	0.68	5.5E-14
0.797	0.67	0.14	0.776	0.68	0.21	0.798	0.65	0.19	0.737	0.68	0.14	0.732	0.68	0.21
0.819	0.68	0.13	0.798	0.68	0.21	0.821	0.66	0.5	0.759	0.68	0.29	0.754	0.68	0.14
0.841	0.67	0.07	0.82	0.68	0.22	0.843	0.67	0.2	0.781	0.69	0.22	0.776	0.68	-0.07
0.863	0.68	0.14	0.842	0.69	0.07	0.866	0.67	0.27	0.803	0.69	0.08	0.798	0.68	0.14
0.885	0.68	0.07	0.864	0.69	0.15	0.888	0.68	0.07	0.824	0.7	0.15	0.82	0.69	-7.22E-16
0.907	0.68	0.36	0.885	0.7	0.37	0.91	0.68	-0.07	0.846	0.7	0.15	0.842	0.68	0.15
0.929	0.69	0.21	0.907	0.7	-0.08	0.932	0.68	0	0.867	0.7	0.16	0.864	0.69	0
0.951	0.69	-0.07	0.928	0.7	0.08	0.954	0.68	-1.13E-13	0.888	0.71	-0.15	0.886	0.68	0.15
0.973	0.69	0.15	0.95	0.71	-8.33E-14	0.976	0.68	0	0.91	0.7	0.16	0.907	0.7	0.29
0.994	0.7	1.18E-13	0.971	0.7	2.08E-13	0.999	0.68	1.13E-13	0.931	0.71	0.15	0.929	0.69	1.22E-13
1.016	0.69	-1.71E-13	0.992	0.71	0.23	1.021	0.68	0.14	0.952	0.7	0	0.95	0.7	0.23
1.037	0.7	0	1.013	0.71	-0.08	1.043	0.68	0.14	0.973	0.71	0.32	0.972	0.7	0
1.059	0.69	1.72E-13	1.035	0.7	0.24	1.065	0.68	0.14	0.994	0.72	-8.75E-14	0.993	0.7	1.72E-13
1.081	0.7	0.15	1.056	0.72	0.08	1.086	0.69	0.22	1.015	0.71	4.16E-16	1.014	0.7	-8.32E-14
1.102	0.7	0.15	1.077	0.71	0.17	1.108	0.69	0.07	1.036	0.72	0.08	1.036	0.7	0.08
1.124	0.7	0.23	1.098	0.72	0.24	1.13	0.69	0	1.057	0.72	0.08	1.057	0.71	1.68E-13
1.145	0.71	1.23E-13	1.119	0.72	-0.08	1.151	0.69	0.07	1.078	0.72	1.24E-13	1.078	0.7	0.08
1.166	0.7	0	1.139	0.72	0.08	1.173	0.69	-0.07	1.099	0.72	0.17	1.1	0.71	0
1.187	0.71	-0.15	1.16	0.72	-0.16	1.195	0.69	-0.15	1.119	0.73	-1.85E-13	1.121	0.7	-0.16
1.209	0.7	0	1.181	0.71	1.86E-13	1.216	0.69	-7.49E-16	1.14	0.72	0	1.142	0.7	0.16
1.23	0.71	0.08	1.202	0.72	-0.16	1.238	0.69	-0.07	1.161	0.73	0.08	1.163	0.71	-0.15
1.251	0.7	-0.08	1.223	0.71	-3.44E-15	1.26	0.68	0.07	1.182	0.72	-0.17	1.185	0.7	0.08
1.273	0.7	0.16	1.244	0.72	0.08	1.282	0.69	0.14	1.203	0.72	0.17	1.206	0.71	-3.22E-15
1.294	0.71	-0.08	1.265	0.71	-0.17	1.303	0.69	-0.07	1.223	0.73	-0.08	1.227	0.7	-3.09E-13
1.315	0.7	0.16	1.286	0.71	0.08	1.325	0.69	0.15	1.244	0.72	0.09	1.249	0.71	0.15
1.336	0.71	1.23E-13	1.307	0.71	-5.29E-14	1.347	0.7	0.07	1.265	0.73	0.17	1.27	0.7	-0.23
1.358	0.7	-0.08	1.328	0.71	-1.92E-27	1.368	0.69	-0.08	1.285	0.72	-0.09	1.291	0.7	0
1.379	0.71	0.15	1.349	0.71	-0.16	1.39	0.69	-0.07	1.306	0.73	-0.17	1.313	0.7	-0.3
1.4	0.71	-0.16	1.37	0.71	-5.11E-15	1.412	0.69	1.67E-13	1.327	0.72	-0.25	1.334	0.69	0.08
1.421	0.7	0.08	1.391	0.71	-0.3	1.433	0.69	-0.21	1.348	0.72	0.08	1.356	0.71	1.49E-13

1.442	0.71	-0.08	1.413	0.69	-0.32	1.455	0.68	-0.29	1.368	0.72	-0.16	1.377	0.69	-0.08
1.464	0.7	0	1.434	0.7	0.07	1.477	0.68	0.14	1.389	0.71	0.08	1.399	0.7	0.15
1.485	0.71	0.08	1.455	0.7	1.18E-13	1.499	0.69	0.07	1.41	0.72	1.25E-13	1.42	0.69	-0.15
1.506	0.7	-0.08	1.477	0.7	0.08	1.521	0.68	0.15	1.431	0.71	-0.08	1.442	0.7	0.15
1.527	0.71	-3.28E-15	1.498	0.7	0.08	1.542	0.69	0.14	1.452	0.72	0.08	1.463	0.7	-0.07
1.549	0.7	-3.28E-15	1.52	0.7	0.32	1.564	0.69	0.23	1.473	0.71	-0.08	1.485	0.69	-0.15
1.57	0.71	-3.28E-15	1.541	0.71	-0.08	1.586	0.7	0.15	1.494	0.72	0	1.506	0.69	-0.29
1.591	0.7	-1.26E-13	1.562	0.7	-0.08	1.607	0.7	-0.15	1.515	0.71	-0.16	1.528	0.68	-0.15
1.612	0.71	0.08	1.583	0.71	2.9E-13	1.629	0.7	-0.07	1.536	0.71	-0.16	1.55	0.69	-0.14
1.634	0.71	1.73E-13	1.604	0.7	-0.24	1.65	0.69	-0.29	1.557	0.71	-0.23	1.572	0.68	-0.28
1.655	0.71	0.08	1.626	0.7	4.34E-27	1.672	0.68	-0.07	1.578	0.7	-0.16	1.594	0.68	0.07
1.676	0.71	-0.15	1.647	0.7	-0.22	1.694	0.69	-0.07	1.6	0.7	-0.08	1.616	0.68	0
1.697	0.7	-0.08	1.669	0.69	-0.08	1.716	0.68	-1.56E-13	1.621	0.7	-3.14E-15	1.639	0.68	-0.07
1.719	0.71	-1.67E-15	1.69	0.7	-0.07	1.737	0.69	-0.07	1.643	0.7	-1.71E-13	1.661	0.68	9.04E-27
1.74	0.7	-0.08	1.712	0.69	-7.41E-14	1.759	0.68	-0.07	1.664	0.7	0.08	1.683	0.68	-2.25E-13
1.761	0.7	0.08	1.734	0.7	0.07	1.781	0.69	0.07	1.686	0.7	0.08	1.705	0.68	-0.07
1.782	0.7	4.44E-27	1.755	0.69	-1.72E-13	1.803	0.68	-0.14	1.707	0.7	-0.23	1.727	0.67	-0.2
1.804	0.7	0.08	1.777	0.7	0.15	1.825	0.68	0.14	1.728	0.69	-0.15	1.75	0.67	-0.13
1.825	0.71	-0.08	1.798	0.7	-0.07	1.847	0.69	1.57E-13	1.75	0.69	-0.15	1.772	0.67	-0.13
1.846	0.7	-0.08	1.82	0.69	0.15	1.869	0.68	0.15	1.772	0.69	-0.15	1.795	0.66	0.07
1.868	0.7	1.67E-13	1.841	0.7	-0.07	1.891	0.69	-0.07	1.793	0.69	9.49E-27	1.817	0.67	-1.94E-15
1.889	0.7	-0.15	1.863	0.69	-0.08	1.913	0.68	-2.36E-13	1.815	0.69	-0.14	1.84	0.66	-0.07
1.91	0.7	0.16	1.884	0.7	0.07	1.934	0.69	0.21	1.837	0.68	0.07	1.863	0.67	0.06
1.932	0.71	0	1.906	0.69	-0.23	1.956	0.69	-0.22	1.859	0.69	0.07	1.885	0.66	0.07
1.953	0.7	-1.74E-13	1.928	0.69	0.08	1.978	0.68	-2.91E-15	1.881	0.68	-1.63E-13	1.908	0.67	0.2
1.974	0.71	0.15	1.949	0.7	0.15	2	0.69	-0.07	1.903	0.69	0.14	1.93	0.67	0
1.996	0.7	0.08	1.971	0.7	0	2.022	0.68	1.6E-13	1.924	0.69	0	1.952	0.67	0.07
2.017	0.71	0.08	1.992	0.7	-0.07	2.044	0.69	-0.07	1.946	0.69	0.07	1.975	0.68	-0.07
2.038	0.71	-2.49E-13	2.014	0.69	-0.08	2.066	0.68	-0.07	1.968	0.69	-0.07	1.997	0.67	-0.2
2.059	0.71	-0.08	2.035	0.69	-0.07	2.088	0.68	0.14	1.989	0.69	2.42E-13	2.019	0.67	-0.07
2.08	0.7	-0.23	2.057	0.69	-3.3E-13	2.109	0.68	-0.14	2.011	0.69	-0.07	2.042	0.66	-0.13
2.102	0.7	-0.08	2.079	0.69	0	2.131	0.68	0	2.033	0.68	-0.15	2.065	0.66	2.92E-13
2.123	0.7	-2.02E-27	2.1	0.69	3.3E-13	2.153	0.68	0.07	2.055	0.69	-0.07	2.087	0.66	-2.8E-13
2.145	0.7	-0.08	2.122	0.69	0.07	2.175	0.68	-5.75E-15	2.077	0.68	-0.14	2.11	0.66	0.2
2.166	0.7	0.16	2.144	0.69	0.08	2.197	0.68	-0.07	2.099	0.68	-1.11E-26	2.132	0.67	0.26
2.187	0.71	0.15	2.165	0.7	0.15	2.219	0.68	0.07	2.121	0.68	0	2.155	0.67	0.07
2.209	0.7	0	2.187	0.7	0.08	2.241	0.69	0.14	2.143	0.68	1.09E-26	2.177	0.68	0.07
2.23	0.71	-0.08	2.208	0.7	-2.44E-13	2.263	0.68	0.07	2.165	0.68	-2.3E-13	2.199	0.68	-2.25E-13
2.251	0.7	-0.08	2.229	0.7	5.84E-13	2.285	0.69	0.07	2.187	0.68	5.42E-13	2.221	0.68	2.25E-13
2.273	0.7	-0.08	2.251	0.7	-3.4E-13	2.307	0.69	4.47E-15	2.209	0.68	-0.07	2.244	0.68	-0.2
2.294	0.7	-0.15	2.272	0.7	0.08	2.328	0.69	3.22E-13	2.231	0.68	-3.12E-13	2.266	0.67	-0.07

2.315	0.7	-1.98E-26	2.294	0.7	0	2.35	0.69	-0.07	2.253	0.68	0.14	2.288	0.67	0.07
2.337	0.7	-1.98E-26	2.315	0.7	-0.08	2.372	0.69	0.07	2.275	0.68	0.07	2.311	0.67	-5.33E-15
2.358	0.7	-0.07	2.336	0.7	-0.08	2.394	0.69	0.07	2.297	0.68	0.15	2.333	0.67	0.2
2.38	0.69	-6.16E-15	2.358	0.7	0.16	2.415	0.69	0.07	2.318	0.69	0.22	2.355	0.68	0.35
2.402	0.7	0.15	2.379	0.71	0.23	2.437	0.69	3.28E-13	2.34	0.69	0.23	2.377	0.69	0.14
2.423	0.7	-0.07	2.4	0.71	-5.97E-13	2.459	0.69	0.08	2.362	0.7	-5.71E-13	2.399	0.68	8.2E-14
2.444	0.69	-1.08E-13	2.422	0.71	0.08	2.48	0.7	0.07	2.383	0.69	2.43E-13	2.421	0.69	2.31E-13
2.466	0.7	-1.25E-14	2.443	0.71	0.08	2.502	0.69	-0.08	2.405	0.7	3.31E-13	2.443	0.68	-0.07
2.487	0.69	-0.08	2.464	0.71	-0.08	2.524	0.69	-0.15	2.426	0.69	-0.08	2.465	0.68	-1.87E-26
2.509	0.7	-0.07	2.485	0.71	-6.66E-15	2.545	0.69	-0.15	2.448	0.7	-0.07	2.487	0.68	-1.87E-26
2.531	0.69	-0.15	2.506	0.71	-0.08	2.567	0.69	3.39E-26	2.469	0.69	-0.15	2.508	0.68	-1.87E-26
2.552	0.69	0.07	2.527	0.7	0.08	2.589	0.69	0.07	2.491	0.69	0.07	2.53	0.68	-0.07
2.574	0.69	-0.14	2.548	0.71	0.08	2.611	0.69	-5.56E-13	2.513	0.69	-2.4E-13	2.552	0.68	-0.07
2.596	0.68	-0.22	2.569	0.71	0.08	2.632	0.69	-0.07	2.534	0.69	0.08	2.574	0.68	2.3E-13
2.618	0.68	0.07	2.591	0.72	0.24	2.654	0.69	-0.07	2.556	0.7	0.15	2.596	0.68	0.07
2.64	0.69	-0.07	2.611	0.72	0.08	2.676	0.68	-0.14	2.577	0.7	0.08	2.618	0.68	0.07
2.661	0.68	0	2.632	0.72	5.27E-15	2.698	0.68	-0.14	2.599	0.7	0.08	2.64	0.68	0.07
2.683	0.69	0.07	2.653	0.72	-0.08	2.72	0.68	-3.12E-13	2.62	0.7	-0.08	2.662	0.69	0.14
2.705	0.68	-0.14	2.674	0.72	-3.94E-26	2.742	0.68	0	2.642	0.7	-0.08	2.684	0.69	0.07
2.727	0.68	-0.14	2.695	0.72	-6.19E-13	2.764	0.68	8.31E-14	2.663	0.7	-0.07	2.705	0.69	-0.14
2.749	0.68	-0.07	2.716	0.72	-0.08	2.786	0.68	0.29	2.685	0.69	-0.22	2.727	0.68	-0.07
2.771	0.68	-0.14	2.737	0.71	0	2.808	0.69	0.07	2.706	0.69	-0.07	2.749	0.69	-0.07
2.793	0.67	-0.27	2.758	0.72	-0.08	2.83	0.68	-0.07	2.728	0.69	-0.21	2.771	0.68	0
2.816	0.67	0	2.779	0.71	0.08	2.852	0.69	0	2.75	0.68	-0.15	2.793	0.69	0.14
2.838	0.67	0.13	2.8	0.72	0.16	2.873	0.68	-0.14	2.772	0.68	0.14	2.815	0.69	0.15
2.861	0.67	0.14	2.821	0.72	0.08	2.895	0.68	-0.07	2.794	0.68	0.07	2.836	0.69	0.22
2.883	0.68	0.21	2.841	0.72	0.17	2.917	0.68	1.69E-27	2.816	0.69	0.07	2.858	0.7	0.15
2.905	0.68	0.14	2.862	0.72	0.09	2.939	0.68	-0.07	2.838	0.69	-3.21E-13	2.879	0.7	0.08
2.927	0.68	0.07	2.883	0.73	0.09	2.961	0.68	0.07	2.859	0.69	0.07	2.901	0.7	-0.15
2.949	0.68	-0.07	2.903	0.73	-0.17	2.983	0.68	0.07	2.881	0.69	-0.07	2.922	0.69	-0.08
2.971	0.68	-0.14	2.924	0.72	-0.09	3.005	0.68	-5.75E-15	2.903	0.68	-0.15	2.944	0.7	-0.07
2.993	0.68	-3.16E-13	2.945	0.72	-3.7E-13	3.027	0.68	0.14	2.925	0.68	0.07	2.965	0.69	-0.22
3.015	0.68	0	2.966	0.72	-0.25	3.049	0.69	0	2.947	0.69	-0.07	2.987	0.69	0.07
3.037	0.68	0.07	2.986	0.71	-0.08	3.071	0.68	0.07	2.968	0.68	-5.64E-13	3.009	0.69	0.15
3.059	0.68	0.21	3.007	0.72	-1.03E-13	3.093	0.69	0.07	2.99	0.69	0.14	3.031	0.69	-0.07
3.081	0.69	0.07	3.028	0.71	-0.08	3.115	0.69	-0.07	3.012	0.69	2.34E-13	3.052	0.69	5.67E-13
3.102	0.69	0.07	3.049	0.71	-1.43E-13	3.136	0.69	-1.48E-13	3.034	0.69	0.07	3.074	0.69	-0.14
3.124	0.69	0.07	3.07	0.71	0	3.158	0.69	-0.14	3.056	0.69	-0.07	3.096	0.68	-0.76
3.146	0.69	0.07	3.091	0.71	-0.08	3.18	0.68	0.79	3.078	0.68	-0.62	3.118	0.66	-4.39
3.168	0.69	0	3.112	0.71	-4.39	3.202	0.72	-3.39	3.1	0.66	-4.32	3.149	0.38	-2.9
3.189	0.69	-2.4	3.14	0.43	-4.57	3.229	0.45	-5.48	3.13	0.39	-2.82	3.183	0.51	2.5

3.213	0.57	-5.1	3.173	0.49	6.17	3.262	0.45	6.62	3.164	0.52	5.26	3.211	0.55	2.82
3.245	0.4	1.98	3.198	0.74	-0.32	3.288	0.77	1.52	3.189	0.68		3.236	0.66	0.32
3.274	0.67	0.15	3.227	0.4	-10.58	3.326	0.27					3.261	0.57	-3.25
3.309	0.32	0.06	3.508	0.03								3.511	0.03	-0.97
3.344	0.67											3.799	0.15	0.02
												3.908	0.13	-0.13
												4.028	0.12	

Trial 19			Trial 20			Trial 21			Trial 22			Trial 23		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.033	0.69	-0.07	0.018	0.64	0.18	0.016	0.64	0.18	0.029	0.47	0.65	0.032	0.6	0
0.054	0.69	0.07	0.041	0.64	0.06	0.04	0.65	0.06	0.06	0.48	0.49	0.057	0.6	0.19
0.076	0.69	0.15	0.064	0.64	0.06	0.063	0.65	1.58E-30	0.091	0.5	0.44	0.082	0.61	0.2
0.097	0.7	0.07	0.088	0.65	0.12	0.086	0.65	0.06	0.12	0.51	0.57	0.107	0.61	0.2
0.119	0.7	1.03E-14	0.111	0.65	0.12	0.109	0.65	0.12	0.149	0.53	0.41	0.131	0.62	0.26
0.14	0.7	0.08	0.134	0.65	0.06	0.132	0.65	7.37E-15	0.178	0.53	0.35	0.155	0.62	0.27
0.162	0.7	1.59E-14	0.157	0.65	0.06	0.155	0.65	0.06	0.205	0.55	0.28	0.179	0.63	0.22
0.183	0.7	-0.08	0.18	0.66	0.06	0.178	0.66	0.06	0.233	0.55	0.19	0.203	0.63	0.11
0.205	0.7	0.15	0.203	0.66	0.06	0.201	0.65	-9.59E-15	0.26	0.56	0.27	0.227	0.64	0.06
0.226	0.7	3.11E-14	0.226	0.66	0.19	0.224	0.66	0.12	0.286	0.56	0.2	0.25	0.64	2.59E-14
0.248	0.7	-0.08	0.248	0.66	0.06	0.247	0.66	0.06	0.313	0.57	0.2	0.274	0.64	9.41E-15
0.269	0.7	0.08	0.271	0.66	-2.64E-14	0.27	0.66	0.06	0.339	0.57	0.3	0.298	0.64	0.06
0.29	0.7	0.08	0.293	0.66	0.06	0.292	0.66	0.06	0.365	0.59	0.31	0.321	0.64	0.12
0.312	0.7	0.23	0.316	0.66	6.14E-14	0.315	0.66	2.69E-14	0.39	0.59	0.28	0.345	0.64	0.12
0.333	0.71	0.08	0.339	0.66	0.07	0.338	0.66	0.13	0.416	0.6	0.24	0.368	0.64	0.24
0.354	0.71	-0.16	0.361	0.67	0.07	0.36	0.67	0.13	0.44	0.6	0.15	0.391	0.65	0.18
0.375	0.7	-0.16	0.384	0.67	-1.73E-14	0.383	0.67	-0.07	0.465	0.61	0.15	0.414	0.65	0.13
0.397	0.7	0.08	0.406	0.67	0.07	0.405	0.66	-0.07	0.49	0.61	0.1	0.437	0.66	0.12
0.418	0.71	-0.08	0.429	0.67	5.49E-14	0.428	0.66	-0.06	0.514	0.61	0.21	0.46	0.66	8.76E-14
0.439	0.7	-0.23	0.451	0.67	0.07	0.451	0.66	-5.45E-14	0.539	0.62	0.16	0.483	0.66	-8.76E-14
0.461	0.7	0.15	0.474	0.67	0.07	0.473	0.66	0.13	0.563	0.62	0.11	0.505	0.66	0.06
0.482	0.7	0.15	0.496	0.67	-0.13	0.496	0.67	0.2	0.587	0.62	0.27	0.528	0.66	-0.12
0.504	0.7	0.16	0.518	0.67	-0.07	0.518	0.67	0.13	0.611	0.63	0.17	0.551	0.65	-0.19
0.525	0.71	0.16	0.541	0.67	3.53E-28	0.541	0.67	-7.72E-14	0.635	0.63	0.11	0.574	0.65	0.06
0.546	0.71	-6.05E-14	0.563	0.67	0.07	0.563	0.67	-0.07	0.658	0.64	0.11	0.597	0.66	0.06
0.567	0.71	-0.08	0.586	0.67	0.07	0.585	0.67	-0.07	0.682	0.64	0.12	0.62	0.66	0

0.588	0.71	-0.08	0.608	0.67	-1.98E-14	0.608	0.67	5.45E-14	0.705	0.64	0.12	0.643	0.66	0.06
0.609	0.71	-0.08	0.631	0.67	0.07	0.63	0.67	-0.13	0.728	0.64	0.06	0.666	0.66	0.06
0.631	0.7	-0.08	0.653	0.67	0.14	0.653	0.66	-0.13	0.752	0.65	0.12	0.688	0.66	0
0.652	0.7	-0.15	0.675	0.68	0.07	0.675	0.66	1.25E-13	0.775	0.65	0.06	0.711	0.66	0.06
0.673	0.7	-0.08	0.697	0.68	-0.07	0.698	0.66	-0.06	0.798	0.65	0.06	0.734	0.66	-0.06
0.695	0.7	8.16E-14	0.72	0.67	0	0.72	0.66	3.05E-16	0.821	0.65	0.12	0.757	0.66	-0.06
0.716	0.7	0	0.742	0.68	0.14	0.743	0.66	0	0.844	0.66	0.12	0.78	0.66	0.06
0.738	0.7	-8.28E-14	0.764	0.68	0.07	0.766	0.66	0.07	0.867	0.66	0.26	0.802	0.66	0
0.759	0.7	-1.19E-13	0.786	0.68	-1.13E-13	0.788	0.67	0.13	0.889	0.67	0.19	0.825	0.66	0.06
0.781	0.7	8.28E-14	0.808	0.68	0.14	0.811	0.67	-0.07	0.912	0.67	0.07	0.848	0.66	0.06
0.802	0.7	-0.08	0.83	0.68	0.07	0.833	0.66	0.07	0.934	0.67	0.07	0.871	0.66	0.06
0.824	0.7	0.15	0.852	0.68	0.07	0.856	0.67	-6.97E-14	0.957	0.67	-0.07	0.893	0.66	0.06
0.845	0.7	-0.07	0.874	0.69	0.07	0.878	0.66	-0.07	0.979	0.67	-0.2	0.916	0.66	-1.06E-27
0.866	0.69	-0.23	0.896	0.68	7.98E-14	0.901	0.67	-0.06	1.002	0.66	7.51E-14	0.938	0.66	0
0.888	0.69	-0.07	0.918	0.69	1.08E-15	0.924	0.66	-0.07	1.024	0.67	0.13	0.961	0.66	-0.06
0.91	0.69	-1.17E-13	0.94	0.68	-1.96E-13	0.946	0.66	-0.13	1.047	0.67	-1.09E-13	0.984	0.66	-0.06
0.931	0.69	0.07	0.961	0.69	0.07	0.969	0.66	1.44E-13	1.069	0.67	0.07	1.006	0.66	9.09E-28
0.953	0.69	-0.07	0.983	0.69	-0.07	0.992	0.66	0.06	1.092	0.67	0.13	1.029	0.66	1.79E-13
0.975	0.69	8.2E-14	1.005	0.68	-8.02E-14	1.014	0.66	-7.46E-14	1.114	0.67	0	1.052	0.66	0.06
0.996	0.69	-0.07	1.027	0.69	-2.19E-15	1.037	0.66	0.06	1.137	0.67	2E-15	1.074	0.66	0.06
1.018	0.69	-0.15	1.049	0.68	-0.07	1.06	0.66	0.07	1.159	0.67	0	1.097	0.66	-0.06
1.04	0.69	-0.07	1.071	0.68	0.07	1.082	0.67	-0.13	1.181	0.67	-1.54E-13	1.12	0.66	0
1.062	0.68	-0.07	1.093	0.69	0.07	1.105	0.66	-0.13	1.204	0.67	0.07	1.142	0.66	1.89E-15
1.083	0.68	-0.07	1.114	0.69	-1.6E-13	1.128	0.66	0.19	1.226	0.67	0.07	1.165	0.66	0.07
1.105	0.68	-0.14	1.136	0.69	-0.07	1.151	0.66	-2.5E-13	1.248	0.68	0.14	1.188	0.67	0.13
1.127	0.68	-0.07	1.158	0.68	0	1.173	0.66	-0.13	1.27	0.68	0	1.21	0.67	0.07
1.15	0.68	1.13E-13	1.18	0.69	2.77E-13	1.196	0.66	2.47E-13	1.292	0.68	1.12E-13	1.233	0.67	0.07
1.172	0.68	0	1.202	0.68	-0.07	1.219	0.66	0.19	1.315	0.68	0	1.255	0.67	0.14
1.194	0.68	0	1.224	0.68	0.07	1.241	0.67	0.06	1.337	0.68	-0.07	1.277	0.68	0.13
1.216	0.68	0.07	1.245	0.69	2.19E-15	1.264	0.66	-1.48E-13	1.359	0.68	-0.07	1.3	0.68	-0.07
1.238	0.68	0	1.267	0.68	-0.14	1.286	0.67	0.13	1.381	0.67	-0.14	1.322	0.67	0
1.26	0.68	-2.7E-13	1.289	0.68	-0.07	1.309	0.67	3.64E-14	1.404	0.67	-1.14E-13	1.344	0.68	-1.1E-13
1.282	0.68	-0.07	1.311	0.68	-0.07	1.331	0.67	-0.07	1.426	0.67	0.07	1.366	0.67	-0.07
1.304	0.68	-0.21	1.333	0.68	0.07	1.354	0.67	0.07	1.448	0.67	-0.07	1.389	0.67	0.07
1.326	0.67	-0.14	1.355	0.68	0	1.376	0.67	6.66E-16	1.471	0.67	0.14	1.411	0.68	-0.07
1.349	0.67	-0.07	1.377	0.68	-0.14	1.399	0.67	0	1.493	0.68	0.13	1.433	0.67	-0.07
1.371	0.67	0.07	1.399	0.68	0	1.421	0.67	0.13	1.515	0.68	6.94E-16	1.455	0.67	0.07
1.393	0.67	0.07	1.422	0.68	-0.07	1.443	0.67	-1.45E-13	1.537	0.68	0.07	1.478	0.67	0.07
1.416	0.67	0.07	1.444	0.68	-0.14	1.466	0.67	-1.12E-13	1.559	0.68	-1.13E-13	1.5	0.68	0.14
1.438	0.68	0.07	1.466	0.67	-0.14	1.488	0.67	2.56E-13	1.581	0.68	-0.14	1.522	0.68	0.14
1.46	0.68	9.04E-27	1.488	0.67	-0.13	1.51	0.67	-0.07	1.604	0.67	-0.07	1.544	0.68	0.07

1.482	0.68	8.94E-27	1.511	0.67	-0.07	1.533	0.67	-0.07	1.626	0.68	0.07	1.566	0.68	-0.14
1.505	0.68	9.04E-27	1.533	0.67	0	1.555	0.67	-0.07	1.648	0.68	0.07	1.588	0.68	-0.14
1.527	0.68	8.94E-27	1.556	0.67	0.07	1.578	0.67	0.07	1.67	0.68	0.07	1.61	0.68	-0.07
1.549	0.68	9.04E-27	1.578	0.67	0.07	1.6	0.67	0.13	1.692	0.68	0.14	1.633	0.67	-0.2
1.571	0.68	0.07	1.601	0.67	-0.07	1.623	0.67	-2.21E-13	1.714	0.68	0.21	1.655	0.67	-0.13
1.593	0.68	2.26E-13	1.623	0.67	-0.13	1.645	0.67	-0.13	1.736	0.69	0.07	1.678	0.67	-0.07
1.615	0.68	-0.2	1.646	0.66	1.47E-13	1.667	0.67	0	1.758	0.69	0.07	1.7	0.66	-0.13
1.638	0.67	-0.14	1.668	0.67	0	1.69	0.67	-0.07	1.78	0.69	-0.07	1.723	0.66	0.07
1.66	0.67	-0.07	1.691	0.66	-0.13	1.712	0.66	-0.13	1.801	0.68	0	1.745	0.67	-1.39E-13
1.683	0.67	0.07	1.713	0.66	-1.45E-13	1.735	0.66	0.07	1.823	0.69	0	1.768	0.66	-0.07
1.705	0.67	0.2	1.736	0.66	0	1.758	0.67	-0.06	1.845	0.68	-0.07	1.791	0.66	0.06
1.727	0.68	0.14	1.759	0.66	-0.13	1.78	0.66	-0.07	1.867	0.69	-0.07	1.813	0.66	-2.19E-13
1.749	0.68	0.07	1.781	0.66	0.06	1.803	0.66	0.13	1.889	0.68	-0.21	1.836	0.66	-0.06
1.772	0.68	-0.07	1.804	0.66	-1.42E-13	1.825	0.67	0	1.911	0.68	0	1.859	0.66	-0.06
1.794	0.68	-0.07	1.827	0.66	-0.13	1.848	0.66	-0.13	1.933	0.68	-6.94E-16	1.881	0.66	-0.13
1.816	0.68	-8.94E-27	1.85	0.66	1.41E-13	1.871	0.66	-1.45E-13	1.955	0.68	0.14	1.904	0.66	-0.19
1.838	0.68	-9.04E-27	1.872	0.66	-0.06	1.893	0.66	0.13	1.977	0.69	0.29	1.927	0.65	-0.12
1.86	0.68	-8.94E-27	1.895	0.66	0	1.916	0.67	0.07	1.999	0.69	0.15	1.95	0.65	-0.12
1.882	0.68	-2.25E-13	1.918	0.66	-7.29E-14	1.938	0.67	-2.2E-13	2.02	0.69	0.07	1.973	0.65	-0.06
1.905	0.68		1.941	0.66	0.06	1.961	0.67	2.2E-13	2.042	0.69	-0.07	1.996	0.65	7.42E-14
			1.964	0.66	0.06	1.983	0.67	-0.13	2.064	0.69	-0.15	2.02	0.65	2.68E-13
			1.986	0.66	1.42E-13	2.006	0.66	-0.13	2.085	0.69	-0.14	2.043	0.65	0.06
			2.009	0.66	0.06	2.029	0.66	0.06	2.107	0.68	-0.07	2.066	0.65	0
			2.032	0.66	1.69E-26	2.051	0.66	5.05E-15	2.129	0.68	1.87E-26	2.089	0.65	-0.06
			2.055	0.66	-0.13	2.074	0.66	-0.13	2.151	0.68	0.15	2.112	0.65	0.06
			2.077	0.66	-0.13	2.097	0.66	-0.06	2.173	0.69	0.07	2.135	0.65	0.06
			2.1	0.66	2.76E-13	2.119	0.66		2.195	0.69	0	2.159	0.65	0
			2.123	0.66	-2.13E-13				2.216	0.69	-2.35E-13	2.182	0.65	-0.06
			2.146	0.66	-0.06				2.238	0.69	-0.07	2.205	0.65	-0.06
			2.169	0.65	-0.06				2.26	0.69	5.28E-27	2.228	0.65	-0.06
			2.192	0.65	0				2.282	0.69	3.21E-13	2.251	0.64	-0.12
			2.215	0.65	-0.06				2.303	0.69	-3.21E-13	2.275	0.64	2.68E-13
			2.238	0.65	4.72E-15				2.325	0.69	-0.07	2.298	0.64	0.12
			2.261	0.65	-0.06				2.347	0.68	-0.21	2.321	0.65	0.06
			2.284	0.65	-0.18				2.369	0.68	-0.21	2.344	0.65	-3.46E-27
			2.308	0.64	-0.29				2.391	0.68	0	2.368	0.65	-2.68E-13
			2.331	0.63	-0.5				2.413	0.68	0.07	2.391	0.65	0.06
			2.355	0.62					2.435	0.68	-0.07	2.414	0.65	0.06
									2.458	0.68	1.39E-15	2.437	0.65	-0.06
									2.48	0.68	0.07	2.46	0.65	0
									2.502	0.68	-0.07	2.483	0.65	0.12

									2.524	0.68	-0.14	2.506	0.65	-0.06
									2.546	0.67	-0.2	2.53	0.65	-0.12
									2.569	0.67	-0.07	2.553	0.65	
									2.591	0.67	2.2E-13			
									2.614	0.67	-0.07			
									2.636	0.67	-0.07			
									2.659	0.66	-0.13			
									2.681	0.66	-0.19			
									2.704	0.66	-0.19			
									2.727	0.65	-0.06			
									2.75	0.65	0.06			
									2.773	0.66	0.06			
									2.796	0.66	-0.12			
									2.819	0.65	-0.12			
									2.842	0.65	-0.12			
									2.865	0.64	-0.18			
									2.888	0.64				

Trial 24			Trial 25a			Trial 25b			Trial 26			Trial 27		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.02	0.67	-0.27	0.02	0.67	-0.27	0.033	0.63	0.11	0.029	0.61	-0.46	0.023	0.63	0.28
0.043	0.67	-2.41E-15	0.043	0.67	-2.41E-15	0.056	0.64	0.11	0.054	0.61	0.05	0.047	0.64	0.17
0.065	0.67	0.14	0.065	0.67	0.14	0.08	0.64	0.12	0.078	0.62	0.26	0.071	0.64	0.06
0.087	0.68	0.07	0.087	0.68	0.07	0.103	0.64	0.06	0.102	0.62	0.11	0.094	0.64	4.72E-15
0.11	0.68	0.07	0.11	0.68	0.07	0.127	0.64	0.12	0.126	0.62	0.05	0.118	0.64	0.06
0.132	0.68	0.14	0.132	0.68	0.14	0.15	0.65	0.12	0.15	0.63	0.22	0.141	0.64	0.12
0.154	0.68	0.29	0.154	0.68	0.29	0.173	0.65	0.06	0.174	0.63	0.22	0.164	0.64	0.12
0.176	0.69	0.14	0.176	0.69	0.14	0.196	0.65	0.18	0.198	0.64	0.23	0.188	0.65	0.18
0.197	0.69	-0.07	0.197	0.69	-0.07	0.219	0.66	0.19	0.221	0.64	0.29	0.211	0.65	0.18
0.219	0.69	3.06E-14	0.219	0.69	3.06E-14	0.242	0.66	0.19	0.244	0.65	0.06	0.234	0.66	0.19
0.241	0.69	-0.14	0.241	0.69	-0.14	0.265	0.66	0.13	0.268	0.65	2.64E-14	0.256	0.66	0.26
0.263	0.68	-0.07	0.263	0.68	-0.07	0.287	0.66	-2.7E-14	0.291	0.65	-2.4E-14	0.279	0.67	0.2
0.285	0.68	-0.07	0.285	0.68	-0.07	0.31	0.66	0.13	0.314	0.65	-0.18	0.302	0.67	0.2
0.307	0.68	-0.21	0.307	0.68	-0.21	0.333	0.67	0.07	0.337	0.64	-0.06	0.324	0.68	0.13
0.329	0.68	0.07	0.329	0.68	0.07	0.355	0.67	-0.07	0.361	0.64	0.06	0.346	0.68	0.07
0.351	0.68	0.14	0.351	0.68	0.14	0.378	0.67	0.07	0.384	0.64	0.12	0.368	0.68	0.07
0.373	0.68	0.14	0.373	0.68	0.14	0.4	0.67	0.13	0.407	0.65	0.18	0.39	0.68	-5.78E-14
0.395	0.69	0.14	0.395	0.69	0.14	0.422	0.67	0.07	0.43	0.65	0.25	0.412	0.68	0

0.417	0.69	0.07	0.417	0.69	0.07	0.445	0.67	0.07	0.453	0.66	0.32	0.434	0.68	5.78E-14
0.439	0.69	0.15	0.439	0.69	0.15	0.467	0.68	0.07	0.476	0.67	0.13	0.457	0.68	-5.78E-14
0.46	0.69	0.07	0.46	0.69	0.07	0.489	0.68	0	0.498	0.67	0.07	0.479	0.68	0.07
0.482	0.69	0.08	0.482	0.69	0.08	0.511	0.68	5.5E-14	0.52	0.67	-1.7E-14	0.501	0.68	0.14
0.503	0.7	0.07	0.503	0.7	0.07	0.533	0.68	0	0.543	0.67	-0.07	0.523	0.68	0.07
0.525	0.7	0.08	0.525	0.7	0.08	0.556	0.68	-0.13	0.565	0.67	7.15E-14	0.545	0.68	-7.86E-14
0.546	0.7	-5.95E-14	0.546	0.7	-5.95E-14	0.578	0.67	-0.2	0.588	0.67	-0.13	0.566	0.68	-5.83E-14
0.568	0.7	-0.08	0.568	0.7	-0.08	0.6	0.67	-3.05E-16	0.611	0.66	-0.07	0.588	0.68	0.15
0.589	0.7	-2.32E-14	0.589	0.7	-2.32E-14	0.623	0.67	0.13	0.633	0.66	5.39E-14	0.61	0.69	0.14
0.611	0.7	0.08	0.611	0.7	0.08	0.645	0.67	0.14	0.656	0.66	0.07	0.632	0.69	-5.88E-14
0.632	0.7	0.15	0.632	0.7	0.15	0.667	0.68	0.07	0.678	0.67	0.26	0.654	0.69	0.15
0.654	0.7	0.08	0.654	0.7	0.08	0.69	0.68	0.14	0.701	0.67	0.2	0.675	0.7	0.07
0.675	0.7	0.16	0.675	0.7	0.16	0.712	0.68	0.14	0.723	0.68	0	0.697	0.69	0.08
0.696	0.71	0.24	0.696	0.71	0.24	0.734	0.68	0.07	0.745	0.67	-0.07	0.718	0.7	0.07
0.717	0.71	0.16	0.717	0.71	0.16	0.756	0.68	0.07	0.768	0.67	-0.07	0.74	0.7	-0.08
0.738	0.72	-0.08	0.738	0.72	-0.08	0.778	0.68	-0.07	0.79	0.67	-0.07	0.761	0.7	2.52E-29
0.759	0.71	-0.16	0.759	0.71	-0.16	0.8	0.68	-0.07	0.812	0.67	-0.2	0.783	0.7	-0.07
0.78	0.71	-8.94E-14	0.78	0.71	-8.94E-14	0.822	0.68	0.07	0.835	0.66	-0.2	0.804	0.69	-0.08
0.801	0.71	-0.16	0.801	0.71	-0.16	0.844	0.68	0.07	0.858	0.66	-0.06	0.826	0.69	-0.07
0.822	0.7	-0.23	0.822	0.7	-0.23	0.865	0.68	-3.79E-28	0.88	0.66	-0.13	0.847	0.69	-0.07
0.844	0.7	-0.08	0.844	0.7	-0.08	0.887	0.68	0.07	0.903	0.66	-0.06	0.869	0.69	-0.07
0.865	0.7	-0.15	0.865	0.7	-0.15	0.909	0.69	0.07	0.926	0.66	-7.03E-14	0.891	0.69	-0.07
0.887	0.69	-8.18E-14	0.887	0.69	-8.18E-14	0.931	0.69	-0.07	0.949	0.66	0	0.913	0.69	-1.21E-27
0.908	0.7	0.07	0.908	0.7	0.07	0.953	0.68	-1.97E-13	0.972	0.66	0.06	0.934	0.69	0.15
0.93	0.7	-0.15	0.93	0.7	-0.15	0.975	0.69	0.07	0.995	0.66	7.04E-14	0.956	0.69	0.22
0.951	0.69	-0.08	0.951	0.69	-0.08	0.997	0.69	2.78E-13	1.018	0.66	-0.06	0.978	0.7	0.07
0.973	0.69	-0.07	0.973	0.69	-0.07	1.018	0.69	0.07	1.041	0.66	0.06	0.999	0.7	0.08
0.994	0.69	-0.15	0.994	0.69	-0.15	1.04	0.69	-1.61E-13	1.063	0.66	-0.06	1.021	0.7	0.08
1.016	0.69	-0.22	1.016	0.69	-0.22	1.062	0.69	-7.49E-16	1.086	0.65	-0.06	1.042	0.7	-0.08
1.038	0.68	-0.21	1.038	0.68	-0.21	1.084	0.69	-0.07	1.109	0.66	1.35E-13	1.064	0.7	-0.08
1.06	0.68	-0.14	1.06	0.68	-0.14	1.105	0.68	-0.07	1.132	0.65	-1.37E-13	1.085	0.7	-0.15
1.082	0.68	-0.2	1.082	0.68	-0.2	1.127	0.69	0	1.155	0.66	-0.06	1.107	0.69	-0.15
1.105	0.67	-0.07	1.105	0.67	-0.07	1.149	0.68	-0.21	1.178	0.65	-0.18	1.128	0.69	-2.81E-13
1.127	0.67	0.07	1.127	0.67	0.07	1.171	0.68	-0.07	1.201	0.65	-0.12	1.15	0.69	0.07
1.149	0.67	-0.07	1.149	0.67	-0.07	1.193	0.68	0.07	1.225	0.64	0.06	1.172	0.69	-0.07
1.172	0.67	1.54E-13	1.172	0.67	1.54E-13	1.215	0.68	-0.07	1.248	0.65	0.18	1.193	0.69	0
1.194	0.67	-0.07	1.194	0.67	-0.07	1.237	0.68	0	1.271	0.65	1.35E-13	1.215	0.69	0.07
1.216	0.67	-0.26	1.216	0.67	-0.26	1.259	0.68	0.07	1.294	0.65	0.13	1.237	0.69	0.08
1.239	0.66	-0.19	1.239	0.66	-0.19	1.281	0.68	0.07	1.317	0.66	0.12	1.258	0.7	0.15
1.262	0.66	-0.19	1.262	0.66	-0.19	1.303	0.68	0.14	1.34	0.66	0.06	1.28	0.7	-2.89E-13
1.285	0.65	-0.13	1.285	0.65	-0.13	1.325	0.69	0.07	1.362	0.66	0.13	1.301	0.7	1.68E-13

1.308	0.65	1.06E-13	1.308	0.65	1.06E-13	1.347	0.69	-0.07	1.385	0.66	-0.06	1.323	0.7	1.21E-13
1.331	0.65	0	1.331	0.65	0	1.369	0.68	-1.6E-13	1.408	0.66	1.44E-13	1.344	0.7	-0.07
1.354	0.65	0.13	1.354	0.65	0.13	1.39	0.69	0.07	1.431	0.66	-0.06	1.366	0.69	-0.08
1.377	0.66	0.12	1.377	0.66	0.12	1.412	0.69	-0.07	1.453	0.66	-0.06	1.388	0.69	1.67E-13
1.399	0.66	0.13	1.399	0.66	0.13	1.434	0.68	-0.07	1.476	0.66	0.19	1.409	0.69	0
1.422	0.66	0.13	1.422	0.66	0.13	1.456	0.68	-0.14	1.499	0.66	0.13	1.431	0.69	-0.07
1.445	0.66	1.11E-13	1.445	0.66	1.11E-13	1.478	0.68	-0.21	1.522	0.66	0.07	1.452	0.69	-0.07
1.467	0.66	-1.45E-13	1.467	0.66	-1.45E-13	1.5	0.68	-0.2	1.544	0.67	0.07	1.474	0.69	1.64E-13
1.49	0.66	-0.13	1.49	0.66	-0.13	1.522	0.67	-0.07	1.567	0.67	0.07	1.496	0.69	7.32E-28
1.513	0.66	-0.13	1.513	0.66	-0.13	1.545	0.67	0.14	1.589	0.67	0.13	1.517	0.69	-1.64E-13
1.535	0.66	0.06	1.535	0.66	0.06	1.567	0.68	0.14	1.611	0.67	0.07	1.539	0.69	0
1.558	0.66	0.19	1.558	0.66	0.19	1.589	0.68	0.14	1.634	0.67	0.07	1.561	0.69	1.64E-13
1.581	0.67	0.26	1.581	0.67	0.26	1.611	0.68	0.14	1.656	0.68	-0.07	1.583	0.69	-2.35E-13
1.603	0.67	0.2	1.603	0.67	0.2	1.633	0.68	0.14	1.678	0.67	-0.14	1.604	0.69	0.15
1.625	0.68	0.14	1.625	0.68	0.14	1.655	0.69	0.14	1.701	0.67	-0.07	1.626	0.7	0.07
1.647	0.68	0.07	1.647	0.68	0.07	1.677	0.69	0.15	1.723	0.67	-0.2	1.647	0.69	-0.08
1.67	0.68	0.07	1.67	0.68	0.07	1.699	0.69	0.15	1.746	0.66	-0.13	1.669	0.69	-1.64E-13
1.692	0.68	0.14	1.692	0.68	0.14	1.72	0.7	0.15	1.768	0.66	0.06	1.691	0.69	-0.15
1.714	0.68	0.07	1.714	0.68	0.07	1.741	0.7	-1.65E-13	1.791	0.66	-1.92E-15	1.712	0.69	-0.07
1.735	0.68	0.07	1.735	0.68	0.07	1.763	0.7	-0.15	1.814	0.66	0.07	1.734	0.69	0.07
1.757	0.69	2.33E-13	1.757	0.69	2.33E-13	1.784	0.69	-0.15	1.836	0.67	0.2	1.756	0.69	0.07
1.779	0.68	0	1.779	0.68	0	1.806	0.69	-0.15	1.859	0.67	-0.07	1.777	0.69	0.07
1.801	0.69	0.07	1.801	0.69	0.07	1.828	0.69	-0.07	1.881	0.66	1.48E-13	1.799	0.69	1.64E-13
1.823	0.69	0.07	1.823	0.69	0.07	1.85	0.69	-0.07	1.904	0.67	0.07	1.821	0.69	0
1.845	0.69	0	1.845	0.69	0	1.872	0.68	0	1.926	0.67	-0.13	1.842	0.69	-0.07
1.866	0.69	-0.07	1.866	0.69	-0.07	1.893	0.69	0.07	1.949	0.66	-0.13	1.864	0.69	-0.07
1.888	0.69		1.888	0.69		1.915	0.69	0.07	1.971	0.66	-0.06	1.886	0.69	-7.16E-14
						1.937	0.69	0.07	1.994	0.66	-0.06	1.907	0.69	2.35E-13
						1.959	0.69	0.15	2.017	0.66	0.13	1.929	0.69	-1.64E-13
						1.98	0.7	0.15	2.04	0.67	0.19	1.951	0.69	0.07
						2.002	0.7	3.36E-13	2.062	0.67	0	1.972	0.69	0.07
						2.023	0.7	-0.07	2.085	0.67	0.07	1.994	0.69	-0.15
						2.045	0.69	-0.15	2.107	0.67	-0.07	2.016	0.69	-0.15
						2.066	0.69	-0.15	2.13	0.66	-0.07	2.037	0.69	-3.21E-13
						2.088	0.69	-0.22	2.152	0.67	0.13	2.059	0.69	-0.14
						2.11	0.68	-0.21	2.175	0.67	-2.2E-13	2.081	0.68	-0.14
						2.132	0.68	5.41E-13	2.197	0.67	5.17E-13	2.103	0.68	-0.07
						2.154	0.68	0.07	2.219	0.67	-0.07	2.125	0.68	
						2.176	0.68	0.07	2.242	0.66	-0.07			
						2.198	0.68	-3.15E-13	2.264	0.67	2.9E-13			
						2.22	0.68		2.287	0.66	-0.07			

									2.31	0.66				

Trial 28			Trial 29			Trial 30			Trial 31			Trial 32		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.03	0.62	0.37	0.032	0.65	-0.41	0.037	0.55	0.8	0.033	0.65	0.37	0.033	0.61	-0.06
0.054	0.63	0.27	0.055	0.64	-0.06	0.064	0.56	0.26	0.056	0.66	0.12	0.057	0.61	0.26
0.077	0.63	0.17	0.079	0.64	0.12	0.09	0.56	0.2	0.079	0.66	-0.06	0.081	0.62	0.16
0.101	0.64	0.17	0.102	0.65	-0.12	0.117	0.57	0.24	0.102	0.65	0.06	0.105	0.62	-0.16
0.125	0.64	0.06	0.125	0.64	-5E-16	0.143	0.57	0.26	0.125	0.66	0.19	0.129	0.62	0.22
0.148	0.64	-0.06	0.149	0.65	0.42	0.169	0.59	0.26	0.148	0.66	0.13	0.153	0.63	0.05
0.171	0.64	0.06	0.172	0.66	0.31	0.194	0.59	0.04	0.171	0.66	0.2	0.177	0.62	-0.22
0.195	0.64	0.12	0.194	0.66	0.06	0.22	0.59	0.23	0.193	0.67	0.13	0.202	0.62	0.39
0.218	0.64	0.12	0.217	0.66	-2.69E-14	0.245	0.6	0.33	0.216	0.67	0.07	0.225	0.64	0.33
0.241	0.65	0.18	0.24	0.66	0.06	0.27	0.6	0.2	0.238	0.67	4.47E-14	0.249	0.64	-0.06
0.265	0.65	0.25	0.262	0.66	0.2	0.295	0.61	0.05	0.26	0.67	-0.07	0.272	0.64	0.24
0.287	0.66	0.19	0.285	0.67	0.2	0.319	0.61	-0.1	0.283	0.67	-0.07	0.296	0.65	0.29
0.31	0.66	0.13	0.307	0.67	-0.2	0.344	0.6	0.31	0.305	0.67	-0.07	0.319	0.65	-0.18
0.333	0.66	0.2	0.33	0.66	-0.26	0.369	0.62	0.52	0.328	0.67	0.13	0.342	0.64	2.63E-14
0.355	0.67	0.13	0.352	0.66	0.33	0.393	0.63	-0.05	0.35	0.67	0.13	0.365	0.65	0.18
0.378	0.67	-0.13	0.375	0.68	0.06	0.417	0.62	-0.06	0.372	0.67	-1.01E-28	0.388	0.65	-0.06
0.4	0.66	0.07	0.397	0.66	-0.2	0.441	0.63	0.33	0.395	0.67	0.07	0.412	0.65	-0.06
0.423	0.67	0.13	0.42	0.67	0.07	0.464	0.64	-0.05	0.417	0.68	0.07	0.435	0.65	0.06
0.445	0.67	-0.13	0.442	0.67	0.2	0.488	0.63	-0.17	0.439	0.68	0.14	0.458	0.65	-0.06
0.468	0.67	-0.13	0.465	0.68	0.41	0.512	0.63	0.22	0.461	0.68	0.14	0.481	0.64	-0.06
0.49	0.66	-0.13	0.487	0.68	0.07	0.536	0.64	0.17	0.483	0.68	-0.07	0.504	0.65	0.18
0.513	0.66	0.07	0.509	0.68	-0.28	0.559	0.64	0.12	0.505	0.68	1.73E-14	0.528	0.65	0.25
0.535	0.67	0.2	0.531	0.67	0.07	0.583	0.64	0.17	0.527	0.68	0	0.55	0.66	0.25
0.558	0.67	0.07	0.553	0.68	0.14	0.606	0.64	-0.06	0.549	0.68	-0.07	0.573	0.66	6.11E-16
0.58	0.67	0.07	0.575	0.68	-0.07	0.63	0.64	-4.76E-14	0.571	0.68	-0.07	0.596	0.66	-0.13
0.603	0.67	0.07	0.597	0.68	0.07	0.653	0.64	0.06	0.593	0.68	-0.07	0.619	0.66	-7.06E-14
0.625	0.67	0.07	0.619	0.68	-0.14	0.676	0.64	-0.06	0.616	0.68	0.14	0.641	0.66	-0.12
0.647	0.68	0.07	0.641	0.67	-0.07	0.7	0.64	-0.06	0.638	0.68	0.07	0.664	0.65	-5.33E-14
0.669	0.68	-0.07	0.664	0.68	0.35	0.723	0.64	0.24	0.66	0.68	-0.07	0.687	0.66	0.19
0.692	0.67	5.49E-14	0.686	0.69	0.14	0.747	0.65	0.18	0.682	0.68	-0.07	0.71	0.66	-0.06
0.714	0.68	-0.07	0.707	0.68	-0.21	0.77	0.65	-0.06	0.704	0.68	-0.07	0.733	0.66	-0.06
0.736	0.67	-0.27	0.729	0.68	-0.07	0.793	0.65	0.06	0.726	0.68	0.07	0.756	0.66	0.26
0.759	0.66	-0.07	0.751	0.68	0.21	0.816	0.65	0	0.748	0.68	-0.13	0.778	0.67	-0.06
0.781	0.67	7.04E-14	0.773	0.69	-0.21	0.839	0.65	0.06	0.771	0.67	-0.07	0.801	0.66	-0.19

0.804	0.66	-1.82E-13	0.795	0.67	-0.35	0.862	0.65	0.12	0.793	0.68	0.07	0.824	0.66	0.13
0.826	0.67	9.44E-16	0.818	0.67	-0.07	0.885	0.66	0	0.815	0.67	-0.07	0.847	0.66	-0.19
0.849	0.66	1.11E-13	0.84	0.67	3.48E-14	0.908	0.65	3.54E-14	0.838	0.67	1.12E-13	0.869	0.65	-0.06
0.871	0.67	0.13	0.862	0.67	9.99E-16	0.931	0.66	0.06	0.86	0.67	0	0.892	0.66	0.06
0.894	0.67	0.13	0.885	0.67	7.8E-14	0.954	0.66	-0.12	0.882	0.67	0	0.915	0.65	-0.31
0.916	0.67	0	0.907	0.67	0.07	0.977	0.65	-0.12	0.904	0.67	0	0.938	0.64	0.19
0.938	0.67	-0.07	0.929	0.67	0	1	0.65	0.19	0.927	0.67	-7.72E-14	0.961	0.66	0.24
0.961	0.67	0.14	0.952	0.67	0.21	1.023	0.66	0.06	0.949	0.67	-1.12E-13	0.984	0.66	-0.06
0.983	0.68	0.13	0.974	0.68	-0.07	1.046	0.65	0.06	0.971	0.67	1.52E-13	1.007	0.66	0.13
1.005	0.68	-0.07	0.996	0.67	-0.28	1.069	0.66	0.25	0.994	0.67	0.07	1.03	0.66	-0.12
1.028	0.67	0	1.018	0.67	0.14	1.091	0.66	0.13	1.016	0.68	-0.07	1.053	0.65	-0.06
1.05	0.68	-2.03E-15	1.041	0.68	-0.19	1.114	0.67	0.07	1.038	0.67	-0.14	1.076	0.66	-0.06
1.072	0.67	-0.14	1.063	0.66	-0.46	1.136	0.67	-0.07	1.061	0.67	0.07	1.099	0.65	-0.37
1.094	0.67	2E-15	1.086	0.66	-0.06	1.159	0.66	-0.07	1.083	0.67	0	1.122	0.64	-0.06
1.117	0.67	-1.45E-13	1.109	0.66	0	1.181	0.66	0	1.105	0.67	-0.26	1.145	0.65	0.18
1.139	0.67	-0.07	1.132	0.66	0.06	1.204	0.66	-0.06	1.128	0.66	-0.13	1.168	0.65	0.06
1.162	0.67	3.72E-14	1.155	0.66	-0.12	1.227	0.66	-0.06	1.151	0.66	-2.5E-13	1.191	0.65	-0.06
1.184	0.67	0.14	1.177	0.65	-2.44E-15	1.249	0.66	-0.19	1.173	0.66	-0.13	1.215	0.65	0.12
1.206	0.68	0.13	1.2	0.66	0.19	1.272	0.65	-0.19	1.196	0.66	-0.13	1.238	0.66	0.12
1.228	0.68	0	1.223	0.66	-0.13	1.295	0.65	-0.06	1.219	0.66	-0.06	1.26	0.65	1.37E-13
1.251	0.68	0.07	1.246	0.66	-0.12	1.318	0.65	-0.12	1.242	0.66	-1.31E-27	1.283	0.66	-0.06
1.273	0.68	0	1.269	0.65	-0.06	1.341	0.65	-2.3E-15	1.265	0.66	-0.06	1.306	0.65	-0.12
1.295	0.68	-6.94E-16	1.292	0.65	-0.06	1.365	0.65	-2.33E-15	1.288	0.65	-0.06	1.33	0.65	0
1.317	0.68	-1.13E-13	1.315	0.65	-0.18	1.388	0.65	0.06	1.31	0.65	-0.06	1.353	0.65	-1.03E-13
1.339	0.68	-0.07	1.338	0.64	1.35E-13	1.411	0.65	0.25	1.334	0.65	-0.12	1.376	0.65	0
1.361	0.68	1.13E-13	1.361	0.65	0.24	1.434	0.66	2.41E-13	1.357	0.65	-0.18	1.399	0.65	0.25
1.384	0.68	0.07	1.384	0.66	0.12	1.457	0.65	-1.41E-13	1.38	0.64	-0.06	1.422	0.66	0.31
1.406	0.68	0.07	1.407	0.66	0.13	1.48	0.66	0	1.403	0.64	0.12	1.444	0.66	0.13
1.428	0.68	0.07	1.43	0.66	0.19	1.502	0.65	-0.13	1.427	0.65	-0.06	1.467	0.67	0
1.45	0.68	0.14	1.453	0.66	-1.92E-15	1.525	0.65	-0.12	1.45	0.64	0	1.49	0.66	-1.45E-13
1.472	0.68	4.3E-14	1.475	0.66	0.07	1.549	0.65	-0.06	1.473	0.65	0.06	1.512	0.67	-0.19
1.494	0.68	-0.07	1.498	0.67	1.12E-13	1.572	0.65	1.03E-13	1.496	0.64	-0.12	1.535	0.66	-0.26
1.516	0.68	-1.56E-13	1.521	0.66	-0.13	1.595	0.65	-0.18	1.52	0.64	-0.12	1.558	0.66	-0.24
1.538	0.68	-0.07	1.543	0.66	0.13	1.618	0.64	-1.11E-15	1.543	0.64	-0.06	1.581	0.64	-0.18
1.56	0.68	-0.07	1.566	0.67	0.06	1.641	0.65	0.24	1.567	0.64	-0.17	1.604	0.65	0.18
1.582	0.68	5.05E-29	1.588	0.66	-1.47E-13	1.665	0.65	0.12	1.59	0.63	-0.11	1.627	0.65	0.12
1.604	0.68	-5.05E-29	1.611	0.67	0.13	1.688	0.65	0.13	1.614	0.63	0.17	1.65	0.65	0.13
1.626	0.68	0.07	1.633	0.67	-7.17E-14	1.711	0.66	0.06	1.638	0.64	0.17	1.673	0.66	0.32
1.648	0.68	0.07	1.656	0.67	-0.07	1.733	0.66	-0.06	1.661	0.64	0.12	1.696	0.67	0.13
1.67	0.68	0.07	1.678	0.67	0.07	1.756	0.66	0.06	1.685	0.64	0.18	1.718	0.66	0.07
1.692	0.68	0.07	1.701	0.67	-6.38E-16	1.779	0.66	-0.12	1.708	0.65	1.19E-15	1.741	0.67	0.07

1.714	0.68	0	1.723	0.67	0	1.802	0.65	-0.31	1.731	0.64	-0.12	1.763	0.67	-0.2
1.736	0.68	0.07	1.746	0.67	0.07	1.825	0.64	-0.06	1.754	0.64	-3.23E-27	1.786	0.66	-0.13
1.758	0.69	0.14	1.768	0.67	0	1.849	0.65	0	1.778	0.64	-0.12	1.808	0.66	-0.25
1.78	0.69	-0.07	1.791	0.67	-7.19E-14	1.872	0.64	0.12	1.801	0.64	-0.06	1.831	0.65	-0.25
1.801	0.68	-0.22	1.813	0.67	0.07	1.895	0.65	0.12	1.824	0.64	3.28E-13	1.854	0.65	-0.06
1.823	0.68	-0.21	1.835	0.67	0.14	1.918	0.65	0.19	1.848	0.64	0	1.878	0.65	-0.06
1.845	0.68	-0.21	1.857	0.68	2.05E-15	1.941	0.66	0.25	1.871	0.64	0.12	1.901	0.65	0.12
1.868	0.67	-0.14	1.88	0.67	0	1.964	0.66	0.13	1.895	0.64	0.24	1.924	0.65	0.06
1.89	0.67	-2E-15	1.902	0.68	-2.03E-15	1.986	0.67	0.06	1.918	0.65	0.24	1.947	0.65	-0.06
1.912	0.67	0.07	1.924	0.67	-0.07	2.009	0.66	-0.07	1.941	0.66	0.06	1.97	0.65	-1.37E-13
1.935	0.67	0.07	1.947	0.67	0.07	2.031	0.66	-0.13	1.964	0.66	0.06	1.993	0.65	-0.24
1.957	0.68	2.24E-13	1.969	0.68	0.14	2.054	0.66	-0.06	1.986	0.66	-0.06	2.016	0.64	-0.24
1.979	0.67	0	1.991	0.68	0.07	2.077	0.66	-0.06	2.009	0.65	-0.19	2.04	0.64	-0.17
2.002	0.68	-2.72E-15	2.013	0.68	0.14	2.1	0.66	-0.19	2.032	0.65	-0.24	2.064	0.63	-0.17
2.024	0.67	-0.14	2.035	0.68	0.07	2.123	0.65	-0.06	2.056	0.64	-0.24	2.087	0.63	-0.06
2.046	0.67	-0.2	2.057	0.68	0.07	2.146	0.65	0.06	2.079	0.64	-0.12	2.111	0.63	-0.16
2.069	0.66	-0.2	2.079	0.69	0.14	2.168	0.66	0.06	2.103	0.64	-1.39E-17	2.135	0.62	2.43E-13
2.091	0.66	-0.19	2.101	0.69	-3.39E-26	2.191	0.66	0.13	2.126	0.64	0.17	2.159	0.63	0.05
2.114	0.66	-0.19	2.123	0.69		2.214	0.66	0.26	2.15	0.64		2.183	0.63	0.06
2.137	0.65	-0.12				2.237	0.67	0.06				2.207	0.63	0.22
2.16	0.65					2.259	0.66	-0.13				2.231	0.64	0.06
						2.282	0.66	-0.19				2.254	0.63	-0.06
						2.305	0.66	-0.19				2.278	0.63	-0.06
						2.328	0.65	-0.18				2.302	0.63	-0.11
						2.351	0.65	-0.12				2.326	0.63	-0.06
						2.374	0.65	2.68E-13				2.35	0.63	-0.11
						2.397	0.65	-0.06				2.374	0.62	-0.05
						2.42	0.64	-0.06				2.398	0.63	
						2.444	0.64	0.06						
						2.467	0.65	0.06						
						2.49	0.65	-0.06						
						2.513	0.64	-0.12						
						2.537	0.64	-0.06						
						2.56	0.64	-0.12						
						2.584	0.64	-0.17						
						2.607	0.63	-0.06						
						2.631	0.63	-0.06						
						2.655	0.63							

Trial 33			Trial 34a			Trial 34b			Trial 35a			Trial 35b		
t	v	a	t	v	a	t	v	a	t	v	a	t	v	a
0.028	0.58	0.53	0.028	0.58	0.53	0.029	0.65	0.43	0.028	0.58	0.53	0.03	0.66	0.65
0.054	0.59	0.37	0.054	0.59	0.37	0.052	0.66	0.06	0.054	0.59	0.37	0.052	0.67	0.39
0.079	0.6	0.09	0.079	0.6	0.09	0.075	0.65	-0.06	0.079	0.6	0.09	0.075	0.68	0.27
0.104	0.6	0.3	0.104	0.6	0.3	0.098	0.65	0.06	0.104	0.6	0.3	0.097	0.68	0.28
0.129	0.61	0.29	0.129	0.61	0.29	0.121	0.66	0.25	0.129	0.61	0.29	0.119	0.69	-0.07
0.153	0.61	0.1	0.153	0.61	0.1	0.144	0.66	0.32	0.153	0.61	0.1	0.141	0.68	-0.14
0.177	0.62	0.21	0.177	0.62	0.21	0.166	0.67	0.27	0.177	0.62	0.21	0.163	0.68	0.21
0.202	0.62	-0.05	0.202	0.62	-0.05	0.189	0.68	0.07	0.202	0.62	-0.05	0.185	0.69	0.07
0.226	0.62	0.05	0.226	0.62	0.05	0.211	0.67	-2.75E-14	0.226	0.62	0.05	0.206	0.68	-0.28
0.25	0.62	0.21	0.25	0.62	0.21	0.233	0.68	2.75E-14	0.25	0.62	0.21	0.228	0.68	-0.21
0.274	0.63	0.34	0.274	0.63	0.34	0.255	0.67	-0.07	0.274	0.63	0.34	0.251	0.68	0.36
0.298	0.64	0.22	0.298	0.64	0.22	0.278	0.67	-9.96E-15	0.298	0.64	0.22	0.273	0.69	0.35
0.321	0.64	0.06	0.321	0.64	0.06	0.3	0.67	-2.99E-14	0.321	0.64	0.06	0.294	0.69	-2.81E-14
0.344	0.64	0.3	0.344	0.64	0.3	0.322	0.67	0.07	0.344	0.64	0.3	0.316	0.69	-0.14
0.368	0.65	0.12	0.368	0.65	0.12	0.345	0.68	0.14	0.368	0.65	0.12	0.338	0.68	0.15
0.391	0.65	-0.12	0.391	0.65	-0.12	0.367	0.68	0.14	0.391	0.65	-0.12	0.359	0.7	0.37
0.414	0.65	-0.18	0.414	0.65	-0.18	0.389	0.68	0.21	0.414	0.65	-0.18	0.381	0.7	-0.29
0.437	0.64	0.06	0.437	0.64	0.06	0.411	0.69	0.36	0.437	0.64	0.06	0.403	0.68	-0.23
0.46	0.65	0.12	0.46	0.65	0.12	0.432	0.7	-0.14	0.46	0.65	0.12	0.424	0.69	-0.07
0.483	0.65	-0.06	0.483	0.65	-0.06	0.454	0.68	-0.22	0.483	0.65	-0.06	0.446	0.68	-0.07
0.507	0.65	0.12	0.507	0.65	0.12	0.476	0.69	3.89E-16	0.507	0.65	0.12	0.468	0.69	0.07
0.53	0.65	0.25	0.53	0.65	0.25	0.498	0.68	-0.14	0.53	0.65	0.25	0.49	0.68	-0.14
0.553	0.66	0.19	0.553	0.66	0.19	0.52	0.68	-0.21	0.553	0.66	0.19	0.512	0.68	-0.14
0.575	0.66	0.19	0.575	0.66	0.19	0.542	0.67	-0.14	0.575	0.66	0.19	0.534	0.68	0.14
0.598	0.67	0.13	0.598	0.67	0.13	0.564	0.68	0.07	0.598	0.67	0.13	0.556	0.69	0.29
0.621	0.67	-0.07	0.621	0.67	-0.07	0.586	0.68	0	0.621	0.67	-0.07	0.578	0.69	0.07
0.643	0.66	-0.07	0.643	0.66	-0.07	0.609	0.68	0.07	0.643	0.66	-0.07	0.599	0.69	-0.07
0.666	0.66	0.07	0.666	0.66	0.07	0.631	0.68	-5.51E-14	0.666	0.66	0.07	0.621	0.69	-5.87E-14
0.688	0.67	0.13	0.688	0.67	0.13	0.653	0.68	0	0.688	0.67	0.13	0.643	0.69	-0.07
0.711	0.67	-0.07	0.711	0.67	-0.07	0.675	0.68	0.07	0.711	0.67	-0.07	0.665	0.68	-0.29
0.733	0.66	-0.2	0.733	0.66	-0.2	0.697	0.68	-0.2	0.733	0.66	-0.2	0.687	0.68	-0.21
0.756	0.66	0.07	0.756	0.66	0.07	0.719	0.67	-0.14	0.756	0.66	0.07	0.709	0.68	-0.14
0.778	0.67	0.33	0.778	0.67	0.33	0.742	0.67	0.07	0.778	0.67	0.33	0.731	0.67	-0.14
0.801	0.68	0.2	0.801	0.68	0.2	0.764	0.67	-0.07	0.801	0.68	0.2	0.753	0.67	-7.7E-14
0.823	0.68	-1.13E-13	0.823	0.68	-1.13E-13	0.786	0.67	-0.13	0.823	0.68	-1.13E-13	0.776	0.67	0.14
0.845	0.68	-0.13	0.845	0.68	-0.13	0.809	0.67	3.33E-16	0.845	0.68	-0.13	0.798	0.68	0.14
0.867	0.67	0.07	0.867	0.67	0.07	0.831	0.67	1.83E-13	0.867	0.67	0.07	0.82	0.68	-0.07
0.89	0.68	7.38E-14	0.89	0.68	7.38E-14	0.854	0.67	-0.13	0.89	0.68	7.38E-14	0.842	0.67	0
0.912	0.67	-0.27	0.912	0.67	-0.27	0.876	0.66	0.07	0.912	0.67	-0.27	0.865	0.68	0.21

0.934	0.67	-0.13	0.934	0.67	-0.13	0.899	0.67	-0.06	0.934	0.67	-0.13	0.887	0.68	-0.2
0.957	0.66	-0.19	0.957	0.66	-0.19	0.921	0.66	-0.07	0.957	0.66	-0.19	0.909	0.67	-0.14
0.98	0.66	-0.19	0.98	0.66	-0.19	0.944	0.67	0.06	0.98	0.66	-0.19	0.931	0.68	0.13
1.003	0.66	0.19	1.003	0.66	0.19	0.966	0.66	-0.19	1.003	0.66	0.19	0.953	0.68	-0.13
1.025	0.67	0.13	1.025	0.67	0.13	0.989	0.66	-3.61E-14	1.025	0.67	0.13	0.976	0.67	-0.14
1.048	0.66	-0.13	1.048	0.66	-0.13	1.012	0.66	0.06	1.048	0.66	-0.13	0.998	0.67	7.32E-28
1.071	0.66	0.13	1.071	0.66	0.13	1.035	0.66	1.43E-13	1.071	0.66	0.13	1.02	0.67	-0.13
1.093	0.67	1.39E-13	1.093	0.67	1.39E-13	1.057	0.66	0.06	1.093	0.67	1.39E-13	1.043	0.66	-0.07
1.116	0.66	0	1.116	0.66	0	1.08	0.66	0.07	1.116	0.66	0	1.065	0.67	0.13
1.138	0.67	-0.19	1.138	0.67	-0.19	1.102	0.67	-1.42E-13	1.138	0.67	-0.19	1.088	0.67	-0.13
1.161	0.65	-0.38	1.161	0.65	-0.38	1.125	0.66	0.07	1.161	0.65	-0.38	1.11	0.66	0.07
1.184	0.65	-0.12	1.184	0.65	-0.12	1.147	0.67	0.13	1.184	0.65	-0.12	1.133	0.67	0.26
1.207	0.65	0	1.207	0.65	0	1.17	0.67	-0.2	1.207	0.65	0	1.155	0.67	-0.13
1.23	0.65	-0.12	1.23	0.65	-0.12	1.192	0.66	0.07	1.23	0.65	-0.12	1.178	0.67	2.63E-13
1.254	0.64	0	1.254	0.64	0	1.215	0.67	0.06	1.254	0.64	0	1.2	0.67	0.07
1.277	0.65	0.18	1.277	0.65	0.18	1.237	0.66	-0.07	1.277	0.65	0.18	1.222	0.67	-0.2
1.3	0.65	3.84E-27	1.3	0.65	3.84E-27	1.26	0.67	0.07	1.3	0.65	3.84E-27	1.245	0.66	-0.07
1.323	0.65	0.12	1.323	0.65	0.12	1.282	0.67	0	1.323	0.65	0.12	1.267	0.67	-0.13
1.346	0.66	-0.06	1.346	0.66	-0.06	1.305	0.67	3.71E-14	1.346	0.66	-0.06	1.29	0.66	-0.19
1.369	0.65	0	1.369	0.65	0	1.327	0.67	0	1.369	0.65	0	1.313	0.66	-0.12
1.392	0.66	-0.06	1.392	0.66	-0.06	1.35	0.67	0.2	1.392	0.66	-0.06	1.336	0.65	-0.06
1.415	0.64	-0.24	1.415	0.64	-0.24	1.372	0.68	0	1.415	0.64	-0.24	1.359	0.66	-2.41E-15
1.439	0.64	0.12	1.439	0.64	0.12	1.394	0.67	0.21	1.439	0.64	0.12	1.382	0.65	0.06
1.462	0.65	0.06	1.462	0.65	0.06	1.416	0.68	0.2	1.462	0.65	0.06	1.405	0.66	0.25
1.485	0.65	-2.3E-15	1.485	0.65	-2.3E-15	1.438	0.68	-0.07	1.485	0.65	-2.3E-15	1.427	0.66	0.13
1.508	0.65	0.06	1.508	0.65	0.06	1.46	0.68	0.07	1.508	0.65	0.06	1.45	0.66	0.07
1.531	0.65	0.06	1.531	0.65	0.06	1.482	0.68	-0.07	1.531	0.65	0.06	1.472	0.67	0.13
1.554	0.65	0.12	1.554	0.65	0.12	1.504	0.68	-0.14	1.554	0.65	0.12	1.495	0.67	-0.13
1.577	0.66	0.19	1.577	0.66	0.19	1.527	0.68	-0.07	1.577	0.66	0.19	1.517	0.66	-0.13
1.6	0.66	0.13	1.6	0.66	0.13	1.549	0.68	9.04E-27	1.6	0.66	0.13	1.54	0.66	-0.06
1.623	0.66	-1.05E-26	1.623	0.66	-1.05E-26	1.571	0.68	8.94E-27	1.623	0.66	-1.05E-26	1.563	0.66	-0.19
1.645	0.66	-0.06	1.645	0.66	-0.06	1.593	0.68	0.14	1.645	0.66	-0.06	1.586	0.66	-0.06
1.668	0.66	-2.13E-13	1.668	0.66	-2.13E-13	1.615	0.68	0.14	1.668	0.66	-2.13E-13	1.609	0.66	-2.13E-13
1.691	0.66	-0.12	1.691	0.66	-0.12	1.637	0.68	3.86E-13	1.691	0.66	-0.12	1.631	0.66	3.51E-13
1.714	0.65	0	1.714	0.65	0	1.659	0.68	0.14	1.714	0.65	0	1.654	0.66	0.06
1.737	0.66	0.12	1.737	0.66	0.12	1.681	0.69	-1.57E-13	1.737	0.66	0.12	1.677	0.66	0.13
1.759	0.66	-0.06	1.759	0.66	-0.06	1.703	0.68	-0.14	1.759	0.66	-0.06	1.7	0.66	0.19
1.782	0.66	0	1.782	0.66	0	1.725	0.68	-9.29E-27	1.782	0.66	0	1.723	0.67	0.06
1.805	0.66	1.41E-13	1.805	0.66	1.41E-13	1.747	0.68	-9.29E-27	1.805	0.66	1.41E-13	1.745	0.66	-0.07
1.828	0.66	-0.06	1.828	0.66	-0.06	1.769	0.68	-0.07	1.828	0.66	-0.06	1.768	0.66	0.07
1.851	0.66	-0.12	1.851	0.66	-0.12	1.791	0.68	2.29E-13	1.851	0.66	-0.12	1.79	0.67	-0.13

1.874	0.65	1.4E-13	1.874	0.65	1.4E-13	1.813	0.68	0.14	1.874	0.65	1.4E-13	1.813	0.66	-0.19
1.897	0.66	-1.37E-13	1.897	0.66	-1.37E-13	1.835	0.68	0	1.897	0.66	-1.37E-13	1.836	0.66	8.33E-27
1.919	0.65	-0.06	1.919	0.65	-0.06	1.857	0.68	0.15	1.919	0.65	-0.06	1.858	0.66	-0.06
1.942	0.65	0.13	1.942	0.65	0.13	1.879	0.69	0.14	1.942	0.65	0.13	1.881	0.66	-0.12
1.965	0.66	-1.37E-13	1.965	0.66	-1.37E-13	1.901	0.69	-0.07	1.965	0.66	-1.37E-13	1.904	0.65	2.41E-15
1.988	0.65	-0.06	1.988	0.65	-0.06	1.922	0.69	-0.21	1.988	0.65	-0.06	1.927	0.66	0.12
2.011	0.66	2.13E-13	2.011	0.66	2.13E-13	1.944	0.68	-0.21	2.011	0.66	2.13E-13	1.95	0.66	3.52E-13
2.034	0.65	1.18E-15	2.034	0.65	1.18E-15	1.967	0.68	-0.14	2.034	0.65	1.18E-15	1.973	0.66	
2.057	0.66	0.12	2.057	0.66	0.12	1.989	0.67	-0.14	2.057	0.66	0.12			
2.08	0.66	0	2.08	0.66	0	2.011	0.67	2.88E-27	2.08	0.66	0			
2.103	0.66	-0.06	2.103	0.66	-0.06	2.033	0.67		2.103	0.66	-0.06			
2.126	0.66	5.05E-29	2.126	0.66	5.05E-29				2.126	0.66	5.05E-29			
2.149	0.66	0.06	2.149	0.66	0.06				2.149	0.66	0.06			
2.172	0.66	0	2.172	0.66	0				2.172	0.66	0			
2.194	0.66	0.13	2.194	0.66	0.13				2.194	0.66	0.13			
2.217	0.66	0.06	2.217	0.66	0.06				2.217	0.66	0.06			
2.24	0.66	0.07	2.24	0.66	0.07				2.24	0.66	0.07			
2.262	0.67	0.13	2.262	0.67	0.13				2.262	0.67	0.13			
2.285	0.66	-0.07	2.285	0.66	-0.07				2.285	0.66	-0.07			
2.308	0.66	2.87E-13	2.308	0.66	2.87E-13				2.308	0.66	2.87E-13			
2.33	0.66	-0.06	2.33	0.66	-0.06				2.33	0.66	-0.06			
2.353	0.66	-0.06	2.353	0.66	-0.06				2.353	0.66	-0.06			
2.376	0.66	-0.13	2.376	0.66	-0.13				2.376	0.66	-0.13			
2.398	0.66	-2.14E-13	2.398	0.66	-2.14E-13				2.398	0.66	-2.14E-13			
2.421	0.66	4.92E-13	2.421	0.66	4.92E-13				2.421	0.66	4.92E-13			
2.444	0.66	-0.13	2.444	0.66	-0.13				2.444	0.66	-0.13			
2.467	0.66	-5.05E-29	2.467	0.66	-5.05E-29				2.467	0.66	-5.05E-29			
2.49	0.66	5.05E-29	2.49	0.66	5.05E-29				2.49	0.66	5.05E-29			
2.513	0.66	-0.06	2.513	0.66	-0.06				2.513	0.66	-0.06			
2.536	0.65	-0.24	2.536	0.65	-0.24				2.536	0.65	-0.24			
2.559	0.64		2.559	0.64					2.559	0.64				

Trial 36			Trial 37			Trial 38		
t	v	a	t	v	a	t	v	a
0.023	0.66	-0.25	0.032	0.68	0.35	0.026	0.65	-0.12
0.046	0.65	-0.24	0.054	0.68	-0.07	0.049	0.65	0.12
0.07	0.64	0.19	0.076	0.68	0.22	0.072	0.65	0.25

0.093	0.66	0.24	0.098	0.69	0.58	0.095	0.66	0.32
0.115	0.66	-0.12	0.119	0.7	0.07	0.118	0.67	0.13
0.138	0.65	-0.12	0.141	0.69	-0.23	0.14	0.66	-0.07
0.161	0.65	-5.12E-15	0.162	0.69	-0.07	0.163	0.66	-0.13
0.185	0.65	0.25	0.184	0.69	-1.53E-14	0.185	0.66	0.2
0.207	0.66	0.19	0.206	0.69	0.23	0.208	0.67	0.26
0.23	0.66	0.06	0.227	0.7	0.15	0.23	0.67	-0.26
0.253	0.66	0.26	0.249	0.7	-0.15	0.253	0.66	-0.13
0.275	0.67	0.2	0.27	0.69	0.08	0.275	0.66	0.13
0.297	0.67	-0.07	0.292	0.7	0.23	0.298	0.67	0.34
0.32	0.67	-0.07	0.313	0.7	0.16	0.32	0.68	0.34
0.342	0.67	0.14	0.335	0.71	-0.15	0.342	0.68	-9.88E-15
0.364	0.68	0.13	0.356	0.7	-0.16	0.364	0.68	-0.07
0.387	0.68	-5.5E-14	0.377	0.7	0.15	0.387	0.68	0.21
0.409	0.68	-0.07	0.399	0.7	-0.15	0.408	0.69	0.21
0.431	0.67	5.54E-14	0.42	0.69	6.01E-14	0.43	0.69	-0.14
0.453	0.68	0.21	0.442	0.7	-0.07	0.452	0.68	-0.28
0.476	0.68	0.21	0.463	0.69	-0.3	0.474	0.68	4.13E-14
0.497	0.68	-0.07	0.485	0.69	0.07	0.496	0.68	0.07
0.519	0.68	0.07	0.506	0.69	0.07	0.518	0.68	-4.03E-14
0.541	0.69	0	0.528	0.69	-0.15	0.54	0.68	0.07
0.563	0.68	-0.14	0.55	0.69	0.08	0.562	0.68	-5.55E-14
0.585	0.68	0.14	0.571	0.7	0.15	0.584	0.68	0.07
0.607	0.68	0.07	0.593	0.69	-0.08	0.606	0.68	0.07
0.629	0.68	-0.14	0.615	0.69	-0.07	0.628	0.68	-0.07
0.651	0.68	0	0.636	0.69	-0.15	0.65	0.68	0.15
0.673	0.68	0.14	0.658	0.69	5.83E-14	0.672	0.69	0.14
0.695	0.68	-0.07	0.68	0.69	-0.07	0.694	0.69	-0.07
0.717	0.68	-0.21	0.701	0.68	-0.07	0.716	0.69	-0.14
0.739	0.68	0	0.723	0.69	0.29	0.738	0.68	7.86E-14
0.761	0.68	0.07	0.745	0.7	0.15	0.759	0.69	0.07
0.783	0.68	0.07	0.766	0.69	-0.08	0.781	0.68	-0.21
0.805	0.68	-0.14	0.788	0.69	0.08	0.803	0.68	0.07
0.827	0.67	-0.07	0.81	0.7	0.15	0.825	0.69	0.29
0.85	0.68	0.35	0.831	0.7	-0.07	0.847	0.69	-3.89E-16
0.871	0.69	-0.14	0.853	0.69	-0.08	0.869	0.69	-0.07
0.893	0.68	-0.35	0.874	0.7	-0.07	0.891	0.69	0.22
0.916	0.67	0.14	0.896	0.69	0.08	0.912	0.7	0.15
0.938	0.68	0.07	0.917	0.7	0.07	0.934	0.69	-0.08
0.96	0.68	-0.14	0.939	0.69	-0.23	0.955	0.69	-8.48E-14
0.982	0.68	0.21	0.96	0.69	0.08	0.977	0.69	0.08

1.004	0.68	0.07	0.982	0.7	0.07	0.999	0.7	1.67E-13
1.026	0.68	-0.14	1.004	0.69	-0.29	1.02	0.69	-0.08
1.048	0.68	-1.57E-27	1.025	0.68	-0.22	1.042	0.69	-1.67E-13
1.07	0.68	-0.33	1.047	0.68	0.15	1.063	0.69	1.67E-13
1.093	0.66	-0.27	1.069	0.69	1.6E-13	1.085	0.69	0.15
1.115	0.67	0.27	1.091	0.68	-0.15	1.106	0.7	0.07
1.138	0.68	0.07	1.113	0.68	0.15	1.128	0.7	-0.15
1.16	0.67	-0.2	1.135	0.69	0	1.149	0.69	0
1.182	0.67	0.21	1.156	0.68	1.17E-13	1.171	0.7	0.07
1.205	0.68	0.34	1.178	0.69	0.14	1.192	0.7	1.68E-13
1.227	0.68	7.22E-16	1.2	0.69	-0.07	1.214	0.7	-5.15E-27
1.249	0.68	0.14	1.222	0.69	0.07	1.235	0.7	-0.07
1.271	0.69	6.44E-15	1.243	0.69	-1.61E-13	1.257	0.69	0.08
1.293	0.68	-0.07	1.265	0.69	-0.07	1.278	0.7	0.07
1.315	0.68	-0.07	1.287	0.69	0.07	1.3	0.7	0.08
1.337	0.68	-0.28	1.308	0.69	0.07	1.321	0.7	0.15
1.359	0.67	-0.14	1.33	0.69	-1.61E-13	1.343	0.7	-0.08
1.381	0.67	-0.13	1.352	0.69	0.08	1.364	0.7	-0.08
1.404	0.67	-0.2	1.373	0.7	1.64E-13	1.385	0.7	-0.08
1.426	0.66	-0.07	1.395	0.69	0.08	1.407	0.7	-0.15
1.449	0.66	0.26	1.416	0.7	0.15	1.428	0.69	-0.15
1.472	0.67	0.07	1.438	0.7	-0.08	1.45	0.69	-0.15
1.494	0.67	0.14	1.459	0.7	-0.07	1.472	0.69	-0.14
1.516	0.68	1.47E-13	1.481	0.69	-0.15	1.494	0.68	-1.6E-13
1.539	0.67	-0.21	1.503	0.69	-0.07	1.515	0.69	0.14
1.561	0.67	-0.13	1.524	0.69	7.32E-28	1.537	0.69	0
1.584	0.66	-0.2	1.546	0.69	0.07	1.559	0.69	-0.07
1.606	0.66	-0.19	1.568	0.69	-0.07	1.581	0.69	0.07
1.629	0.65	-0.06	1.589	0.69	0.08	1.603	0.69	-0.14
1.652	0.66	0.06	1.611	0.7	0.15	1.624	0.68	-0.29
1.675	0.66	-0.06	1.633	0.69	-1.67E-13	1.646	0.68	-0.14
1.698	0.66	0.13	1.654	0.7	0.07	1.669	0.68	-0.27
1.721	0.66	1.41E-13	1.676	0.7	-0.07	1.691	0.67	-0.07
1.743	0.66	-0.13	1.697	0.69	-0.08	1.713	0.67	0.07
1.766	0.66	-0.06	1.719	0.69	-0.07	1.736	0.67	-0.07
1.789	0.65	-0.12	1.74	0.69	-0.15	1.758	0.67	0.07
1.812	0.65	-0.18	1.762	0.69	-0.14	1.78	0.67	0.14
1.835	0.64	-0.18	1.784	0.68	-1.6E-13	1.803	0.68	-0.07
1.859	0.64	-0.17	1.806	0.69	2.16E-15	1.825	0.67	-0.07
1.882	0.64	-3.28E-15	1.828	0.68	0.15	1.847	0.67	0
1.906	0.64	0.23	1.849	0.69	0.07	1.87	0.67	-0.2

1.929	0.65	0.18	1.871	0.69	-0.07	1.892	0.66	-0.2
1.952	0.65	0.12	1.893	0.69	2.35E-13	1.915	0.66	-0.06
1.975	0.65	0.12	1.915	0.69	-0.14	1.938	0.66	2.17E-27
1.998	0.66	-1.37E-13	1.937	0.68	-0.07	1.96	0.66	0.06
2.021	0.65	1.39E-13	1.958	0.68	-0.21	1.983	0.66	0.13
2.044	0.66	-0.12	1.981	0.68	-0.21	2.005	0.67	0.13
2.067	0.65	-0.24	2.003	0.68		2.028	0.67	0.13
2.09	0.64	-0.12				2.05	0.67	0.07
2.114	0.64	-0.12				2.073	0.67	-0.13
2.137	0.64	-0.06				2.095	0.67	-0.2
2.161	0.64	-5.67E-14				2.117	0.66	
2.184	0.64	-1.26E-29						
2.208	0.64							