FCM II Programming Assignment 2

Jonathan Engle

February 20th, 2024

1 Executive Summary

In this report we explore the computational powers and accuracy of polynomial interpolation. The two types of interpolation that we consider through this project are: Hermite interpolation using Newtons basis and Cubic Splines. On these interpolation polynomials we will consider are the intuitive even/ uniform meshes on the selected interval $[a, b]$. We will then discuss the draw backs to these types of methods as depicted in Runges counter example ($f_2(x)$). This report is an extension of previously completed results of monomial, Lagrange and Newton basis interpolation. Graphs and other visual aids are provided to further emphasize the error analysis as well as draw backs and advantages.

2 Statement of the Problem

To implement polynomial interpolation techniques, we first need a set of test data or test functions to interpolate over. The two major functions we consider throughout this project are:

$$f_1(x) = (x-2)^9$$

Or

$$f_1(x) = x^9 - 18x^8 + 144x^7 - 672^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$$

$$f_2(x) = \frac{1}{1+x^2}$$

The motivation for selecting these functions is to analyze the accuracy of these methods as well as two different choices of meshes. With the known function, we are able to verify the theoretical results discussed in lecture as well as discuss the efficacy of each method with respect to a higher degree polynomial $f_1(x)$, and a version of Runge's counter example $f_2(x)$. For these two functions we will incorporate Hermite interpolation using Newtons basis as well as Cubic splines. Description of these methods are bellow.

3 Description of the Algorithms and Implementation

This section moves to describe the theoretical aspects of algorithms and implementation that have been completed in this assignment.

3.1 Hermite interpolation using Newton Basis

Our first method of interpolation is: Hermite interpolation using Newton Basis. For this method we will need two pieces of information at each given (computed) point, these two pieces are the function value itself as well as the derivative evaluated at that chosen x_i. More formally our textbook defines through the following relations

$$\frac{d^p}{dx^p}(L_{ik})(x_j) = \begin{cases} 1 \text{ if } i = j \text{ and } k = p, \\ 0 \text{ otherwise.} \end{cases}$$

Defining the polynomials as:

$$l_{ij}(x) = \frac{(x-x_i)^j}{j!} \prod_{\substack{k=0 \\ k\neq i}}^{n} \left(\frac{x-x_k}{x_i-x_k} \right)^{m_k+1}, i=0,\dots,n, j=0,\dots,m_i$$

Which also brings us to the question of interpolation error. For interpolation error the following estimate holds

$$f(x) - H_{N-1}(x) = \frac{f^{(N)}(\xi)}{N!} \Omega_N(x) \forall x \in \mathbb{R},$$

where $\xi \in \mathrm{I}(x; x_0, \dots, x_n)$ and Ω_N is the polynomial of degree N defined by

$$\Omega_N(x) = (x-x_0)^{m_0+1}(x-x_1)^{m_1+1} \cdots (x-x_n)^{m_n+1}.$$

Once again we expect this method to be better than the previous three interpolation methods as the Hermite interpolation targets the error bound making it tighter. For this method we consider the Newtons basis.

3.1.1 Newton Basis

A brief discussion on the chosen basis for Hermite interpolation. Newtons basis functions up to a degree n are defined as $\phi_k(x) = \Pi_{j=0}^{k-1}(x-x_j), k = 0, \dots, n$. As both of the previous methods we obtain a linear system. The linear system now is formed as:

$$A = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \dots & \phi_n(x_n) \end{bmatrix}$$

Where everything above the diagonal entries of Matrix A are zeros leaving us with a lower triangular matrix. Where we then can utilize forward substitution to lessen the computational cost. The Newtons basis provides us with the best of both of the previous methods in that it is easier to work with extensions of $P_n(x)$ such as $P_n'(x)$ and $\int P_n(x)$. While keeping our matrix A relatively simple in that it is easy to compute A^{-1}. For the implementation we use the forward substitution method from our previous FCM I assignments.

3.2 Cubic Splines

We first start off by utilizing the Definition of Spline

Definition 8.1: Let $x_0, \dots, x_n$, be $n+1$ distinct nodes of $[a, b]$, with $a = x_0 < x_1 < \dots < x_n = b$. The function $s_k(x)$ on the interval $[a, b]$ is a spline of degree k relative to the nodes x_j if

$$s_{k|[x_j, x_{j+1}]} \in P_k, j = 0, 1, \dots, n-1$$
$$s_k \in C^{k-1}[a, b].$$

With this definition we move to discuss the importance of Interpolatory Cubic Splines. The two main reasons we consider cubic splines is because:

1. They are the splines of minimum degree that yield C^2 approximations. I.e smallest to ensure a sufficient approximation.

2. They are sufficiently smooth in the presence of small curvatures.

Since our chosen spline is of degree 3 the second-order derivative must be continuous. This condition reflects back to our chosen functions $f_1(x)$ and $f_2(x)$ as they both satisfy this condition. As a matter of fact, the reason that $f_2(x)$ (Runges phenomenon) caused so many issues in the last project was because its higher order derivatives caused issues at the endpoints. Utilizing these cubic splines minimizes these errors. The linear system for this interpolation is of the following tridiagonal form:

$$\begin{bmatrix} 2 & \lambda_0 & 0 & \dots & 0 \\ \mu_1 & 2 & \lambda_1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \mu_{n-1} & 2 & \lambda_{n-1} \\ 0 & \dots & 0 & \mu_n & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

With parameters:

1. $\mu_i = \frac{h_i}{h_i + h_{i+1}}$
2. $\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}$
3. $d_i = \frac{6}{h_i + h_{i+1}} \left(\frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \right)$

Due to its nice tridiagonal form we are able to solve for this system by utilizing previous computational methods seen before such as complete pivoting LU decomposition as well as incorporation of our forward and backward substitution. Note that for the implementation, our code results were checked by using the textbook code which used the splines package. Our code obtained the same results. Now when discussing the issue of errors, we refer to the following property:

Property 8.3 Let $f \in C^4([a, b])$ and fix a partition of $[a, b]$ into subintervals of width h_i such that $h = \max_i h_i$ and $\beta = h / \min_i h_i$. Let s_3 be the cubic spline interpolating f. Then

$$\left\| f^{(r)} - s_3^{(r)} \right\|_\infty \leq C_r h^{4-r} \left\| f^{(4)} \right\|_\infty, \quad r = 0, 1, 2, 3,$$

with $C_0 = 5/384, C_1 = 1/24, C_2 = 3/8$ and $C_3 = \left(\beta + \beta^{-1}\right)/2$.

Showing that our errors satisfy this property, we refer to the figures bellow for visual reference as the computation of the errors is satisfied. Furthermore, for $f_1(x) = (x-2)^9$ we will have a relatively large fourth derivative which will increase our error bound as seen in the property above.

4 Description of the Experimental Design and Results

This section moves to describe the experimental design, testing, and results of the following interpolation techniques as discussed above. We will break up this section with respect to the functions $f_1(x) = (x-2)^9$ and the famous Runges counter example of $f_2(x) = \frac{1}{1+x^2}$. We begin first by discussing that our errors can be bounded for any choice of n by the following theorem.

- **Theorem 8.2**: Let $x_0, x_1, \dots, x_n$ be $n+1$ distinct nodes and let x be a point belonging to the domain of a given function f. Assume that $f \in C^{n+1}(I_x)$, where I_x is the smallest interval containing the nodes $x_0, x_1, \dots, x_n$ and x. Then the interpolation error at the point x is given by

$$E_n(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega_{n+1}(x)$$

where $\xi \in I_x$ and ω_{n+1} is the nodal polynomial of degree $n+1$.

All of our errors for the following computations were bounded by this theorem for small(5), medium (51) and large (151) n. Figures are shown for small cases to emphasize details. We compute the error by taking $\|f(x) - P_n(x)\|_\infty$, note $\|f(x_i) - P_n(x_i)\|_\infty = 0$ since we are using those interpolate values of the given function.

4.1 Hermite interpolation using Newtons basis

In observing the Hermite interpolation using Newtons basis for $f_1(x) = (x-2)^9$ we can see that it handles the nodes near the x-axis well as those nodes have relatively low derivative values near the root. We do see a majority of our error tail off towards the initial jump or change in concavity. Furthermore, when analyzing this interpolation at $f_2(x) = \frac{1}{1+x^2}$ we can see that as n gets larger our error decreases and we obtain a better fitting line at the end. Note that this is a drastic improvement from monomial basis as the monomial basis had massive issues near the endpoints of the Runges counter example type functions.

4.2 Cubic Splines

Next, we observe the implementation of the cubic splines. First, on $f_1(x) = (x-2)^9$ we can see that it does a better job of handling the changes in concavity as n gets larger. Figure 6 only has an $n = 25$ and is nearly a perfect fit with small error. Next, we can observe the cubic spline implementation on $f_2(x) = \frac{1}{1+x^2}$. For this implementation we can see through the multitude of graphs as $n \to \infty$ our error and our interpolation become extremely close to the correct answer. Note that this is what the theory suggested above. Finally, we can see that we chose odd number of points on the interpolated interval. This was motivated to obtain the peak of $f_2(x)$, if we chose an even number it would just cut it short near the peak but convergence to the true solution would still exist as $n \to \infty$. Note that large n graphs are not shown since they are relatively the same figure. Computations were still performed. Furthermore we can conclude that for $f_1(x)$ hermite was better and for $f_2(x)$ cubic splines out performed as well. This matches the theoretical results derived in our inclass notes.

5 Conclusion and Comparison of Methods

Overall this project sought to compare the two different Interpolation methods called Hermite interpolation using Newtons basis as well as cubic splines. As seen in our discussion all of these methods contain their own respective strengths and weaknesses. When observing error it is crucial to check that our methods are accurate and align with the theoretical results of Theorem 8.2. Interpolation techniques are fascinating in that it is how we discretize and visualize continuous functions with approximations. Understanding the motivations and draw backs to multiple methods further expands our intuition behind numerical methods to solve for real world problems.

6 Theorems, Tables, and Figures

6.1 Theorems, Properties, Citations

1. **Theorem 8.1**: Given $n+1$ distinct nodes $x_0, \ldots, x_n$ and $n+1$ corresponding values $y_0, \ldots, y_n$, there exists a unique polynomial $\Pi_n \in \mathbb{P}_n$ such that $\Pi_n(x_i) = y_i$ for $i = 0, \ldots, n$.

2. **Theorem 8.2**: Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct nodes and let x be a point belonging to the domain of a given function f. Assume that $f \in C^{n+1}(I_x)$, where I_x is the smallest interval containing the nodes $x_0, x_1, \ldots, x_n$ and x. Then the interpolation error at the point x is given by

$$E_n(x) = f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x)$$

where $\xi \in I_x$ and ω_{n+1} is the nodal polynomial of degree $n+1$.

3. **Definition 8.1:** Let $x_0, \ldots, x_n$, be $n+1$ distinct nodes of $[a,b]$, with $a = x_0 < x_1 < \ldots < x_n = b$. The function $s_k(x)$ on the interval $[a,b]$ is a spline of degree k relative to the nodes x_j if

$$s_{k|[x_j,x_{j+1}]} \in P_k, j = 0, 1, \ldots, n-1$$
$$s_k \in C^{k-1}[a,b].$$

4. **Property 8.3** Let $f \in C^4([a,b])$ and fix a partition of $[a,b]$ into subintervals of width h_i such that $h = \max_i h_i$ and $\beta = h/\min_i h_i$. Let s_3 be the cubic spline interpolating f. Then

$$\left\| f^{(r)} - s_3^{(r)} \right\|_\infty \leq C_r h^{4-r} \left\| f^{(4)} \right\|_\infty, \quad r = 0, 1, 2, 3,$$

with $C_0 = 5/384, C_1 = 1/24, C_2 = 3/8$ and $C_3 = \left(\beta + \beta^{-1}\right)/2$.

6.2 Tables

1. Table 1: Correctness results for $f_1(x) = (x-2)^9$ on the interval $[-4, 4]$

$\|f_1(x) - p_n(x)\|_\infty$	$n = 4$	$n = 51$	$n = 151$
Hermite interpolation	2.342	$1.742 * 10^{-4}$	$2.393 * 10^{-16}$
Cubic Splines	3408	$5.8997 * 10^{6}$	$1.5485 * 10^{7}$

2. Table 2: Correctness results for $f_2(x) = \frac{1}{1+x^2}$ on the interval $[-4, 4]$

$\|f_2(x) - p_n(x)\|_\infty$	$n = 5$	$n = 51$	$n = 151$
Hermite interpolation	3408	$3.408 * 10^{5}$	$1.541 * 10^{6}$
Cubic Splines	0.4106	$2.1242 * 10^{-3}$	$4.26 * 10^{-8}$

6.3 Figures

6.3.1 Hermite interpolation using Newtons basis

Figure 1: Hermite interpolation using Newtons basis on $f_1(x), n = 3$

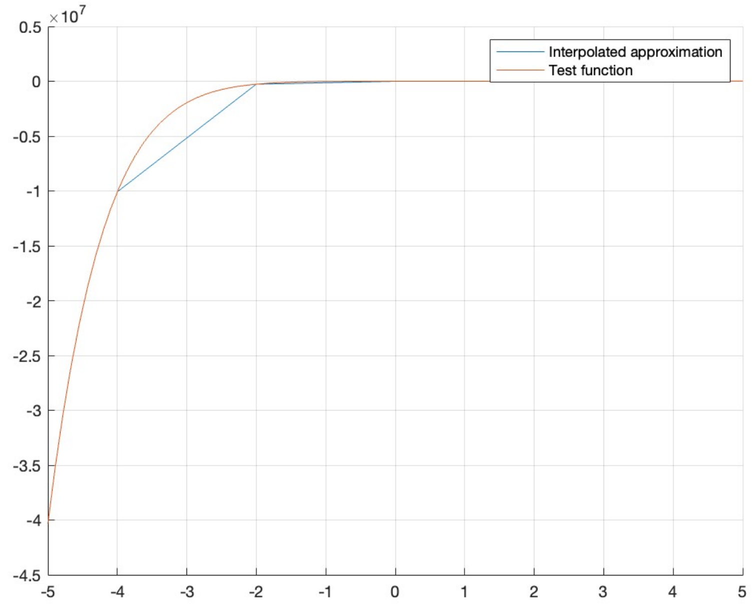


Figure 2: Hermite interpolation using Newtons basis on $f_1(x), n = 10$

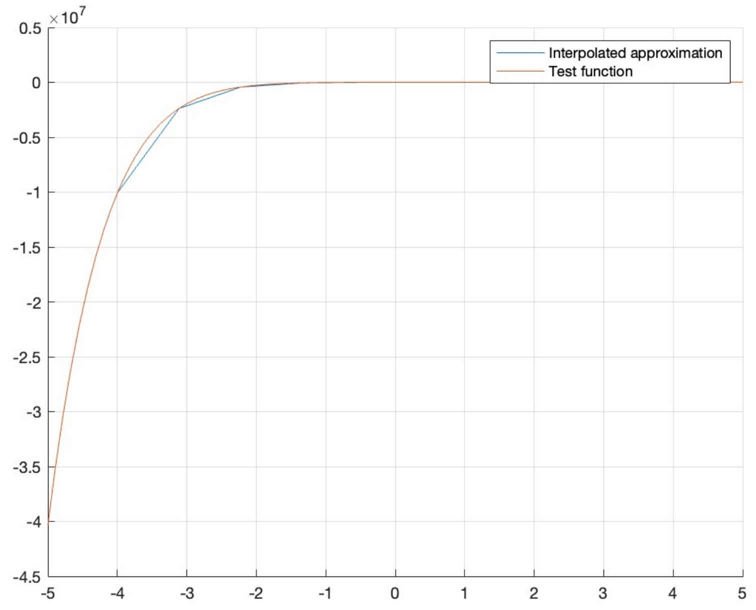


Figure 3: Hermite interpolation using Newtons basis on $f_2(x), n = 5$

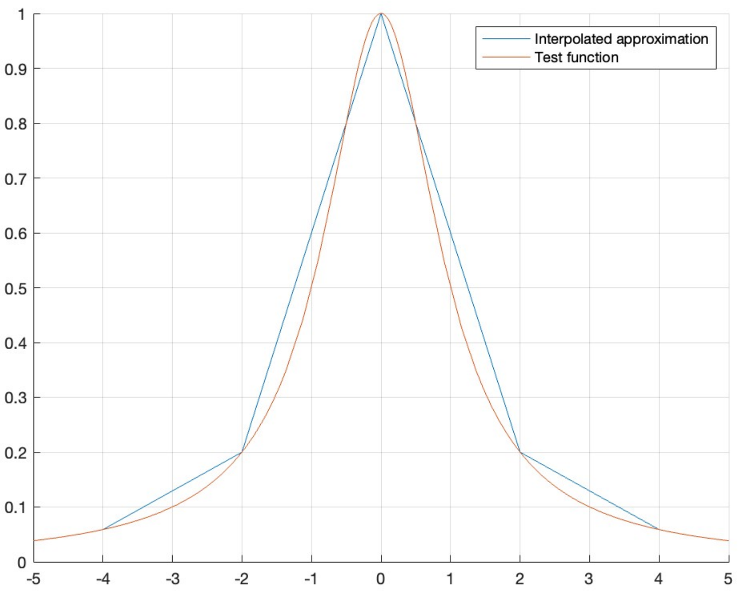
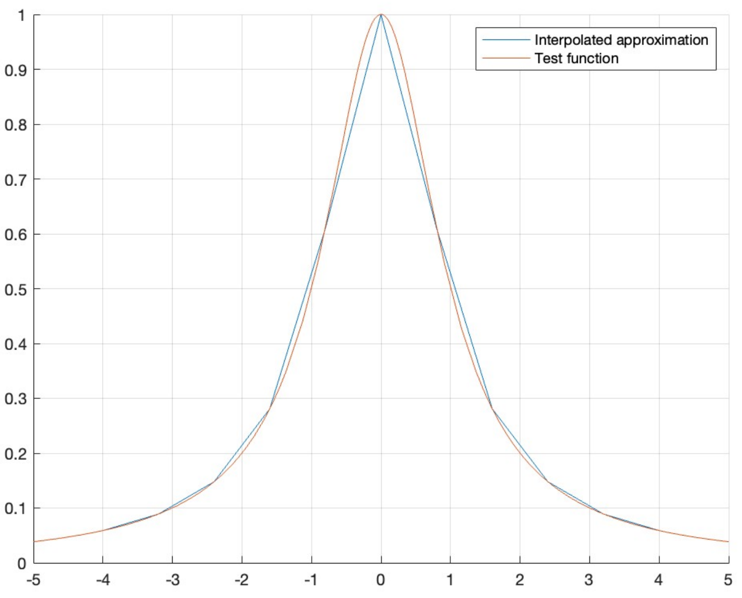


Figure 4: Hermite interpolation using Newtons basis on $f_2(x), n = 11$



6.3.2 Cubic Splines

Figure 5: Cubic spline interpolation on $f_1(x), n = 20$

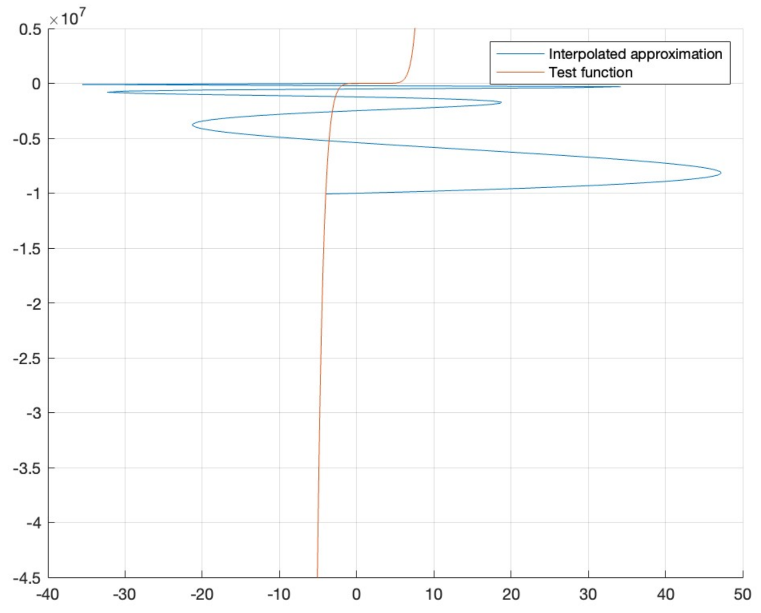


Figure 6: Cubic spline interpolation on $f_1(x), n = 25$

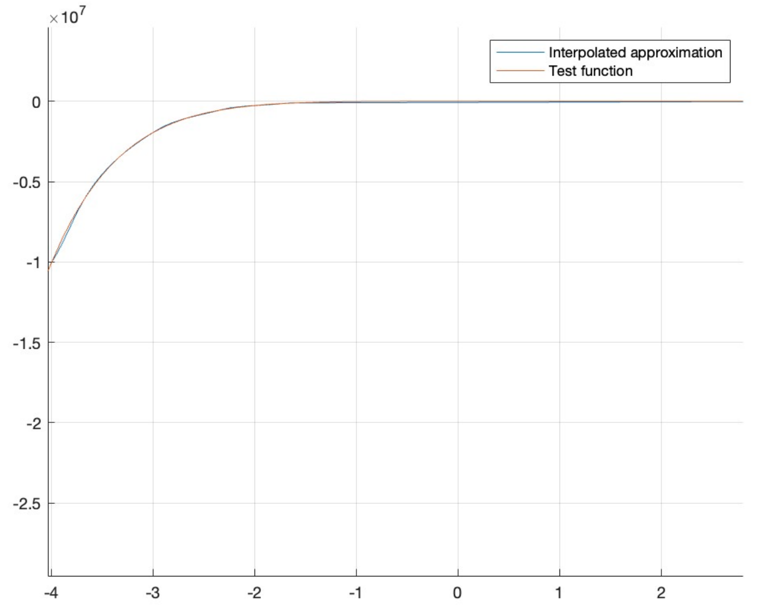


Figure 7: Cubic spline interpolation on $f_2(x), n = 3$

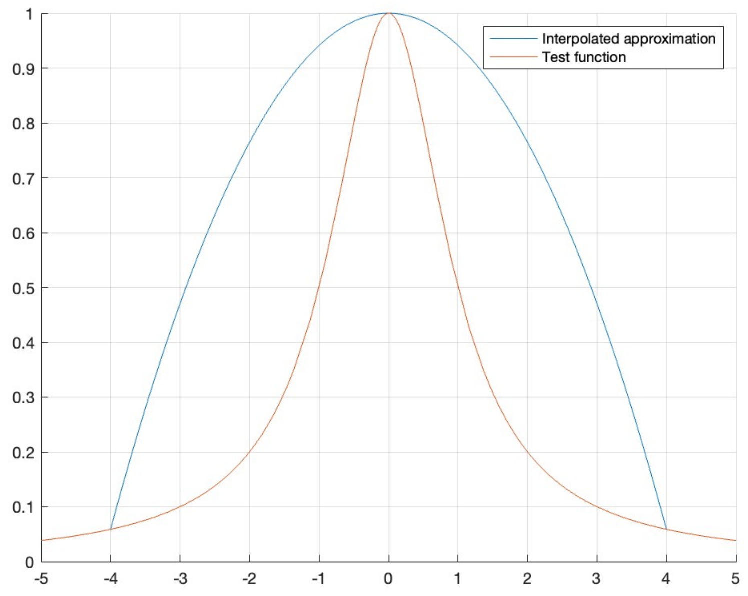


Figure 8: Cubic spline interpolation on $f_2(x), n = 5$

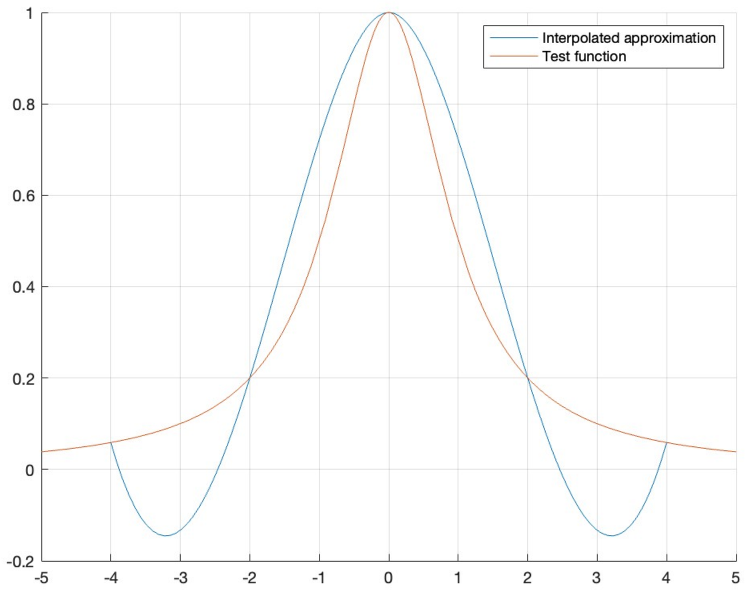


Figure 9: Cubic spline interpolation on $f_2(x), n = 7$

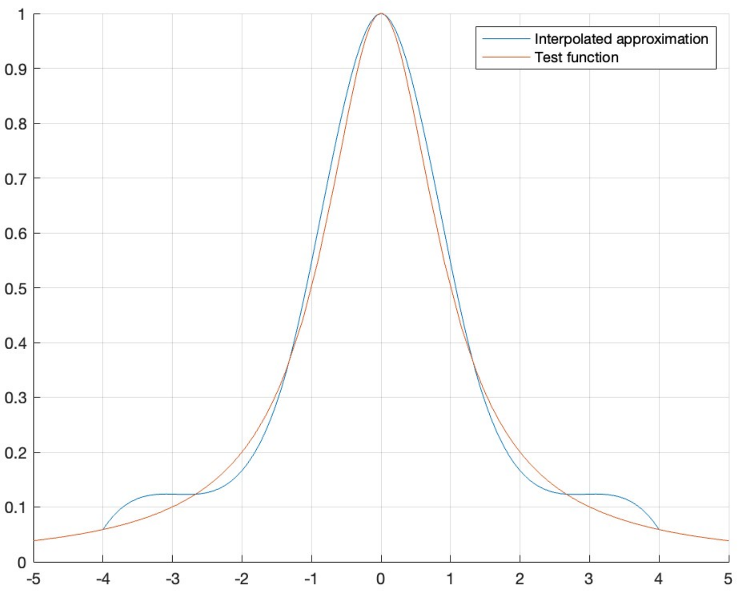
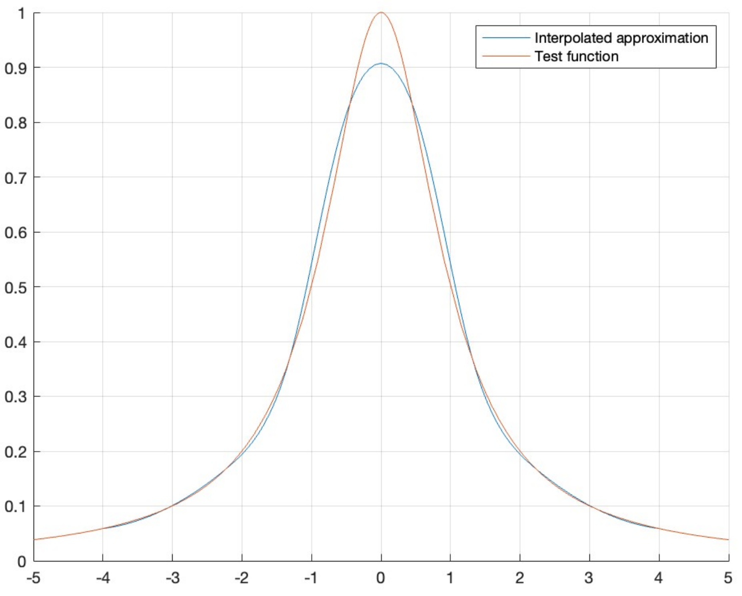


Figure 10: Cubic spline interpolation on $f_2(x), n = 10$. This one is fun to see visually as n increases.


Interpolated approximation
Test function