Development and analysis of the Lagrange interpolation method for solving multidimensional nonlinear equations

Ali Abdulhussein Hlail Alebadi*

Department of Mathematics, College of Mathematical and Computer Sciences, University of Islamic Azad, Isfahan, IRAN

*Corresponding author E-mail: aliali66t55@gmail.com

https://doi.org/10.29072/basjs.20240215

ARTICLE INFO **ABSTRACT**

Keywords

Lagrange interpolation; Multidimensional nonlinear equations; Numerical methods; Error estimates

The development of the Lagrange interpolation methodology and its analysis can enhance its accuracy, efficiency, and flexibility in solving multidimensional nonlinear equations. The article discusses the development and analysis of the Lagrange interpolation method as an effective means of solving multidimensional nonlinear equations. This method is based on estimating the functional values from known points, which provides accurate and fast solutions to complex mathematical problems. The Lagrange interpolation code has been improved for higher performance and greater efficiency through techniques such as the use of the polynomial database.

Received 10 Nov 2024; Received in revised form 29 Nov 2024; Accepted 17 Dec 2024, Published 31 Dec 2024

1. Introduction

Interpolation methods are one of the basic mathematical tools in numerical analysis, as they are used to calculate unknown values from a set of known points. One of the most common of these methods is the Lagrange interpolation method, which allows estimating a function based on its values at specific points. This method can be used in a wide range of applications, including multidimensional nonlinear systems that are difficult to solve using traditional methods. Several studies have demonstrated the benefits of using the Lagrange method in solving mathematical equations. The effectiveness of the interpolation method in improving the accuracy of environmental models was analyzed by using different measurement points. The results showed that the method performs well with discrete data, which led to improved model predictions. The method relies on polynomial functions to create an accurate representation of the underlying data, making it invaluable in modeling and prediction scenarios. Given the significance of Lagrange interpolation, numerous studies have emerged to explore its applications, improvements, and theoretical foundations. Below is a summary of key research contributions that demonstrate the breadth and depth of this mathematical technique. Jiwari introduced two differential quadrature methods aimed at approximating solutions for one- and two-dimensional hyperbolic partial differential equations under Dirichlet and Neumann boundary conditions. These methods utilize Lagrange interpolation and modified cubic B-splines, respectively, effectively transforming the hyperbolic problem into a system of second-order ordinary differential equations in relation to the time variable [1]. Carlberg et al presented a model-reduction strategy that retains the Lagrangian structure while enhancing computational efficiency, particularly in scenarios with high-order nonlinearities and arbitrary parameter dependencies. Their results, based on a parameterized truss structure problem, highlight the practical significance of preserving the Lagrangian framework and demonstrate the advantages of their approach, which reduces computation time while ensuring high accuracy and stability compared to existing nonlinear model-reduction methods that do not maintain structural integrity [2]. Berrut and Trefethen examined the mathematical characteristics of Lagrange interpolation and advocated for barycentric Lagrange interpolation as a more advantageous technique. The authors shed light on the computational efficiency of this method, making it a valuable asset for numerical analysis [3]. The Lagrange representation of the interpolating polynomial can be reformulated into two more computationally viable forms: a modified Lagrange form and a barycentric form. It is demonstrated that the modified Lagrange formula is backward stable. Consequently, the barycentric formula may exhibit significantly lower accuracy than the modified Lagrange formula, particularly when poorly chosen interpolating points are utilized [4]. Luo transformed the traditional global univariate Lagrange interpolation method into a local multivariate interpolation method. This newly developed method is capable of interpolating irregularly distributed data points effectively [5]. The performance of the local multivariate Lagrange interpolation method was assessed through its application in function approximation. Boukhelkhal and Zeghdane proposed accurate computational approaches based on the Lagrange basis and Jacobi-Gauss collocation methods to address a class of nonlinear stochastic Itô-Volterra integral equations (SIVIEs) [6]. Liu et al reported a highly accurate mesh-free method leveraging barycentric Lagrange basis functions to solve both linear and nonlinear multi-dimensional Fredholm integral equations (FIEs) of the second kind. This method represents an enhanced Lagrange interpolation technique that offers high precision alongside a cost-effective procedure [7]. Pagani et al introduced an equivalent single-layer modeling approach for laminated structures, allowing the user to preselect the number of layers consolidated into one. Lagrange points are implemented to locate and potentially unify equivalent single-layer and layer-wise techniques by enforcing displacement continuity in the thickness direction [8]. Ibrahim investigated Newton's interpolation alongside Lagrange's interpolation polynomial method (LIPM). Their research integrated both Newton’s interpolation and Lagrange methods (NIPM) for solving first-order differential equations, resulting in minimal approximation errors [9]. Carrera et al explored the geometrically nonlinear behavior arising from significant displacements and rotations in the cross-sections of thin-walled composite beams subjected to axial loading. The study presents displacement areas and stress distributions for composite structures with varying angles and functionally graded (FG) structures, showcasing the accuracy and computational efficiency of the employed method, which encourages further research [10]. Occorsio et al introduced a novel technique for image resizing that employs Lagrange-Chebyshev interpolation, suitable for both upscaling and downscaling processes. By utilizing a continuous scale combined with Chebyshev zeros on tensor product grids, this method achieves optimal approximation. Performance evaluations using SSIM and PSNR metrics illustrate that the approach yields results comparable to the traditional Bicubic method during upscaling and significantly surpasses it and other recent techniques during downscaling [11]. Sharma et al proposed a new variant of the Butterfly Optimization Algorithm (BOA), referred to as mLBOA, to enhance its performance. The new algorithm incorporates a self-adaptive parameter setting, the Lagrange interpolation formula, and a novel local search strategy enhanced with Levy flight search, aimed at improving exploration and exploitation balance [12].

Boukhelkhal and Zeghdane recommended accurate computational methods based on Lagrange basis and Jacobi-Gauss collocation methods to tackle a range of nonlinear stochastic Itô-Volterra integral equations (SIVIEs). Utilizing Lagrange polynomials and the zeros of Jacobi polynomials, the resultant system of linear and nonlinear stochastic Volterra integral equations is transformed into a linear and nonlinear algebraic equations system [13]. Yuan et al thoroughly examined a high-precision barycentric Lagrange interpolation collocation method for solving nonlinear wave equations. Comparative experiments demonstrated that this method achieves superior computational accuracy and convergence rates for nonlinear wave equations [14]. Zhao noted that the Lagrange interpolation formula does not uniformly converge to any continuous function, raising the need for advancements in its convergence characteristics. The research delves into the application of an improved Lagrangian interpolation formula by Bohr, referencing relevant original documents [15]. Deng et al introduced a generalized fuzzy barycentric Lagrange interpolation method to resolve two-dimensional fuzzy fractional Volterra integral equations. The convergence of this proposed method is analyzed, and an error estimation is provided based on the uniform continuity modulus. Finally, various numerical experiments demonstrate that the method possesses high numerical accuracy for both smooth and non-smooth solutions [16].

2. Lagrange interpolation method

The Lagrange interpolation method is a mathematical technique used to find a polynomial that passes through a given set of points. It is especially useful when you have discrete data points and want to estimate values at other points or perform data fitting.

2.1 Lagrange Polynomial

Given a set of $n+1$ data points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, where: x_i are the distinct x-coordinates of the points and y_i are the corresponding y-coordinates (function values) at those x-coordinates.

The Lagrange polynomial $P(x)$ that interpolates these points can be expressed as:

$$P(x) = \sum_{i=0}^{n} y_i \, L_i(x)$$

where $L_i(x)$ are the Lagrange basis polynomials defined as:

$$L_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}$$

The term $L_i(x)$ is constructed such that $L_i(\,x_i) = 1$ and $L_i(x_j) = 0$ for $j \neq i$. This means that $L_i(x)$ is a polynomial of degree n which is equal to 1 at x_i and equal to 0 at all other x_j. To compute a Lagrange polynomial, we perform the following steps:

We Select the data points (x_i, x_j) for $i = 0, 1, \ldots, n$. Then Compute each Lagrange basis polynomial $L_i(x)$ using the formula for $L_i(x)$. Finally, we sum the contributions $y_i L_i(x)$ for each i to obtain the polynomial $P(x)$.

Example .1.

Let's consider a concrete example with three points: $(1,1), (2,4), (3,9)$ corresponding to the function. We want to find the polynomial that fits these points.

We have the Points: $(x_0,\, y_0) = (1,1),\ (x_1,\, y_1) = (2,4),\ (x_2,\, y_2) = (3,9)$

And then basis pPolynomials:

$$L_0(x) = (\frac{x-2)(x-3)}{(1-2)(1-3)} = \frac{(x-2)(x-3)}{2}$$

$$L_1(x) = (\frac{x-1)(x-3)}{(2-1)(2-3)} = -\,(x-1)(x-3)$$

$$L_2(x) = (\frac{x-1)(x-2)}{(3-1)(3-2)} = \frac{(x-1)(x-2)}{2}$$

Then we have the following Polynomial:

$$P(x) = L_0(x) + 4L_1(x) + 9L_2(x)$$

We substitute $L_0(x), L_1(x),\ L_2(x)$ into this equation and after simplification we get the following final terms.

$$P(x) = x^2 - 36$$

Lagrange interpolation is commonly used in numerical analysis, computer graphics, and data fitting. It provides a straightforward way to approximate functions or interpolate data. The Lagrange interpolation method is a valuable technique for constructing polynomials from discrete data points, allowing for function approximation and interpolation in various applications. The Lagrange interpolation method has the advantage of providing accurate boundaries that pass through all given data points. It is easy to implement and understand, especially for a small amount of data.

The Lagrange interpolation formula can be adapted to multidimensional cases, such as for functions that depend on more than one variable (like \(f(x, y) \)). While the univariate Lagrange interpolation method focuses on interpolating functions of a single variable, the extension to multiple dimensions typically employs a tensor product approach using the concept of multivariate Lagrange interpolation.

2.2 Multidimensional Lagrange Interpolation

For a function $f(x,y)$ defined on a grid of points in a two-dimensional space, we can obtain a polynomial $P(x,y)$ that interpolates the function values at these points. Here's how the method is adapted:

We show how to adapt the method, first the point grid: Suppose we have a set of known data points given as $(x_i,\ y_i, f(x_i,\ y_i))$ for $i\ =\ 0,1,\dots,n$ and $j\ =\ 0,1,\dots,m$. Second, we define the Lagrange basis polynomials: Multivariate polynomials can be constructed using the following basis polynomials for each dimension, similar to the way this is done in a single dimension.

For a fixed y, the univariate Lagrange polynomial in x is:

$$L_i(x) = \prod_{\substack{0\le j\le n \\ j\ne i}} \frac{x - x_j}{x_i - x_j}$$

For a fixed x, the univariate Lagrange polynomial in y is:

$$M_j(y) = \prod_{\substack{0\le k\le m \\ k\ne j}} \frac{y - y_k}{y_j - y_k}$$

Now, we cCombine the basic polynomials in two dimensions, the multivariable Lagrangian polynomial $P(x,y)$ can be defined as:

$$P(x,y) = \sum\nolimits_{i=0}^{n} \sum\nolimits_{j=0}^{m} f(x_i,\ y_i)\ L_i(x)\ M_j(y)$$

This double summation combines the contributions of each basis polynomial from both dimensions. Each term $f(x_i,\ y_i)\ L_i(x)M_j(y)$ accommodates the function values at the grid points determined by $(x_i,\ y_i)$.

The method can be extended further to three or more dimensions. For instance, for a function $f(x,y,z)$ that depends on three variables, the polynomial is expressed as:

$$P(x,y,z) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p} f(x_i,\ y_i, z_k)\ L_i(x)\ M_j(y) N_k(z)$$

where $N_k(z)$ is the univariate Lagrange polynomial in the z dimension.

Example .2. (Two-Dimensional Interpolation):

Consider a set of points $(x_i,\ y_i)$ with respective function values:

$$(1,1,1),(1,2,2),(2,1,3),(2,2,4)$$

To interpolate this data. We compute Lagrange Basis Polynomials:

For x values (e.g., $x_0 = 1, x_1 = 2$):

$$L_0(x) = \frac{(x-2)}{(1-2)}, \qquad L_1(x) = \frac{(x-1)}{(2-1)}$$

For y values (e.g., $y_0 = 1, y_1 = 2$):):

$$M_0(x) = \frac{(y-2)}{(1-2)}, \qquad M_1(x) = \frac{(y-1)}{(2-1)}$$

Then, we construct the Polynomial as following:

$$P(x,y) = \sum_{i=0}^{1}\sum_{j=0}^{1} f(x_i,\ y_i)\ L_i(x) M_j(y)$$

Finally, we substitute values into the equation to compute the polynomial at any point (x, y). Multidimensional interpolation can be applied to image processing where resampling of images can involve 2D or bicubic interpolation. It can also be applied to geographic information systems (GIS) and computational physics where simulations in multiple dimensions often require interpolating fields on a grid. Multidimensional interpolation is a natural extension of the univariate case, allowing the creation of multiple bounds that fit data in multiple variables. By using the product of the multidimensional boundary tensor and combining contributions from distinct variable dimensions, this method provides a structured and efficient way to perform interpolation in higher dimensional spaces.

3. Error analysis in multidimensional interpolation

Error analysis in multidimensional interpolation is a crucial aspect of numerical analysis and computational mathematics, as it helps to quantify how accurate the interpolation results are in representing the underlying function from which sample data is drawn.

Given a function $f : \mathbb{R}^n \to \mathbb{R}$ defined on a domain $\Omega \subset \mathbb{R}^n$, we want to interpolate f using known values $f(x_1), f(x_2), \dots, f(x_m)$ at m points in Ω.

3.1 Interpolation Error Definition

The error in interpolation is typically defined as the difference between the actual function value and the interpolated value at a point X:

$$E(X) = f(X) - P(X)$$

where $P(X)$ is the interpolating polynomial or function.

3.2 Lagrange Interpolation Error

For Lagrange interpolation in n dimensions, the interpolation polynomial $P_n(x)$ can be expressed as:

$$P_n(X) = \sum_{i=0}^{n} f(X_i)\, L_i(X)$$

The error can be expressed involving the derivatives of f:

$$E(X) = f(X) - P_n(X) = \frac{f^{(n+1)}(\eta)}{(n+1)!} \prod_{i=0}^{n} (X - X_i)$$

Here, η is some point in the domain of interpolation, and $f^{(n+1)}(\eta)$ denotes the $(n+1)$-th derivative of f.

In multidimensional spaces, error analysis can become complex due to the interaction of multiple dimensions. The overall error can be characterized by evaluating the maximum error among the dimensions:

$$E_{max} = \max_{X \in D} |f(X) - P(X)|$$

Analytical forms will differ based on the interpolation method used, but key considerations include: The order of the polynomial used in interpolation, the behavior of derivatives of the function in higher dimensions, and the distribution and density of data points.

Error analysis in multidimensional interpolation provides a framework to understand how closely an interpolated function approximates the actual function.

4. Numerical examples

We provide a first example demonstrating the Lagrange interpolation method and how to find the real and approximate solutions for a set of nonlinear equations.

Example .3.

Let's use a simple nonlinear function for demonstration

$$f(x) = x^2 - 2$$

which has the real solution $x = \sqrt{2}$.

We'll generate sample points and perform interpolation. Then we'll compute the error between the real solution and the interpolated solution.

Code Matlab for example .3.

```
function lagrange_interpolation()
    real_solution = sqrt(2);
    f = @(x) x.^2 - 2;
    x_points = [0, 1, 2];
    y_points = f(x_points);
    x_interpolate = linspace(0, 2, 100);
     y_interpolated = lagrange_interpolate(x_points, y_points, x_interpolate);
    approximate_solution = find_approximate_solution(x_points, y_points);
    error = abs(real_solution - approximate_solution);
    fprintf('Real solution: %.5f\n', real_solution);
    fprintf('Approximate solution: %.5f\n', approximate_solution);
    fprintf('Approximation error: %.5f\n', error);
plot_error_graph(x_interpolate, y_interpolated, f, real_solution, x_points, y_points);
end
function y = lagrange_interpolate(x_points, y_points, x)
    n = length(x_points);
    y = zeros(size(x));
    for k = 1:n
        L = ones(size(x));
        for j = [1:k-1, k+1:n]
            L = L .* (x - x_points(j)) / (x_points(k) - x_points(j));
        end
        y = y + y_points(k) * L;
    end
end
function approx = find_approximate_solution(x_points, y_points)
    approx = lagrange_interpolate(x_points, y_points, sqrt(2));
end
function plot_error_graph(x_interpolate, y_interpolated, f, real_solution, x_points, y_points)
    exact_values = f(x_interpolate);
    errors = abs(exact_values - y_interpolated);
    figure;
    subplot(2, 1, 1);
    plot(x_interpolate, errors, 'r-', 'LineWidth', 2);
    xlabel('x', 'FontSize', 12);
    ylabel('Error', 'FontSize', 12);
    title('Error of Lagrange Interpolation', 'FontSize', 14);
    grid on;
    subplot(2, 1, 2);
    plot(x_interpolate, exact_values, 'b-', 'LineWidth', 2); hold on;
    plot(x_interpolate, y_interpolated, 'g--', 'LineWidth', 2);
    plot(x_points, y_points, 'ro', 'MarkerFaceColor', 'r');
    plot(real_solution, real_solution^2 - 2, 'ko', 'MarkerFaceColor', 'k');
    xlabel('x', 'FontSize', 12);
    ylabel('Function value', 'FontSize', 12);
    title('Lagrange Interpolation vs Exact Function', 'FontSize', 14);
    legend('Exact function', 'Lagrange Interpolated', 'Sample points', 'Real solution', 'Location',
'Best');
    grid on;
end
```

The provided MATLAB code in example 3 implements Lagrange interpolation to approximate the roots of a nonlinear function, specifically $f(x) = x^2 - 2$.

We generate sample points for interpolation: $x = [0, 1, 2]$, and calculate their corresponding function values y. Then find the approximate value of the function at $x = \sqrt{2}$ using Lagrange interpolation. The error between the real solution and the approximate solution is calculated and displayed. The top graph of Figure 1 displays the error in the Lagrange interpolation across the range of x values from 0 to 2. The y-axis represents the error magnitude, which is very small (on the order of 10^{-16}, indicating that the interpolation is highly accurate. The oscillations in the error graph suggest that while the interpolation is generally accurate, there are small fluctuations in the error values.

While Bottom graph of Figure 1 compares the exact function $f(x) = x^2 - 2$ (shown in blue) with the Lagrange interpolated values (shown in green dashed line). The red dots represent the sample points used for interpolation, while the black dot indicates the real solution at $x = \sqrt{2}$. The close alignment of the green dashed line with the blue line indicates that the Lagrange interpolation closely approximates the actual function across the range.

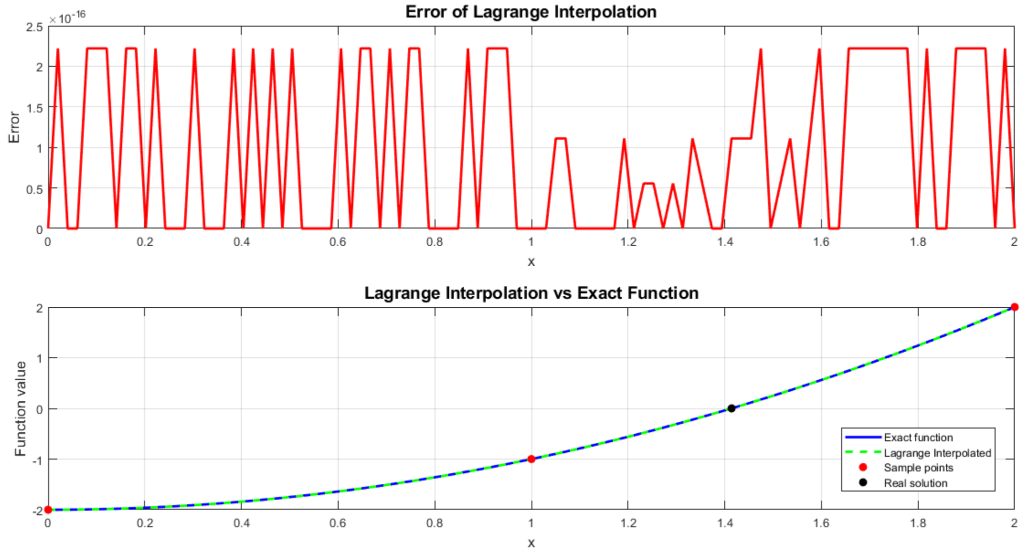

Figure 1: Error of Lagrange interpolation (top graph) and Lagrange interpolation vs exact function (bottom graph) $f(x) = x^2 - 2$

Table 1: Lagrange interpolation results for example 3

Lagrange interpolation	
Real solution	1.41421
Approximate solution	1.41421
Approximation error	0.00000

Example .4.

In the example we use the function:

$$f(x) = sin(x)$$

This function oscillates and has more intricate behavior, making it a good candidate for demonstrating Lagrange interpolation.

We'll calculate an approximate value for $f(\pi/4)$, which is

$$sin(\pi/4) = \frac{\sqrt{2}}{2} \approx 0.7071.$$

We write MATLAB code to utilize this new function and compute the approximate solution for $x = \frac{\pi}{4}$.

Code Matlab for example .4.

```
function lagrange_interpolation()
   real_solution = sin(pi/4);
   f = @(x) sin(x);
   x_points = [0, pi/6, pi/3, pi/2];
   y_points = f(x_points);
   x_interpolate = linspace(0, pi/2, 100);
   y_interpolated = lagrange_interpolate(x_points, y_points, x_interpolate);
   approximate_solution = lagrange_interpolate(x_points, y_points, pi/4);
   error = abs(real_solution - approximate_solution);
   fprintf('Real solution (sin(pi/4)): %.5f\n', real_solution);
   fprintf('Approximate solution: %.5f\n', approximate_solution);
   fprintf('Approximation error: %.5f\n', error);
   plot_error_graph(x_interpolate, y_interpolated, f, real_solution, approximate_solution, x_points,
y_points);
end
function y = lagrange_interpolate(x_points, y_points, x)
   n = length(x_points);
   y = zeros(size(x));
   for k = 1:n
      L = ones(size(x));
      for j = [1:k-1, k+1:n]
         L = L .* (x - x_points(j)) / (x_points(k) - x_points(j));
      end
      y = y + y_points(k) * L;
   end
end
function plot_error_graph(x_interpolate, y_interpolated, f, real_solution, approximate_solution,
x_points, y_points)
   exact_values = f(x_interpolate);
   errors = abs(exact_values - y_interpolated);
   figure;
   subplot(2, 1, 1);
   plot(x_interpolate, errors, 'r-', 'LineWidth', 2);
   xlabel('x', 'FontSize', 12);
```

```
    ylabel('Error', 'FontSize', 12);
    title('Error of Lagrange Interpolation', 'FontSize', 14);
    grid on;
    subplot(2, 1, 2);
    plot(x_interpolate, exact_values, 'b-', 'LineWidth', 2); hold on;
    plot(x_interpolate, y_interpolated, 'g--', 'LineWidth', 2);
    plot(x_points, y_points, 'ro', 'MarkerFaceColor', 'r');
    plot(pi/4, real_solution, 'ko', 'MarkerFaceColor', 'k');
    xlabel('x', 'FontSize', 12);
    ylabel('Function value', 'FontSize', 12);
    title('Lagrange Interpolation vs Exact Function', 'FontSize', 14);
    legend('Exact function', 'Lagrange Interpolated', 'Sample points', 'Real solution', 'Location',
'Best');
    grid on;
end
```

Points $[0, \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}]$ were chosen to better fit our interpolation for $\frac{\pi}{4}$. The plot annotations for the real and approximate solutions are adjusted to reflect the new function and point of interest (see Figure 2).

Table 2 explain the real solution for $sin(\pi/4)$, the approximate solution using Lagrange interpolation, and showing how well the interpolation fits the sine function, along with the associated errors.

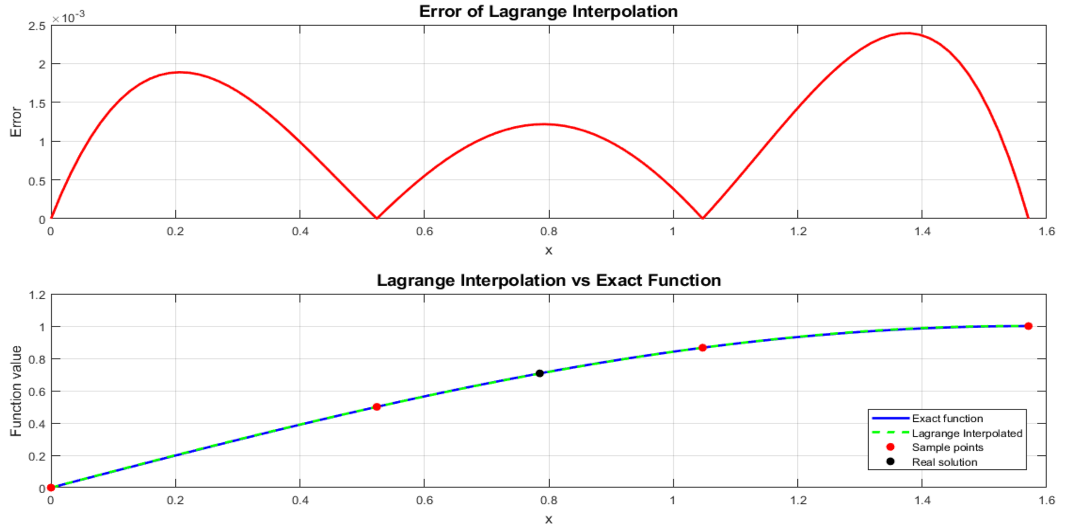

Figure 2: Error of Lagrange interpolation (top graph) and Lagrange interpolation vs exact function (bottom graph) for $f(x) = sin(x)$

Table 2: Lagrange interpolation results for example 4

Lagrange interpolation	
Real solution	0.70711
Approximate solution	0.70589
Approximation error	0.00122

Example .5.

Let the function be:

$$u(x) = 6x - x^3 + 0.5 \int_0^{x^2} (6t - t^3)\, dt$$

Code Matlab for Eexample .5.

```
function lagrange_interpolation()
   real_solution = u(sqrt(2));
   x_points = [0, 1, 2];
   y_points = u(x_points);
   x_interpolate = linspace(0, 2, 100);
   y_interpolated = lagrange_interpolate(x_points, y_points, x_interpolate);
   approximate_solution = find_approximate_solution(x_points, y_points);
   error = abs(real_solution - approximate_solution);
   fprintf('Real solution: %.5f\n', real_solution);
   fprintf('Approximate solution: %.5f\n', approximate_solution);
   fprintf('Estimation error: %.5f\n', error);
   plot_results(x_interpolate, y_interpolated, real_solution, x_points, y_points);
end
function y = lagrange_interpolate(x_points, y_points, x)
   n = length(x_points);
   y = zeros(size(x));
   for k = 1:n
      L = ones(size(x));
      for j = [1:k-1, k+1:n]
         L = L .* (x - x_points(j)) / (x_points(k) - x_points(j));
      end
      y = y + y_points(k) * L;
   end
end
function approx = find_approximate_solution(x_points, y_points)
   approx = lagrange_interpolate(x_points, y_points, sqrt(2));
end
```

```
function u_value = u(x)
   u_value = zeros(size(x));
   for i = 1:length(x)
      t_squared = x(i).^2;
      u_value(i) = 6*x(i) - x(i)^3 + 0.5 * integral(@(t) u_func(t), 0, t_squared);
   end
end
function val = u_func(t)
   val = 6*t - t.^3;
end
function plot_results(x_interpolate, y_interpolated, real_solution, x_points, y_points)
   exact_values = u(x_interpolate);
   errors = abs(exact_values - y_interpolated);
   figure;
   plot(x_interpolate, exact_values, 'b-', 'LineWidth', 2); hold on;
   plot(x_interpolate, y_interpolated, 'g--', 'LineWidth', 2);
   plot(x_points, y_points, 'ro', 'MarkerFaceColor', 'r');
   xlabel('x', 'FontSize', 12);
   ylabel('Function Value', 'FontSize', 12);
   title('Lagrange Interpolation vs Exact Function', 'FontSize', 14);
   legend('Exact Function', 'Lagrange Interpolated', 'Sample Points', 'Location', 'Best');
   grid on;
   figure;
   plot(x_interpolate, errors, 'm-', 'LineWidth', 2);
   xlabel('x', 'FontSize', 12);
   ylabel('Error', 'FontSize', 12);
   title('Interpolation Error', 'FontSize', 14);
   grid on;
end
```

Figure 3 shows how the interpolation error varies across the defined range. The peak error signifies a point where the interpolation fails to accurately follow the original function. This graph illustrates the difference between the original function (blue line) and the interpolated function using Lagrange (dashed line). It shows that the interpolation may fit well at certain sample points (red dots) but diverges in other areas. These results in Table 3 indicate that the choice of sample points and their impact on the accuracy of interpolation plays a crucial role. It is essential to evaluate how well the interpolation aligns with the original function across specific ranges to ensure accurate estimations. In this case, the approximate solution is far from the true value, highlighting the need to improve point selection.

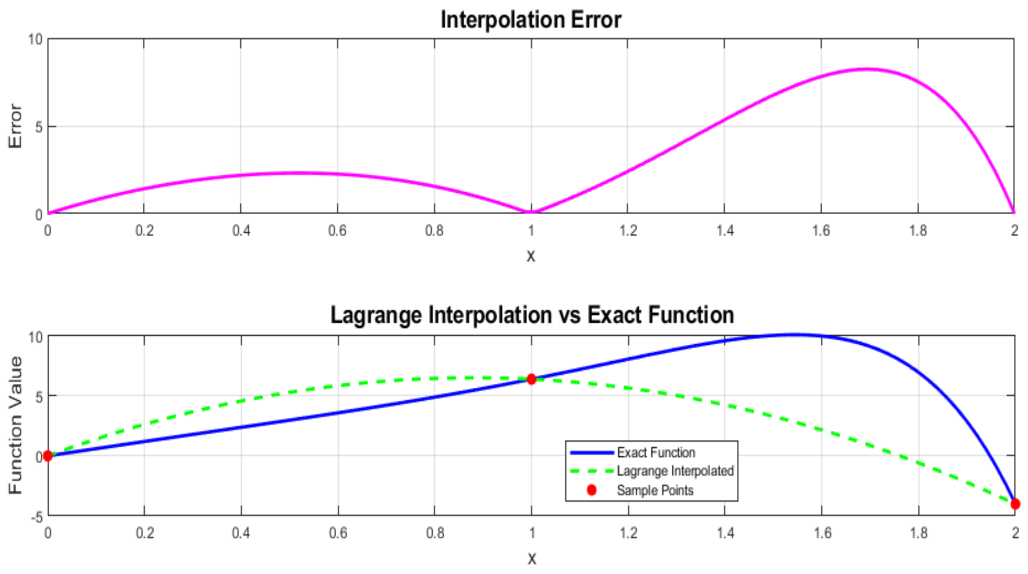

Figure 3: Error of Lagrange interpolation (top graph) and Lagrange interpolation vs exact function (bottom graph) for function in example 5

Table 3: Lagrange interpolation results for example 5

Lagrange interpolation	
Real solution	9.65685
Approximate solution	4.10965
Estimation error	5.54720

Example 6.

Let the function

$$f(x) = x^2 + \frac{1}{21}x^4 + \int_0^x (t - x)\,dt$$

and we perform Lagrange interpolation.

Code Matlab for example .6.

```
function lagrange_interpolation()
    f = @(x) x.^2 + (1/21) * x.^4 + arrayfun(@(x) integral(@(t) (t - x) .* ones(size(t)), 0, x), x);
    real_solution = f(sqrt(2));
    x_points = [0, 1, 2];
    y_points = f(x_points);
    x_interpolate = linspace(0, 2, 100);
    y_interpolated = lagrange_interpolate(x_points, y_points, x_interpolate);
    approximate_solution = find_approximate_solution(x_points, y_points);
    error = abs(real_solution - approximate_solution);
    fprintf('Real solution: %.5f\n', real_solution);
    fprintf('Approximate solution: %.5f\n', approximate_solution);
    fprintf('Approximation error: %.5f\n', error);
    plot_error_graph(x_interpolate, y_interpolated, f, real_solution, x_points, y_points);
end
function y = lagrange_interpolate(x_points, y_points, x)
    n = length(x_points);
    y = zeros(size(x));
    for k = 1:n
        L = ones(size(x));
        for j = [1:k-1, k+1:n]
            L = L .* (x - x_points(j)) / (x_points(k) - x_points(j));
        end
        y = y + y_points(k) * L;
    end
end
function approx = find_approximate_solution(x_points, y_points)
    approx = lagrange_interpolate(x_points, y_points, sqrt(2));
end
function plot_error_graph(x_interpolate, y_interpolated, f, real_solution, x_points, y_points)
    exact_values = f(x_interpolate);
    errors = abs(exact_values - y_interpolated);
    figure;
    subplot(2, 1, 1);
    plot(x_interpolate, errors, 'r-', 'LineWidth', 2);
    xlabel('x', 'FontSize', 12);
    ylabel('Error', 'FontSize', 12);
    title('Error of Lagrange Interpolation', 'FontSize', 14);
    grid on;
    subplot(2, 1, 2);
    plot(x_interpolate, exact_values, 'b-', 'LineWidth', 2); hold on;
    plot(x_interpolate, y_interpolated, 'g--', 'LineWidth', 2);
    plot(x_points, y_points, 'ro', 'MarkerFaceColor', 'r');
    plot(sqrt(2), real_solution, 'ko', 'MarkerFaceColor', 'k');
    xlabel('x', 'FontSize', 12);
    ylabel('Function value', 'FontSize', 12);
    title('Lagrange Interpolation vs Exact Function', 'FontSize', 14);
    legend('Exact function', 'Lagrange Interpolated', 'Sample points', 'Real solution', 'Location',
'Best');
```

```
    grid on;
end
```

The Lagrange interpolation results in Table 4 indicate that the approximate solution 1.26261 is relatively close to the real solution 1.19048. The approximation error of 0.07213 signifies a small deviation, suggesting that the interpolation method performed quite well in this case. The Figure 4 shows the error across the range, indicating that the error fluctuates but remains relatively small, peaking around \(0.05\). The interpolation closely follows the exact function, demonstrating good accuracy in this example. The Lagrange interpolation method yielded a satisfactory approximation in this case, with a minor error, indicating that the interpolation closely represents the original function over the chosen range.

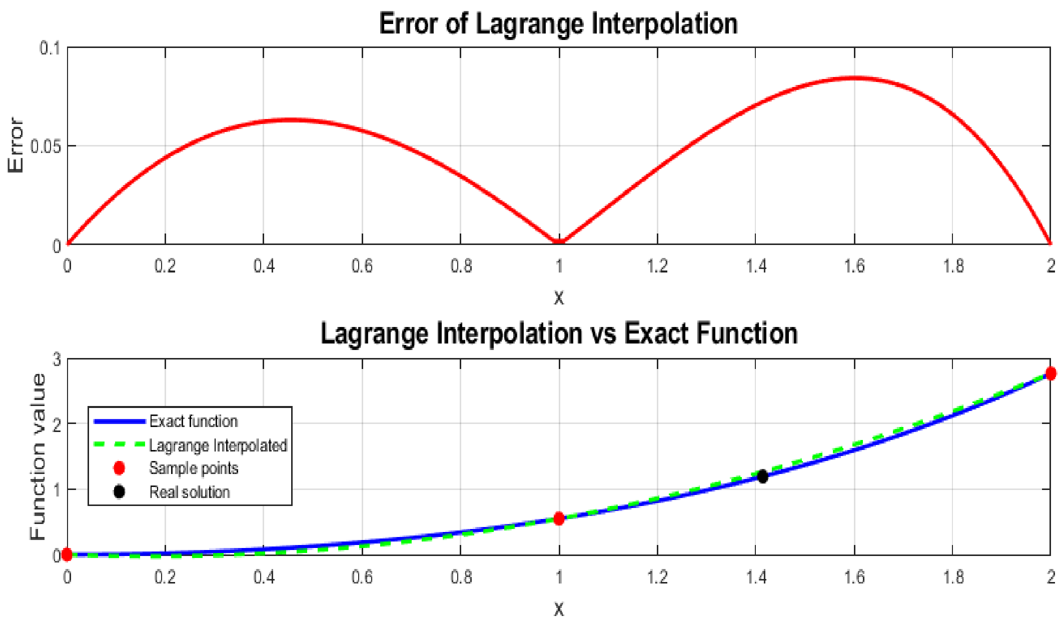

Figure 4: Error of Lagrange interpolation (top graph) and Lagrange interpolation vs exact function (bottom graph) for function in example 6

Table 4: Lagrange interpolation results for example 6

Lagrange interpolation	
Real solution	1.19048
Approximate solution	1.26261
Approximation error	0.07213

Example 7.

Let the function

$$f(x,y) = \sqrt{(x^2 + y^2) + exp(x + y)}$$

and we perform Lagrange interpolation.

Code Matlab for example 7.

```
function nonlinear_interpolation()
   f = @(x, y) sqrt(x.^2 + y.^2) + exp(x + y);
   x_points = linspace(-2, 2, 5);
   y_points = linspace(-2, 2, 5);
   [X, Y] = meshgrid(x_points, y_points);
   Z_exact = f(X, Y);
   x_interpolate = linspace(-2, 2, 100);
   y_interpolate = linspace(-2, 2, 100);
   [X_interp, Y_interp] = meshgrid(x_interpolate, y_interpolate);
   Z_approx = lagrange_interpolate(x_points, y_points, f, X_interp, Y_interp);
   Z_exact_interp = f(X_interp, Y_interp);
   absolute_error = abs(Z_exact_interp - Z_approx);
   figure;
   subplot(1, 2, 1);
   surf(X_interp, Y_interp, Z_exact_interp, 'FaceAlpha', 0.5);
   hold on;
   surf(X_interp, Y_interp, Z_approx, 'FaceAlpha', 0.5);
   title('Exact and Approximated Solutions');
   xlabel('X-axis');
   ylabel('Y-axis');
   zlabel('Z-axis');
   legend('Exact Solution', 'Approximated Solution');
   grid on;
   subplot(1, 2, 2);
   surf(X_interp, Y_interp, absolute_error);
   title('Absolute Error of Approximations');
   xlabel('X-axis');
   ylabel('Y-axis');
   zlabel('Absolute Error');
   colorbar;
   grid on;
   fprintf('Numerical Values of Absolute Errors:\n');
   disp(absolute_error);
```

```
    fprintf('Note: It is important to consider the numerical values or use the absolute error graph to
assess the accuracy of the approximation compared to the exact solution.\n');
end
function Z = lagrange_interpolate(x_points, y_points, f, X, Y)
    Z = zeros(size(X));
    m = length(x_points);
    n = length(y_points);
    for i = 1:m
        for j = 1:n
            Lx = 1;
            Ly = 1;
            for k = 1:m
                if k ~= i
                    Lx = Lx .* (X - x_points(k)) / (x_points(i) - x_points(k));
                end
            end
            for l = 1:n
                if l ~= j
                    Ly = Ly .* (Y - y_points(l)) / (y_points(j) - y_points(l));
                end
            end
            Z = Z + f(x_points(i), y_points(j)) * Lx .* Ly;
        end
    end
end
```

In Figure 5, the exact solution in blue and the approximate solution in orange (left figure), allowing for comparison of the two figures. The absolute error between the two solutions is shown (right figure), which helps in assessing the accuracy of the interpolation.

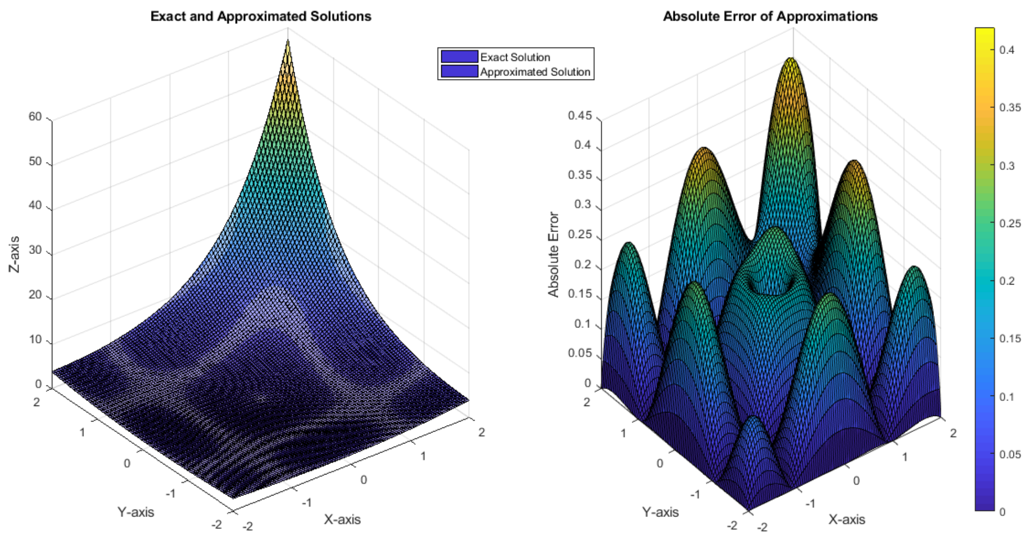

Figure 5: The exact solution (in blue) and the approximate solution (in orange) in the left figure, and the absolute error between the two solutions (right figure) for example 7

Example 8

We repeat the previous example and calculate the exact solution and approximate solutions using different numbers of points $[1, 5, 10, 20]$.

$$f(x,y) = \sqrt{(x^2 + y^2) + exp(x + y)}$$

Code Matlab for example 8

```
function nonlinear_interpolation_multiple_iterations_combined()
   f = @(x, y) sqrt(x.^2 + y.^2) + exp(x + y);
   iterations = [1, 5, 10, 20];
   x_interpolate = linspace(-2, 2, 500);
   y_interpolate = linspace(-2, 2, 500);
   [X_exact, Y_exact] = meshgrid(x_interpolate, y_interpolate);
   Z_exact = f(X_exact, Y_exact);
   absolute_errors = zeros(size(iterations));
   figure;
   for k = 1:length(iterations)
      num_points = iterations(k);
      x_points = linspace(-2, 2, num_points);
      y_points = linspace(-2, 2, num_points);
      Z_approx = lagrange_interpolate(x_points, y_points, f, X_exact, Y_exact);
      absolute_error = abs(Z_exact - Z_approx);
```

```
        max_absolute_error = max(absolute_error(:));
        absolute_errors(k) = max_absolute_error;
        subplot(2, length(iterations), k);
        surf(X_exact, Y_exact, Z_exact, 'FaceAlpha', 0.5, 'EdgeColor', 'none');
        title('Exact Solution');
        xlabel('X-axis');
        ylabel('Y-axis');
        zlabel('Z-axis');
        grid on;
        view(3);
        subplot(2, length(iterations), k + length(iterations));
        surf(X_exact, Y_exact, Z_approx, 'FaceAlpha', 0.3, 'EdgeColor', 'none');
        title(['Approximation with ', num2str(num_points), ' Points']);
        xlabel('X-axis');
        ylabel('Y-axis');
        zlabel('Z-axis');
        grid on;
        view(3);
    end
    disp('Maximum Absolute Errors for each iteration:');
    disp(array2table(absolute_errors', 'VariableNames', {'MaximumAbsoluteError'}, 'RowNames',
cellstr(num2str(iterations'))));
end
function Z = lagrange_interpolate(x_points, y_points, f, X, Y)
    Z = zeros(size(X));
    m = length(x_points);
    n = length(y_points);
    for i = 1:m
        for j = 1:n
            Lx = 1;
            Ly = 1;
            for k = 1:m
                if k ~= i
                    Lx = Lx .* (X - x_points(k)) / (x_points(i) - x_points(k));
                end
            end
            for l = 1:n
                if l ~= j
                    Ly = Ly .* (Y - y_points(l)) / (y_points(j) - y_points(l));
                end
            end
            Z = Z + f(x_points(i), y_points(j)) * Lx .* Ly;
        end
    end
end
```

The resulting plot shows the exact solution and approximate solutions using different numbers of points (see Figure 6). Exact Solution shown in the top row of the plot. It represents the exact surface of the given function, which appears smooth and curved. The peak in the middle indicates the highest values of the function, while the lowest values appear at the ends. Approximate solutions shown in the bottom row. Each plot is an approximation using a different number of points (1, 5, 10, 20). As the number of points increases, the approximate solutions get closer to the exact solution. When using a single point, the surface appears very flat, which means that the approximation is inaccurate. As the number of points increases, the surface becomes more complex and approaches the exact shape. Table 5 shows the maximum absolute errors for the iterations (1, 5, 10, 20).

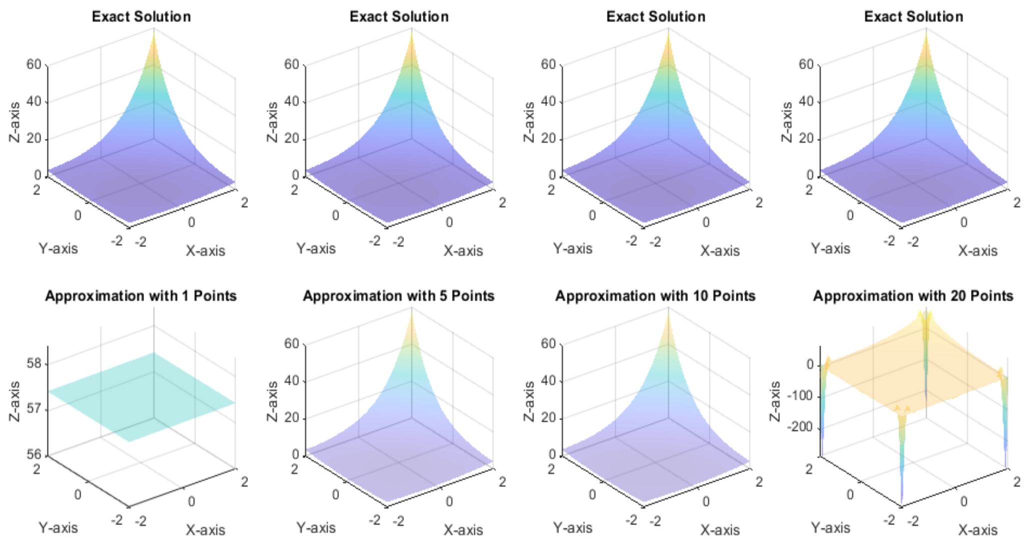

Figure 6: The exact solution and approximate solutions using different numbers of points for example 8

Table 5: Maximum Absolute Errors for each iteration

Iteration	Maximum Absolute Errors
1	56.474
5	0.41982
10	0.18032
20	0.10601

5. Computational results

Developing the Lagrange interpolation methodology and its analysis can enhance its accuracy, efficiency, and flexibility in solving multidimensional nonlinear equations. This can open new horizons for more complex and innovative applications in various fields. The Lagrange interpolation method may be more accurate in estimating values at unknown points. A good analysis can lead to the detection of errors and dimensions that can be improved. As the algorithm is improved, it is possible to reduce the number of points required to achieve a given level of accuracy. This can reduce computational complexity. Developing an analytical study of the algorithm can help to obtain accurate estimates of errors, allowing us to understand how and to what extent nonlinear complexities affect the results. Increasing the flexibility of the algorithm in dealing with certain constraints or nonlinear functions can open the way to solve a wider range of nonlinear equations.

6. Discussion and conclusion

The current study provides a comprehensive analysis of the Lagrange method and its improvements. The results confirm that improving the code using techniques such as analysis and the Howern method leads to increased efficiency and reduced time required to calculate the interpolated values. The use of the Lagrange method can be expanded to include new types of mathematical models, such as those used in data science and network analysis. It is useful to combine the Lagrange method with artificial intelligence and machine learning techniques to enhance modeling and prediction capabilities. Further studies should be conducted to analyze typical errors and how to improve the method to address problems related to numerical analysis.

References

[1] R. Jiwari, Lagrange interpolation and modified cubic B-spline differential quadrature methods for solving hyperbolic partial differential equations with Dirichlet and Neumann boundary conditions, Computer Physics Communications, 193(2015)55-65, https://doi.org/10.1016/j.cpc.2015.03.021

[2] K. Carlberg, R. Tuminaro, P. Boggs, Preserving Lagrangian structure in nonlinear model reduction with application to structural dynamics, SIAM J. Sci Com., 37(2015) B153-B184, https://doi.org/10.1137/140959602

[3] J. P. Berrut, L. N. Trefethen, Barycentric lagrange interpolation, SIAM review, 46(2004) 501-517, https://doi.org/10.1137/S0036144502417715

[4] N. J. Higham, The numerical stability of barycentric Lagrange interpolation, IM. J. NA., 24(2004)547-556, https://doi.org/10.1093/imanum/24.4.547

[5] Y. Luo, A local multivariate Lagrange interpolation method for constructing shape functions, Int. J. biom eng., 26(2010)252-261, https://doi.org/10.1002/cnm.1149

[6] I. Boukhelkhal, R. Zeghdane, Lagrange interpolation polynomials for solving nonlinear stochastic integral equations, Num Algo, 96(2024)583-618, https://doi.org/10.1007/s11075-023-01713-8

[7] H. Liu, J. Huang, W. Zhang, Y Ma, Meshfree approach for solving multi-dimensional systems of Fredholm integral equations via barycentric Lagrange interpolation, App. Math. Com, 346(2019)295-304, https://doi.org/10.1016/j.amc.2018.10.024

[8] A. Pagani, E. Carrera, R. Augello, D. Scano, Use of Lagrange polynomials to build refined theories for laminated beams, plates and shells, Com. Str, 276(2021)114505, https://doi.org/10.1016/j.compstruct.2021.114505

[9] S. Ibrahim, Application of Lagrange Interpolation Method to Solve First-Order Differential Equation Using Newton Interpolation Approach, Eur. J. Sci. Eng., 9(2023) 89-98, https://doi.org/10.23918/eajse.v9i1p89 توحيد كتابة العنوان

[10] E. Carrera, M. D. Demirbas, R. Augello, Evaluation of stress distribution of isotropic, composite, and FG beams with different geometries in nonlinear regime via Carrera-Unified Formulation and Lagrange polynomial expansions, App. Sci, 11 (2021)10627, https://doi.org/10.3390/app112210627

[11] D. Occorsio, G. Ramella, W. Themistoclakis, Lagrange–Chebyshev Interpolation for image resizing, Math. Com. Sim, 197(2022)105-126, https://doi.org/10.1016/j.matcom.2022.01.017

[12] S. Sharma, S. Chakraborty, A. K. Saha, S. Nama, S. K. Sahoo, mLBOA: A modified butterfly optimization algorithm with lagrange interpolation for global optimization, J. Bio. Eng., 19 (2022) 1161-1176, https://doi.org/10.1007/s42235-022-00175-3

[13] I. Boukhelkhal, R. Zeghdane, Lagrange interpolation polynomials for solving nonlinear stochastic integral equations, Num. Algo, 96(2024)583-618, https://doi.org/10.1007/s11075-023-01713-8

[14] H. Yuan, X. Wang, J. Li, Solving Nonlinear Wave Equations Based on Barycentric Lagrange Interpolation, J. Nonl. Math. Phy., 31 (2024)41, https://doi.org/10.1007/s44198-024-00200-5

[15] N. Zhao, Study on the Application of Borel's Improved Lagrange Interpolation Formula, Int. J. Edu. Hum., 9 (2023) 21-23, https://doi.org/10.54097/ijeh.v9i1.9030

[16] T. Deng, J. Huang, Y. Wang, H. Li, A generalized fuzzy barycentric Lagrange interpolation method for solving two-dimensional fuzzy fractional Volterra integral equations, Num. Algo, (2024)1-24, https://doi.org/10.1007/s11075-024-01814-y

تطوير وتحليل طريقة استيفاء(التعديل الداخلي) لاغرانج لحل المعادلات غير الخطية متعددة الأبعاد

المستخلص

إن تطوير منهجية الاستيفاء لاغرانج وتحليلها يمكن أن يعزز من دقتها وكفاءتها ومرونتها في حل المعادلات غير الخطية متعددة الأبعاد. تناقش المقالة تطوير وتحليل طريقة الاستيفاء لاغرانج كوسيلة فعالة لحل المعادلات غير الخطية متعددة الأبعاد. تعتمد هذه الطريقة على تقدير القيم التابعة من نقاط معروفة، مما يوفر حلولاً دقيقة وسريعة للمشكلات الرياضية المعقدة. تم تحسين كود الاستيفاء لاغرانج لتحقيق أداء أعلى وكفاءة أكبر من خلال تقنيات مثل استخدام قاعدة بيانات الحدود.