MATH 245 Computational Mathematics Project 1

August 1, 2019

1 Niels Clayton 300437590

1.1 Question 1

Compute the sum of vector x using a for loop:

$$s_1 = \sum_{i=1}^{5} x_i$$

[3]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
x = [0.1, 1.3, -1.5, 0, 12.3]
s1 = 0
for i in x:
    s1 += i
print("s1 = {}".format(s1))
```

```
s1 = 12.200000000000001
```

Compute the sum of vector x^i using a for loop:

$$s_2 = \sum_{i=1}^{5} x_i^i$$

[4]:
```
s2 = 0

for i in range(5):
    s2 += (x[i]**i)
print("s2 = {}".format(s2))
```

```
s2 = 22893.214100000005
```

Find the max value of vector x using a for loop:
$M = max\{|x_i|, 1, 2, ..., 5\}$

[5]:
```
max = float('-inf')

for i in x:
    if (abs(i) > max):
```

```
        max = abs(i)

print("M = {}".format(max))
```

```
M = 12.3
```

Find the min value of vector x using a for loop:
$m = min\{|x_i|, 1, 2, ..., 5\}$

[6]:
```
min = float('inf')

for i in x:
    if (abs(i) < min):
        min = abs(i)

print("m = {}".format(min))
```

```
m = 0
```

Compute the sum, max, and min of vector x using built in python functions:

[7]:
```
print("sum calculated function = {}".format(sum(x)))
print("max calculated function = {}".format(np.max(np.abs(x))))
print("min calculated function = {}".format(np.min(np.abs(x))))
```

```
sum calculated function = 12.200000000000001
max calculated function = 12.3
min calculated function = 0.0
```

It can be seen that when we calculate the sum, max, or min using the built in python functions, we get the exact same values as we would when calculating them manually with loops.

1.2 Question 2

[8]:
```
x = np.linspace(2,3,5)                          # Vector of 5 equally spaces␣
 ↪values between 2 and 3
x[1] += 1                                       # Add 1 to element 2 of x
y = np.array([4,6,8,10,12])                     # Create array y with successive␣
 ↪even values starting at 4
A = np.array([x, np.array([0,0,0,0,0]) , y]) # Form into a matrix
z = []
for i in range(5):
    average = (A[0,i] + A[1,i] + A[2, i])/3  # Average the columns and append␣
 ↪them to z
    z.append(average)
np.asarray(z)
print("vector x = {}\n".format(x))
print("vector y = {}\n".format(y))
```

```
print("matrix A = \n{}\n".format(A))
print("vector z = {}\n".format(z))
```

```
vector x = [2.   3.25 2.5  2.75 3.  ]

vector y = [ 4  6  8 10 12]

matrix A =
[[ 2.    3.25  2.5   2.75  3.  ]
 [ 0.    0.    0.    0.    0.  ]
 [ 4.    6.    8.   10.   12.  ]]

vector z = [2.0, 3.0833333333333335, 3.5, 4.25, 5.0]
```

1.3 Question 3

Compute $C_1 = A + B$ and $C_2 = A - B$

[9]:
```
A = np.array([[1,2], [4,-1]])      # Declare matrix A and B
B = np.array([[4,-2], [-6, 3]])
c_1 = A+B
print("matrix C_1: \n{}\n".format(c_1))
c_2 = A-B
print("matrix C_2: \n{}\n".format(c_2))
```

```
matrix C_1:
[[ 5  0]
 [-2  2]]

matrix C_2:
[[-3  4]
 [10 -4]]
```

Compute $A.B$ & $B.A$

[10]:
```
d_1 = np.dot(A,B)
print("matrix D_1: \n{}\n".format(d_1))
d_2 = np.dot(B,A)
print("matrix D_2: \n{}\n".format(d_2))
```

```
matrix D_1:
[[ -8   4]
 [ 22 -11]]

matrix D_2:
[[ -4  10]
```

```
 [  6 -15]]
```

Compute $F_{ij} = B_{ij} + A_{ij}B_{ij}^{\frac{1}{3}}$

[11]:
```
temp_row = []
for i in range (2):
    for j in range (2):
        val = B[i][j] + (A[i][j] * (np.sign(B[i][j])*(np.power(np.abs(B[i][j]),␣
↪1/3))))
        temp_row.append(val)
F = np.asarray(temp_row)
F.shape = (2,2)
print("matrix F: \n{}\n".format(F))
```

```
matrix F:
[[  5.58740105  -4.5198421 ]
 [-13.26848237   1.55775043]]
```

Find if A and B are singular by computing their determinants, If the determinant is 0, it is singular. If not singular, find the inverse matrix.

[12]:
```
print("Determinant of A: {}\n".format(np.linalg.det(A)))
print("Determinant of B: {}\n".format(np.linalg.det(B)))
print("Inverse of A: \n{}".format(np.linalg.inv(A)))
```

```
Determinant of A: -8.999999999999998

Determinant of B: 0.0

Inverse of A:
[[ 0.11111111  0.22222222]
 [ 0.44444444 -0.11111111]]
```

Find the Eigenvalues of matrix B

[13]:
```
print("The Eigenvalues of B are: {0} and {1}".format(round(np.linalg.
↪eigvals(B)[0]),round(np.linalg.eigvals(B)[1])))
```

```
The Eigenvalues of B are: 7.0 and -0.0
```

Because matrix B is singular, it would be expected that one of its eigenvalues is 0. the eigenvalues calculates confirm this.

1.4 Question 4

Write a function to compute the factorial $n!$

[14]:
```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n*(factorial(n-1))
```

Create a vector of 100 elements between 0 and 100 using the formula: $a_n = \frac{(-1)^n \pi^{2n}}{(2n)!}$

[15]:
```
a = np.linspace(0,100,101)
for i in range (101):
    a[i] = (np.power(-1, a[i])*(np.power(np.pi, (2*a[i]))))/(factorial(2*a[i]))
```

```
/usr/lib/python3.7/site-packages/ipykernel_launcher.py:5: RuntimeWarning:
overflow encountered in double_scalars
  """
```

Sum all the values of a:

[16]:
```
print(sum(a))
```

```
-1.0000000000000002
```

We can see that the series converges on -1, which is what was expected since the function is actually the $cos(\pi)$ power series after its all been summed, and $cos(\pi)$ is -1.

If we attempt the same calculation using math.factorial() we will get an error that the integer value produced by math.factorial() is too large to be converted to a float.

[17]:
```
a = np.linspace(0,100,101)
for i in range (101):
    a[i] = (np.power(-1, a[i])*(np.power(np.pi, (2*a[i]))))/(math.
 ↪factorial(2*a[i]))
```

```
      ␣
↪---------------------------------------------------------------------------

       OverflowError                                Traceback (most recent call␣
↪last)

       <ipython-input-17-9a41e8c41b3a> in <module>
         1 a = np.linspace(0,100,101)
         2 for i in range (101):
   ----> 3     a[i] = (np.power(-1, a[i])*(np.power(np.pi, (2*a[i]))))/(math.
↪factorial(2*a[i]))
         4

       OverflowError: int too large to convert to float
```

1.5 Question 5

Create two column vectors x and y of 100 elements with random values between 10 and 10.

[18]:
```
x = np.array(-10 + (20*np.random.random(100)))
y = np.array(-10 + (20*np.random.random(100)))
```

Write a function to compute the dot product of two vectors

[19]:
```
def mydot(x, y):
    total = 0.0
    for i in range (100):
        total += (x[i]*y[i])
    return total
```

Compare my function to numpy.dot() and numpy.inner()

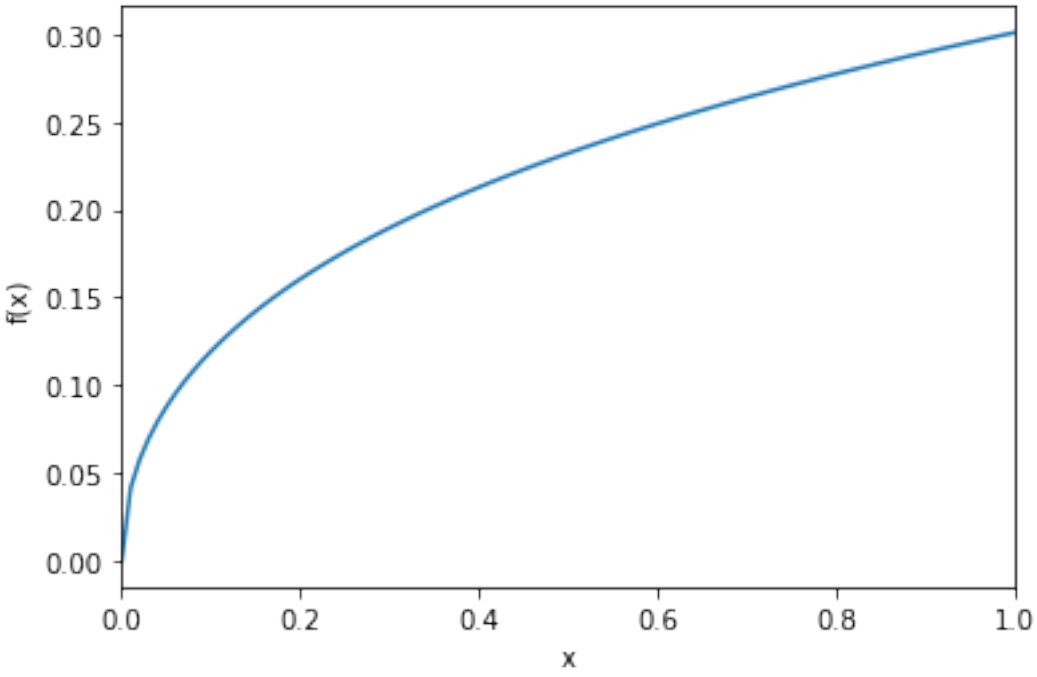
[20]:
```
print("Output using numpy.dot() function:   {}".format(np.dot(x,y)))
print("Output using numpy.inner() function: {}".format(np.inner(x,y)))
print("Output using my function function:   {}".format(mydot(x,y)))
```

```
Output using numpy.dot() function:   897.8973007368861
Output using numpy.inner() function: 897.8973007368861
Output using my function function:   897.8973007368861
```

It can be seen that both numpy.dot() and numpy.inner() functions output identical values. The function mydot() that I have written outputs almost the same valve, but differs in the last 3 decimal points (past 1×10^{-11} the functions are different.) ## Question6 Plot the following graphs:

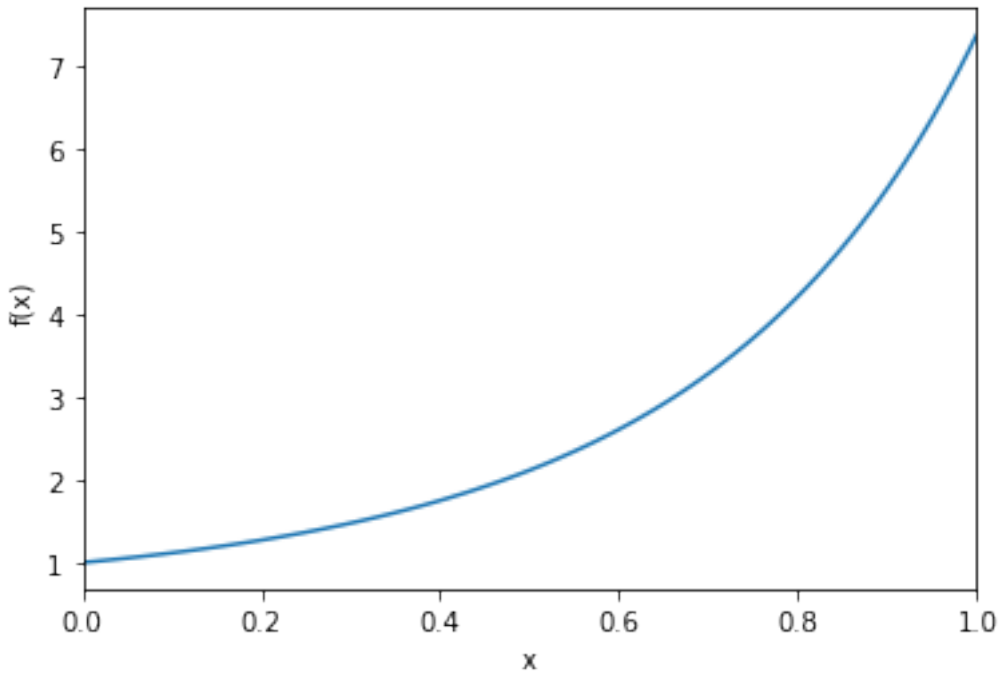
$y(x) = \log_{10}(1 + \sqrt{x})$

[21]:
```
import matplotlib.pyplot as plt
x= np.linspace(0,1,101)
y = np.log10((1+np.sqrt(x)))
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0,1)
plt.show()
```



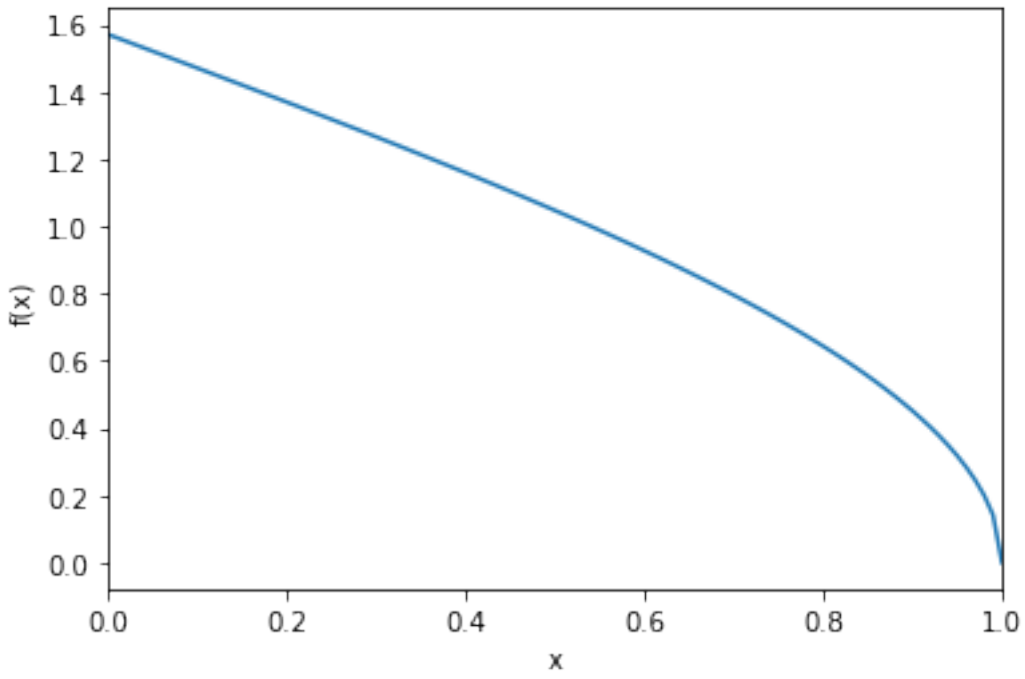
$y(x) = e^{x+x^2}$

[22]:
```
y = np.power(np.e, (x + np.power(x,2)))
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0,1)
plt.show()
```



$$y(x) = arcos(x)$$

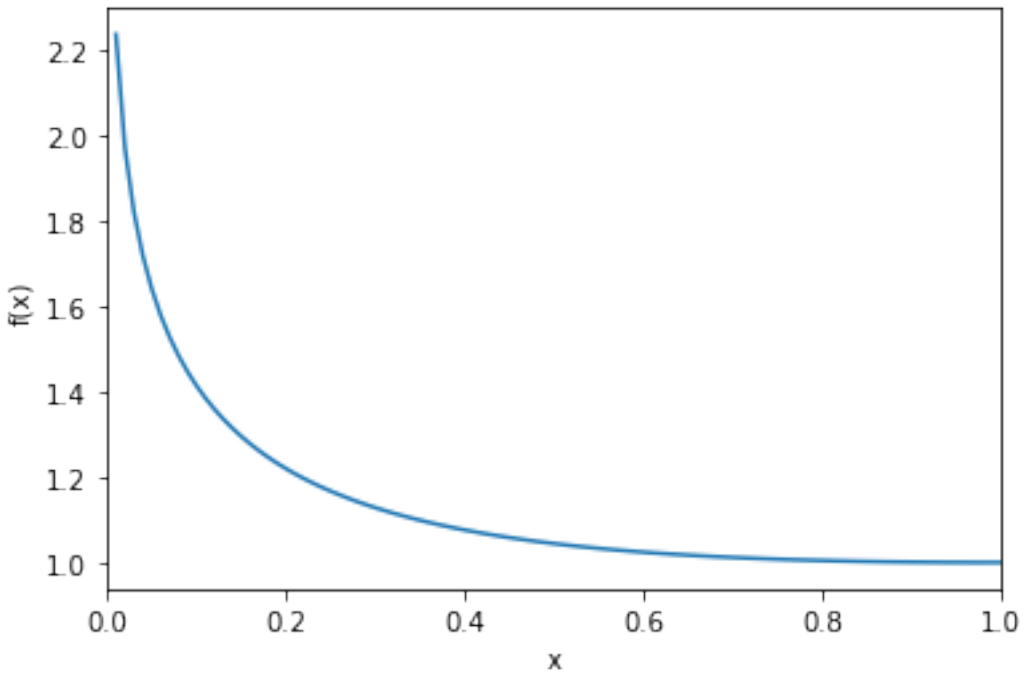
[23]:
```
y = np.arccos(x)
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0,1)
plt.show()
```



$$y(x) = \sqrt{1 + \log^2(x)}$$

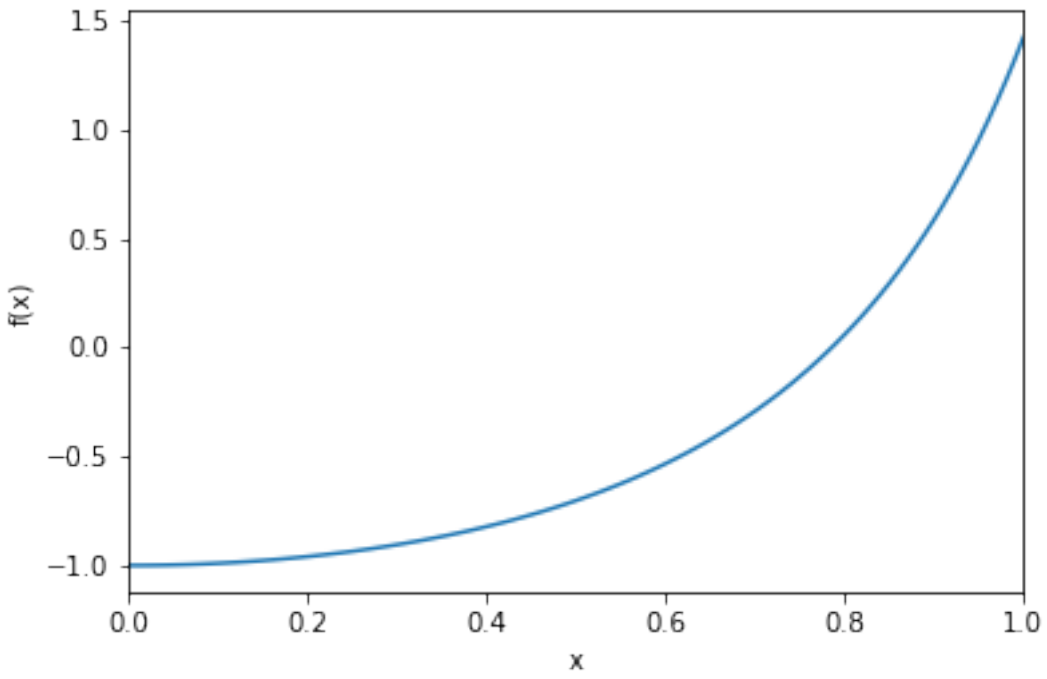
[24]:
```
y = np.sqrt((1 + np.power(np.log10(x),2)))
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0,1)
plt.show()
```

```
/usr/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning: divide
by zero encountered in log10
  """Entry point for launching an IPython kernel.
```



$$y(x) = \tan^2(x) - 1$$

[25]:
```
y = np.power(np.tan(x),2) -1
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.xlim(0,1)
plt.show()
```



1.6 Question 7

The while loop will terminate when $s + t = s$. This will occur when the value of t is so small that it is known as machine epsilon.

[58]:
```
def powersin(x):
    s = 0
    t = x
    n = 1
    count = 0
    max = 0
    while s+t != s:
        s += t
        t = -x ** 2/((n + 1) * (n + 2)) * t
        n += 2
        count += 1
        if max < np.abs(t):
            max = np.abs(t)
    return s
```

It can be seen that the accuracy of the computation for $x = \pi/2$, $x = 11\pi/2$, $x = 21\pi/2$, and $31\pi/2$ becomes drastically less accurate the larger the value of x.

[28]:
```
print("compute pi/2 with sin           {}".format(np.sin((np.pi)/2)))
print("compute pi/2 with powersin      {}\n".format(powersin((np.pi)/2)))
print("compute 11pi/2 with sin         {}".format(np.sin((11*np.pi)/2)))
```

```
print("compute 11pi/2 with powersin     {}\n".format(powersin((11*np.pi)/2)))
print("compute 21pi/2 with sin          {}".format(np.sin((21*np.pi)/2)))
print("compute 21pi/2 with powersin     {}\n".format(powersin((21*np.pi)/2)))
print("compute 31pi/2 with sin          {}".format(np.sin((31*np.pi)/2)))
print("compute 31pi/2 with powersin     {}\n".format(powersin((31*np.pi)/2)))
```

```
compute pi/2 with sin            1.0
compute pi/2 with powersin       1.0000000000000002

compute 11pi/2 with sin          -1.0
compute 11pi/2 with powersin     -1.0000000002128728

compute 21pi/2 with sin          1.0
compute 21pi/2 with powersin     0.9998667640418495

compute 31pi/2 with sin          -1.0
compute 31pi/2 with powersin     -5822.01852702401
```

Compute how many terms are required to reach the result

[35]:
```
print("compute terms required for pi/2:         {}\n".format(powersin((np.pi)/
↪2)))
print("compute terms required for 11pi/2:       {}\n".format(powersin((11*np.
↪pi)/2)))
print("compute terms required for 21pi/2:       {}\n".format(powersin((21*np.
↪pi)/2)))
print("compute terms required for 31pi/2:       {}\n".format(powersin((31*np.
↪pi)/2)))
```

```
compute terms required for pi/2:         11.0

compute terms required for 11pi/2:       37.0

compute terms required for 21pi/2:       60.0

compute terms required for 31pi/2:       78.0
```

Compute the largest term in the series

[57]:
```
print("compute Largest term for pi/2:         {}\n".format(powersin((np.pi)/2)))
print("compute Largest term for 11pi/2:       {}\n".format(powersin((11*np.pi)/
↪2)))
print("compute Largest term for 21pi/2:       {}\n".format(powersin((21*np.pi)/
↪2)))
print("compute Largest term for 31pi/2:       {}\n".format(powersin((31*np.pi)/
↪2)))
```

```
0.6459640975062462


compute Largest term for pi/2:          0.6459640975062462

3066514.6373838116


compute Largest term for 11pi/2:        3066514.6373838116

14672596728254.967


compute Largest term for 21pi/2:        14672596728254.967

7.988994169819993e+19


compute Largest term for 31pi/2:        7.988994169819993e+19
```

It can be seen that the accuracy of the computation of $\sin(x)$ for $x = \pi/2$, $x = 11\pi/2$, $x = 21\pi/2$, and $31\pi/2$ becomes drastically less accurate the larger the value of x. This is because the larger the value of x, the more of the available bits contained by the float datatype will be used to hold integer values, decreasing the accuracy of the computation. Because of this, using a power series to compute $\sin(x)$ is only reliable for small values of x.

1.7 Question 8

Compute the value to which y_n converges

$y_n = \frac{x_n}{(1+\sqrt{2}/2)^n}$

[11]:
```
def y(x, n):
    return x / ((1 + (1/np.sqrt(2))) ** n)

x1 = 1
x2 = 2
y1 = y(x1, 1)
y2 = y(x2, 2)
i = 3

while y1 != y2:
    xt = x2
    x2 = (2 * xt) - (0.5 * x1)
    x1 = xt
    y1 = y2
    y2 = y(x2, i)
    i += 1
```

```
print(y2)
```

```
0.7071067811865481
```

The series will converge on $\sqrt{2}/2$