15.03.2019, Gliwice

Numerical methods

Calculation of function value

Damian Kucharski

Magdalena Rogalska

Presentation of the problem given

1. Compare time of polynomial value calculation at the given point (x_0) using standard formula $W_n(x) = \sum_{i=0}^{i=n} a_i x^i$ and Horner scheme. Analyze relationship between this time and polynomial degree.

2. Implement programme determining interval in which roots of polynomial are contained.

3. Implement programme determining value of given function with the aid of Taylor series. Evaluate the number of terms for given accuracy.

Listings of the programmes or the correct directory path

As it would be very impractical to put whole source code here, we will provide only listings of actual functions.
Whole code - with helper functions and everything else - you can find by following this link:
https://github.com/Slajni/Numerical-Methods-Lab---Calculation-of-function-value/tree/master

```python
def normalEvaluation(coefs,val):
   """Evaluates function value in standard way


   Keyword arguments:
   coefs -- dictionary with coefficients and their values
   val -- value of a parameter for which function is gonna be evaluated

   """
   valToReturn = 0
   for key, value in coefs.items():
       valToReturn += val**key * value
   return valToReturn
```

```python
def hornerEvaluation(coefit,val, checker=False):
   """Evaluates function value using horners algorithm or checks if general condition is
satisfied


   Keyword arguments:
   coefits -- dictionary with coefficients and their values
   val -- value of a parameter for which function is gonna be evaluated
   checker -- false by default, if true function instead of the function value returns
the logical value informing if the general condition is satisfied

   """
   satisfies = True
   coefs = coefit.copy()
   hValue = coefs[len(coefs)-1]
   del coefs[len(coefs)-1]
   for i in range(len(coefs)-1,-1,-1):
       hValue = hValue*val + coefs[i]
       if hValue < 0 and checker == True:
           satisfies = False
           break
   if(checker):
       return satisfies
   else:
    return hValue
```

```python
def findBoundsOfRoots(coefs,step=0.1,start=0.1):
   """returns the interval in which roots of polynomial are included

   if evaluation of algorithm takes too long it is assumed that polynomial
is unbounded on one or both sides


   Keyword arguments:
   coefs -- dictionary with coefficients and their names
   step -- the rate in which we change value of evaluation to search for
bounds
   start -- starting value for searching

   """

   interval = [None,None]
   jumper = start
   timer = time.time()
   while(horner Evaluation(coefs,jumper,checker=True)==False):
       jumper += step
       if float(time.time() - timer) > 10:
           interval[1] = float("inf")
           #break
   if interval[1] == None:
       interval[1] = jumper
   jumper = start
   coefficients = coefs.copy()
   if((len(coefficients)-1)%2 == 1):
       for key in coefficients.keys():
           coefficients[key] *= -1
   for key in coefficients.keys():
       if(key %2 == 1):
           coefficients[key] *= -1 # TODO: check the version with locals
   while(hornerEvaluation(coefficients,jumper,checker=True)==False):
       jumper += step
       if float(time.time() - timer) > 10:
           interval[0] = float("inf")
           break
   if interval[0] == None:
       interval[0] = -1*jumper
   return interval
```

```python
def calculateSineFromTaylorPolynomial(value, numberOfTerms):
   """returns the value of sine for given argument (in radians)


   Keyword arguments:
   numberOfTerms -- number of terms of taylor polynomial used to calculate
sine
   value -- argument for sine
   """

   rsum = 0
   if numberOfTerms < 1:
       return rsum
   else:
       term = value
       rsum += term
       for i in range(3,numberOfTerms,2):
           term = term * value**2 * (-1) / mt.factorial(i)
           rsum += term
       return rsum
```

```python
def calculateNForGivenAccuracy(accuracy, startingPoint):
   """returns the number of taylor polynomial terms needed to achieve given
accuracy in place of derivative calculations

   Keyword arguments:
   accuracy -- accuracy we want to achieve
   startingPoints -- argument for which we want to be the center of our
approximations

   """

   N = 0
   while(x**(N+1)/mt.factorial(N+1)) > accuracy:
       N = N + 1

   return N
```

Results of computations made with aid of the programmes

For the first part of the exercise we were performing computations on given polynomial:

$x^4 + 7x^3 - 94x^2 - 328x + 960$

for $x = 10$

Result obtained for calculation with general formula: 5280

In time: $1.7642974853515625e - 05\ s$

Result obtained for calculation with horner scheme: 5280

In time: $2.1457672119140625e - 05\ s$

As we can see the evaluation of Horner Scheme took longer.
That can be surprising but actually it is not. As polynomial was of small degree the actual calculation didn't took long. Most of the time consumed by both functions in program were not calculations themselves but other external instructions. For polynomial of high degrees the advantage of horner scheme is very visible.

The bounds of roots we obtained for that polynomial are : $(-13.9, 8.6)$

For the third task we decided to evaluate value of sine for $x = 0.1$
Our program calculated that the number of terms needed to calculate it with accuracy $\varepsilon = 0.001$ is $N = 2$
The value our program returned is $y = 0.09983$ which gives real accuracy $\varepsilon_0 = 0.00016658$

Conclusions based on the results obtained from the numerical experiments

From our numerical experiments we conclude that numerical algorithms are very good tool that we can use to approximate or calculate the value of some functions. In some cases like in horner scheme it saves us a lot of time by limiting the number of calculations we have to perform. Also - this scheme can be used to find the bounds of roots of a polynomial and even a value of its derivatives.
Also, sometimes it is very hard to get exact value of function if it is a complicated one. Taylor series is very good tool here. We can use it to create a polynomial that - in the place that we are interested to calculate function value in - reminds the original function in very accurate way. And then again - value of polynomial can be much easily calculated - for example using horner scheme.