Ni Jiasheng

jn2294

2022-2-6

Clarifications

Notation

The estimation number of float operations are considered to be the basic calculation performed during the program.

- **Addition**, **subtraction**, **multiplication**, **division**, **root**, **power** are considered **1** ops. (Although multiplication and division are much more complicated in its basicnimplementation in machine language, here we consider them to be one float ops for simplification)
- **Function operation** $f(x) = cosx - x^3$ contains **3** ops, one for cos, one for x^3 and one for subtraction, same logic for $f'(x)$
- **Function operation** $f'(x) = -sinx - 3x^2$ contains **5** steps, $-sinx$ is considered as one multiplication and sin operation. $3x^2$ is considered a x^2 operation and $3 \times x^2$ multiplication.
- **Comparison** is **not** considered an ops here.
- `self.printHelper()` contains a ops, but we **don't** count it since it is not part of the algorithms themselves. just for printing results.
- `#1` means the number of ops in a particular line of code.

Code

All the codes are done in python 3.8 with anaconda enmvironment

External libraries used are:

- numpy
- matplotlib
- sympy
- scipy

Run

Type in `python hw1.py` in the console. The results as well as the figure in 1.5 will pop out.

Problem 1.1

- Total number of iteration: 12;
- The root found: 0.865478515625;
- Final Absolute Error: 4.51562500003444e-06;
- The estimated number of floating operation: 120, 10 operations(excluding comparisons) for each iteration.

Code Snippet

```
def bisectionMethod(self,start,end):
    """
        Implementation for bisection method
        :param start: Left endpoint
        :param end:  Right endpoint
        :return: None
        """
    print("1.1 Performing bisection method!")
    iteration = 0
    while (end-start)/2 > self.TOL: #1
        iteration += 1
        mid = (start+end)/2  #1
        if np.abs(mid -  float(self.true)) < self.error:  #1
            self.printHelper(iteration,1, mid, np.abs(mid -
float(self.true)), iteration * 6, 6)  #0
            return mid
        if self.f(self.func,start)*self.f(self.func,mid) < 0: #7
            end = mid
        else:
            start = mid
```

Problem 1.2

- Total number of iteration: 7;
- The root found: 0.8654740525339539;
- Final Absolute Error: 5.253395396476179e-08;
- The estimated number of floating operation: 70, 10 operations(excluding comparisons) for each iteration.

Code Snippet

```
def newtonMethod(self, max_iter = 10000, initial = 0.3):
    """
        Implementation for newton method
        :param max_iter: To prevent the possible divergence in
the method
        :param initial: x0
        :return: None
        """

    print("1.2 Performing newton method !")

    # g'(x) = -sinx -3x^2
    first_diff_func = diff(self.func, self.x)


    iteration = 0
    t = initial
    while iteration < max_iter:
        iteration+=1
        if(np.abs(t-self.true)) < self.error: #1
            self.printHelper(iteration,1,t,np.abs(t-
self.true),iteration*10,10)
            break

            # x_{i+1} = x_{i} - g(x)/g'(x)
            t = t -
self.f(self.func,t)/self.f(first_diff_func,t) #9
```

Problem 1.3

- Total number of iteration: 6;
- The root found: 0.8654321018259392;

- Final Absolute Error: 4.189817406075047e-05;
- The estimated number of floating operation: 90, 15 operations(excluding comparisons) for each iteration.

Code Snippet

```
def secantMethod(self,x0,x1,max_iter=10000):
    """
        Implementation for secant method
        :param x0: initial root
        :param x1: initial root
        :param max_iter: To prevent the possible divergence in
the method
        :return: None
        """
    print("1.3 Performing secant method !")

    iteration = 0
    x0 = x0
    x1 = x1
    while iteration < max_iter:
        iteration+=1
        if(np.abs(x0-self.true)) < self.error: #1
            self.printHelper(iteration,1,x0,np.abs(x0-
self.true),iteration*15,15)
            break

            # x_{i+1} = x_{i} - ( (x_{i]-x_{i-1})f(xi) / (f(xi)-
f(x_{i-1})) )
            x0 = x1 - (x1-
x0)*self.f(self.func,x1)/(self.f(self.func,x1)-
self.f(self.func,x0)) #14
```

Problem 1.4

Based on $g(x) = cosx - x^3$, we have the following three ways for representation:

$x_n = cos(x_{n-1}) - x_{n-1}^3$ failed. Maximum iteration reached, the fixed point method diverges !

$x_n = (cos(x_{n-1}) - x_{n-1})^{(1/3)}$ failed. Maximum iteration reached, the fixed point method diverges !

$x_n = \frac{cos(x_{n-1})}{(x_{n-1}^2+1)}$ converged!

- Total number of iteration: 166;
- The fixed point found: 0.603566873265286;
- Final Absolute Error: 0.000046873265286l847;
- The estimated number of floating operation: 830, 5 operations(excluding comparisons) for each iteration.

Code Snippet

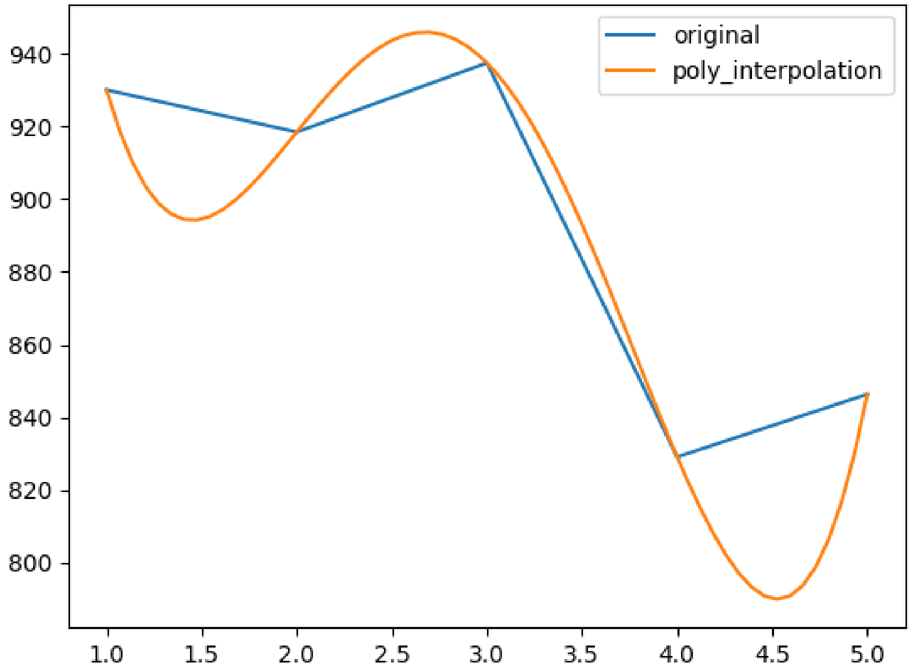
```
def fixedPointMethod(self,max_iter=200,initial=0,fixed_point =
0.60352):
    """
        The implementation of fixedPointMethod
        :param max_iter: To prevent the possible divergence in
the method
        :param initial: x0
        :param fixed_point: fc
        :return: None
        """
    print("1.4 Performing fixed point method !")

    iteration = 0
    x = initial

    x1 = symbols('x')
    alternative_func = real_root(cos(x1)-x1,3)

    x2 = symbols('x')
    alternative_func2 = cos(x2)/(x2**2+1)


    while iteration<max_iter:
        iteration+=1
        if(np.abs(x-fixed_point)) < self.error:
            self.printHelper(iteration,0,x,np.abs(x-
fixed_point),iteration*4,4)
            return
```

```
        x = self.f(self.func,x)

        print("x_{n} = cos(x_{n-1})-x_{n-1}^3 failed. Maximum
iteration reached, the fixed point method diverges !\n ")


        x = initial
        iteration = 0
        while iteration<max_iter:
            iteration+=1
            if(np.abs(x-fixed_point)) < self.error:
                self.printHelper(iteration,0,x,np.abs(x-
fixed_point),iteration*5,5)
                return

            x = alternative_func.evalf(subs={x1:x})

            print("x_{n} = (cos(x_{n-1})-x_{n-1})^(1/3) failed.
Maximum iteration reached, the fixed point method diverges !\n")


            x = initial
            iteration = 0
            while iteration < max_iter:
                iteration += 1
                if (np.abs(x - fixed_point)) < self.error: #1
                    print(
                        "x_{n} = cos(x_{n-1})/(x_{n-1}^2+1)
succeeded!\n")
                    self.printHelper(iteration, 0, x, np.abs(x -
fixed_point), iteration * 5, 5)
                    return

                x = alternative_func2.evalf(subs={x1: x}) #4

                print(
                    "x_{n} = cos(x_{n-1})/(x_{n-1}^2+1) failed.
Maximum iteration reached, the fixed point method diverges !\n")
```

Problem 1.5

The polynomial interpolation takes the form of:

$$P_4(x) = 17.1170833333336x^4 - 197.492500000003x^3 + 772.332916666677x^2 \\ -1202.90750000002x + 1540.95000000001$$



Code Snippet

```
def interpolation(self,data_point, order):
    """
        Implementation of polynomial interpolation
        :param data_point: The key-value pair form of data point
        :param order: The order of the polynomial interpolation
        :return: None
        """

    print("1.5 Performing interpolation !")

    A = self.construct_A(data_point,order)
    b = self.construct_b(data_point)
    x = linalg.solve(A,b)
    x0,func,formatted_func = self.formatPoly(order,x)
    print("The polynomial interpolation takes the form
of:\n{}".format(formatted_func))

    self.plotInterpolation(data_point,func,x0)
```

```
def plotInterpolation(self,data_point,func,x0):
    x_list = [pair[0] for pair in data_point]
    y_list = [pair[1] for pair in data_point]
    x_poly = np.linspace(1,5,60)
    y_poly = [float(func.evalf(subs={x0:x})) for x in x_poly]
    plt.plot(x_list,y_list,label="original")
    plt.plot(x_poly,y_poly,label="poly_interpolation")
    plt.legend()
    plt.show()
```