CS/MATH 375: Homework 9

Adam Fasulo

November 3, 2025

Problem 1: Interpolation with Chebyshev Points

In this problem, we revisit the function $f(x) = 1/(1+25x^2)$ from the previous homework. Instead of using equally spaced points, we now use **Chebyshev points**, which are clustered near the endpoints of the interval $[-1, 1]$. The goal is to see if this choice of points mitigates Runge's phenomenon.

The Chebyshev points are given by the formula:

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n+2}\right), \quad i = 0, \ldots, n$$

The experiment was run for $n = 2, 4, \ldots, 20$ using Lagrange interpolation. The results are shown in Figure 1.

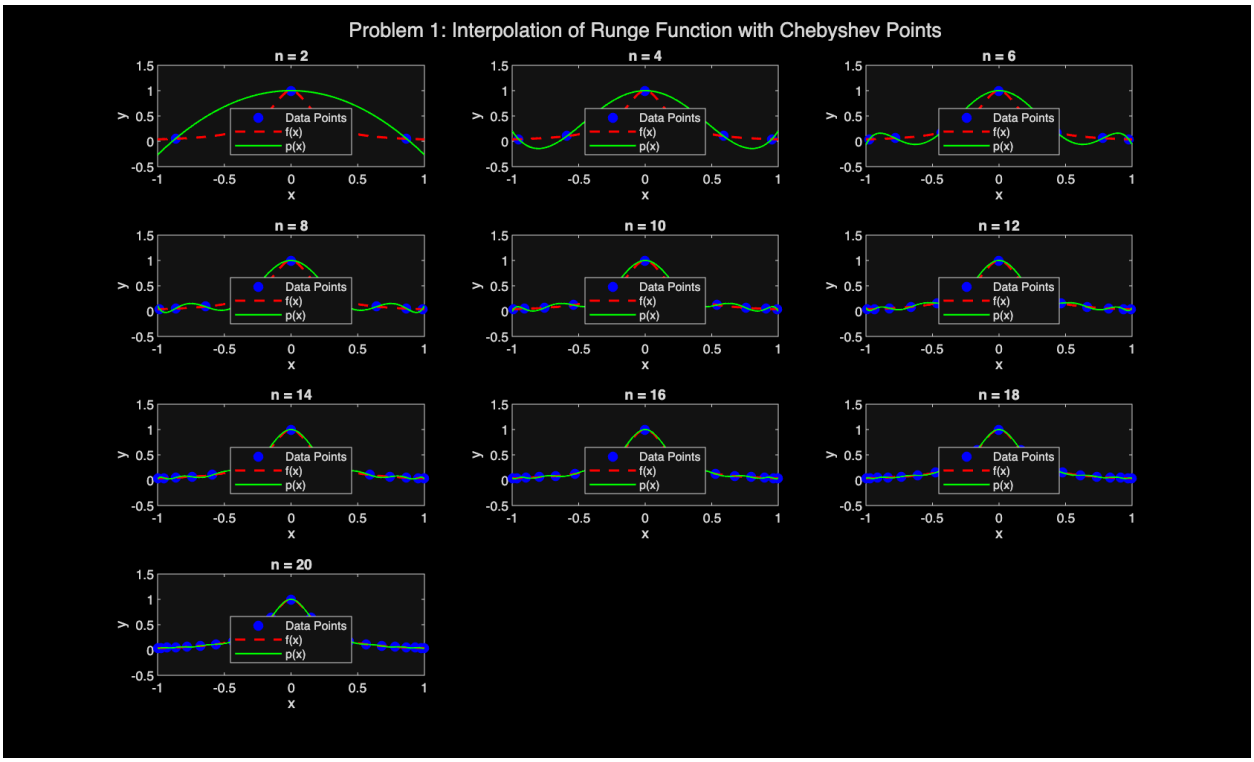

Figure 1: Interpolation of $f(x)$ using Chebyshev points for $n = 2, 4, \ldots, 20$.

As seen in the plots, the interpolation is dramatically improved. The wild oscillations near the endpoints are gone. As n increases, the interpolating polynomial converges nicely to the actual function across the entire interval.

Does the interpolation fail?

No, the interpolation does not fail for any of the tested values of n between 2 and 20. Unlike with equispaced points, where the method failed numerically, the Chebyshev points provide a stable and accurate interpolation that improves as n increases. This demonstrates that the choice of interpolation points is crucial for achieving convergence for high-degree polynomial interpolation.

Problem 2: Convergence for Different Functions

Here, we examine the convergence of Lagrange interpolation with Chebyshev points for two different functions: $f_1(x) = \sin(x)$ and $f_2(x) = |x|$. We calculated the maximum error on a fine grid for $n = 1, \ldots, 16$. The results are plotted in Figure 2.

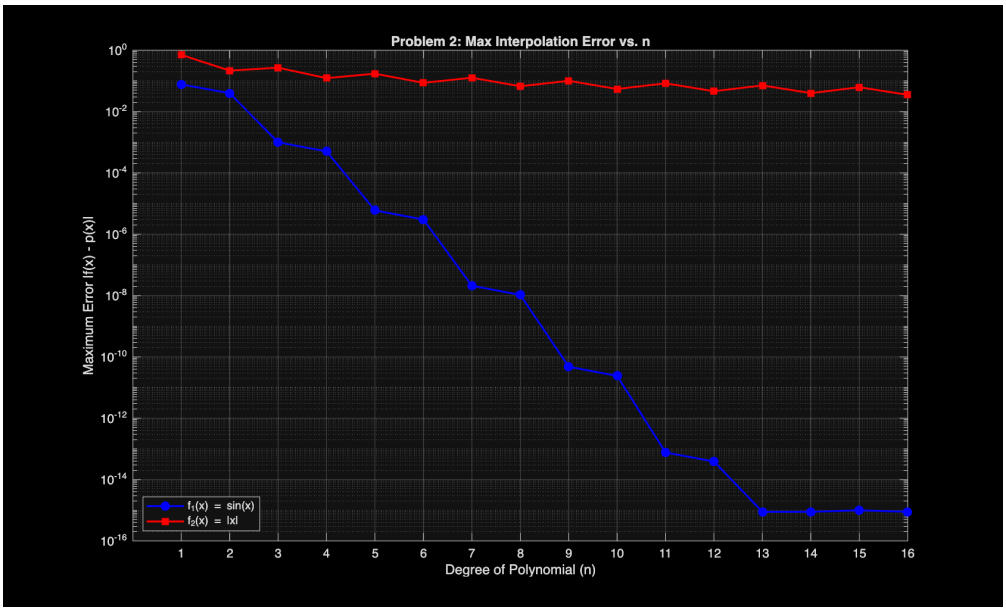

Figure 2: Maximum interpolation error vs. n for $f_1(x) = \sin(x)$ and $f_2(x) = |x|$.

Is the interpolating polynomial converging?

For $f_1(x) = \sin(x)$: **Yes, it is converging rapidly.** The error decreases exponentially (a straight line on a log-linear plot), which is characteristic of interpolating a smooth, infinitely differentiable function (like sine) with Chebyshev points. The error quickly reaches machine precision.

* **For $f_2(x) = |x|$:** **No, it is not converging well.** The error decreases, but very slowly (it is not a straight line on the log-linear plot).

What is going wrong?

The slow convergence for $f_2(x) = |x|$ is due to the function's lack of smoothness. Polynomial interpolation theory states that the convergence rate depends on the derivatives of the function. The function $f_2(x) = |x|$ has a **discontinuity in its first derivative** at $x = 0$ (a sharp corner). High-degree polynomials are infinitely smooth, so they struggle to approximate this "kink." This lack of smoothness severely limits the rate of convergence, even with the well-behaved Chebyshev points.

Problem 3: Natural Cubic Splines

We return to approximating $f(x) = 1/(1 + 25x^2)$, but this time with a natural cubic spline using $n = 10$ equispaced intervals (11 knots).

(a) MATLAB function spline3 coeff.m

This function sets up and solves the tridiagonal system for the second derivatives (z_i) at the knots of a natural cubic spline, where $z_0 = z_n = 0$.

```
function z = spline3_coeff(t, y)
    % Takes knot vector t and values y and returns the vector z of
    % second derivatives for a natural cubic spline.
    n = length(t) - 1;
    h = t(2:n+1) - t(1:n);

    % Setup the tridiagonal system Az = v
    % A is an (n-1)x(n-1) matrix for the interior points z_1, ..., z_(n-1)

    % Main diagonal
    diag_main = 2 * (h(1:n-1) + h(2:n));

    % Sub and super diagonals
    diag_sub = h(2:n-1);
    diag_super = h(2:n-1);

    A = diag(diag_main) + diag(diag_super, 1) + diag(diag_sub, -1);

    % Right-hand side vector v
    v = 6 * ( (y(3:n+1)-y(2:n))./h(2:n) - (y(2:n)-y(1:n-1))./h(1:n-1) );

    % Solve for the interior z values
    z_interior = A \ v;

    % Form the final z vector with natural boundary conditions (z0=zn=0)
    z = [0; z_interior; 0];
end
```

Listing 1: spline3_coeff.m

(b) Spline Interpolation Plot and Analysis

Using the function above, we computed the natural cubic spline for $f(x)$ with 11 equispaced knots. The resulting spline and the original function are plotted in Figure 3.

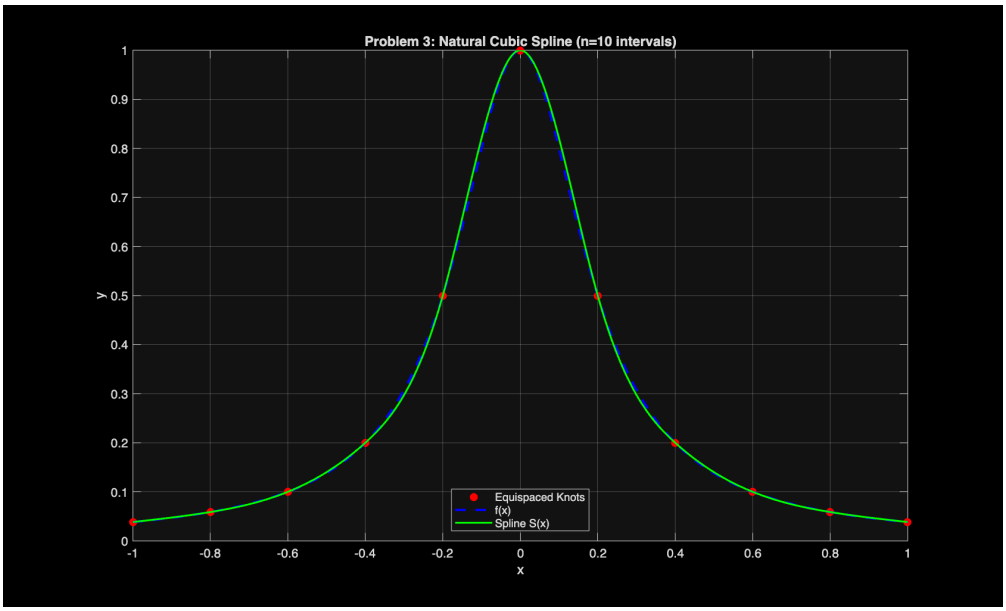

Figure 3: Natural cubic spline $S(x)$ vs. original function $f(x)$ for $n = 10$ equispaced intervals.

How does the spline behave relative to polynomial interpolation?

The natural cubic spline provides a **much better approximation** than the high-degree polynomial interpolation with equispaced points from the last homework.

* The spline is visually indistinguishable from the function $f(x)$ over most of the interval. * Most importantly, the spline **does not exhibit Runge's phenomenon**. It remains smooth and follows the function closely near the endpoints without any wild oscillations. * This is because a spline is a **piecewise polynomial**. It uses low-degree (cubic) polynomials between each pair of knots, which avoids the instability inherent in a single high-degree polynomial. It provides a stable and accurate approximation even with equispaced points.

Appendix: MATLAB Scripts

In accordance with the feedback from the previous assignment, all MATLAB code used to generate the results and figures in this report is included below.

Main Script: 'hw9 main.m'

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main Script for HW 9
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; close all;

% Define the functions
f_runge = @(x) 1 ./ (1 + 25*x.^2);
f1 = @(x) sin(x);
f2 = @(x) abs(x);

%% --- Problem 1: Chebyshev Interpolation for Runge Function ---
```

```
n_values = 2:2:20;
figure('Name', 'Chebyshev Interpolation');
t = tiledlayout(ceil(length(n_values)/3), 3);
title(t, 'Problem 1: Interpolation with Chebyshev Points');

for n = n_values
    % 1. Generate Chebyshev points
    i = 0:n;
    x = cos((2*i + 1) * pi / (2*n + 2))';
    y = f_runge(x);

    % 2. Generate fine grid for plotting
    n_fine = n * 100;
    x_fine = linspace(-1, 1, n_fine)';
    y_fine_f = f_runge(x_fine);

    % 3. Evaluate Lagrange polynomial on the fine grid
    p_fine = eval_lagrange(x, y, x_fine);

    % 4. Plot
    nexttile;
    plot(x, y, 'bo', 'MarkerFaceColor', 'b'); % Data points
    hold on;
    plot(x_fine, y_fine_f, 'r--', 'LineWidth', 1.5); % Original function
    plot(x_fine, p_fine, 'g-', 'LineWidth', 1); % Interpolating polynomial
    hold off;

    % 5. Format plot
    title(['n = ' num2str(n)]);
    xlabel('x'); ylabel('y');
    legend('Data Points', 'f(x)', 'p(x)', 'Location', 'South');
    ylim([-0.5, 1.5]);
end
sgtitle('Problem 1: Interpolation of Runge Function with Chebyshev Points');
saveas(gcf, 'chebyshev_runge.png');


%% --- Problem 2: Convergence for sin(x) and |x| ---
n_values_conv = 1:16;
max_errors_f1 = zeros(length(n_values_conv), 1);
max_errors_f2 = zeros(length(n_values_conv), 1);
x_fine_err = linspace(-1, 1, 2000)'; % Very fine grid for error check

for k = 1:length(n_values_conv)
    n = n_values_conv(k);

    % Generate Chebyshev nodes
    i = 0:n;
    x_nodes = cos((2*i + 1) * pi / (2*n + 2))';

    % --- Function 1: sin(x) ---
    y1_nodes = f1(x_nodes);
    p1_fine = eval_lagrange(x_nodes, y1_nodes, x_fine_err);
    max_errors_f1(k) = max(abs(p1_fine - f1(x_fine_err)));

    % --- Function 2: |x| ---
    y2_nodes = f2(x_nodes);
    p2_fine = eval_lagrange(x_nodes, y2_nodes, x_fine_err);
    max_errors_f2(k) = max(abs(p2_fine - f2(x_fine_err)));
```

```
end

% Plot the errors
figure('Name', 'Convergence Errors');
semilogy(n_values_conv, max_errors_f1, 'b-o', 'LineWidth', 1.5, 'MarkerFaceColor',
    'b');
hold on;
semilogy(n_values_conv, max_errors_f2, 'r-s', 'LineWidth', 1.5, 'MarkerFaceColor',
    'r');
hold off;
title('Problem 2: Max Interpolation Error vs. n');
xlabel('Degree of Polynomial (n)');
ylabel('Maximum Error |f(x) - p(x)|');
legend('f_1(x) = sin(x)', 'f_2(x) = |x|', 'Location', 'SouthWest');
grid on;
xticks(1:16);
saveas(gcf, 'convergence_errors.png');


%% --- Problem 3: Natural Cubic Spline ---
n_spline = 10;
t_knots = linspace(-1, 1, n_spline + 1)';
y_vals = f_runge(t_knots);

% (a) Get spline coefficients
z = spline3_coeff(t_knots, y_vals);

% (b) Evaluate spline on a fine grid
% Assuming eval_spine.m is in the same directory
x_fine_spline = linspace(-1, 1, 500)';
S_x = eval_spine(t_knots, y_vals, z, x_fine_spline);

% Plot the result
figure('Name', 'Cubic Spline');
plot(t_knots, y_vals, 'ro', 'MarkerFaceColor', 'r', 'DisplayName', 'Equispaced
    Knots');
hold on;
plot(x_fine_spline, f_runge(x_fine_spline), 'b--', 'LineWidth', 2, 'DisplayName',
    'f(x)');
plot(x_fine_spline, S_x, 'g-', 'LineWidth', 1.5, 'DisplayName', 'Spline S(x)');
hold off;
title(['Problem 3: Natural Cubic Spline (n=', num2str(n_spline), ' intervals)']);
xlabel('x');
ylabel('y');
legend('Location', 'South');
grid on;
saveas(gcf, 'spline_vs_f.png');

%% --- Helper Functions from Last HW ---
function p_eval = eval_lagrange(x_nodes, y_nodes, x_eval)
    % Evaluates the Lagrange polynomial defined by
    % (x_nodes, y_nodes) at the points in x_eval.
    n = length(x_nodes) - 1;
    p_eval = zeros(size(x_eval));

    for j = 1:(n+1)
        L_j = 1;
        for k = 1:(n+1)
            if j ~= k
```

```
                L_j = L_j .* (x_eval - x_nodes(k)) / (x_nodes(j) - x_nodes(k));
            end
        end
        p_eval = p_eval + y_nodes(j) * L_j;
    end
end

function S = eval_spine(t, y, z, x_eval)
    % Evaluates the cubic spline at points x_eval
    n = length(t) - 1;
    S = zeros(size(x_eval));

    for i = 1:n
        % Find points in this interval
        idx = find(x_eval >= t(i) & x_eval <= t(i+1));
        if isempty(idx)
            continue;
        end

        h_i = t(i+1) - t(i);
        x = x_eval(idx);

        % Spline formula
        term1 = z(i+1)/(6*h_i) * (x - t(i)).^3;
        term2 = z(i)/(6*h_i) * (t(i+1) - x).^3;
        term3 = (y(i+1)/h_i - z(i+1)*h_i/6) * (x - t(i));
        term4 = (y(i)/h_i - z(i)*h_i/6) * (t(i+1) - x);

        S(idx) = term1 + term2 + term3 + term4;
    end
end
```

Listing 2: hw9_main.m