Innovation Week – Lorenzo Mazzeo

Group: Makerspace
Project: Graphing Software With Algebraically Scaling Axes

How it works:

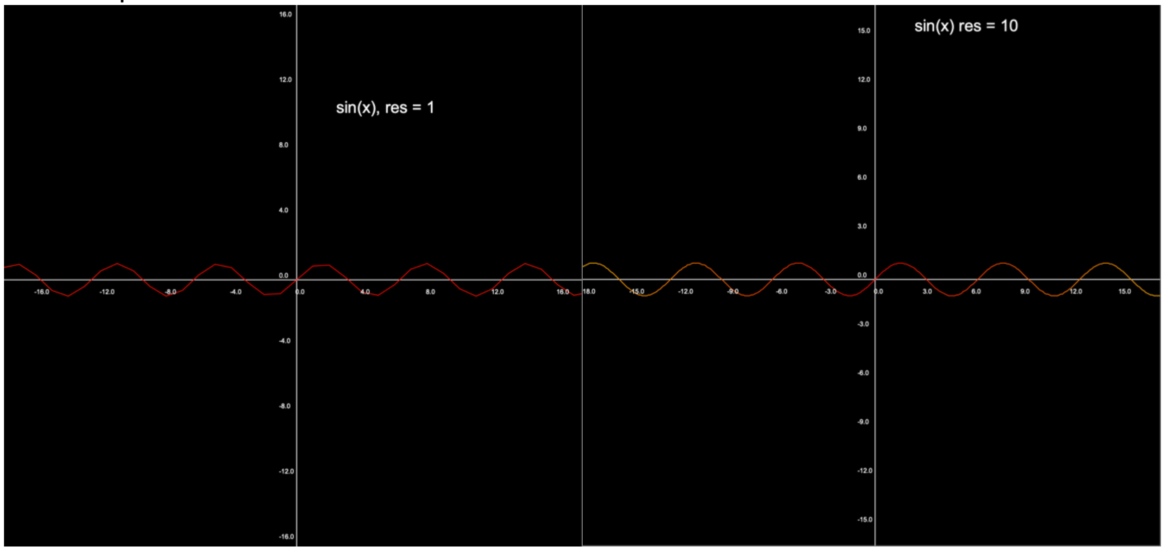
Basics:
The program gets two points on a coordinate plane, $(x, f(x)), (x+r, f(x+r))$ with r (called resolution in the source code) being the x distance between said points. The program then draws a line between a and b and after which it sets a to b $(x, f(x)), (x+r, f(x+r)) \rightarrow (x+r, f(x)), (x+2r, f(x+2r))$ and draws again. This process is repeats until the program reaches the edge of the coordinate plane (which has a pre-defined size). The entire process is then done again, but for negative numbers.

Algebraically Scaling Axes
Essentially this means that the axis can by distorted a function. Let's call the x distortion function $\varkappa(x)$ and the y distortion function $\mathfrak{H}(y)$. If a distortion function is given, it is applied to the points before a line is drawn:

$$(\varkappa(x), \mathfrak{H}(f(x))), (\varkappa(x+r), \mathfrak{H}(f(x+r)))$$

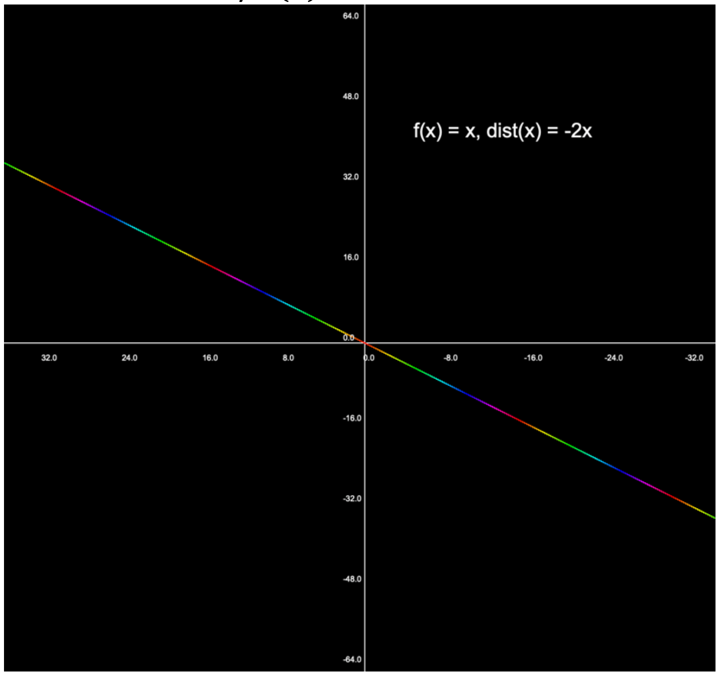
Because this program is written in python, any python function can also be used to distort, such as one for generating random numbers.

Visual explanation of resolution:



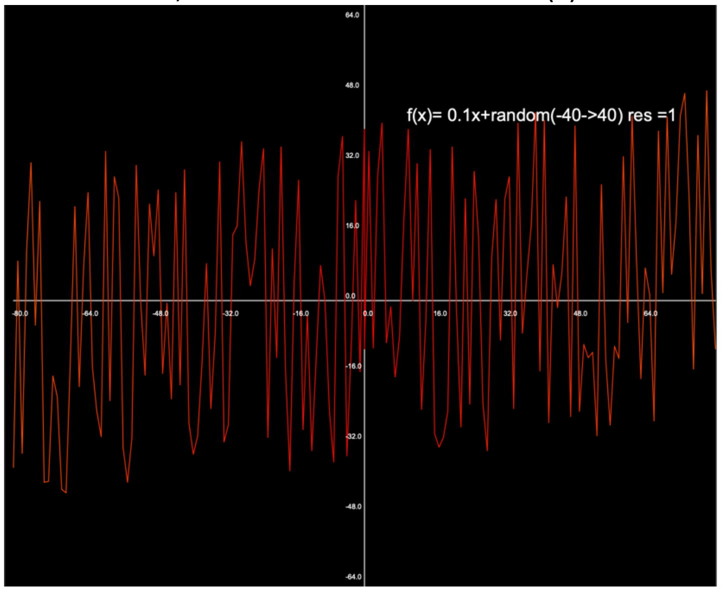
As you can see, with a resolution of only 1, the sine wave seems much more jagged than the sine wave with a resolution of 10. You might also notice that the color changes the further the sine wave moves from the origin. This color change is necessary if you want to be able to make sense of a graph when lines start crossing over themselves and looping back around.

X axis distortion by $\varkappa(x) = -2x$



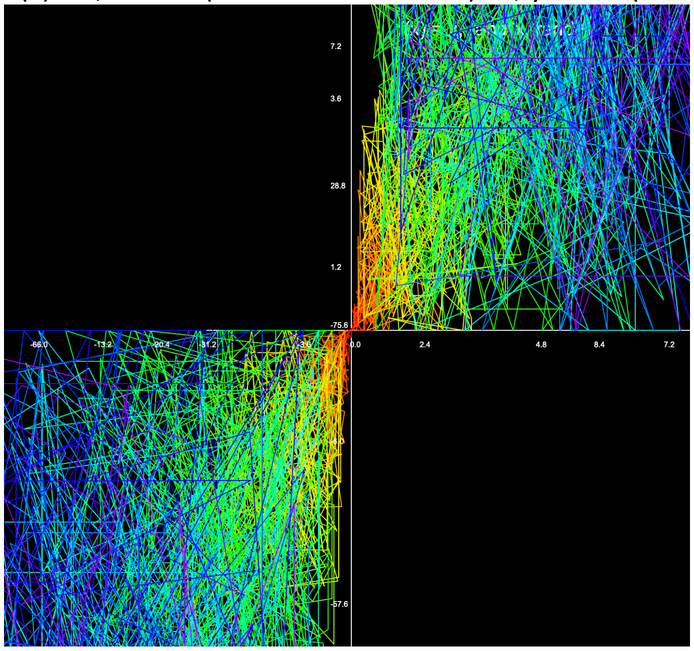
As you can see, a distortion of -2x appears to mirror and stretch the graph. However, if you inspect the axes, you will quickly see that this graph is still f(x) = x, as the x axis numbering has been mirrored and stretched along with the graph.

No distortion, random number added to f(x)



This example demonstrates use of python's random function by adding it to f(x)

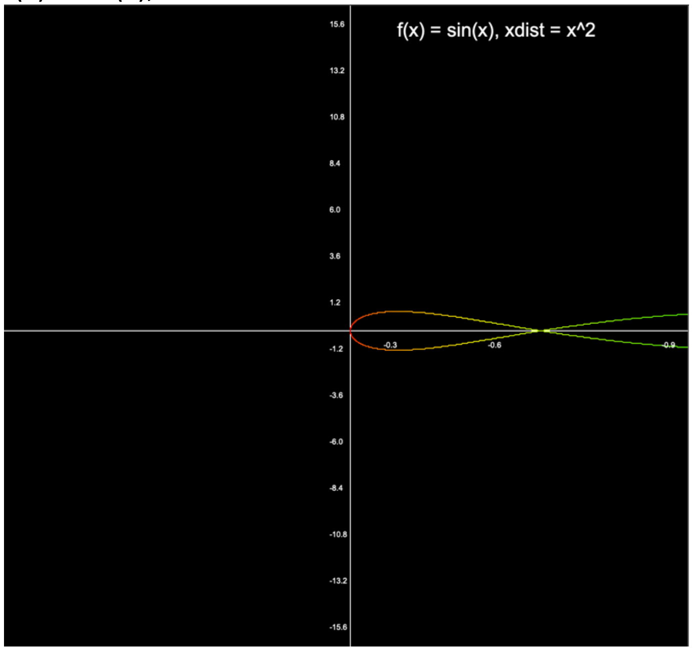
F(x) = x, xdist = (random from 1->10) *x, y dist = (random from 0->20)+y



This example also demonstrates use of python's random function by distorting the axis with random numbers

You'll notice that the x axis goes from 4.8 to 8.4 to back down to 7.2, and y goes from 1.2 to 28.8, then back down to 3.6, creating this mess.

f(x) = sin(x), xdist = x^2



This is a graph of sin(x) with a x distortion of x^2. As you can see, no x axis numbering is present left of the y axis, as that is outside of the domain.

The numbers that do occur on the x axis are negative interestingly enough, which is because the program is designed to prevent two axis numbers from occupying the same space, and numbers are generated left to right. (this is still valid, as any negative number^2 is positive)

Source Code (212 Lines)

```
from math import *
import turtle as t
from random import *

def main
t hideturtle
t bgcolor "black"
t pencolor "white"
t penup
eq = t textinput "equation" "f(x) = "
t colormode 255
xMod = t textinput "X Mod" "X Scaling: "
yMod = t textinput "Y Mod" "Y Scaling: "

x = 0
y = 0
fCoords =  x y
rgb =  255 0 0
colorStep = 0
start =  x y

title = t textinput "Title" "Graph Title: "
canvW = int t numinput "canvW" "Window Width: "
canvH = int t numinput "canvH" "Window Height: "
zoom = float t numinput "Zoom" "Zoom Value: "
res = 1/ float t numinput "GRes" "Graph Resolution: "
axisRes = int t numinput "ARes"  "Axis Resolution "

t screensize canvW  canvH

t tracer 0  0
t setposition 50   canvH/2
t write title  align="left"  font= 'Arial'  20  'normal'
t home
t pendown
t setposition canvW 0
t setposition -1*canvW 0
t home
t setposition 0 canvH
t setposition 0 -1*canvH
t pencolor rgb 0  rgb 1  rgb 2
```

```
t penup

t setx x
t sety eval eq

## t.dot(4)

def forward x  eq
    try
        y = eval eq
        return y
    except
        pass

def back x eq
    try
        y = eval eq
        return y
    except
        pass


while  abs fCoords 0   < canvW  and  abs x  < 1000/res
    if colorStep == 0
        rgb 1  += 1
        if rgb 1  == 255
        colorStep = 1
    elif colorStep == 1
        rgb 0  -= 1
        if rgb 0  == 0
        colorStep = 2
    elif colorStep == 2
        rgb 2  += 1
        if rgb 2  == 255
        colorStep = 3
    elif colorStep == 3
        rgb 1  -= 1
        if rgb 1  == 0
        colorStep = 4
    elif colorStep == 4
        rgb 0  += 1
        if rgb 0  == 255
        colorStep = 5
    elif colorStep == 5
        rgb 2  -= 1
        if rgb 2  == 0
        colorStep = 0
    t pencolor rgb 0  rgb 1  rgb 2
```

```
        t pendown
        try
                y = eval eq

        except
                pass

        try
                fCoords 0  = eval xMod *zoom
                fCoords 1  = eval yMod *zoom
        except
                pass


        t speed 0
        t setposition fCoords 0  fCoords 1


        x += res
        ##print(x)

print "P done"
t penup
t setx start 0
t sety start 1
x = 0
colorStep = 0
rgb =  255  0  0
fCoords =  0 0
while  abs fCoords 0   < canvW  and  abs x  < 1000/res

        if colorStep == 0
                rgb 1  += 1
                if rgb 1  == 255
                colorStep = 1
        elif colorStep == 1
                rgb 0  -= 1
                if rgb 0  == 0
                colorStep = 2
        elif colorStep == 2
                rgb 2  += 1
                if rgb 2  == 255
                colorStep = 3
        elif colorStep == 3
                rgb 1  -= 1
                if rgb 1  == 0
                colorStep = 4
        elif colorStep == 4
                rgb 0  += 1
                if rgb 0  == 255
                colorStep = 5
        elif colorStep == 5
                rgb 2  -= 1
                if rgb 2  == 0
                colorStep = 0
        t pencolor rgb 0  rgb 1  rgb 2


        try
                y = eval eq
        except
                pass

        try
                fCoords 0  = eval xMod *zoom
                fCoords 1  = eval yMod *zoom
        except
                pass

        t pendown
        t speed 0
        t setposition fCoords 0  fCoords 1


        x -= res
        ##print(x)

print "N done"
t penup
t color "white"
offset = -20
t setposition -1*canvW  offset
lastPos =  t xcor   t ycor
move = canvW/axisRes
overlap =
for i in range 0  2*axisRes
        nover =True
        t penup
        t setposition  i*move -canvW  offset
        x = t xcor
        t setposition eval xMod    offset
        for j in range  len overlap
                if  abs overlap j  - t xcor    < 48
                nover = False
```

```
    if  nover == True
        t write round x  1 /zoom
        overlap append t xcor

t setposition offset  -1*canvH
lastPos =  t xcor   t ycor
move = canvH/axisRes
overlap =
for i in range 0  2*axisRes
    nover =True
    t penup
    t setposition offset   i*move -canvH
    y = t ycor
    t setposition offset  eval yMod
    for j in range  len overlap
        if  abs overlap j  - t ycor    < 48
        nover = False
    if  nover == True
        t write round y  1 /zoom
        overlap append t ycor

t update
t done


if __name__ == "__main__"
main
```