Course Project

Q.1:

Solution:

(a) Use Composite Trapezoidal Rule:

$$h = \frac{b-a}{n} \quad , \quad r_j = a + jh \tag{1}$$

$$f(r) = \rho v 2\pi r \tag{2}$$

$$\int_0^R f(r)\mathrm{d}r = \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(r_j) + f(b)\right] \tag{3}$$

(b) Use Composite Simpson's Rule:

$$\int_0^R f(r)\mathrm{d}r = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{(n/2)-1} f(r_{2j}) + 4\sum_{j=1}^{n/2} f(r_{2j-1}) + f(b)\right] \tag{4}$$

(c) Select a method of your choice that you think is more accurate than the two methods above:

Composite Simpson Rule is more accurate because the remainder term is $o(h^4)$ while that of the trapezoidal method is $o(h^2)$.

Python Code Output:

Composite Trapezoidal Rule= 0.1790632414322496 kg/s
Composite Simpson Rule= 0.18621350631182 kg/s

Q.2 :

Solution:

$$C = 1.83; t_0 = 1, k(t_0) = 1, \epsilon(t_0) = 0.2176 \tag{5}$$

(a) Use the Euler method:

$$\begin{cases} k(t_{i+1}) = k(t_i) + h(-\epsilon(t_i)) \\ \epsilon(t_{i+1}) = \epsilon(t_i) + h(-C\frac{\epsilon(t_i)^2}{k(t_i)}) \end{cases} \tag{6}$$

(b) Use the modified Euler method:

$$\begin{cases} \bar{k}(t_{i+1}) = k(t_i) + h(-\epsilon(t_i)) & (7) \\ \bar{\epsilon}(t_{i+1}) = \epsilon(t_i) + h(-C\dfrac{\epsilon(t_i)^2}{k(t_i)}) & (8) \\ k(t_{i+1}) = k(t_i) + \dfrac{h}{2}(-\epsilon(t_i) - \bar{\epsilon}(t_{i+1})) & (9) \\ \epsilon(t_{i+1}) = \epsilon(t_i) + \dfrac{h}{2}\left(-C\dfrac{\epsilon(t_i)^2}{k(t_i)} - C\dfrac{\bar{\epsilon}(t_{i+1})^2}{\bar{k}(t_{i+1})}\right) & (10) \end{cases}$$

(c) Use the 4th order Runge-Kutta method:

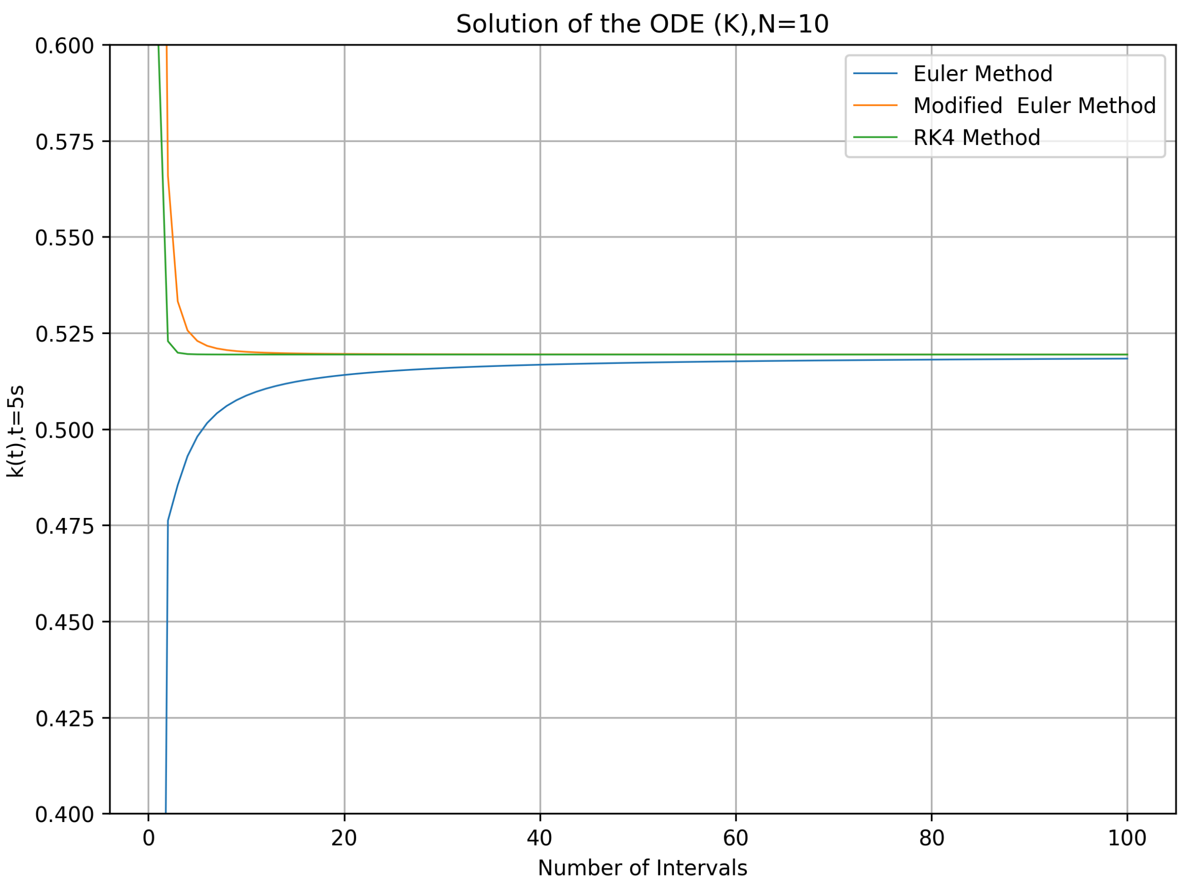
$$
\begin{cases}
K_{k,1} = -\epsilon & (11) \\
K_{\epsilon,1} = -C\dfrac{\epsilon^2}{k} & (12) \\
K_{k,2} = -\epsilon + h\dfrac{K_{k,1}}{2} & (13) \\
K_{\epsilon,2} = -C\dfrac{\epsilon^2}{k} + h\dfrac{K_{\epsilon,1}}{2} & (14) \\
K_{k,3} = -\epsilon + h\dfrac{K_{k,2}}{2} & (15) \\
K_{\epsilon,3} = -C\dfrac{\epsilon^2}{k} + h\dfrac{K_{\epsilon,2}}{2} & (16) \\
K_{k,4} = -\epsilon + hK_{k,3} & (17) \\
K_{\epsilon,4} = -C\dfrac{\epsilon^2}{k} + hK_{\epsilon,3} & (18)
\end{cases}
$$

Comment:

1. RK4 Method converges the fastest following the Modified Euler Method, the Euler Method converges the slowest.
2. RK4 Method and Modified Euler Method has almost the same accuracy,while the Euler Method has the worst accuracy.

Python Code Output:

Results of k at $t = 0.5s$,with different number of intervals dividing [1,5]



	Euler method	modified Euler method	Runge-Kutta 4 method
$k(t), t = 5; N = 40$	0.51674	0.51943	0.51939

Q.3:

Solution:

Let $y_+ = \frac{u_\tau y}{\nu}, U_+ = \frac{U}{u_\tau}$,

$$f(u_\tau) = -y_+ + U_+ + e^{-\kappa B}\left[e^{\kappa U_+} - 1 - \kappa U_+ - \frac{1}{2!}(\kappa U_+)^2 - \frac{1}{3!}(\kappa U_+)^3 - \frac{1}{4!}(\kappa U_+)^4\right] = 0 \quad (19)$$

$$\tau_{wall} = {u_\tau}^2 \rho \quad (20)$$

(a) Use bisection method

Since $f(0.1) = 3.0524259021078516e + 36 > 0, f(2) = -1318.893031144245 < 0$, let $a = 0.1, b = 2, mid = \frac{b+a}{2}$, if $f(a)f(mid) < 0$, $b = mid$, else $a = mid$, repeat until $toler = |b - a| < 10^{-3}$

(b) Use Newton's method

Take initial root approximation $p_0 = 1$, use $p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$ to find the approximation root. $f'(x_0) = \frac{f(x_0+h)-f(x_0)}{h}, h = 0.0001$

(c) Try to find a fixed point iteration formula that converges

$$\begin{gathered} f(U_+) = -y_+ + U_+ + e^{-\kappa B}\left[e^{\kappa U_+} - 1 - \kappa U_+ - \frac{1}{2!}(\kappa U_+)^2 - \frac{1}{3!}(\kappa U_+)^3 - \frac{1}{4!}(\kappa U_+)^4\right] = 0 \\ U_+ = U_+ - \frac{f(U_+)}{U_+^4} \end{gathered} \quad (21)$$

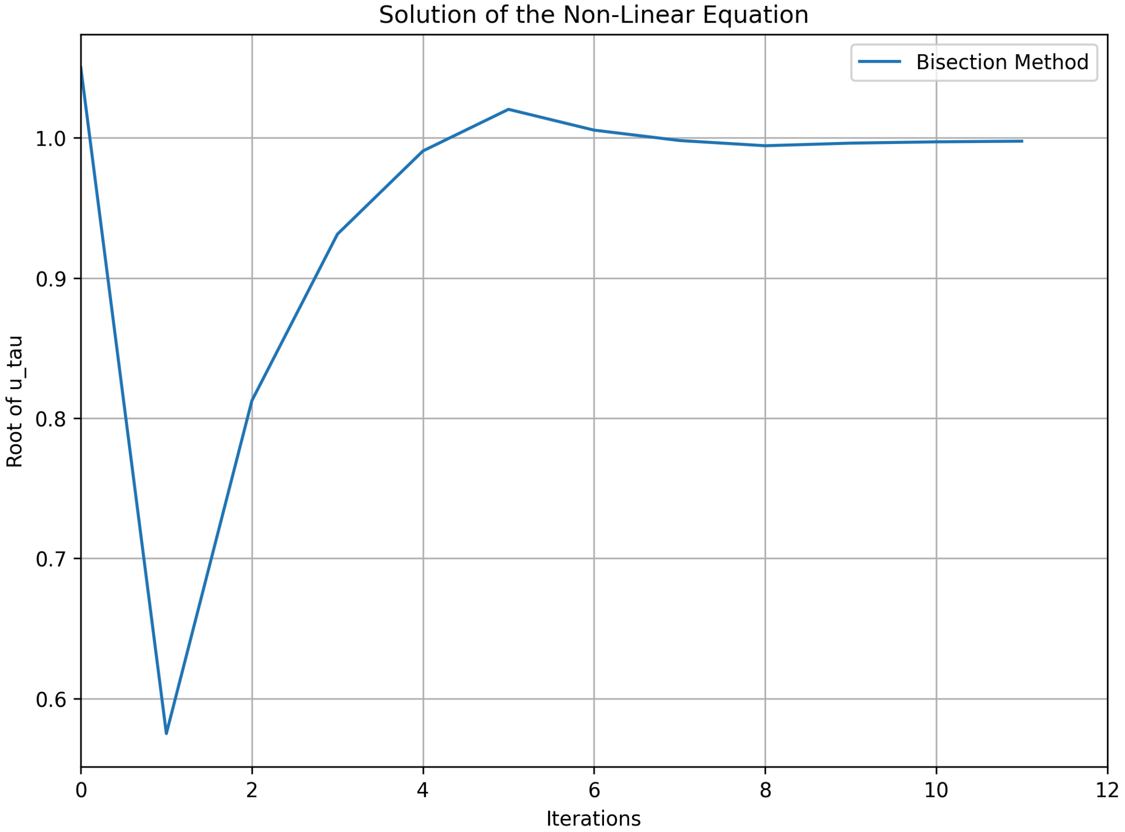
is a fixed point iteration formula that converges.

Python Code Output:

1. Output of Bisection Method

   Root of u_tau= **0.9975830078125001**

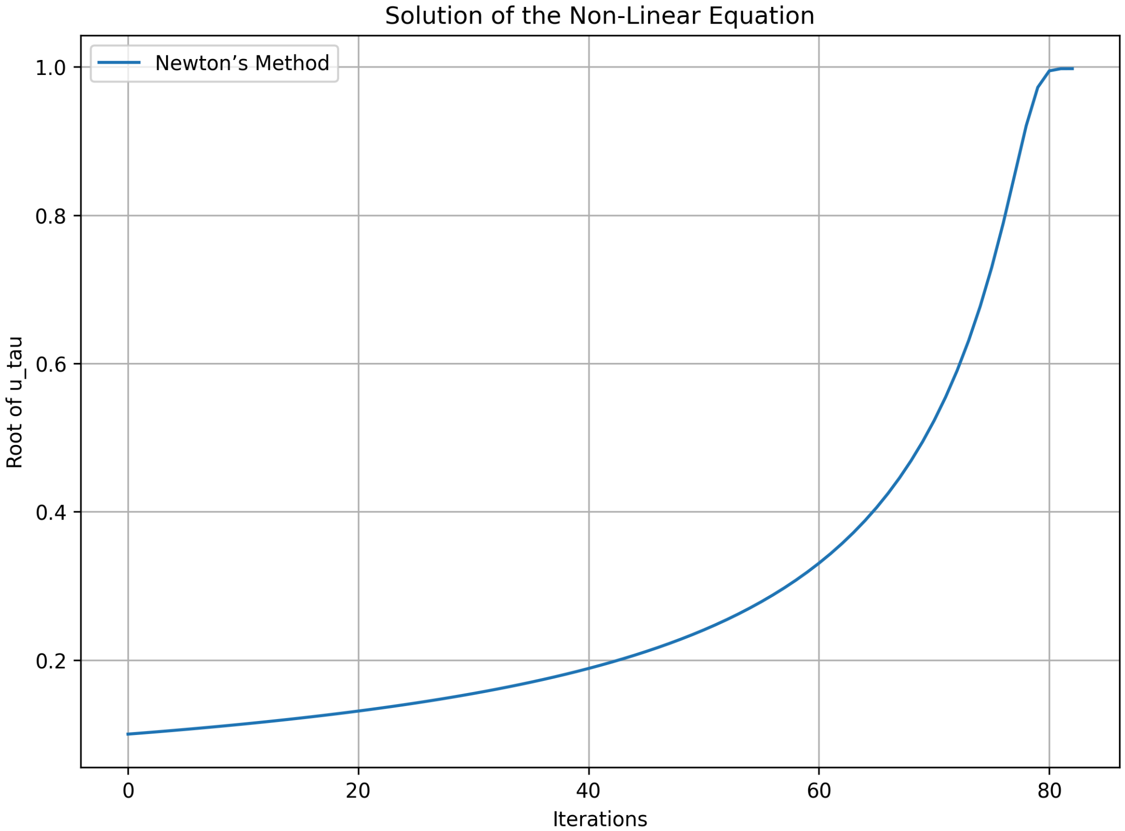
   Wall Shear Stress(bisection method)= **1.2439648218452932**



2. Output of the Newton's Method

   Root of u_tau= **0.9976727452597954**

   Wall Shear Stress(Newton's method)= **1.2441886332927707**



3. Output of a fixed point iteration that converges
   Wall Shear Stress(Fixed Point Iteration)= **1.3218985298910881**
   Iterations= **321**
   Tolerance= **0.0009987081191376035**

Q.4:

Solution:

First use the Householder transformation transfer matrix A into a tridiagonal matrix and than use the Gram-Schmidt process to decompose A into QR,than let A=RQ, do the QR decomposition again until the sum of the non-diagonal element of A is no bigger than 1e-10.Than the diagonal elements are the eigenvalue of A.

Python Code Output:

Eigenvalues: [132.62787533 52.4423   -11.54113078 -3.52904455]
Number of positive eigenvalues: 2
Number of negative eigenvalues: 2

Q.5:

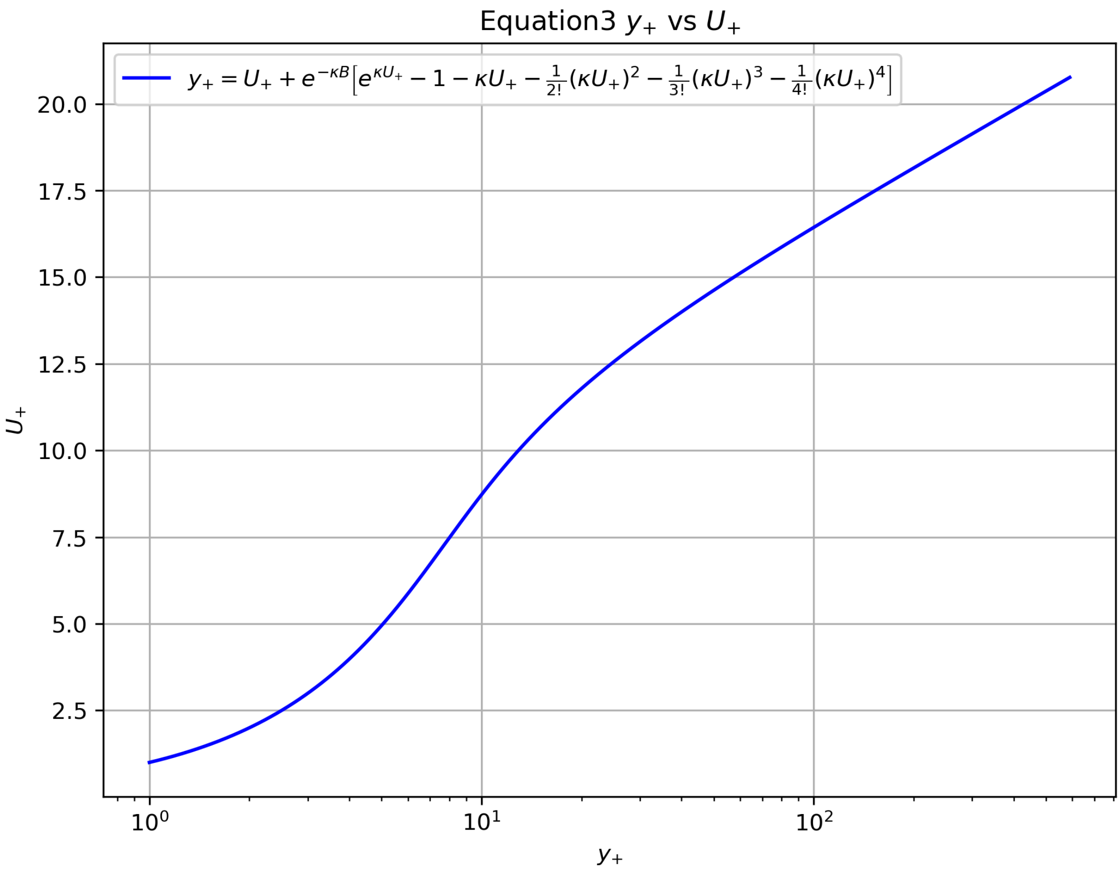
Solution:

$$y_+ = U_+ + e^{-\kappa B}\left[e^{\kappa U_+} - 1 - \kappa U_+ - \frac{1}{2!}(\kappa U_+)^2 - \frac{1}{3!}(\kappa U_+)^3 - \frac{1}{4!}(\kappa U_+)^4\right] \tag{22}$$

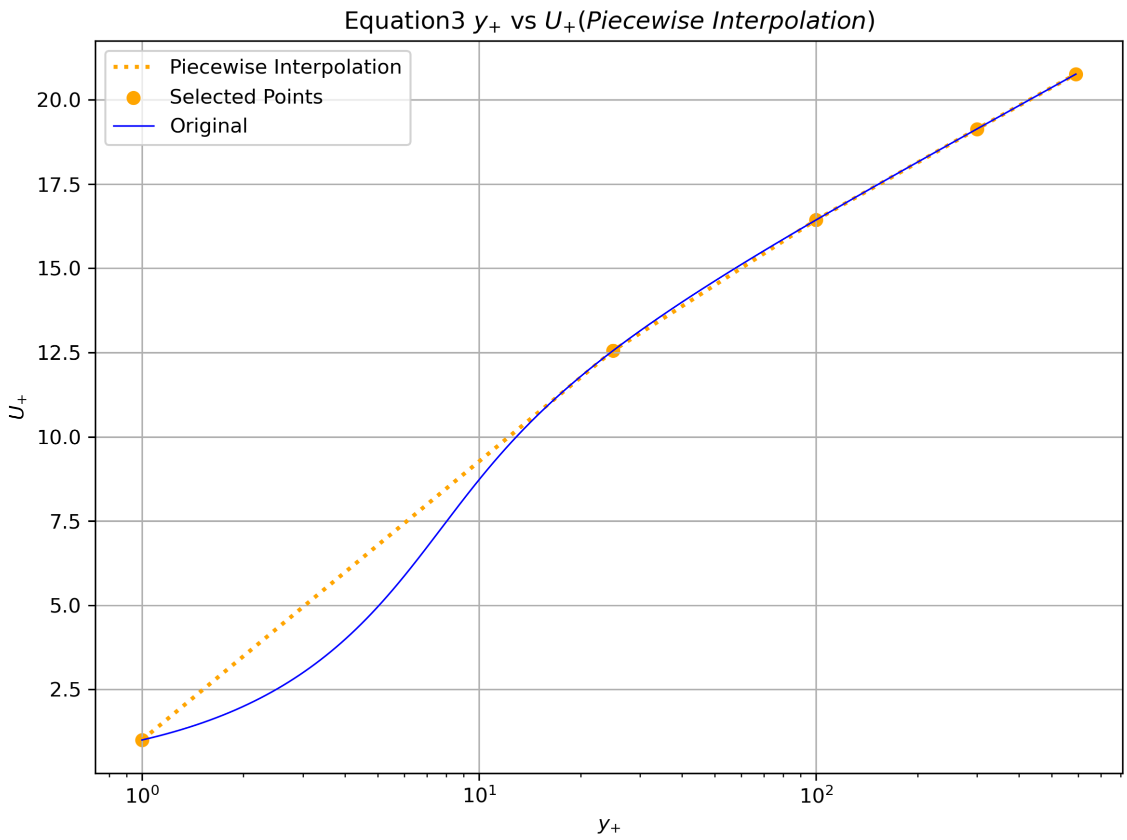
Use the fsolve function to calculate the value of U_+ at $y_+ = 1, 590$.Use these value as an interval to plot equation 22

Python Code Output:

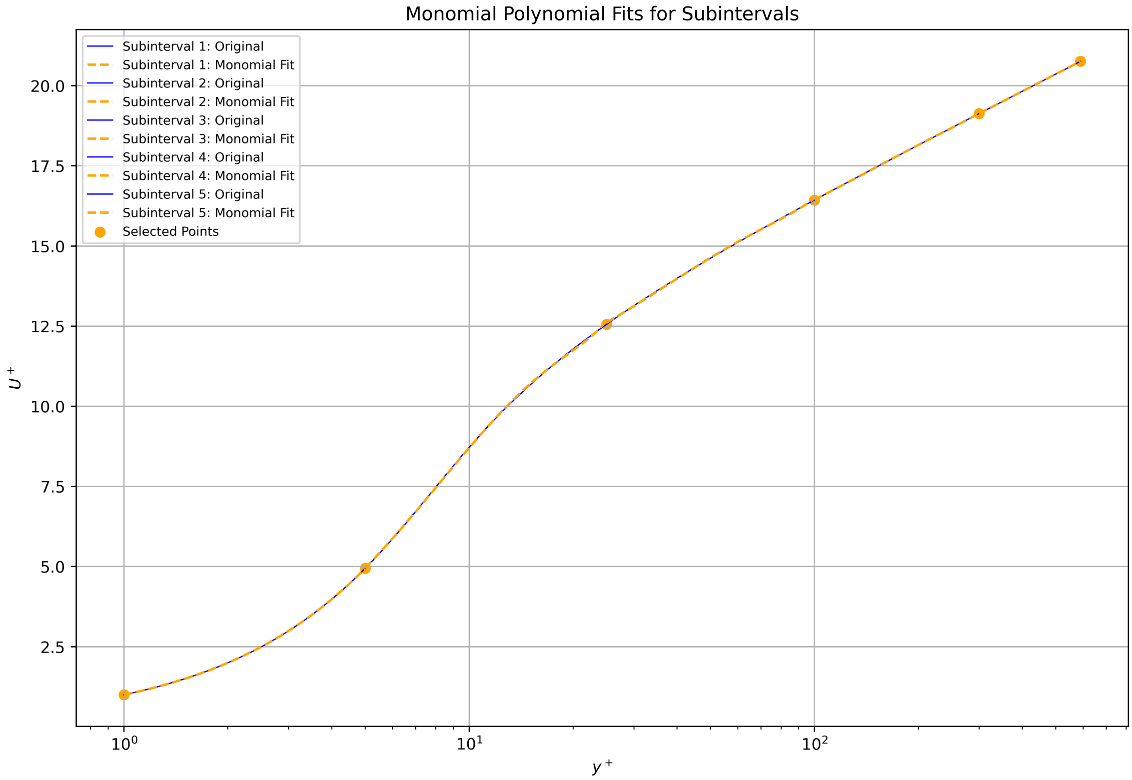
1. Draw the y_+ vs U_+,Plotting the y_+ range using the log10 scale.



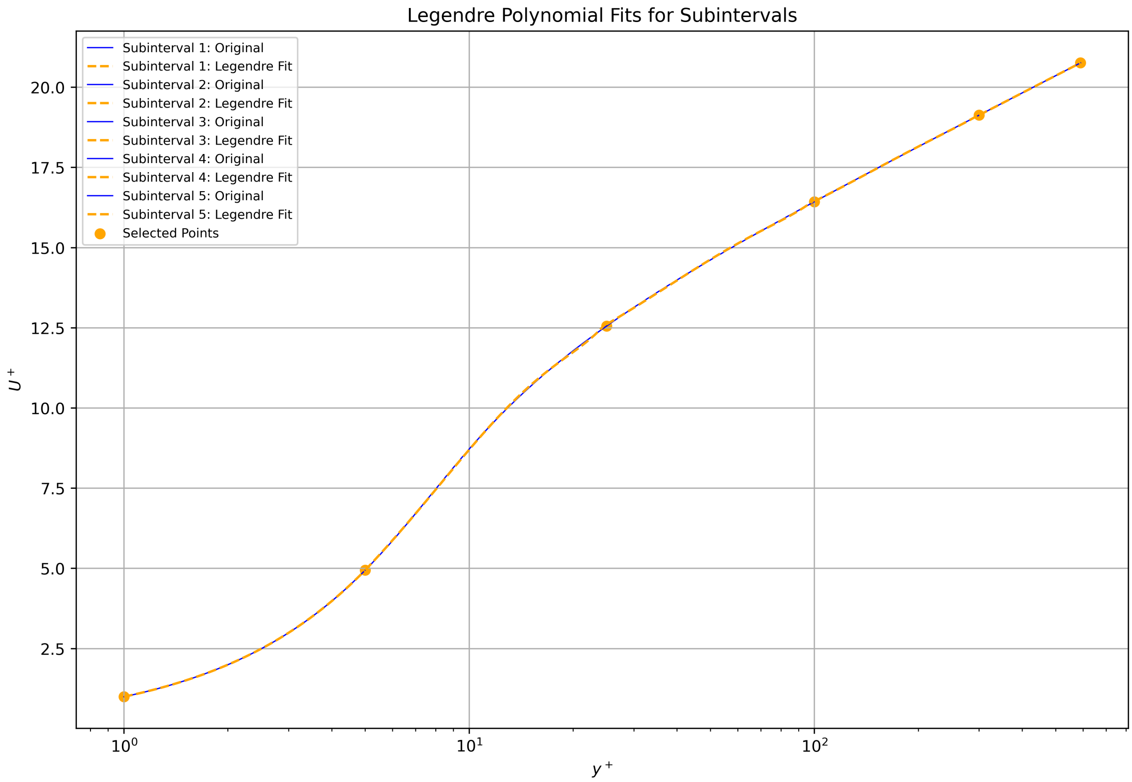
2. Construct piecewise linear interpolation, and draw on a figure:



3. For each subinterval you choose, use the Monomial polynomials up to order 3 to construct least-squares approximation and plot the answers in a figure: The interval I choose is [1, 5, 25,100,300, 590]



4. Repeat the previous step by using the Legendre polynomials up to order 3:



Q.6:

Solution:

(a) Use N equally spaced points in [−1,1] to interpolate the Runge function.

Use Newton interpolation method to do the interpolation, use N point to divide the interval.

To obtain the divided-difference coefficients of the interpolatory polynomial P on the $(n+1)$ distinct numbers $x_0, x_1, \ldots, x_n$ for the function f:

INPUT numbers $x_0, x_1, \ldots, x_n$; values $f(x_0), f(x_1), \ldots, f(x_n)$ as $F_{0,0}, F_{1,0}, \ldots, F_{n,0}$.

OUTPUT the numbers $F_{0,0}, F_{1,1}, \ldots, F_{n,n}$ where

$$P_n(x) = F_{0,0} + \sum_{i=1}^{n} F_{i,i} \prod_{j=0}^{i-1}(x - x_j). \quad (F_{i,i} \text{ is } f[x_0, x_1, \ldots, x_i].)$$

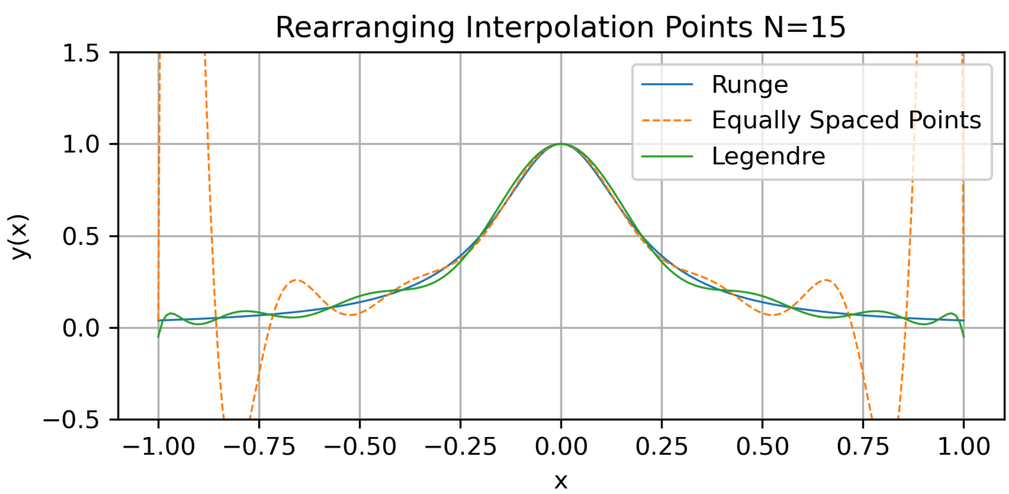
Step 1 For $i = 1, 2, \ldots, n$
For $j = 1, 2, \ldots, i$
set $F_{i,j} = \dfrac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$. $\quad (F_{i,j} = f[x_{i-j}, \ldots, x_i].)$

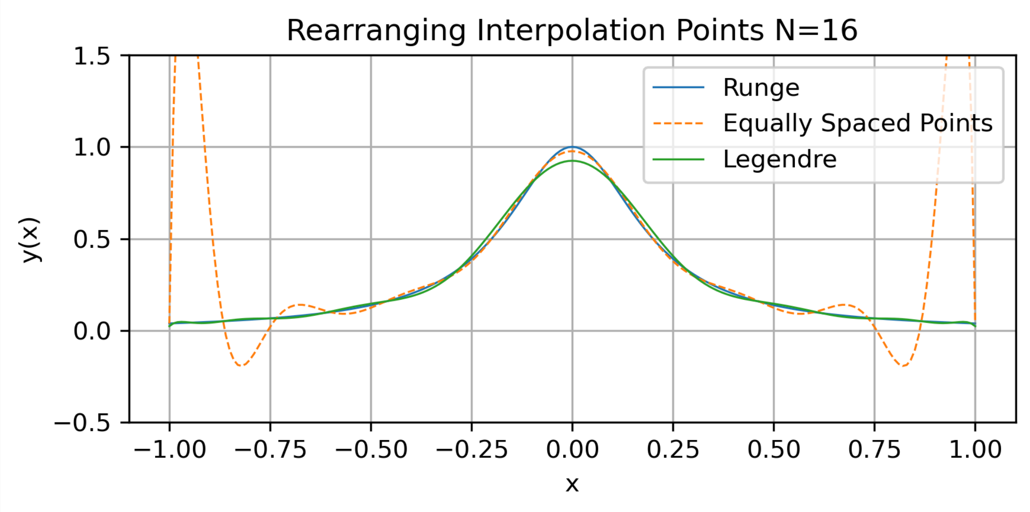
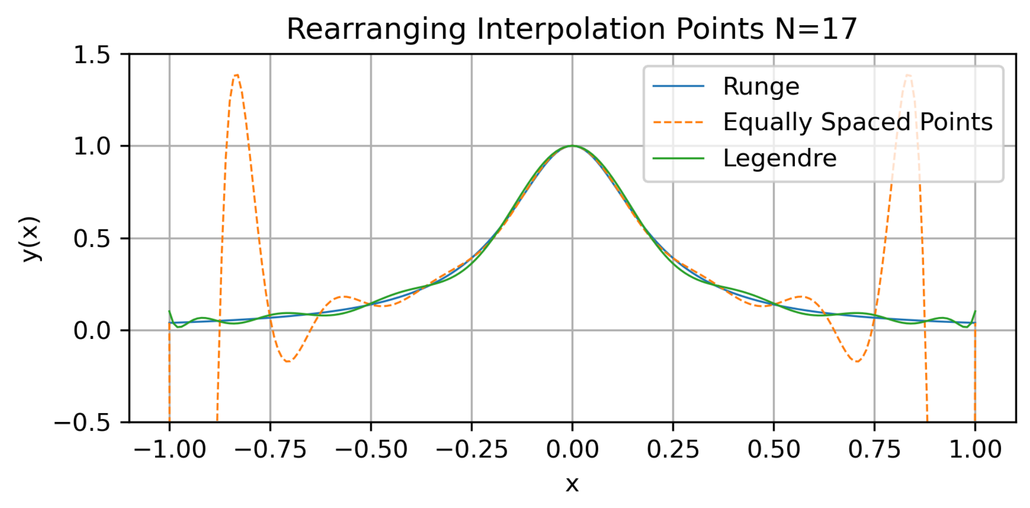
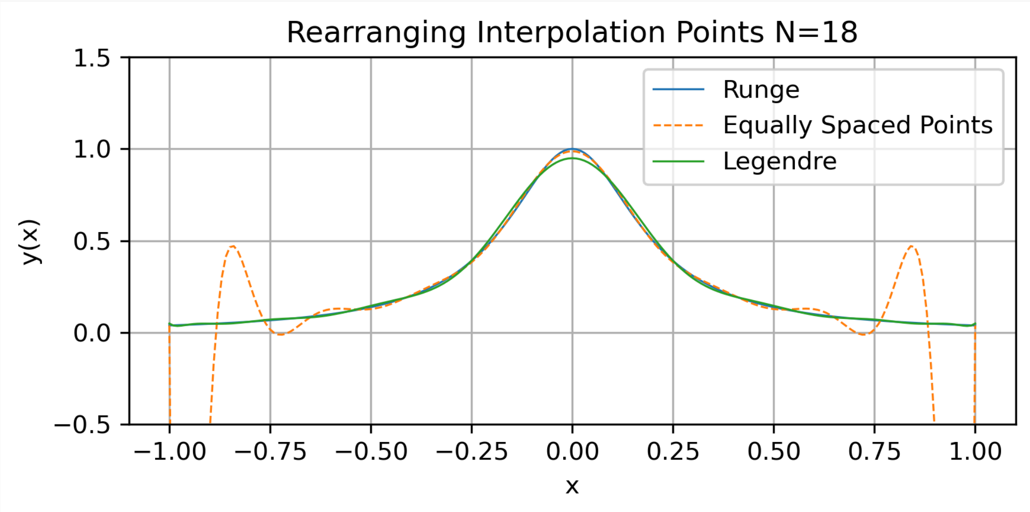
Step 2 OUTPUT $(F_{0,0}, F_{1,1}, \ldots, F_{n,n})$;
STOP.

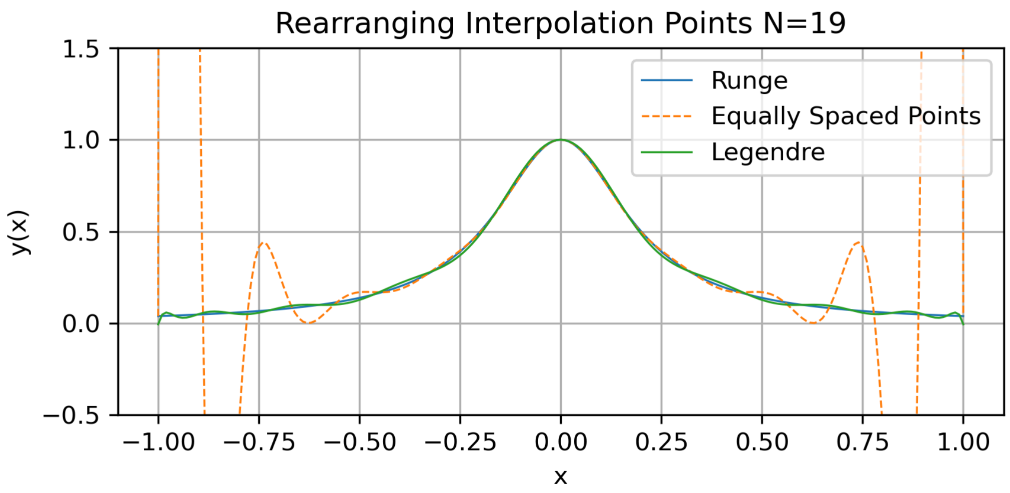
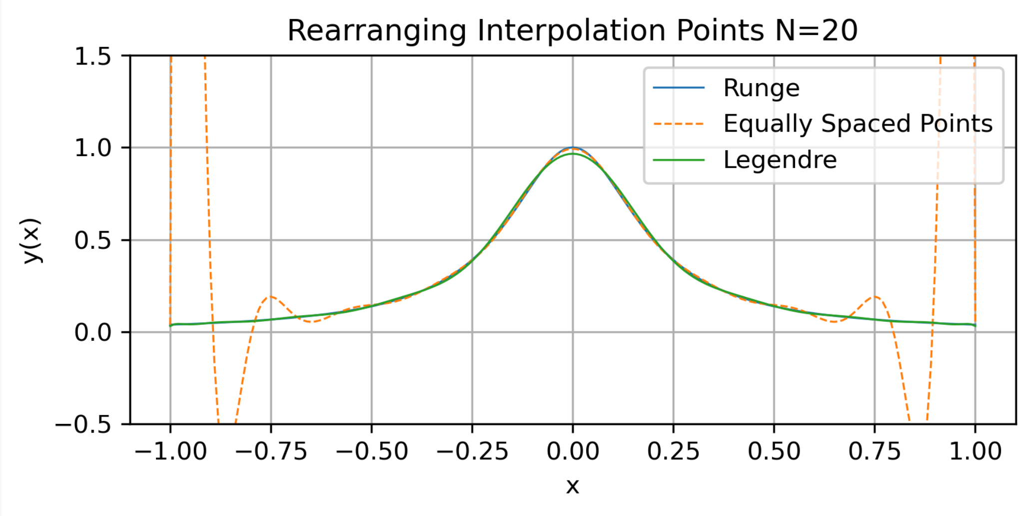
(b) Use roots of Legendre polynomial of degree (N−1), to interpolate the Runge function

Use scipy and numpy to compute the roots of Legendre polynomial of degree (N−1) and do (a) again.

(c) Plot

N	(c)
N=15	

N	(c)
N=16	
N=17	
N=18	

<table>
<tr><th>N</th><th>(c)</th></tr>
<tr><td>N=19</td><td>
</td></tr>
<tr><td>N=20</td><td>
</td></tr>
</table>

(d) Use least square approximation to approximate the Runge function

Since the Legendre polynomials are orthogonal polynomials on the interval [-1, 1] with respect to the power function 1, which constitute a set of standard orthogonal bases of the polynomial space, the least squares approximation of the Runge function can be obtained by projecting the Runge function onto this set of bases.

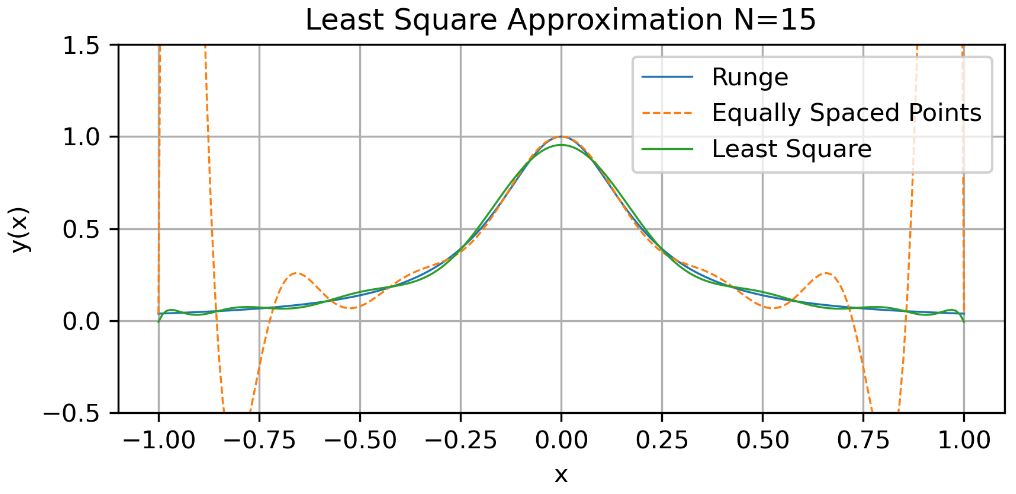
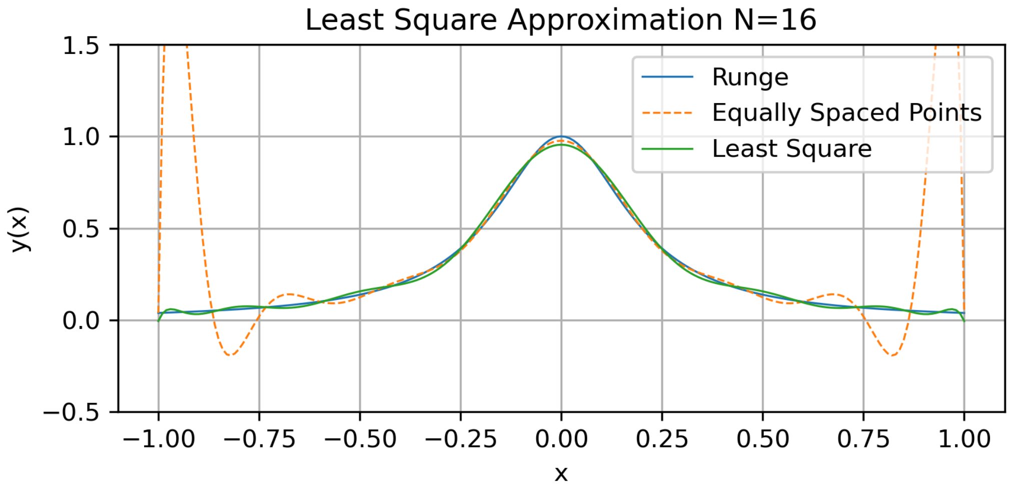
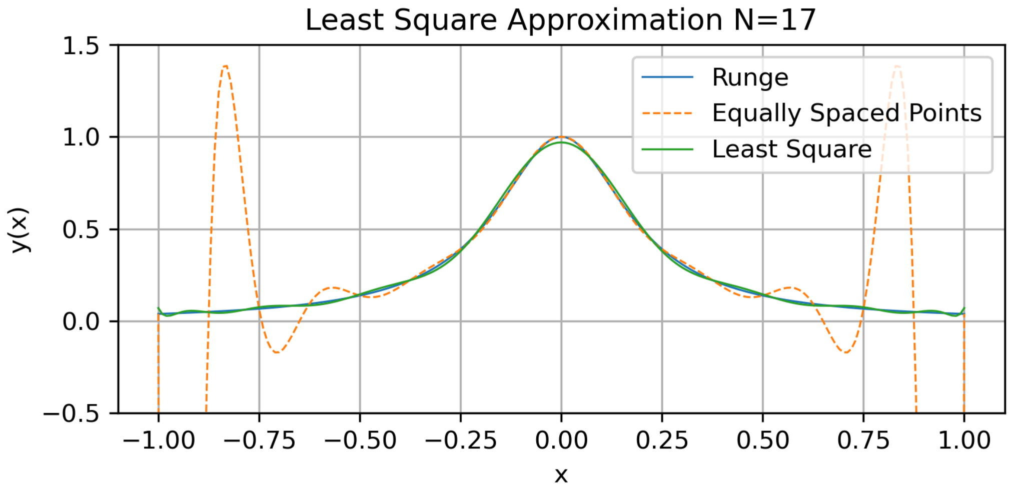
P_i is i-th order Legendre polynomials.$R(x) = \frac{1}{1+25x^2}$.The least square approximation of $R(x)$:

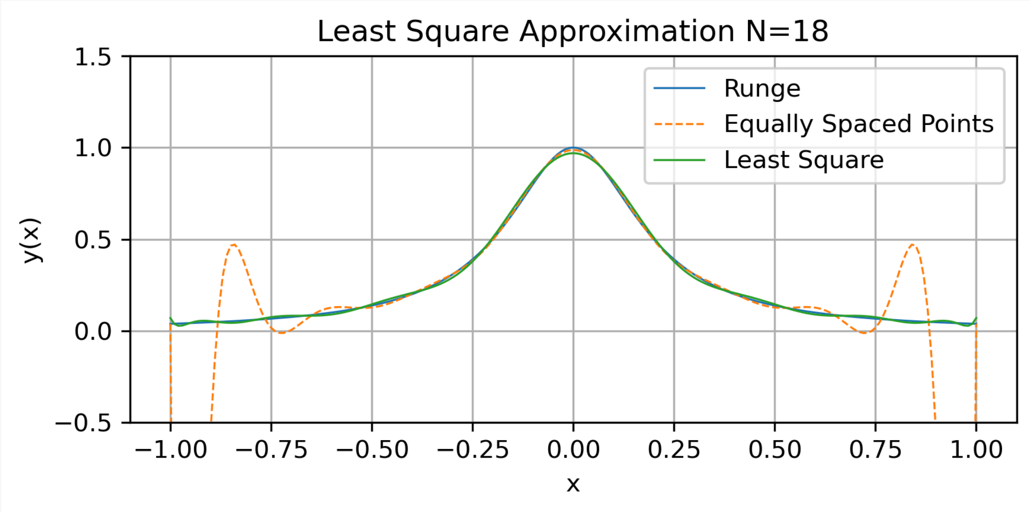
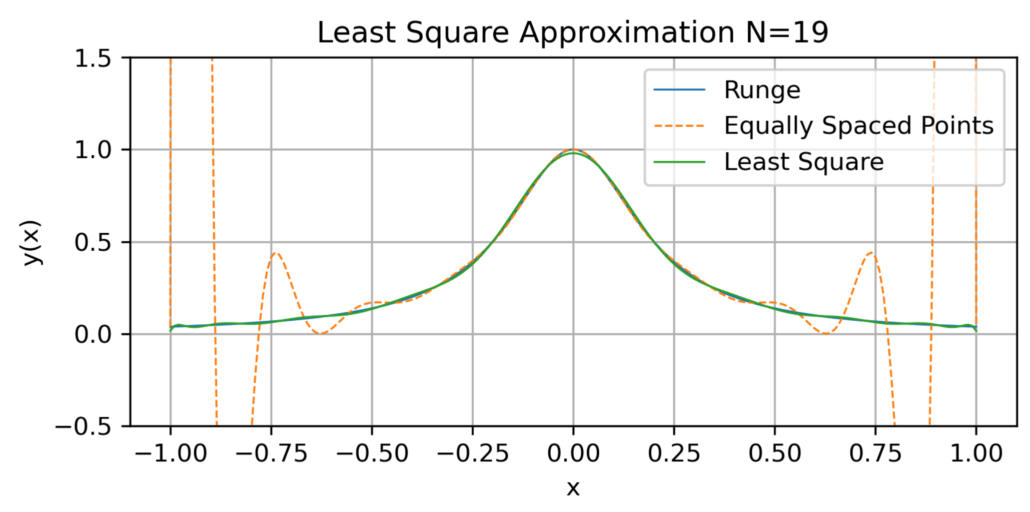
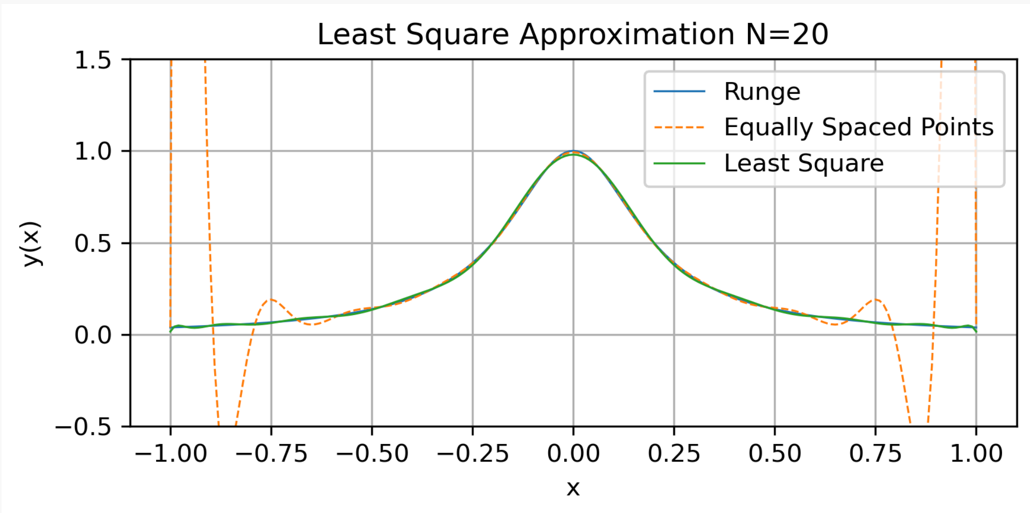
$$L(x) = \frac{\langle P_0, R(x)\rangle}{\langle P_0, P_0\rangle} P_0 + \frac{\langle P_1, R(x)\rangle}{\langle P_1, P_1\rangle} P_1 + \cdots + \frac{\langle P_{N-1}, R(x)\rangle}{\langle P_{N-1}, P_{N-1}\rangle} P_{N-1} \tag{23}$$

$$\langle P_{N-1}, R(x)\rangle = \int_{-1}^{1} P_{N-1} R(x) \mathrm{d}x \tag{24}$$

Use sympy to calculate the integral (24) analytically.

(e) Plot

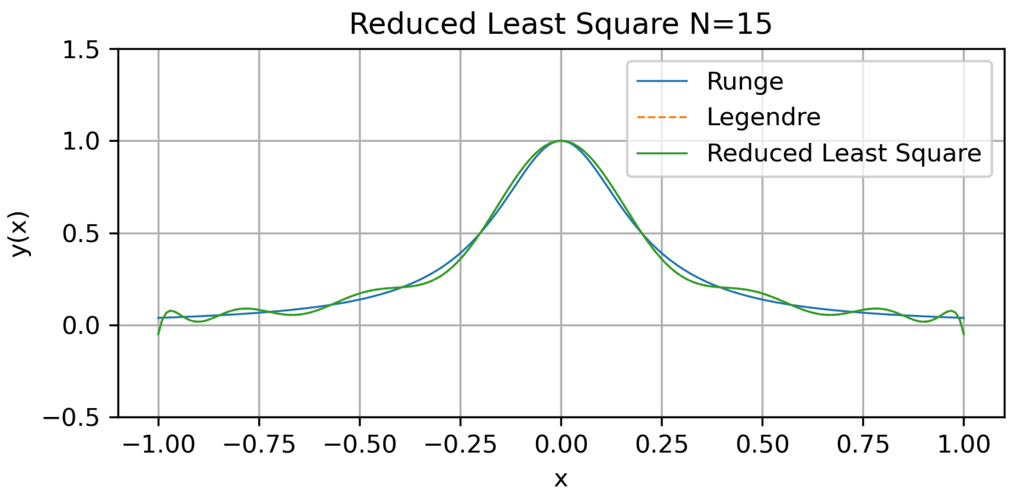
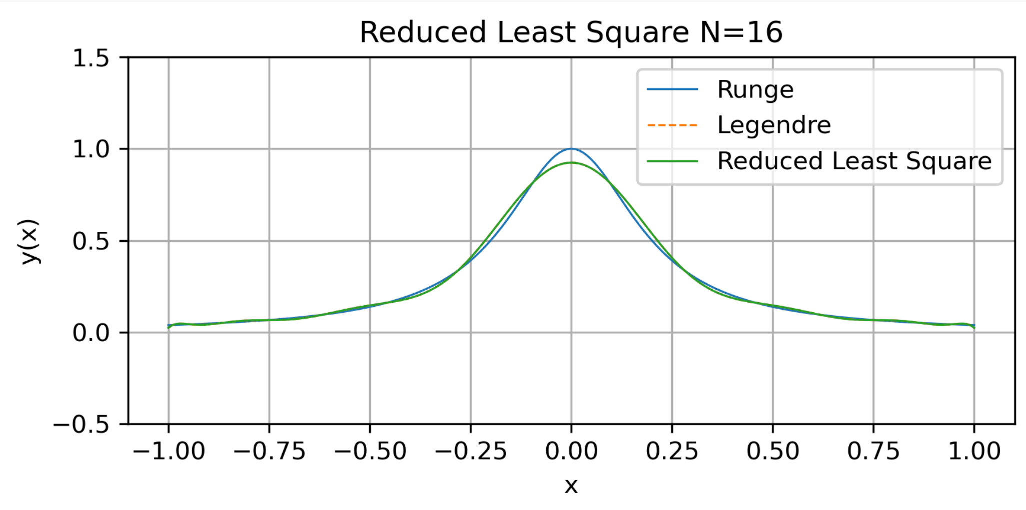
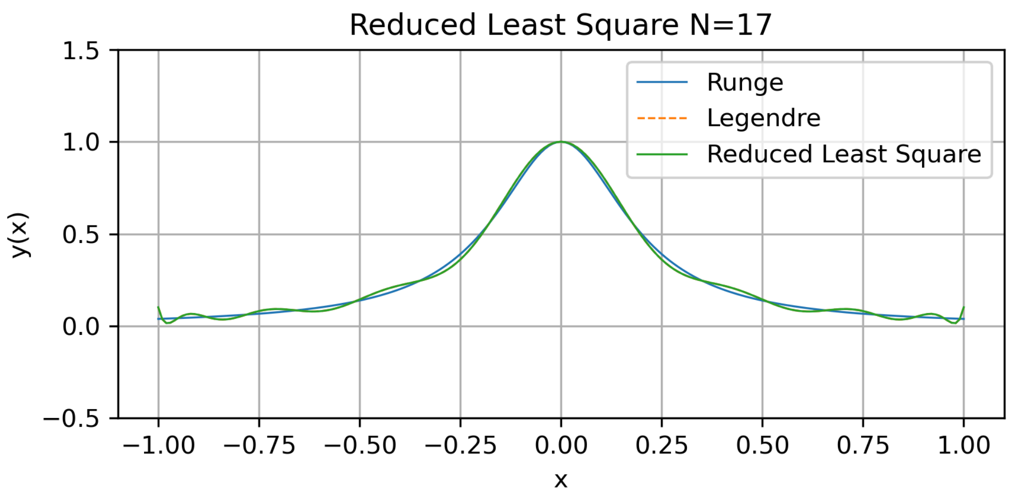
N	(e)
N=15	
N=16	
N=17	

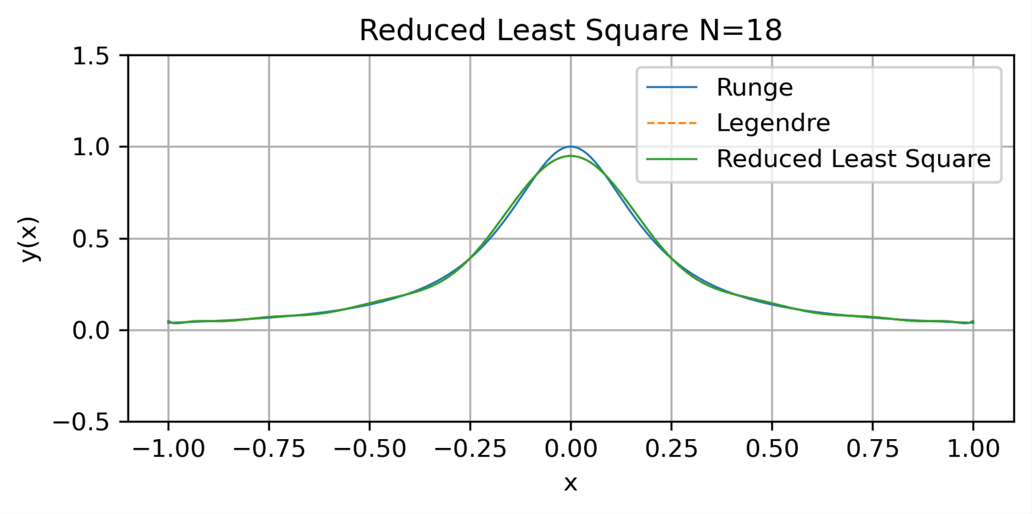
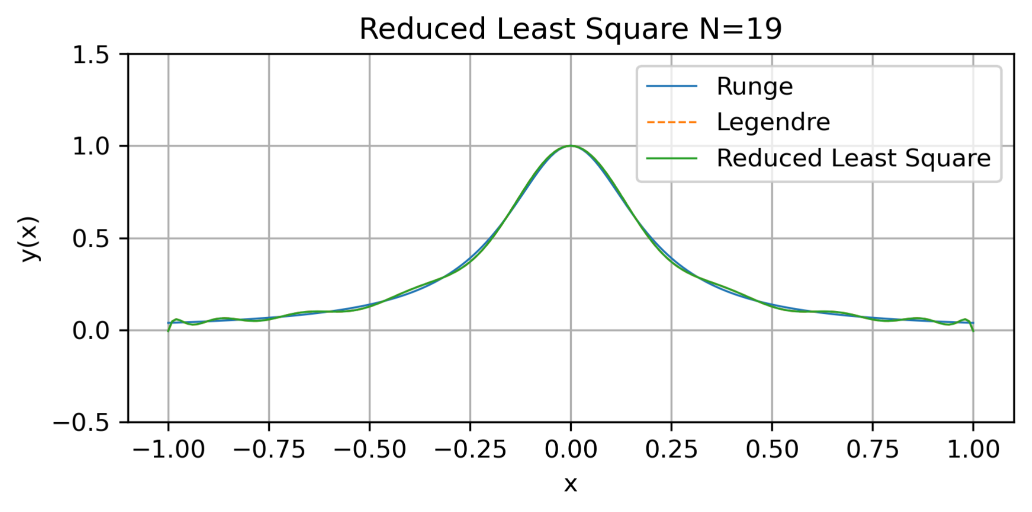
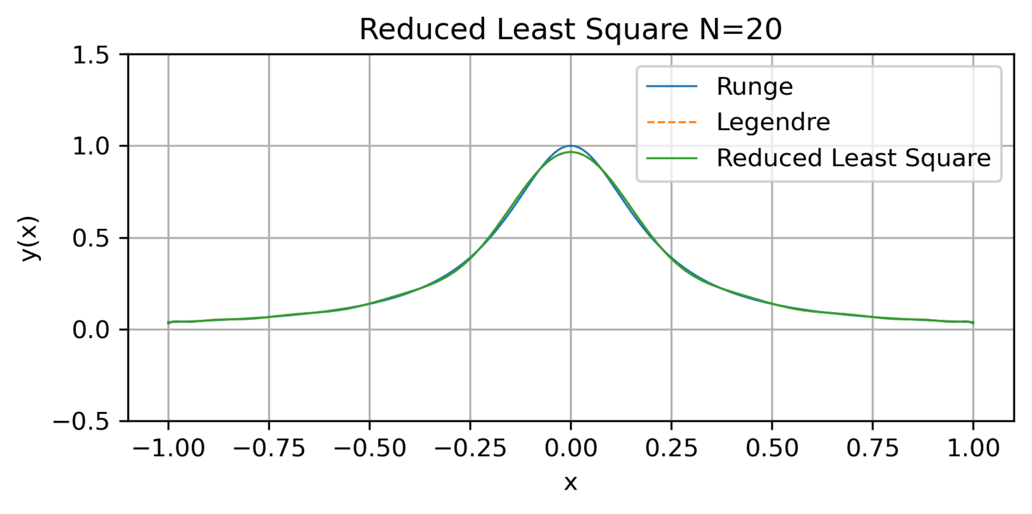
N	(e)
N=18	
N=19	
N=20	

(f) Now we use a reduced Gauss-Legendre quadrature and do Q.6d again.

Use Gauss-Legendre quadrature to compute the integral 24 numerically.

(g) Plot

N	(g)
N=15	
N=16	
N=17	

N	(g)
N=18	
N=19	
N=20	

As we can see from the plot the result of Legendre polynomials to interpolate a function is the same as using polynomials and reduced quadrature to approximate a function.