TMA4135 Matematikk 4D
Exercise 8

Odd André Owren

October 2019

1a. Table of divided differences:

x	y	$f(x_i, x_j)$	$f(x_i, x_j, x_k)$
-2	1		
		$\frac{1}{3}$	
1	2		$-\frac{1}{60}$
		$\frac{1}{5}$	
6	3		

This gives the following second-order interpolation formula:

$p(x) = 1 + (x+2) \cdot \frac{1}{3} + (x+2)(x-1) \cdot -\frac{1}{60} =$

$\underline{\underline{-\frac{1}{60}x^2 + \frac{19}{60}x + \frac{17}{10}}}$

1b. Updated table of divided differences:

x	y	$f(x_i, x_j)$	$f(x_i, x_j, x_k)$	$f(x_i, x_j, x_k, x_l)$
-2	1			
		$\frac{1}{3}$		
1	2		$-\frac{1}{60}$	
		$\frac{1}{5}$		$\frac{1}{315}$
6	3		$-\frac{4}{315}$	
		$\frac{2}{9}$		
$-\frac{3}{4}$	$\frac{3}{2}$			

Now, the new interpolation polynomial becomes the following:

$p(x) = -\frac{1}{60}x^2 + \frac{19}{60}x + \frac{17}{10} + (x+2)(x-1)(x-6) \cdot \frac{1}{315} =$

$\underline{\underline{\frac{1}{315}x^3 - \frac{41}{1260}x^2 + \frac{367}{1260} + \frac{117}{70}}}$

1c. Considering $f(x) = x^2 - 3$, we can see that
$f(1) = -2 \;\; f(2) = 1 \;\; f(3) = 6 \;\; f(\frac{3}{2}) = -\frac{3}{4}$

, thus we can use inverse interpolation and our results from 1a to make an approximation of f(x)=0.
We begin by estimating f(x)=0 with the first interpolation polynomial.

$x_{est} \approx p(0) = \frac{17}{10}$

This gives an error of $0 - f(x_{est}) = \underline{0.11}$.

Now looking at the second interpolation polynomial, from 1b, we get an approximation of f(x)=0 of:

$x_{est} \approx p(0) = \frac{117}{70}$

This gives an error of $0 - f(x_{est}) = \frac{1011}{4900}$, which is lower than previous, thus it is clear that we can edge closer to the real solution by interpolating more points.

1d. See figure 1.

```
In [10]: def f(x):
             return sqrt(x + 3)

         n = 2
         count=0
         x=linspace(-1,3,101)
         while n <= 16:
             xdata = linspace(1, 3, n + 1)
             ydata = [f(x) for x in xdata]

             F = divdiff(xdata, ydata)

             x2 = [0]
             p = newton_interpolation(F, xdata, x2)
             p2= newton_interpolation(F,xdata,x)

             subplot(221+count)
             plot(x, p2)
             plot(xdata, ydata, 'o')

             print(f'n = {f"{n}".rjust(2)}: x = {p[0]:.5f}, error: {str(f"{sqrt(3) - p[0]:.2e}").rjust(9)}')

             n *= 2
             count+=1

executed in 2.90s, finished 11:56:42 2019-10-29
n =  2: x = 1.74129, error: -9.24e-03
n =  4: x = 1.73270, error: -6.54e-04
n =  8: x = 1.73206, error: -6.12e-06
n = 16: x = 1.73205, error:  8.21e-07
```

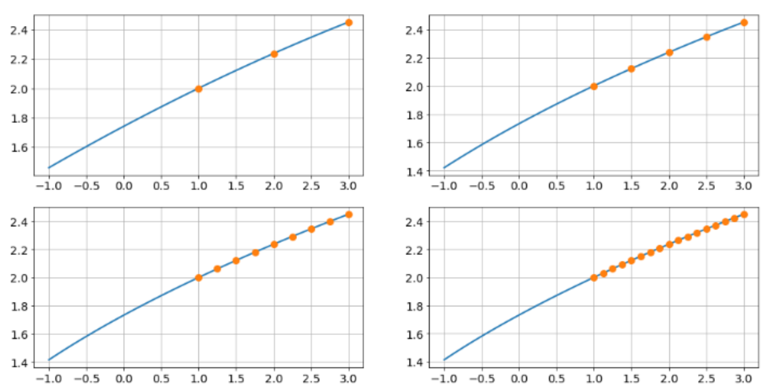

Figure 1: Solution to task 1d

3a. We have that $S_n = \frac{b-a}{6}(f(a)+4f(c)+f(b))$, where $c = \frac{a+b}{2}$. For 1 interval we then have:

$S_1(1,3) = \frac{3-1}{6}(e^{-1} + 4e^{-2} + e^{-3}) = \underline{0.31967}$

For 2 intervals we have:

$S_2(1,3) = \frac{2-1}{6}(e^{-1} + 4e^{-1.5} + e^{-2}) + \frac{3-2}{6}(e^{-2} + 4e^{-2.5} + e^{-3}) = \underline{0.31820}$

The actual errors are then:

$\int_1^3 e^{-x}dx - S_1(1,3) = \underline{1.578 \cdot 10^{-3}}$

$\int_1^3 e^{-x}dx - S_2(1,3) = \underline{1.076 \cdot 10^{-4}}$

For the error estimate we have that

$E_1(a,b) = I(a,b) - S_1(a,b) \approx \frac{16}{15}(S_2(a,b) - S_1(a,b)) = \xi_1$
and
$E_2(a,b) = I(a,b) - S_2(a,b) \approx \frac{1}{15}(S_2(a,b) - S_1(a,b)) = \xi_2$

Thus we have

$\xi_1 = \frac{16}{15}(S_2(1,3) - S_1(1,3)) = \underline{-1.56832 \cdot 10^{-3}}$

$\xi_2 = \frac{1}{15}(S_2(1,3) - S_1(1,3)) = \underline{-9.802 \cdot 10^{-4}}$

3b. We have that the formula for error in composite Simpson method is given by $E_m = -\frac{b-a}{180}h^4 f^{(4)}(\xi)$ where $h = \frac{b-a}{2m}$. We know that $f^{(4)}(x) = e^{-x}$, such that $f^{(4)}(x) \leq e^{-1}$. Inserting this into the formula and solving with regard to m gives us:

$|-\frac{(b-a)^5}{180\cdot(2m)^4} \cdot e^{-1}| \leq 10^{-8} \Rightarrow \frac{e^{-1}}{90\cdot 10^{-8}} \leq m^4 \Rightarrow m \geq \sqrt[4]{\frac{e^{-1}}{90\cdot 10^{-8}}} \Rightarrow \underline{m \geq 25.28}$

Thus, m must be 26 for the error to be minimum 10^{-8}

3c. See figure 2.

In [11]:
```python
def simpson(f, a, b, m=10):
    n = 2*m
    x_noder = linspace(a, b, n+1)
    h = (b-a)/n
    S1 = f(x_noder[0]) + f(x_noder[n])
    S2 = sum(f(x_noder[1:n:2]))
    S3 = sum(f(x_noder[2:n-1:2]))
    S = h*(S1 + 4*S2 + 2*S3)/3
    return S

def f(x):
    return exp(-x)

a, b = 1, 3
m = 22

i = exp(-1) - exp(-3)
s = simpson(f, a, b, m=m)

print(f'Estimate: {s}, actual: {i}, error: {i - s:.2e}')
```
executed in 15ms, finished 11:57:58 2019-10-29

```
Estimate: 0.3180923803455209, actual: 0.3180923728035784, error: -7.54e-09
```

Figure 2: Solution to task 3c

4a. First, we will have to find the nodes for m=3:

$L_3(t) = \frac{d^3}{dt^3}(t^2-1)^3 = \frac{d^3}{dt^3}(t^6 - 3t^4 + 3t^2 - 1) = 120t^3 - 72t$

Solving for $L_3(t) = 0$ gives us $t_0 = 0 \;\; t_1 = -\sqrt{\frac{3}{5}} \;\; t_2 = \sqrt{\frac{3}{5}}$

Now, we need to create the cardinals for the nodes:

$\ell_0(t) = \frac{(t-0)(t-\sqrt{\frac{3}{5}})}{(-\sqrt{\frac{3}{5}}-0)(-\sqrt{\frac{3}{5}}-\sqrt{\frac{3}{5}})} = \frac{5t^2 - g\cdot\sqrt{\frac{3}{5}}t}{6} = \frac{5}{6}t^2 - \frac{\sqrt{15}}{6}t$

$\ell_1(t) = \frac{(t-\sqrt{\frac{3}{5}})(t+\sqrt{\frac{3}{5}})}{(0+\sqrt{\frac{3}{5}})(0-\sqrt{\frac{3}{5}})} = -\frac{5}{3}t^2 + 1$

$\ell_2(t) = \frac{(t+\sqrt{\frac{3}{5}})(t-0)}{(\sqrt{\frac{3}{5}}+\sqrt{\frac{3}{5}})(\sqrt{\frac{3}{5}}-0)} = \frac{5}{6}t^2 + \frac{\sqrt{15}}{6}t$

Next, we find the weights for the quadrature, which is found by finding the integral of the cardinals on [-1,1].

$\omega_0 = \int_{-1}^{1} \ell_0(t)dt = \frac{5}{18}t^3 - \frac{\sqrt{15}}{12}t^2|_{-1}^{1} = \frac{5}{9}$

$\omega_1 = \int_{-1}^{1} \ell_1(t)dt = -\frac{5}{6}t^3 + t|_{-1}^{1} = \frac{8}{9}$

$\omega_2 = \int_{-1}^{1} \ell_2(t)dt = \frac{5}{18}t^3 + \frac{\sqrt{15}}{12}t^2|_{-1}^{1} = \frac{5}{9}$

Now, the Gauss-Legendre quadrature is given by $\int_{-1}^{1} f(t)dt \approx \int_{-1}^{1} p_{m-1}(t)dt = \Sigma_{n=0}^{m-1}\omega_n f(t_n)$. This gives us:

$\int_{-1}^{1} f(t)dt \approx \underline{\frac{1}{9}(5f(t_0) + 8f(t_1) + 5f(t_2))}$

4b. To confirm the degree of precision of the quadrature, we will have to confirm that the quadrature is correct for $\int_{-1}^{1} t^n dt$, n=0,1,2,...,2m-1.

n	$\int_{-1}^{1} t^n dt$	$\int_{-1}^{1} p_2(t)dt$
0	2	2
1	0	0
2	$-\frac{2}{3}$	$-\frac{2}{3}$
3	0	0
4	$\frac{2}{5}$	$\frac{2}{5}$
5	0	0
6	$\frac{2}{7}$	$\frac{6}{25}$

We can see from the table above that $\int_{-1}^{1} t^n dt = \int_{-1}^{1} p_2(t)dt$ for $0 \leq n \leq 5$, thus we have that the degree of precision is 5.

4c. Here we start by selecting $h = \frac{b-a}{2}$, $c = \frac{b+a}{2}$, $x = ht + c$ and $dx = hdt$. This then makes it possible to generalize the Gauss-Legendre quadrature.

$\int_a^b f(x)dx = h\int_{-1}^{1} f(ht+c)dt \approx \frac{h}{9}(5f(ht_0+c) + 8f(ht_1+c) + 5f(ht_2+c)) = \underline{\frac{h}{9}(5f(ht_0+c) + 8f(c) + 5f(ht_2+c))}$

Now, when approximating $\int_1^3 e^{-x}dx$ we have that h=1 and c=2. This gives us the following approximation and error:

$\int_1^3 e^{-x}dx \approx \frac{1}{9}(5e^{-(-\sqrt{\frac{3}{5}}+2)} + 8e^{-2} + 5e^{-(\sqrt{\frac{3}{5}}+2)}) = \underline{0.31808}$

$E = \int_1^3 e^{-x}dx - 0.31808 = -e^{-3} + e^{-1} - 0.31808 = 0.31809 - 0.31808 = 10^{-5}$

4d. Here we are given that $E(a,b) = \int_a^b f(x)dx - Q(a,b) = \frac{(b-a)^7}{2016000} f^{(6)}(\eta)$. Using this, we can get:

$\int_a^b f(x)dx - Q_m(a,b) = \Sigma_{i=0}^{m-1}(\int_i^{i+1} f(x)dx - Q(x_i, x_{i+1}) = \Sigma_{i=0}^{m-1} \frac{(x_{i+1}-x_i)^7}{2016000} f^{(6)}(\eta)$

Since we have a uniform distribution on the interval [a,b]; $h = x_{i+1} - x_i = \frac{b-a}{m}$, such that

$\int_a^b f(x)dx - Q_m(a,b) = \Sigma_{i=0}^{m-1} \frac{(x_{i+1}-x_i)^7}{2016000} f^{(6)}(\eta) = \frac{h^7}{2016000} m f^{(6)}(\eta) = \underline{\frac{(b-a)h^6}{31500 \cdot 2^6} f^{(6)}(\eta)}$

Using this result to find m such that the error is less than 10^{-8}:

$\frac{(b-a)h^6}{31500 \cdot 2^6} f^{(6)}(\eta) \leq 10^{-8} \Rightarrow m \geq \sqrt[6]{\frac{(3-1)^7 e^{-1}}{2016000 \cdot 10^{-8}}} = \underline{3.64}$

Thus, we need m=4 to get a error less than 10^{-8}, which is considerably lower than with Simpson's method.

4e. By following the same procedure as the error estimate in 3a, we have:

$E_1(a,b) = \frac{(b-a)^7}{2016000} f^{(6)}(\xi) = CH^7$, where $C = \frac{f^{(6)}(\xi)}{315000 \cdot 2^6}$ and $H = b - a$

$E_1(a,b) = I(a,b) - Q_1(a,b) \approx CH^7$

$E_2(a,b) = I(a,b) - Q_2(a,b) = I(a,c) - Q_1(a,c) + I(b,c) - Q_1(b,c) \approx 2C(\frac{H}{2})^7 = \frac{1}{64}CH^7$

By taking $E_2 - E_1$ we have:

$CH7 \approx \frac{64}{63}(Q_2(a,b) - Q_1(a,b))$

and

$\xi_1 = \frac{64}{63}(Q_2(a,b) - Q_1(a,b))$

$\xi_2 = \frac{1}{63}(Q_2(a,b) - Q_1(a,b))$

Compared to the estimate for Simpson's method, it is clear to see that we for each degree of precision get about 4 times more precise by using Gauss-Legendre quadrature.

4f. See figure 3.

```
In [13]:
def gauss(f, a, b):
    c = (a + b) / 2
    h = (a - b) / 2
    return h * (5 * f(-sqrt(3 / 5) * h + c) \
                + 8 * f(c) \
                + 5 * f(sqrt(3 / 5) * h + c)) / 9

def gauss_basic(f, a, b):
    S1 = gauss(f, a, b)
    c = (a + b) / 2
    S2 = gauss(f, a, c) + gauss(f, c, b)
    error_estimate = (S2 - S1) / 63
    return S2, error_estimate

def f(n=1):
    def g(x):
        return x ** n
    return g

for n in range(0, 7):
    s2, error = gauss_basic(f(n), -1, 1)
    print(f'n={n}: value={f"{s2:.4f}".rjust(8)}, error={f"{error:.4f}".rjust(9)}')

executed in 13ms, finished 11:59:49 2019-10-29

n=0: value= -2.0000, error=   0.0000
n=1: value=  0.0000, error=   0.0000
n=2: value= -0.6667, error=   0.0000
n=3: value=  0.0000, error=   0.0000
n=4: value= -0.4000, error=   0.0000
n=5: value=  0.0000, error=   0.0000
n=6: value= -0.2850, error=  -0.0007
```

Figure 3: Solution to task 4f

4g. See figure 4. We can see that in most cases the error is lower than the tolerance, except for tolerance 10^{-3} in function ii) and iii)

```
In [33]: def gauss_adaptive(f, a, b, tol, level=0, max_level=15, silent=True):

             Q, error_estimate = gauss_basic(f, a, b)

             if not silent:
                 # -----------------------------------------------
                 x = linspace(a, b, 101)
                 plot(x, f(x), [a, b], [f(a), f(b)], '.r')
                 title('The integrand and the subintervals')
                 # -----------------------------------------------

             if level >= max_level:
                 print('Warning: Maximum number of levels used.')
                 return Q

             if abs(error_estimate) < tol:         # Accept the result, and return
                 result =  Q+error_estimate
             else:
                 # Divide the interval in two, and apply the algorithm to each interval.
                 c = 0.5*(b+a)
                 result_left  = gauss_adaptive(f, a, c, tol = 0.5*tol, level = level+1,silent=True)
                 result_right = gauss_adaptive(f, c, b, tol = 0.5*tol, level = level+1,silent=True)
                 result = result_left + result_right
             return result

         def f1(x):
             return exp(-x)

         def f2(x):
             return 1 / (1 + 16 * x ** 2)

         def f3(x):
             return 1 / ((x - 0.3) ** 2 + 0.01) + 1 / ((x - 0.9) ** 2 + 0.04)

         executed in 88ms, finished 12:19:06 2019-10-29
```

```
In [34]: testcases = [
             {
                 "a": 1,
                 "b": 3,
                 "f": f1,
                 "exact": exp(-1) - exp(-3),
             },
             {
                 "a": 0,
                 "b": 5,
                 "f": f2,
                 "exact": arctan(20) / 4,
             },
             {
                 "a": 0,
                 "b": 2,
                 "f": f3,
                 "exact": 41.326213804391148551,
             },
         ]

         tolerances = [1.e-3, 1.e-6, 1.e-9]

         for i, case in enumerate(testcases, start=1):
             print(f'\nFunction {i}:\n')
             for tol in tolerances:

                 print(f'With tolerance {tol}:')

                 result = -gauss_adaptive(case['f'], case['a'], case['b'], tol=tol, silent=True)

                 print(f'Numerical solution = {result:8f}, exact solution = {case["exact"]:8f}')

                 err = case['exact'] - result
                 print(f'error = {abs(err):.3e}\n')

         executed in 23ms, finished 12:19:08 2019-10-29
```

```
Function 1:

With tolerance 0.001:
Numerical solution = 0.318092, exact solution = 0.318092
error = 1.436e-08

With tolerance 1e-06:
Numerical solution = 0.318092, exact solution = 0.318092
error = 1.436e-08

With tolerance 1e-09:
Numerical solution = 0.318092, exact solution = 0.318092
error = 2.476e-13


Function 2:

With tolerance 0.001:
Numerical solution = 0.394190, exact solution = 0.380209
error = 1.398e-02

With tolerance 1e-06:
Numerical solution = 0.380209, exact solution = 0.380209
error = 1.045e-07

With tolerance 1e-09:
Numerical solution = 0.380209, exact solution = 0.380209
error = 1.688e-11


Function 3:

With tolerance 0.001:
Numerical solution = 41.335690, exact solution = 41.326214
error = 9.476e-03

With tolerance 1e-06:
Numerical solution = 41.326214, exact solution = 41.326214
error = 1.388e-07

With tolerance 1e-09:
Numerical solution = 41.326214, exact solution = 41.326214
error = 1.990e-13
```


Figure 4: Solution to task 4g