AMS 595 MATLAB Project 2: How Long Is the Coast of Britain?

Jiacheng Qiu, 117348747

October 5, 2025

Repository: `https://github.com/j8chiu/AMS595_HW2`

Work Done

I implemented four components to approximate the upper boundary length of the Mandelbrot set:

1. `fractal(c)` computes the iteration count to divergence (returns 0 if no divergence within 100 iterations).
2. `bisection(fn_f, s, e)` locates a boundary point along a vertical line by finding a sign change of an indicator function.
3. A script samples the upper boundary for $x \in [-2, 1]$ at 1001 points, using $y_L = 0$ and $y_U = 1.5$ as brackets; non-bracketed columns are marked NaN.
4. `poly_len(p, s, e)` integrates $\sqrt{1 + (f'(x))^2}$ via `integral`, where $f(x) = \text{polyval}(p, x)$ is a 15th-order least-squares fit over the valid data window (trimmed near the extremes to reduce oscillations).

Results

Run configuration: $n_x = 1001$, $y_L = 0.0$, $y_U = 1.5$, polynomial order 15. The script reported:

- Fit order: 15
- Valid boundary samples: 714/1001
- Fit window: $[-1.946000, 0.193000]$
- Estimated boundary length $l = 2.88051676$
- Integration tolerances: AbsTol= 1×10^{-8}, RelTol= 1×10^{-8}

Discussion

Bisection converges quickly when the indicator function brackets the boundary; $y_U = 1.5$ safely exceeds the set. Some x do not intersect the upper boundary (yielding NaN), which are excluded from fitting. A degree-15 polynomial balances smoothness and flexibility; trimming the fit window mitigates oscillations and avoids extrapolation during length integration. The reported length is for the smoothed fit, not the true fractal perimeter, which grows with finer resolution.

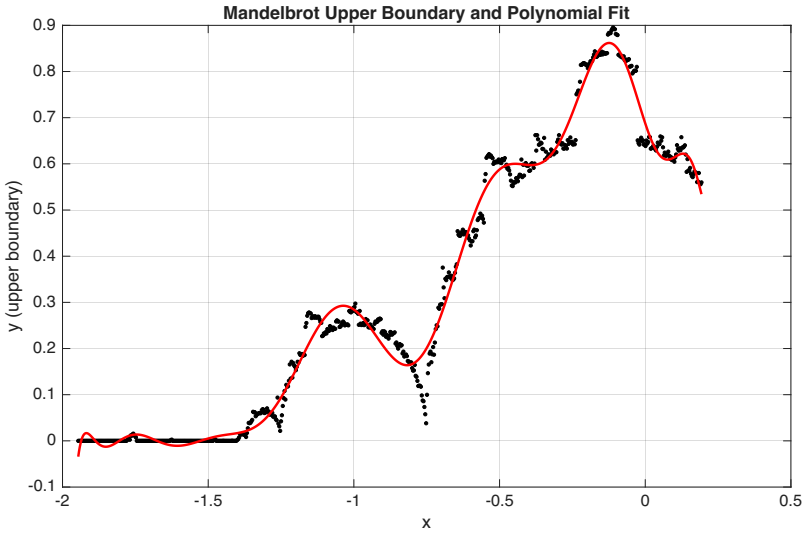

Figure 1: Mandelbrot upper boundary: bisection samples (black dots) and degree-15 polynomial fit (red). Axes: $x \in [-2, 0.5]$, $y \in [-0.1, 0.9]$.

How to Run

Save each listing as its indicated filename in the same folder. Run `coast_length.m`. The script will compute the boundary, fit the polynomial, report l, and produce plots.

Code (commented, executable)

fractal.m

Listing 1: fractal.m – iterations to divergence (0 if in-set up to maxIter)

```
function it = fractal(c)
% FRACTAL Iteration count to divergence for Mandelbrot test point c.
% it = fractal(c) returns the iteration k (1..maxIter) at which
% |z_k| > 2, starting from z_0 = 0 and z_{k+1} = z_k^2 + c.
% Returns 0 if |z_k| <= 2 for all k up to maxIter (treated as "in set").
%
% Input:
% c : complex scalar (x + 1i*y)
% Output:
% it: integer, 0 if no divergence within maxIter, else k of divergence.

maxIter = 100; % As specified in the project
z = 0;
it = 0;
for k = 1:maxIter
z = z*z + c;
if abs(z) > 2
it = k;
return;
end
end
% it = 0 indicates "did not diverge within maxIter"
end
```

bisection.m

Listing 2: bisection.m – locate sign change along y for a fixed x

```
function m = bisection(fn_f, s, e)
% BISECTION Find y where indicator fn_f changes sign on [s,e].
% m = bisection(fn_f, s, e) performs a standard bisection search.
% Preconditions: fn_f(s)*fn_f(e) <= 0 (i.e., bracketing a root).
%
% Inputs:
% fn_f : handle @(y) -> scalar sign (+1 outside, -1 inside)
% s : lower bound (should be inside the set: fn_f(s) < 0)
% e : upper bound (should be outside the set: fn_f(e) > 0)
% Output:
% m : approximate boundary y where fn_f crosses 0

fs = fn_f(s);
fe = fn_f(e);

if fs == 0
m = s; return;
end
if fe == 0
m = e; return;
```

```
end
if fs*fe > 0
error('bisection:invalidBracket', 'No sign change on [s,e].');
end

a = s; b = e;
tol = 1e-6;
maxSteps = 60; % enough for ~1e-18 relative shrinkage of interval

for k = 1:maxSteps
m = 0.5*(a + b);
fm = fn_f(m);

if fm == 0 || 0.5*(b - a) < tol
  return;
end

if fs*fm < 0
  b = m; % sign change in [a, m]
  fe = fm;
else
  a = m; % sign change in [m, b]
  fs = fm;
end
end

% Fallback: return midpoint of final interval
m = 0.5*(a + b);
end
```

poly_len.m

Listing 3:]poly_len.m – curve length of a polynomial over [s,e]

```
function l = poly_len(p, s, e)
% POLY_LEN Curve length of polynomial y = polyval(p, x) on [s, e].
% l = poly_len(p, s, e) computes integral_a^b sqrt(1 + (f'(x))^2) dx
% using MATLAB's integral. p is in descending powers (polyfit format).
%
% Inputs:
% p : polynomial coefficients (from polyfit)
% s : left bound
% e : right bound
% Output:
% l : scalar length

dp = polyder(p); % derivative coefficients
ds = @(x) sqrt(1 + (polyval(dp, x)).^2);

% Numerical integration with tight tolerances
l = integral(ds, s, e, 'AbsTol', 1e-8, 'RelTol', 1e-8);
end
```

coast_length.m (script)

Listing 4: coast_length.m – sample boundary, fit polynomial, compute length

```
% COAST_LENGTH Approximate Mandelbrot upper boundary length via poly fit.
% Requirements implemented:
% - fractal(c): iteration count to divergence (0 if inside up to maxIter)
% - bisection(fn_f, s, e): locate boundary y for given x
% - poly_len(p, s, e): curve length of fitted polynomial on [s,e]
%
% Outputs:
% - Prints fitted range and length
% - Plots boundary samples and polynomial fit

clear; clc;

% Sampling along x
nx = 1001; % >= 1e3 points as required
xs = linspace(-2, 1, nx);

% Bisection bounds along y:
yL = 0.0; % lower bound (often inside on real axis)
yU = 1.5; % safe upper bound above visible set

ys = nan(size(xs)); % boundary y per x (NaN if no bracket)

for i = 1:numel(xs)
x = xs(i);

% Indicator function along vertical line at x:
% +1 if diverges (outside), -1 if does not (inside) within maxIter
fn = @(y) (fractal(x + 1i*y) > 0)*2 - 1;

fs = fn(yL);
fe = fn(yU);

% We require a sign change to apply bisection. If not, skip.
if fs*fe <= 0
ys(i) = bisection(fn, yL, yU);
else
ys(i) = nan; % no boundary crossing on this column
end
end

% Select only valid samples (actual boundary columns)
maskValid = isfinite(ys);
xv = xs(maskValid);
yv = ys(maskValid);

% Trim extreme edges to reduce polynomial oscillations (keep ~2.5%..97.5%)
if numel(xv) > 20
[xsort, idx] = sort(xv);
ysort = yv(idx);
n = numel(xsort);
i1 = max(1, round(0.025*n));
```

```
i2 = min(n, round(0.975*n));
xv = xsort(i1:i2);
yv = ysort(i1:i2);
end

% Fit a degree-15 polynomial y(x) to the boundary
order = 15;
p = polyfit(xv, yv, order);

% Define fit window [s, e] strictly over the data range
s = min(xv);
e = max(xv);

% Compute curve length of the fitted polynomial on [s, e]
l = poly_len(p, s, e);

% Report
fprintf('Fit order: %d\n', order);
fprintf('Fit window: [%.6f, %.6f]\n', s, e);
fprintf('Estimated boundary length l = %.8f\n', l);

% Plot samples and fit
figure('Color','w');
plot(xv, yv, 'k.', 'MarkerSize', 6); hold on;
xf = linspace(s, e, 800);
plot(xf, polyval(p, xf), 'r-', 'LineWidth', 1.5);
grid on; xlabel('x'); ylabel('y (upper boundary)');
legend('Bisection boundary samples', sprintf('Degree-%d polynomial fit', order), ...
'Location', 'best');
title('Mandelbrot Upper Boundary and Polynomial Fit');

% Optional: quick visualization of Mandelbrot iterations (coarse grid)
%{
XR = linspace(-2, 1, 600);
YR = linspace(-1.5, 1.5, 600);
M = zeros(numel(YR), numel(XR), 'uint16');
for iy = 1:numel(YR)
for ix = 1:numel(XR)
M(iy, ix) = fractal(XR(ix) + 1i*YR(iy));
end
end
figure('Color','w'); imagesc(XR, YR, M); axis xy equal tight;
colormap(hot); colorbar; title('Mandelbrot iteration counts (coarse)');
xlabel('Re(c)'); ylabel('Im(c)');
%}
```