Arnav Dani - Mathematica Project 2

Part 1 - Rectangles

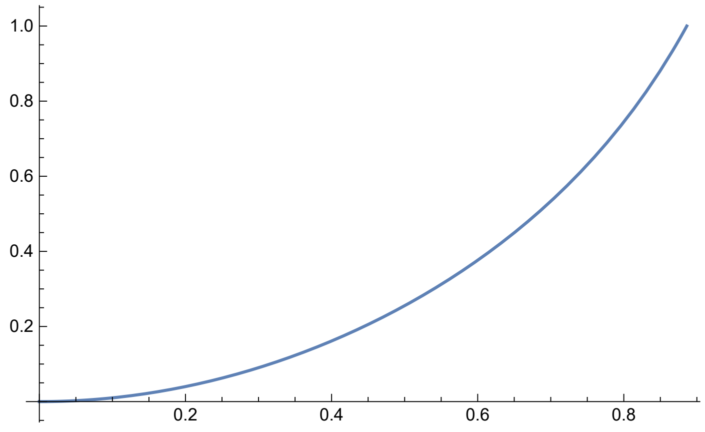
In[◦]:=

```
f[x_] = Tan[x^2]
Plot[f[x], {x, 0, π^(1/2)/2}]
(*i made a truncate method which truncates every decimal past the thousandths place -
 I made it for ease of use when I have to truncate inside
  a loop to make the conde more understandable ad readable

  In addition, I defined the basic function and graphed
 it to provide a visual image of the area I am trying to find.*)
Trunc[i_] = Floor[i, 0.001]
```

Out[◦]= $\text{Tan}[x^2]$

Out[◦]=



Out[◦]= 75 000.

Riemann’s sums define an area as a set of rectangles with equal widths and specified heights. An area would be a certain width * the value of the function at an endpoint of that width to represent the height. The total area would be the sum of all these individual rectangles. As expected, this method is not very accurate when only a few rectangles are used because a small amount of rectangles cannot accurately represent curves like the one I am trying to find the area of. However, when the number of rectangles becomes exceedingly high, the method is supposed to be very accurate.

1 Part 1 - Riemann’s Sums

LHRS

A Riemann' s sum needs every rectangle to have a width and a height. Since I am making a general function that will find the area if N rectangles are passed in, the width of each rectangle is the left bound of the integral - the right bound divided by the number of rectangles, or in this case n.

In this problem, I am trying to find the area between 0 and $\frac{\pi^{1/2}}{2}$so the width of each rectangle is $\frac{(\pi^{1/2}/2-0)}{n}$which equals $\frac{\pi^{1/2}}{2n}$. Since I am doing a left hand sum, the height of each rectangle is the leftmost point of the rectangle. Therefore I start k at 0 so the first rectangle has a height of f(0). In this case, the height is 0 + f(k*width) or $f\left[k * \frac{\pi^{1/2}}{2\,n}\right]$. I limit k to n-1 so that the last rectangle goes from n-1 to n and I maintain the left hand side area rule I set.

In[∘]:=
```
TanLeftA[n_] = Sum[π^(1/2)/(2 n) f[k * π^(1/2)/(2 n)], {k, 0, n - 1}]
```

Out[∘]=

$$\sum_{k=0}^{-1+n} \frac{\sqrt{\pi}\ \text{Tan}\left[\frac{k^2 \pi}{4 n^2}\right]}{2\,n}$$

Rounded

I am seeing whether rounding or truncating the area has an impact on the total sum of all the areas. The key difference between the two is that rounding rounds the number to 3 places while truncating just cuts off every other digit. For example, 0.1236 rounded to 3 decimal places is 0.124 while truncating it to 3 decimal places returns 0.123. I accomplish rounding using the NumberForm command.
I am trying to find the number of rectangles until the output is accurate to 3 decimal places. When using rounding, this means that I am looking for 2 consecutive outputs that are the same since I round before printing the table.
The way I accomplished this was by using a ‘do’ loop. The loop finds the area when using 1 rectangle up to a specified amount, in this case, 50, rectangles for me to access later. To find the first place where the outputs are the same or precise to 3 decimal places, I went through all the outputs and I have printed a table of where the output is the same for 2 consecutive numbers of rectangles.

In[∘]:=
```
Do[{LeftTanRoundArea_n = NumberForm[N[TanLeftA[n]], 3]}, {n, 1, 50}]
(*testing various numbers of rectangles to find where 2 numbers are the same
 For[i = 1, i ≤ 50, i++,Print[i," ", LeftTanRoundArea_i]]*)
TableForm[Table[ {i, LeftTanRoundArea_i}, {i, 22, 25}],
 TableHeadings → {None, {index, "LHRS Area Rounded"}}]
```

Out[∘]//TableForm=
```
index    LHRS Area Rounded
22       0.237
23       0.238
24       0.238
25       0.239
```

As shown in the table, the 23rd and 24th output are the same. This means that the rounded area when 23 and 24 rectangles are used is the same which meets the example criteria of being precise to 3 decimal places.

Number of rectangles necessary for answer to be good to 3 places using LHRS and rounding: 23 rectangles with an area of 0.238 units^2

Truncated

As explained before, truncating is slightly different from rounding and therefore I expect a slightly different result in the number of rectangles where precision is achieved. I use the same function for area except I truncate the area instead of rounding it using the method I defined.

In[∘]:=

```
Do[{LeftTanTruncArea_n = Trunc[N[TanLeftA[n]]]}, {n, 1, 50}]
(*testing various numbers of rectangles to find where 2 numbers are the same
 For[i = 1, i ≤ 50, i++,Print[i," ", LeftTanTruncArea_i]]*)
TableForm[Table[{i, LeftTanTruncArea_i} , {i, 24, 27}],
 TableHeadings → {None, {index, "LHRS Area Truncated"}}]
```

Out[∘]//TableForm=

```
index     LHRS Area Truncated
24        0.238
25        0.239
26        0.239
27        0.24
```

Number of rectangles necessary for answer to be good to 3 places using LHRS and truncating: 25 rectangles at an area of 0.239 units^2, which is slightly different than the 0.238 units^2 when rounding, but not a significant enough difference to say that truncating has a major impact.

RHRS

As explained earlier, a right hand sum is when the right endpoint of every rectangle is used to calculate the height of each individual rectangle. I accomplish this by making k go from 1 to n instead of 0 to n-1. Otherwise, all of the other properties are the same like the width and the basic function and its structure.

In[∘]:= TanRightA[n_] = Sum$\left[\frac{\pi^{1/2}}{2\,n}\,f\left[k * \frac{\pi^{1/2}}{2\,n}\right], \{k, 1, n\}\right]$

Rounded

In[∘]:=

```
Do[{RightTanRoundArea_n = NumberForm[N[TanRightA[n]], 3]}, {n, 1, 50}]
(*testing various numbers of rectangles to find where 2 numbers are the same
 For[i = 1, i ≤ 50, i++,Print[i," ", RightTanRoundAreaArea_i]]*)
TableForm[Table[ {i, RightTanRoundArea_i}, {i, 24, 27}],
 TableHeadings → {None, {index, "RHRS Area Rounded"}}]
```

Number of rectangles necessary for answer to be good to 3 places using RHRS and rounding: 25 rectangles with an area of 0.274 units^2. This Area is drastically different from the LHRS when rounded, showing that Riemann’s sums accurate to the 3rd decimal place do not provide an accurate area for this function due to the major contrast.

Truncated

similar to before, I am seeing whether truncating the right sum has an impact compared to rounding as I did with the left sum. I use the same method as before, utilizing the Trunc function I defined.

```
In[∘]:= Do[{RightTanTruncArea_n = Trunc[N[TanRightA[n]]]}, {n, 1, 50}]
        (*testing various numbers of rectangles to find where 2 numbers are the same
         For[i = 1, i ≤ 50, i++,Print[i," ", RightTanTruncArea_i]]*)
        TableForm[Table[ {i, RightTanTruncArea_i}, {i, 25, 28}],
         TableHeadings → {None, {index, "RHRS Area Truncated"}}]
```

Number of rectangles necessary for answer to be good to 3 places using RHRS and truncating: 26 rectangles with an area of 0.273 units^2. This is similar to the 25 rectangles needed when rounding with almost the same area output, showing that truncating the decimal has a minimal impact on the final area regardless of the side used for summing.

1 Part 3 – Trapezoid

As seen above, the side one chooses to sum from has a drastic impact on the results. Therefore, the trapezoid is also a viable strategy. Instead of using the leftmost or rightmost point, a trapezoid works by taking the height as the middle point of the rectangle by averaging out the height of the left side and the right side as the bases of the trapezoid and using the previous width of the rectangle as the common height. In this case, $\frac{\pi^{1/2}}{2n}$is the height while $f\left[k * \frac{\pi^{1/2}}{2n}\right]$ is one base and $f\left[(k+1) * \frac{\pi^{1/2}}{2n}\right]$ is the other base of the trapezoid with the diagonal being at the top of the function. The loop must go from 0 to n - 1 because I call the function of k+1 which would not exist if k went up to n.

In[∘]:= $\text{TanTrap}[n\_] = \text{Sum}\left[\frac{\pi^{1/2}}{2n} * \frac{f\left[k * \frac{\pi^{1/2}}{2n}\right] + f\left[(k+1) * \frac{\pi^{1/2}}{2n}\right]}{2}, \{k, 0, n-1\}\right]$

Out[∘]= $$\sum_{k=0}^{-1+n} \frac{\sqrt{\pi}\left(\text{Tan}\left[\frac{k^2 \pi}{4 n^2}\right] + \text{Tan}\left[\frac{(1+k)^2 \pi}{4 n^2}\right]\right)}{4 n}$$

Rounded

Since I have proven that rounding or truncating does not have an impact, what I choose will not have significant impact on my final answer. However, I want to be clear about the method I use and be consistent with it. From here on out in the project, I will be rounding every value to 3 places because of the higher accuracy which is desired when finding an area even though the benefit might be marginal.

```
Do[{TrapRoundedArea_n = NumberForm[N[TanTrap[n]], 3]}, {n, 1, 50}]
(*testing various numbers of trapezoids to find where 2 numbers are the same
 For[i = 1, i ≤ 50, i++,Print[i," ", TrapRoundedArea_i]]*)
TableForm[Table[ {i, TrapRoundedArea_i}, {i, 8, 11}],
 TableHeadings → {None, {index, "Trapezoid Sum"}}]
```

Out[∘]//TableForm=

index	Trapezoid Sum
8	0.26
9	0.259
10	0.259
11	0.258

Number of trapezoids necessary for answer to be good to 3 places using rounding : 9 rectangles - the answer was 0.259 units^2 which is near the middle ground between my answer for LHRS and RHRS, meaning that this is probably the closest to the actual area. In addition, this method took significantly less trapezoids than the rectangle methods took rectangles, meaning that this could possibly be a more efficient solution. This could be true because the trapezoid follows the shape of the curve better because the curve is increasing and concave up as shown in the initial graph.

1 Part 2 - Experimental Functions

sin (x) from 0 to π/2

From 0 to π/2, sin x is increasing but is concave down for the entirety of the domain.
From 0 to π/2, the area of sin(x) is -cos(π/2)-(-cos(0)) which is 0 + 1 = 1. I will test the accuracy of Rieman-n's sums using the output compared to the correct answer.
I use the same fundamental definition of the integral and apply it to sin(x).
Since the range is from 0 to π/2, $\frac{b-a}{n} = \frac{\pi/2}{n}$ and $x_k = k * \frac{\pi/2}{n}$
Since I am testing how different functions impact the number of rectangles, the type of sum I use should be constant to isolate the functions as the independent variable. To clarify, I am using RHR Sums and rounding for all the experimental functions

In[∘]:=
```
s[x_] = Sin[x]
SinArea[n_] = Sum[π/(2 n) s[k * π/(2 n)], {k, 1, n}]
```

Out[∘]= Sin[x]

Out[∘]= $$\frac{\pi + \pi \,\mathrm{Csc}\left[\frac{\pi}{4 n}\right] \mathrm{Sin}\left[\frac{(-1+2 n)\pi}{4 n}\right]}{4 n}$$

```
In[∘]:= Do[{SinAreaRectₙ = NumberForm[N[SinArea[n]], 4]}, {n, 1, 50}]
        (*testing various numbers of rectangles to find where 2 numbers are the same
         For[i = 1, i ≤ 50, i++,Print[i," ", SinAreaRectᵢ]]*)
        TableForm[Table[ {i, SinAreaRectᵢ}, {i, 31, 34}], TableHeadings → {None, {index, area}}]
```

Out[∘]//TableForm=

index	area
31	1.025
32	1.024
33	1.024
34	1.023

Number of rectangles necessary for answer to be good to 3 places using RHRS and rounding : 32 rectangles with an area of 1.024 units^2. This shows that while more rectangles were taken possibly because of the function’s shape, the output is still not perfectly accurate. However it was closer to the correct answer than either sum for $\tan(x^2)$ because the actual area is probably an average of the LHRS and RHRS and around 0.25.
This is significantly more than the 23-26 required for the primary function. Since the original function was increasing and concave up while sin(x) was increasing and concave down over the domain, concavity might have an impact on the area under the function, with the concave down functions taking more rectangles to achieve a more accurate and precise area.

x^2 from 0 to 1

From 0 to 1, x^2 is increasing and concave up. The area of the function over the domain is $1/3*(1)^3$ which is 1/3 *1 = 1/3.
When applying Riemann’s sums, $\frac{b-a}{n} = \frac{1}{n}$and $x_k = k * \frac{1}{n}$

```
In[∘]:= p[x_] = x^2
        ParabolaArea[n_] = Sum[1/n p[k * 1/n], { k, 1, n}]
```

Out[∘]= x^2

Out[∘]= $\frac{(1+n)\ (1+2\,n)}{6\,n^2}$

```
In[∘]:= Do[{ParabAreaRectₙ = NumberForm[N[ParabolaArea[n]], 3]}, {n, 1, 50}]
        (*testing various numbers of rectangles to find where 2 numbers are the same
         For[i = 1, i ≤ 50, i++,Print[i," ", ParabAreaRectᵢ]]*)
        TableForm[Table[ {i, ParabAreaRectᵢ}, {i, 23, 26}],
         TableHeadings → {None, {index, area}}]
```

Out[∘]//TableForm=

index	area
23	0.355
24	0.354
25	0.354
26	0.353

Number of rectangles necessary for answer to be good to 3 places using RHRS and rounding : 24 rectangles with an area of 0.354 units^2.

This is similar to the 23-26 required for tan(x^2). Since the original function was increasing and concave up just like this function, we can observe that functions with similar patterns take a similar number of rectangles.

In addition, the answer I got was off the actual answer of 1/3rd by about 0.2, which is similar to the amount I think the Riemann's sum is inaccurate by using estimates and averages. This shows that functions with similar patterns will get similar outputs when using Riemann's sums accurate to 3 places.

$1/x$ from 1 to 2

From 1 to 2, 1/x is decreasing and concave up. The area of the function over the domain is ln(2) - 1n(1) = ln(2) = 0.693 (rounded to 3 places).

When applying Riemann's sums, $\frac{b-a}{n} = \frac{1}{n}$and $x_k = 1 + k * \frac{1}{n}$

```
In[∘]:= inv[x_] = 1/x

        InvArea[n_] = Sum[1/n * inv[1 + k * 1/n], {k, 1, n}]

Out[∘]= 1/x

Out[∘]= -PolyGamma[0, 1 + n] + PolyGamma[0, 1 + 2 n]

In[∘]:= Do[{InvAreaRect_n = NumberForm[N[InvArea[n]], 3]}, {n, 1, 50}]
        (*testing various numbers of rectangles to find where 2 numbers are the same
         For[i = 1, i ≤ 50, i++,Print[i," ", InvAreaRect_i]]*)
        TableForm[Table[ {i, InvAreaRect_i}, {i, 16, 19}], TableHeadings → {None, {index, area}}]

Out[∘]//TableForm=
```

index	area
16	0.678
17	0.679
18	0.679
19	0.68

Number of rectangles necessary for answer to be good to 3 places using RHRS and rounding : 17 rectangles with an area of 0.679 units^2

This is less than the 23-26 required for the primary function and the 25 rectangles for increasing and concave up for x^2. Since the original function was increasing and concave up but this was decreasing and concave up, it can be determined that whether the function is increasing/decreasing has a bigger impact on the number of rectangles than the rate of increase and possibly concavity. Nevertheless, my answer was still decently far from the correct answer of 0.693 but more accurate than the others meaning that increasing and concave up functions' area is calculated with a lesser accuracy when

Riemann’s sums are precise to the 3rd decimal.

$-x^3$ from 0 to 1

From 0 to 1, $-x^3$ is decreasing and concave down. The area of the function over the domain is $\frac{-1}{4}*1 - 0 =$ -0.250(rounded to 3places).

When applying Riemann’s sums, $\frac{b-a}{n} = \frac{1}{n}$and $x_k = k * \frac{1}{n}$

In[∘]:=
```
cubic[x_] = -x^3
CubicArea[n_] = Sum[1/n * cubic[k * 1/n], {k, 1, n}]
```

Out[∘]= $-x^3$

Out[∘]= $-\frac{(1+n)^2}{4 n^2}$

In[∘]:=
```
Do[{CubicAreaRect_n = NumberForm[N[CubicArea[n]], 3]}, {n, 1, 50}]
(*testing various numbers of rectangles to find where 2 numbers are the same
 For[i = 1, i ≤ 50, i++,Print[i," ", CubicAreaRect_i]]*)
TableForm[Table[ {i, CubicAreaRect_i}, {i, 24, 27}],
 TableHeadings → {None, {index, area}}]
```

Out[∘]//TableForm=

index	area
24	-0.271
25	-0.27
26	-0.27
27	-0.269

Number of rectangles necessary for answer to be good to 3 places using RHRS and rounding : 25 Rectangles with an area of -0.27 units^2.

This is very similar to the 23-26 required for tan(x^2) and x^2. Since the original function was increasing and concave up, but this was decreasing and concave concave down, it can be determined that functions that are concave up and increasing or concave down and decreasing take significantly more rectangles since it is harder to model those functions using rectangles due to the massive increase creating excess area. In addition, my area was inaccurate to the same degree, being about 8-10% off the actual answer of -0.25. This shows that Riemann’s sums are the least effective for these type of functions when made precise to the 3rd decimal place.

Part 2 - Darts

Basic Setup

describe random values and ranges

In[∘]:=
```
CountHits[x_, y_] := If[y < f[x], 1, 0]

dartrectarea = π^(1/2)/2 * 1
```

Out[∘]= $\frac{\sqrt{\pi}}{2}$

Function to find area with n darts

In[∘]:=
```
TanDartArea[ndarts_] :=
 (hits = 0; Do[{x_a = RandomReal[] * π^(1/2)/2, y_a = RandomReal[] * 1}, {a, 1, ndarts}];
  For[i = 1, i ≤ ndarts, i++,
   hits += CountHits[x_i, y_i]];
  dArea = N[dartrectarea * hits / ndarts];
  Return[dArea])
```

finding number of darts to make area accurate to 3 places

explain how its mathematically impossible to be the same before 1000

In[∘]:=
```
FindTanAccToT[] :=
 (Do[{NDartArea_n = NumberForm[N[TanDartArea[n]], 3]}, {n, 75000, 75010}];
  i = 75000;
  Print[NDartArea_i];
  Print[NDartArea_i+1];
  Print[FullSimplify[N[NDartArea_i ] == N[NDartArea_i+1]]];
  Print[0.256 === 0.256];
  While[(! FullSimplify[NDartArea_i == NDartArea_i+1]) && i < 75010, Print[NDartArea_i];
   Print[NDartArea_i+1];
   Print[FullSimplify[NDartArea_i == NDartArea_i+1]];
   i++;
   Print[""]];
  Return[i])
```

In[∘]:=
```
match = FindTanAccToT[]
Print[match]
```

```
0.256

0.256

0.256 == 0.256

True

Out[ ]= 75 000

75 000
```

Part 3 - Polynomials

3 Part 1

defining all the base functions

lowercase tan refers to the approximated version given

TanML refers to tan Minus Last (term) and is the original without the last term
TanM2L refers to tan minus 2 Last (terms)
TanM3L refers to tan minus the 3 Last (terms)

$$\text{tan}[x\_] = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9$$

$$\text{TanML}[x\_] = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7$$

$$\text{TanM2L}[x\_] = x + \frac{1}{3}x^3 + \frac{2}{15}x^5$$

$$\text{TanM3L}[x\_] = x + \frac{1}{3}x^3$$

Plotting the full tan approximation compared to the actual Tan function to see whether the approximation works and using a table of differences to ensure that the approximation is precise to 3 decimal places

In[ ]:= $\text{Plot}\left[\{\text{tan}[x], \text{Tan}[x]\}, \left\{x, 0, \frac{\pi}{4}\right\}\right]$

$\text{Table}\left[i\,\text{":"}\,(\text{Tan}[i] - \text{tan}[i]), \left\{i, 0.2, \frac{\pi}{4}, \frac{\frac{\pi}{4} - 0.2}{8}\right\}\right]$

the difference is less than 0.001 for all values tested. combined with the graph we can see that using all 4 terms makes the polynomial an accurate approximation to 3 places

I repeat the same process as above except I use the approximation without the last term to see if the last term had any impact

```
In[ ]:= Plot[{TanML[x], Tan[x]}, {x, 0, π/4}]
        Table[(TanML[i] - Tan[i]), {i, 0.2, π/4, (π/4 - 0.2)/8}]
        Table[(i), {i, 0.2, π/4, (π/4 - 0.2)/8}]
```

The approximation was accurate to 3 places for the majority of the range tested, at least 6/8ths of the way through, but as the value got closer to $\frac{\pi}{4}$, where the approximated function was 0.001 then 0.003 off the tangent function, meaning that the function is not accurate to 3 places without the last term over the domain. Therefore, all 4 terms in the approximated polynomial are necessary for the function to be accurate to 3 places across the entire domain. However, the approximation without the last term was accurate for values up to 0.639 which is 81% of the way to $\frac{\pi}{4}$. This is because the x becomes large and the lack of the x^9 term means that the approximated function can't keep up with the rise in the actual Tan function as x gets bigger. Since the problem only occurs in the last 75% of the domain, a 4 term approximation would work if x was closer to 0 and I hypothesize that a 3 term approximation might work from 0.2 to $\frac{\pi}{8}$because the issues that plague the approximation will not arise because x is too small.

```
In[ ]:= Plot[{TanM2L[x], Tan[x]}, {x, 0, π/8}]
        Table[(TanM2L[i] - Tan[i]), {i, 0.2, π/8, (π/8 - 0.2)/8}]
```

As predicted, the first 3 terms of the approximation are sufficiently accurate to 3 places on a domain that is closer to 0 than it is to 1 as the difference between the approximation and actual output are to the 10^{-5} which means its precise to 3 places.

3 Part 2

if $\tan(x) = x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9$,
$\tan(x^2) = x + \frac{1}{3}x^6 + \frac{2}{15}x^{10} + \frac{17}{315}x^{14} + \frac{62}{2835}x^{18}$since I should just substitute x^2 for x.

Defining the functions for tan(x^2) using the earlier functions made - also defining function to output difference between the actual Tan(x^2) and Tan(x) function and all the approximations

```
tanSq [x_] = tan[x^2]
tanSqML[x_] = TanML[x^2]
tanSqM2L[x_] = TanM2L[x^2]
tanSqM3L[x_] = TanM3L[x^2]

(*these are used later to see how tan(x^2)
 and tan(x) differ from the different approximations*)

findTanSqDiff[x_] := (Print[": with full appproximation " (Tan[x^2] - tanSq[x])];
  Print[": with 4 terms " (Tan[x^2] - tanSqML[x])];
  Print[": with 3 terms " (Tan[x^2] - tanSqM2L[x])];
  Print[": with 2 terms " (Tan[x^2] - tanSqM3L[x])])

findTanDiff[x_] := (Print[": with full appproximation " (Tan[x] - tan[x])];
  Print[": with 4 terms " (Tan[x] - TanML[x])];
  Print[": with 3 terms " (Tan[x] - TanM2L[x])];
  Print[": with 2 terms " (Tan[x] - TanM3L[x])])
```

Out[∘]= $x^2 + \frac{x^6}{3} + \frac{2\,x^{10}}{15} + \frac{17\,x^{14}}{315} + \frac{62\,x^{18}}{2835}$

Out[∘]= $x^2 + \frac{x^6}{3} + \frac{2\,x^{10}}{15} + \frac{17\,x^{14}}{315}$

Out[∘]= $x^2 + \frac{x^6}{3} + \frac{2\,x^{10}}{15}$

Out[∘]= $x^2 + \frac{x^6}{3}$

checking how many terms are needed for the approximation to be fully accurate to 3 places

I use the same process as I did when executing this procedure for tan (x)

In[∘]:= `Plot[{tanSq[x], Tan[x^2]}, {x, 0, π/4}]`

`Table[(Tan[i^2] - tanSq[i]), {i, 0.2, π/4, (π/4 - 0.2)/8}]`

As can be seen in the graph and confirmed by the data, the full approximation is fully precise to 3 terms across the entirety of the x range from 0.2 to $\pi/4$ as none of the differences are large enough to be in the thousandths place. Therefore, the next step is to test the approximations full precision over the domain without the last term.

```
In[•]:= Plot[{tanSqML[x], Tan[x^2]}, {x, 0, π/4}]

        Table[(Tan[i^2] - tanSqML[i]), {i, 0.2, π/4, (π/4 - 0.2)/8}]
```

Unlike the tan(x) function, the tan(x^2) function without the last term is accurate to 3 places since the difference between the function and approximation was in the ten-thousandths meaning the approximation was precise to 3 places. The next step is to test without the last *TWO* places.

```
In[•]:= Plot[{tanSqM2L[x], Tan[x^2]}, {x, 0, π/4}]

        Table[(Tan[i^2] - tanSqM2L[i]), {i, 0.2, π/4, (π/4 - 0.2)/8}]

        Table[i, {i, 0.2, π/4, (π/4 - 0.2)/8}]
```

With only 3 terms, the approximation is mostly precise to 3 terms over the domain except for the last term, $\frac{\pi}{4}$, where the difference between the expected and estimated was 0.002 which means that the 3 term approximation was not precise to 3 places. However, tan(x^2) was precise when using 4 terms which is different to the 5 terms needed for an accurate approximation of tan(x). Therefore, the 4 term approximation is the smallest polynomial that is fully precise for the full domain.
or $x^2 + \frac{x^6}{3} + \frac{2\,x^{10}}{15} + \frac{17\,x^{14}}{315}$ is the approximation with 4 terms that is fully accurate.

checking whether the same precision is needed for individual x values

i will check Tan (x) from the 8 evenly spaced values between 0.2 and $\pi/4$ and check Tan (x^2) from the same 8 evenly spaced values. The table is built with the input value at the top, the difference between the 4 approximations of tan(x^2) compared to the actual value following, and the difference between the approximations and actual value of tan(x) at the bottom with a blank line to finish to help clarify what output corresponds to what input.

```
In[•]:= Table[(Print[i];
          Print["tan sq and approx"];
          findTanSqDiff[i];
          Print["tan and approx"];
          findTanDiff[i];
          Print[""];
          i), {i, 0.2, π/4, (π/4 - 0.2)/8}]
```

As can be seen, both functions initially can be approximated with 2 terms until the 4th input, 0.419, was passed in. For this input, tan(x^2) could be approximated to 2 terms while tan(x) had to be approximated to 3 terms because the 2 term approximation was off by .0018 which makes it imprecise to 3 terms.

Tan(x^2) could be approximated for 2 terms while tan(x) had to use 3 for the next term as well, 0.492, and tan(x) had to use a 4 term approximation by the 6th term, 0.566, because the 3 term approximation had a difference of 0.001 making it imprecise to 3 places. Meanwhile, the 2 term approximation is still precise for tan(x^2). By the 7th term, 0.639, tan(x^2) needed a 3 term approximation because the 2 term difference was 0.0016, which made it imprecise. However, tan(x^2) still needs less terms in its approximation than tan(x) at the 7th term. At the 8th term, 0.712, tan(x)'s 4 term approximation was not precise since the difference was .0012 so the full, 5 term approximation was needed for tan(x) which tan(x^2) was still precise using a 3 term approximation. However, at the last term, tan(x^2) needed the fourth term to be precise since the difference increased to .002. The final result is that tan(x) needed a full 5 term approximation while tan(x^2) only needed a 4 term approximation for the values to be precise to 3 decimal places. Overall, less terms were needed for tan(x^2) to be precise compared to tan(x) as tan(x) required more terms at a faster rate as well. Tan(x^2) required less terms for 5/9 inputs and more for 0/9.

3 Part 3

assuming "integrate" in the question means use integrate function

I integrate each of the approximations of tan(x^2) and compare their values

```
In[•]:= (*5 TERMS*)
N[Integrate[tanSq[x], {x, 0, π^(1/2)/2}]]
(*4 TERMS*)
N[Integrate[tanSqML[x], {x, 0, π^(1/2)/2}]]
(*3 TERMS*)
N[Integrate[tanSqM2L[x], {x, 0, π^(1/2)/2}]]
(*2 TERMS*)
N[Integrate[tanSqM3L[x], {x, 0, π^(1/2)/2}]]
```

Increasing the number of terms has an impact on my answer for the earlier increases in the number of terms. For example, going from 2 terms to 3 terms had an impact of 0.0023 on the area, which is important when being precise to 3 places. However, going from 3 places to 4 places had an impact of 0.0006 units^2 which is much smaller. But, because of rounding, this still had an impact on the result, changing it from 0.255 to 0.256, making it significant. Finally, the transition from 4 terms to 5 terms had no impact as predicted because the rounded area of 0.256 was maintained. This can be corroborated by the earlier experimentation which showed that the approximation tan(x^2) with only 4 out of 5 terms was precise to 3 decimal places. In conclusion, increasing the terms does have a significant impact if the number of terms is less than the previously agreed upon sufficient amount that would make the approximation precise to 3 places.

Part 4 - Conclusion

In this experiment, I was trying to find the area of tan(x^2) from 0 to $\frac{\pi^{\frac{1}{2}}}{2}$. I used various different approaches namely summation, probability, and approximation. Each method has its advantages and disadvantages depending on the function it is trying to find the area of.When using Riemann’s sums precise to 3 places, I got 0.238 and 0.274 units^2 as my area, for the left and right handed sum, respectively. When using a trapezoid I got the area of 0.259 units^2. After using probability and dart throwing to find the area precise to 3 decimal places, I got the final area of ____ and when using a polynomial application I got a final area of 0.256 units^2. When deciding which method I would recommend, I weighed multiple factors like perceived accuracy and effectiveness along with efficiency. From the beginning, while sums are efficient and run relatively quickly, my answer was highly dependent on the side that I chose to sum from, meaning that neither are close to the answer and are probably 2 extremes of an average, which turned out to be 0.256 in this case. Therefore, I think an efficient and accurate solution would be to use an average of the left and right sums, which turns out to be the trapezoid method. Compared to the sums, this method was much faster, taking only 9 shapes instead of the 20-30 taken using the rectangle method. In addition, this method was probably more accurate due to its central location between the different sums. Compared to the other styles taken, I still think the trapezoid is the most effective solution. The probability method, which revolved around throwing thousands of “darts” or random points and using a fraction to calculate area, was not only relatively inaccurate but also was the least efficient. When throwing tens of thousands of darts, my answers still varied by 0.05 to 0.1 which was a large amount considering that I am looking for a precise area. In addition, since every point was randomized for every run, finding the amount of darts that make the area precise to 3 decimal places took by far the most time, taking minutes to possibly hours to calculate. For the time investment, the probability method does not return the accuracy desired and its marginal accuracy benefits are not worth the loss in efficiency compared to the trapezoid. Finally, the last method tested was the polynomial approximation method. Since polynomials are integratable, I was able to use a well known approximation for tan(x) to find the area. I would have picked this as the most effective way because its the simplest and has the least complex procedure and equations. However, I realized that its not a guarantee that every function that cannot be solved using normal integration will not have a ready made replacement. Finding that replacement will take a computer time or might not exist. While this solution is efficient, it assumes that a polynomial approximation exists, which I cannot do when recommending a wholesale solution. Because of the unpredictability of this method, I would recommend using trapezoids over approximations because a trapezoid is guaranteed a somewhat accurate answer when precise to 3 places while the other methods take too long or are unpredictable.