**INFO 6205 Summer 1 2023 Project**

* Part3 Introduction
  + Aim:

To find another correction term for MGL series expect for the three terms provided in the requirement.

* Program
  + Algorithm

As mentioned in the video, the fourth correction term was given. So I tested the term to see the result. The term is:

A picture containing font, text, screenshot, symbol

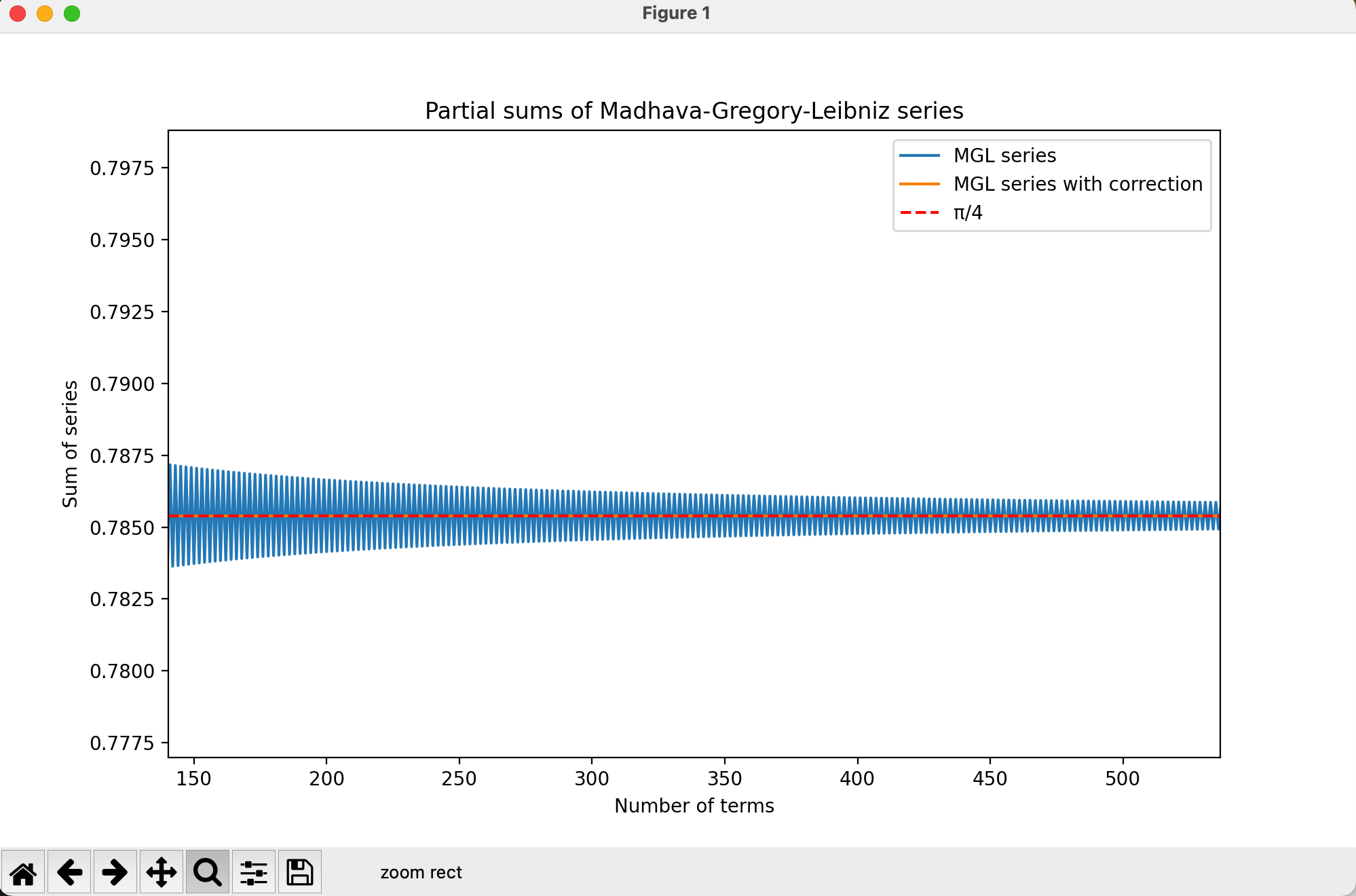
Description automatically generated

* Observations & Graphical Analysis

Using python to draw a graph to compare the difference between MGL series and MGL series after added a correction term we mentioned in the Algorithm, we can clearly see the difference.

A screen shot of a graph

Description automatically generated with medium confidence



A screenshot of a computer

Description automatically generated with medium confidence

So, we can see that the value of MGL series fluctuates up and down from pi/4, and after adding the correction term, the error between value of MGL series and pi/4 greatly reduced. (The black area is the gap/difference between MGL series and MGL series after corrected)

* Results & Mathematical Analysis

Using Java, to ensure that the correction terms make sense to reduce the error between MGL series and pi/4, I compare the MGL series and that after being corrected. I found that the fourth correction term makes sense because the error is smaller than 1E-6 when n is 1000 (such a small number).

* Unit tests
* @Test  
  public void testQuarterPi() {  
   *assertEquals*(Rational.*apply*("0"), Madhava.*quarterPi*(1, Madhava::*termFirst*));  
   *assertEquals*(Rational.*apply*("7/6"), Madhava.*quarterPi*(2, Madhava::*termFirst*));  
   *assertEquals*(Rational.*apply*("1/5"), Madhava.*quarterPi*(1, Madhava::*termSecond*));  
   *assertEquals*(Rational.*apply*("58/51"), Madhava.*quarterPi*(2, Madhava::*termSecond*));  
   *assertEquals*(Rational.*apply*("1/9"), Madhava.*quarterPi*(1, Madhava::*termThird*));  
   *assertEquals*(Rational.*apply*("142/123"), Madhava.*quarterPi*(2, Madhava::*termThird*));  
   *assertEquals*(Rational.*apply*("5/27"), Madhava.*quarterPi*(1, Madhava::*termFourth*));  
   *assertEquals*(Rational.*apply*("90/79"), Madhava.*quarterPi*(2, Madhava::*termFourth*));  
  }  
    
  @Test  
  public void testTerm() {  
   *assertEquals*(Rational.*apply*("-22/27"), Madhava.*termFourth*(1));  
   *assertEquals*(Rational.*apply*("112/237"), Madhava.*termFourth*(2));  
   *assertEquals*(Rational.*apply*("-38/117"), Madhava.*termFourth*(3));  
  }  
    
  @Test  
  public void testPi() {  
   *assertEquals*(approximatePi.divide(4).doubleValue(), Madhava.*quarterPi*(1000, Madhava::*termFirst*).toDouble(), 1E-6);  
   *assertEquals*(approximatePi.divide(4).doubleValue(), Madhava.*quarterPi*(1000, Madhava::*termSecond*).toDouble(), 1E-6);  
   *assertEquals*(approximatePi.divide(4).doubleValue(), Madhava.*quarterPi*(1000, Madhava::*termThird*).toDouble(), 1E-6);  
   *assertEquals*(approximatePi.divide(4).doubleValue(), Madhava.*quarterPi*(1000, Madhava::*termFourth*).toDouble(), 1E-6);  
  }  
    
  @Test  
  public void testError() {  
   *assertTrue*(Math.*abs*(Madhava.*mglSeries*(1001).toDouble() - approximatePi.divide(4).doubleValue()) > Math.*abs*(Madhava.*quarterPi*(1000, Madhava::*termFirst*).toDouble() - approximatePi.divide(4).doubleValue()));;  
   *assertTrue*(Math.*abs*(Madhava.*mglSeries*(1001).toDouble() - approximatePi.divide(4).doubleValue()) > Math.*abs*(Madhava.*quarterPi*(1000, Madhava::*termSecond*).toDouble() - approximatePi.divide(4).doubleValue()));;  
   *assertTrue*(Math.*abs*(Madhava.*mglSeries*(1001).toDouble() - approximatePi.divide(4).doubleValue()) > Math.*abs*(Madhava.*quarterPi*(1000, Madhava::*termThird*).toDouble() - approximatePi.divide(4).doubleValue()));;  
   *assertTrue*(Math.*abs*(Madhava.*mglSeries*(1001).toDouble() - approximatePi.divide(4).doubleValue()) > Math.*abs*(Madhava.*quarterPi*(1000, Madhava::*termFourth*).toDouble() - approximatePi.divide(4).doubleValue()));;  
  }
* Conclusion

The fourth correction term should be:

A picture containing font, text, screenshot, symbol

Description automatically generated

* References
  + https://www.youtube.com/watch?v=ypxKzWi-Bwg&t=1375s
* Part4 Introduction
  + Aim:

To find if there are any other sequences to calculate approximation of pi

* Program
  + Algorithm

After searching, I found the most precise one is Nilakantha’s formula, which looks like:

A picture containing text, font, line, screenshot

Description automatically generated

* Observations & Graphical Analysis

From the unit test, I found that the error between the new sequence and pi is smaller than 1E-9 (much smaller than that of MGL series).

* Results & Mathematical Analysis

With 12 terms, the sequence gives out the result of pi = 3.141479689, which is only accurate to 3 decimal places. So the Nilakantha’s formula must be a reliable one to help approximate the value of pi.

* Unit tests
* @Test  
  public void testPi() {  
   *assertEquals*(approximatePi.doubleValue(), Nilakantha.*calculatePi*(1000).toDouble(), 1E-9);  
   *assertEquals*(approximatePi.doubleValue(), Nilakantha.*calculatePi*(3000).toDouble(), 1E-10);  
  }
* Conclusion

The new sequence is:

A picture containing text, font, line, screenshot

Description automatically generated

* References
  + http://www.maeckes.nl/Formule%20voor%20pi%20%28Nilakantha%29%20GB.html