**Numerical Analysis And Computation Lab Final**

**Algorithm: Gauss Forward Interpolation**

1. **Input:**
   * **A set of data points (x0,y0),(x1,y1),…,(xn,yn) are equidistant.**
   * **The value x where you want to estimate y(x).**
2. **Preprocessing:**
   * **Compute the forward differences Δky0 using: Δyi = yi+1−yi .**
   * **Repeat this process to compute higher-order differences.**
3. **Determine the value of uuu:**
   * **Compute u=x−x0/ hu​​, where h is the uniform interval between xi.**
4. **Apply the Gauss Forward Interpolation Formula:**

**y(x)=y0+ uΔy0+ u(u−1)/ 2! \* Δ^2y0+u(u−1)(u−2)/ 3! \*Δ^3y0+ …………**

**Here:**

* + **u is the normalized difference.**
  + **Δ^ky0 are the forward differences.**

1. **Terminate:**
   * **Stop the computation once the required accuracy or the highest difference level (based on the number of data points) is reached.**
2. **Output:**
   * **The interpolated value y(x).**

**Code for guass forward interpolation:**

**% Gauss Forward Interpolation Method**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 2.5; % The point at which interpolation is to be performed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the forward difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = 1:(n-j+1)**

**diff\_table(i, j) = diff\_table(i+1, j-1) - diff\_table(i, j-1);**

**end**

**end**

**% Display the forward difference table**

**disp('Forward Difference Table:');**

**disp(diff\_table);**

**% Calculate t**

**t = (xp - x(1)) / h;**

**% Perform Gauss forward interpolation**

**yp = diff\_table(1, 1); % Initialize interpolated value with f(x0)**

**factorial\_term = 1; % To hold factorial values**

**t\_term = 1; % To hold t terms**

**for j = 1:n-1**

**t\_term = t\_term \* (t - (j-1));**

**factorial\_term = factorial\_term \* j;**

**yp = yp + (t\_term \* diff\_table(1, j+1)) / factorial\_term;**

**end**

**% Display the result**

**fprintf('The interpolated value at x = %.2f is y = %.6f\n', xp, yp);**

**Stirlings interpolation:**

**Algorithm :**

1. **Input:**
   * **A set of nnn data points (x0,y0),(x1,y1),…,(xn−1,yn−1), where xi are equidistant.**
   * **The value x for which y(x) needs to be interpolated.**
2. **Preprocessing:**
   * **Calculate the forward differences Δy, Δ^2y, Δ^3y..., up to the highest-order difference.**
3. **Identify the Central Point:**
   * **Find the central point xm such that xm ​ is the closest to x.**
4. **Calculate the Value of uuu:**
   * **Compute u=x−xm/ h ,where h is the uniform interval between xi.**
5. **Apply Stirling’s Interpolation Formula:**
   * **Stirling’s formula is given by:**
   * **y(x)=ym+ u/1! \*Δym+ u2/2!\* Δ^2ym+ u(u^2−1)/3!\*(Δ^3ym−1 + Δ^3ym)/2 + ……………**
   * **Alternate the terms from the central point xm and use both forward and backward differences.**
6. **Compute Higher Terms (if needed):**
   * **Include additional terms until the required precision is achieved.**
7. **Output:**
   * **The interpolated value y(x).**

**Code for stirlings:**

**% Stirling's Interpolation Method**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 3.5; % The point at which interpolation is to be performed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the central difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = 1:(n-j+1)**

**diff\_table(i, j) = diff\_table(i+1, j-1) - diff\_table(i, j-1);**

**end**

**end**

**% Display the central difference table**

**disp('Central Difference Table:');**

**disp(diff\_table);**

**% Find the central point index**

**mid = ceil(n / 2);**

**% Calculate t**

**t = (xp - x(mid)) / h;**

**% Stirling's interpolation formula**

**yp = diff\_table(mid, 1); % Initialize interpolated value with f(x\_mid)**

**factorial\_term = 1; % To hold factorial values**

**t2 = t \* t; % To hold t^2**

**t\_term = t; % To hold t terms**

**% Perform interpolation**

**for j = 1:n-1**

**if mod(j, 2) == 1 % Odd terms**

**if mod(j, 4) == 1 % Positive term**

**yp = yp + (t\_term \* diff\_table(mid - floor(j / 2), j+1)) / factorial\_term;**

**else % Negative term**

**yp = yp + (t\_term \* diff\_table(mid - floor(j / 2) + 1, j+1)) / factorial\_term;**

**end**

**t\_term = t\_term \* t2;**

**else % Even terms**

**factorial\_term = factorial\_term \* j;**

**yp = yp + (t2 \* diff\_table(mid - j / 2, j+1)) / factorial\_term;**

**t2 = t2 \* (t2 - (j-1));**

**end**

**end**

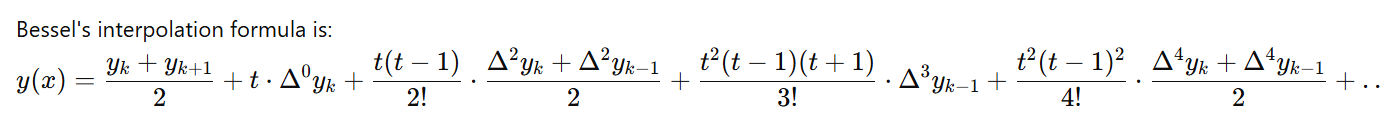
**% Display the result**

**fprintf('The interpolated value at x = %.2f is y = %.6f\n', xp, yp);**

**Bessels Interpolation:**

**Algorithm:**

1. **Given Data:**
   * **A set of evenly spaced data points (x0,y0), (x1,y1),…, (xn,yn).**
   * **The value x at which y(x) needs to be interpolated.**
   * **Step size h=x1−x0**
2. **Find the central index:**
   * **Identify the central point xk​ is closest to x.**
   * **Let t=x−xk/ h ​​ (normalized distance from the central point).**
3. **Compute central differences:**
4. **Apply Bessel’s formula:**

****

1. **Stop when the terms become negligible:**
   * **Continue adding terms from the formula until the desired accuracy is achieved or the additional terms become negligible.**
2. **Return the interpolated value y(x).**

**Steps in Practice:**

1. **Construct the central difference table.**
2. **Identify k such that xk​ is closest to x.**
3. **Use Bessel’s formula step by step to calculate the interpolated value.**

**Code for bessels**

**% Bessel's Interpolation Method**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 3.5; % The point at which interpolation is to be performed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the central difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = 1:(n-j+1)**

**diff\_table(i, j) = diff\_table(i+1, j-1) - diff\_table(i, j-1);**

**end**

**end**

**% Display the central difference table**

**disp('Central Difference Table:');**

**disp(diff\_table);**

**% Find the central point index**

**mid = ceil((n - 1) / 2); % Approximate middle point**

**% Calculate t**

**t = (xp - x(mid)) / h;**

**% Initialize interpolated value**

**yp = (diff\_table(mid, 1) + diff\_table(mid + 1, 1)) / 2; % Average of f(mid) and f(mid+1)**

**% Perform Bessel's interpolation**

**% Odd-order terms**

**odd\_factorial = 1; % Factorial term**

**odd\_t\_term = t - 0.5; % For odd-order terms**

**for j = 1:2:(n-1)**

**odd\_factorial = odd\_factorial \* j;**

**yp = yp + (odd\_t\_term \* diff\_table(mid - (j - 1) / 2, j + 1)) / odd\_factorial;**

**odd\_t\_term = odd\_t\_term \* (t - (j + 1) / 2);**

**end**

**% Even-order terms**

**even\_factorial = 1; % Factorial term**

**even\_t\_term = t; % For even-order terms**

**for j = 2:2:n**

**even\_factorial = even\_factorial \* j;**

**yp = yp + (even\_t\_term \* (diff\_table(mid - j / 2 + 1, j + 1) + diff\_table(mid - j / 2, j + 1)) / 2) / even\_factorial;**

**even\_t\_term = even\_t\_term \* (t - j / 2);**

**end**

**% Display the result**

**fprintf('The interpolated value at x = %.2f is y = %.6f\n', xp, yp);**

**Numerical differentiation using forward difference:**

**Algorithm:**

1. **Input:**
   * **A function f(x) for a table of values for f(x).**
   * **The value x0 at which the derivative is to be computed.**
   * **The step size h (spacing between consecutive x-values).**
2. **Forward Difference Formula:**
   * **First-order derivative: f′(x0)≈f(x1)−f(x0)/h, where x1= x0 + h.**
   * **Second-order derivative: f′′(x0)≈f(x2)−2f(x1)+f(x0)/ h^2, where x2=x0 + 2h**
3. **Procedure:**
   * **For the first derivative, calculate the difference between the next point and the current point, divided by h.**
   * **For the second derivative, use the values at the next two points and the current point.**
   * **Ensure the step size h is consistent and small enough for accuracy.**
4. **Output:**
   * **Return the approximated derivative(s).**

**Code for forward difference:**

**% Numerical Differentiation using Forward Difference**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 3; % The point at which derivative is to be computed**

**% Step size (assuming uniform spacing)**

**h = x(2) - x(1);**

**% Find the index of the point nearest to xp**

**[~, idx] = min(abs(x - xp));**

**if idx == length(x)**

**error('Forward difference is not applicable at the last point.');**

**end**

**% Forward difference formula for the first derivative**

**df\_dx = (y(idx + 1) - y(idx)) / h;**

**% Display the result**

**fprintf('The first derivative at x = %.2f is df/dx = %.6f\n', xp, df\_dx);**

**Newton backward differentiation:**

**Algo:**

1. **Input:**
   * **A set of evenly spaced x-values and their corresponding f(x)-values.**
   * **The value x0​ where the derivative needs to be calculated.**
   * **The step size h (difference between consecutive x-values).**
2. **Backward Difference Formulas:**
   * **First-order derivative: f′(x0)≈f(x0)−f(x−1)/h,where x−1=x0 − h.**
   * **Second-order derivative: f′′(x0)≈f(x0)−2f(x−1)+f(x−2)/h^2,where x−2=x0 − 2h**
3. **Procedure:**
   * **Identify the index iii of x0 in the dataset.**
   * **For the first derivative, compute the difference between the current point and the previous point, divided by hhh.**
   * **For the second derivative, use values at x0x​, x−1, and x−2.**
4. **Output:**
   * **Return the approximated derivative(s).**

**Code backward difference:**

**% Numerical Differentiation using Newton's Backward Difference Formula**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 5; % The point at which derivative is to be computed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the backward difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = j:n**

**diff\_table(i, j) = diff\_table(i, j-1) - diff\_table(i-1, j-1);**

**end**

**end**

**% Display the backward difference table**

**disp('Backward Difference Table:');**

**disp(diff\_table);**

**% Find the index of the given point xp**

**idx = find(x == xp, 1);**

**if isempty(idx)**

**error('Point xp is not in the given data points.');**

**elseif idx == 1**

**error('Backward difference is not applicable at the first point.');**

**end**

**% Newton's backward difference formula for the first derivative**

**t = (xp - x(idx)) / h;**

**df\_dx = diff\_table(idx, 2) / h;**

**% Higher-order terms can also be added if needed**

**% Example: Add second-order term**

**if idx > 2**

**df\_dx = df\_dx - (t + 1) \* diff\_table(idx, 3) / (2 \* h);**

**end**

**% Display the result**

**fprintf('The first derivative at x = %.2f is df/dx = %.6f\n', xp, df\_dx);**

**Numerical differentiation using sterlings formula:**

**Algo:**

1. **Input Data:**
   * **Collect evenly spaced xxx-values and their corresponding y=f(x) values.**
   * **Identify the point x0​ where the derivative needs to be calculated.**
2. **Find the Central Point:**
   * **Determine the middle value in the x-values (this is the central point).**
   * **Compute the parameter t, which represents the relative position of x0​ to the central point.**
3. **Construct the Difference Table:**
   * **Create a table where each row calculates the difference between consecutive values of y.**
   * **Continue until you’ve computed all higher-order differences.**
4. **Combine Differences:**
   * **Use the differences from the table to calculate approximations of the first and second derivatives.**
   * **Focus on the rows around the central point for better accuracy.**
5. **Stop When Accurate:**
   * **Continue combining terms until adding more differences has no significant impact on the result.**
6. **Output the Results:**
   * **Return the approximated values of the first and second derivatives at x0.**

**Code:**

**% Numerical Differentiation using Stirling's Formula**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 3.5; % The point at which derivative is to be computed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the central difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = 1:(n-j+1)**

**diff\_table(i, j) = diff\_table(i+1, j-1) - diff\_table(i, j-1);**

**end**

**end**

**% Display the central difference table**

**disp('Central Difference Table:');**

**disp(diff\_table);**

**% Find the central point index**

**mid = ceil((n + 1) / 2);**

**% Calculate t**

**t = (xp - x(mid)) / h;**

**% Stirling's formula for first derivative**

**% Initialize the derivative**

**df\_dx = diff\_table(mid, 2) / (2 \* h); % First term: Δy / 2h**

**% Compute higher-order terms**

**t2 = t^2; % t^2**

**factorial\_term = 1; % Factorial for the denominator**

**odd\_t\_product = t^2 - 1; % (t^2 - 1) for odd terms**

**for j = 3:2:n-1 % Odd terms**

**factorial\_term = factorial\_term \* j;**

**df\_dx = df\_dx + (odd\_t\_product \* diff\_table(mid - (j - 1) / 2, j + 1)) / (factorial\_term \* h);**

**odd\_t\_product = odd\_t\_product \* (t^2 - (j - 1)^2);**

**end**

**% Display the result**

**fprintf('The first derivative at x = %.2f is df/dx = %.6f\n', xp, df\_dx);**

**Numerical Differentiation Using Bessel's Formula**

**Algorithm**

1. **Input Data:**
   * **Collect evenly spaced x-values and their corresponding y=f(x) values.**
   * **Specify the point x0​ where the derivative is to be calculated.**
2. **Identify the Central Interval:**
   * **Find the interval of x-values around x0 such that x0​ lies as close to the middle as possible.**
3. **Construct the Central Difference Table:**
   * **Calculate the differences (Δy) for the given y-values:**
     + **First differences (Δ^1y): Difference between adjacent yyy-values.**
     + **Second differences (Δ^2y): Difference between adjacent first differences.**
     + **Continue for higher-order differences.**
4. **Compute Derivatives Using Bessel's Formula:**
   * **Use the central differences to approximate:**
     + **The first derivative at x0.**
     + **The second derivative at x0​.**
   * **Focus on terms involving values near x0​ to achieve better accuracy.**
5. **Stop When Accurate:**
   * **Include terms from the difference table until the result stabilizes (adding more terms makes no significant change).**
6. **Output the Results:**
   * **Return the approximated first and second derivatives at x0​.**

**Code:**

**% Numerical Differentiation using Bessel's Formula**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**xp = 3.5; % The point at which derivative is to be computed**

**n = length(x); % Number of data points**

**h = x(2) - x(1); % Step size (assuming uniform spacing)**

**% Construct the central difference table**

**diff\_table = zeros(n, n);**

**diff\_table(:, 1) = y'; % First column is y values**

**for j = 2:n**

**for i = 1:(n-j+1)**

**diff\_table(i, j) = diff\_table(i+1, j-1) - diff\_table(i, j-1);**

**end**

**end**

**% Display the central difference table**

**disp('Central Difference Table:');**

**disp(diff\_table);**

**% Find the index of the central point**

**mid = ceil((n - 1) / 2);**

**% Calculate t**

**t = (xp - x(mid)) / h;**

**% Initialize derivative value**

**df\_dx = (diff\_table(mid, 1) - diff\_table(mid + 1, 1)) / (2 \* h); % First term**

**% Bessel's formula includes higher-order terms**

**t\_term = (t^2 - 1); % Initialize t^2 - 1 for higher-order terms**

**factorial\_term = 2; % Start with 2! for denominator**

**for j = 3:2:n-1**

**df\_dx = df\_dx + t\_term \* (diff\_table(mid - (j-1)/2, j) - diff\_table(mid - (j-1)/2 + 1, j)) / (factorial\_term \* 2 \* h);**

**factorial\_term = factorial\_term \* (j + 1) \* (j + 2); % Update factorial**

**t\_term = t\_term \* (t^2 - j^2); % Update t-term for next term**

**end**

**% Display the result**

**fprintf('The first derivative at x = %.2f is df/dx = %.6f\n', xp, df\_dx);**

**Trapezoidal formula:**

**Algo:**

**Input:**

* **Define the function f(x) you want to integrate.**
* **Specify the interval [a,b] over which to integrate.**
* **Choose the number of subintervals n to divide the range [a,b].**

**Divide the Interval:**

* **Compute the width of each subinterval as h=b−a/n**
* **Determine the x-values at the boundaries of each subinterval.**

**Compute Area:**

* **Calculate the function values at each x point.**
* **Add the first and last function values directly.**
* **Add the remaining intermediate function values twice (since they are shared between two trapezoids).**

**Multiply by the Step Size:**

* **Multiply the sum by h/2 to get the final approximate integral.**

**Output:**

* **Display the computed integral value.**

**Code:**

**% Numerical Integration using Trapezoidal Rule**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**% Step size (assuming uniform spacing)**

**h = x(2) - x(1);**

**% Apply the trapezoidal rule**

**n = length(x); % Number of points**

**area = 0; % Initialize area**

**% Compute the integral**

**for i = 1:(n-1)**

**area = area + (y(i) + y(i+1)) \* h / 2; % Trapezoidal rule formula**

**end**

**% Display the result**

**fprintf('The integral using the Trapezoidal Rule is: %.6f\n', area);**

**Simpsons formula :**

**Algo:**

**Input:**

* **Define the function f(x) to integrate.**
* **Specify the interval [a,b] over which integration is needed.**
* **Choose the number of subintervals nnn (must be even).**

**Divide the Interval:**

* **Compute the width of each subinterval as h=b−a/n.**
* **Determine the x-values at the boundaries of each subinterval.**

**Calculate Function Values:**

* **Compute f(x) at all x-points:**
  + **Add the function values at the first and last points directly.**
  + **Add the function values at odd-indexed points (multiplied by 4).**
  + **Add the function values at even-indexed points (multiplied by 2), excluding the first and last.**

**Multiply by the Step Size:**

* **Multiply the sum by h/3​ to get the final integral.**

**Output:**

* **Display the computed integral value.**

**Code:**

**% Numerical Integration using Simpson's Rule**

**clc;**

**clear;**

**% Input data**

**x = [1, 2, 3, 4, 5]; % Given x values (evenly spaced)**

**y = [2.0, 4.1, 8.3, 16.2, 32.5]; % Corresponding f(x) values**

**% Check if the number of intervals is even**

**n = length(x);**

**if mod(n-1, 2) ~= 0**

**error('Simpson''s Rule requires an even number of intervals (odd number of points).');**

**end**

**% Step size (assuming uniform spacing)**

**h = x(2) - x(1);**

**% Apply Simpson's Rule**

**area = y(1) + y(end); % Add first and last terms**

**% Add terms with coefficients 4 and 2**

**for i = 2:n-1**

**if mod(i, 2) == 0**

**area = area + 4 \* y(i); % Coefficient 4 for odd indices**

**else**

**area = area + 2 \* y(i); % Coefficient 2 for even indices**

**end**

**end**

**% Multiply by h/3**

**area = area \* h / 3;**

**% Display the result**

**fprintf('The integral using Simpson''s Rule is: %.6f\n', area);**

**Euler’s Method:**

**Algorithm:  
Start with initial values x0,y0;**

**Calculate yn+1 using the formula iteratively for n steps.**

**Code:**

**f = @(x,y) -2\*x\*y;**

**x0 = 0;**

**y0 = 1;**

**h = 0.1;**

**n = 10;**

**x = x0;**

**y = y0;**

**for i = 1:n**

**y = y+h\*f(x,y);**

**x = x + h;**

**disp([‘Step ‘, num2str(i), ‘ : x = ‘, num2str(x), ‘, y = ’, num2str(y)]);**

**end**

**Rungge Kutta Method:**

**Code:**

**f = @(x,y) -2\*x\*y;**

**x0 = 0;**

**y0 = 1;**

**h = 0.1;**

**n = 10;**

**x = x0;**

**y = y0;**

**for i = 1:n**

**k1 = f(x,y);**

**k2 = f(x + h/2, y + h/2 \* k1);**

**k3 = f(x + h/2, y + h/2 \* k2);**

**k4 = f(x + h, y + h \* k3);**

**y = y + h/6 \* (k1 + 2\*k2 + 2\*k3 + k4);**

**x = x + h;**

**disp([‘Step ‘, num2str(i), ‘ : x = ‘, num2str(x), ‘, y = ’, num2str(y)]);**

**end**

**Plotting of a Graph(3 functions on a single figure):**

**% Define the x values**

**x = linspace(-2\*pi, 2\*pi, 100); % 100 points from -2π to 2π**

**% Define the functions**

**y1 = sin(x); % Function 1: sin(x)**

**y2 = cos(x); % Function 2: cos(x)**

**y3 = sin(x) .\* cos(x); % Function 3: sin(x) \* cos(x)**

**% Create the figure**

**figure;**

**% Plot the first function**

**plot(x, y1, 'r--', 'LineWidth', 1.5); % Red dashed line**

**hold on; % Hold the plot to overlay other plots**

**% Plot the second function**

**plot(x, y2, 'b-', 'LineWidth', 1.5); % Blue solid line**

**% Plot the third function**

**plot(x, y3, 'g:', 'LineWidth', 1.5); % Green dotted line**

**% Add grid lines**

**grid on;**

**% Add labels and title**

**xlabel('x-axis'); % Label for x-axis**

**ylabel('y-axis'); % Label for y-axis**

**title('Comparison of Three Functions'); % Plot title**

**% Add legend**

**legend('sin(x)', 'cos(x)', 'sin(x) \* cos(x)', 'Location', 'best');**

**% Set axis limits for better visibility (optional)**

**xlim([-2\*pi, 2\*pi]);**

**ylim([-1.5, 1.5]);**

**% Display the figure**

**hold off;**

**Matrix Addition, Subtraction, Product, and Transpose**

**% Define two matrices**

**A = [1 2 3; 4 5 6; 7 8 9];**

**B = [9 8 7; 6 5 4; 3 2 1];**

**% Matrix addition**

**C\_add = A + B;**

**% Matrix subtraction**

**C\_sub = A - B;**

**% Matrix multiplication (standard matrix product)**

**C\_mul = A \* B;**

**% Transpose of matrices**

**A\_transpose = A';**

**B\_transpose = B';**

**% Display results**

**disp('Matrix A:');**

**disp(A);**

**disp('Matrix B:');**

**disp(B);**

**disp('A + B:');**

**disp(C\_add);**

**disp('A - B:');**

**disp(C\_sub);**

**disp('A \* B:');**

**disp(C\_mul);**

**disp('Transpose of A:');**

**disp(A\_transpose);**

**disp('Transpose of B:');**

**disp(B\_transpose);**

**Determinant of a Matrix**

**% Define a matrix**

**M = [4 2 1; 3 5 7; 8 9 6];**

**% Compute determinant**

**determinant = det(M);**

**% Display the determinant**

**disp('Matrix M:');**

**disp(M);**

**disp(['Determinant of M: ', num2str(determinant)]);**

**Check if a Matrix is Invertible and Compute Its Inverse**

**% Define a matrix**

**M = [4 2 1; 3 5 7; 8 9 6];**

**% Check if the determinant is non-zero**

**determinant = det(M);**

**if determinant ~= 0**

**disp('The matrix M is invertible.');**

**% Compute the inverse**

**M\_inverse = inv(M);**

**% Display the inverse**

**disp('Inverse of M:');**

**disp(M\_inverse);**

**else**

**disp('The matrix M is not invertible.');**

**end**

**Question:**

**b. Using MATLAB, perform the following matrix operations for**

**Compute the eigenvalues and eigenvectors of AAA.**

* **Check if AAA is singular or non-singular.**
* **Create a new matrix BBB by adding 5 to all elements of AAA, then compute A×BA \times BA×B.**

**Solution:**

**% Define the matrix A**

**A = [1 2 3; 4 5 6; 7 8 9];**

**% 1. Compute eigenvalues and eigenvectors**

**[eigenvectors, eigenvalues] = eig(A);**

**disp('Eigenvalues:');**

**disp(diag(eigenvalues)); % Extract eigenvalues from the diagonal of the matrix**

**disp('Eigenvectors:');**

**disp(eigenvectors);**

**% 2. Check if A is singular or non-singular**

**determinant = det(A);**

**if determinant == 0**

**disp('Matrix A is singular.');**

**else**

**disp('Matrix A is non-singular.');**

**end**

**% 3. Create a new matrix B by adding 5 to all elements of A**

**B = A + 5;**

**% Compute the product of A and B**

**result = A \* B;**

**disp('Matrix B:');**

**disp(B);**

**disp('Result of A \* B:');**

**disp(result);**

**Power Method**

**% Define the matrix C**

**C = [5 2; 2 1];**

**% Initial guess x0**

**x0 = [1; 0];**

**% Maximum number of iterations and tolerance for convergence**

**max\_iter = 100;**

**tolerance = 1e-6;**

**% Power Method**

**x = x0; % Initial vector**

**lambda\_old = 0; % Placeholder for eigenvalue**

**for k = 1:max\_iter**

**y = C \* x; % Multiply matrix with vector**

**lambda\_new = max(abs(y)); % Approximate dominant eigenvalue**

**x = y / norm(y); % Normalize the vector**

**% Check for convergence**

**if abs(lambda\_new - lambda\_old) < tolerance**

**fprintf('Converged after %d iterations.\n', k);**

**break;**

**end**

**lambda\_old = lambda\_new;**

**end**

**% Display results**

**disp('Dominant Eigenvalue:');**

**disp(lambda\_new);**

**disp('Corresponding Eigenvector:');**

**disp(x);**

Clone URL

https://qasimnauman: ghp\_RgQwwZpXTUd8cYAjREPl4n2EwDJeQ80ZLGic@github.com/qasimnauman/matlabscripts.git