Project 2: Mandelbrot Fractal

Table of Contents

Creation of the set

Initialize variables/constants

```
POLYNOMIAL_ORDER = 15;
N = 1000;
displayGrid = zeros(N,N);

% Create arrays for the ranges of real and complex components and a
% meshgrid of their values
x = linspace(-2, 1, N);
y = linspace(-1, 1, N);
[X, Y] = meshgrid(x, y);

% Compute the complex points in the 1000x1000 meshgrid
mandelbrotPoints = X + Y * 1i;

% For every complex value in the mandelbrotPoints meshgrid, determine
% whether the complex number converges or diverges using the
% fractalIterationsToDivergence function.
% The function will return either the number of iterations (divergence)
% or -1 (convergence), and the values will be stored as such
for valX = 1:N
    for valY = 1:N
       iters = fractalIterationsToDivergence(mandelbrotPoints(valY, valX));
       if iters == -1
           displayGrid(valY, valX) = 100;
       else
           displayGrid(valY, valX) = iters;
       end
    end
end

% % Display the Mandelbrot plot using the following:
% figure;
% imshow(displayGrid, [0 100], "InitialMagnification", 400);
```

Determination of the Mandelbrot boundary

```
% Use the bisection function at each x-value to find the y-value that
% bounds the convergent and divergent complex numbers
yBoundary = [numel(x)];
for column = 1:numel(x)
    yBoundary(column) = bisection(indicatorFunctionForCol(x(column)), 0, 1);
end
```

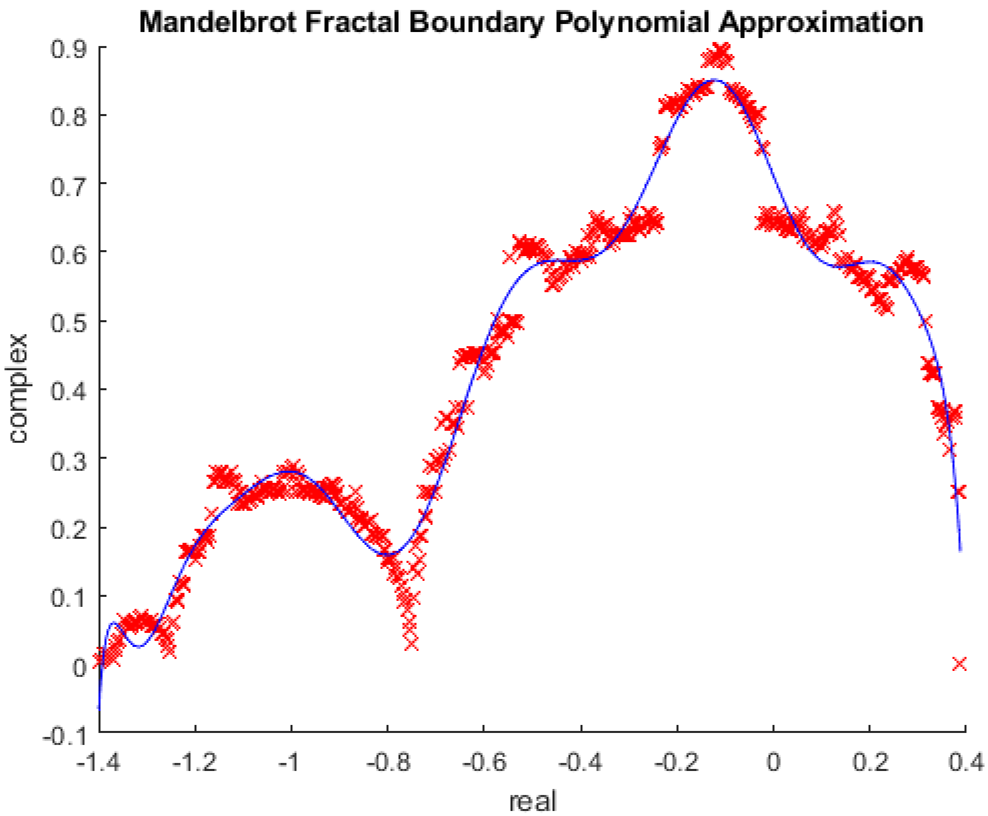
Polynomial Fitting

```
% First, cut off all the zero values on either side of the point curve
% (currently amounts to a range of 201-795)
init = 1;
termin = 1;
for i = 1:numel(yBoundary)
    if yBoundary(i) ~= 0 && init == 1
        init = i;
    elseif yBoundary(i) == 0 && init ~= 1 && termin == 1
        termin = i;
    end
end

% Resize arrays
yBoundary = yBoundary(init:termin);
xBoundary = x(init:termin);

% Plot a chart as a sanity check
figure;
scatter(xBoundary, yBoundary, 'xr');
xlabel('real');
ylabel('complex');
title('Mandelbrot Fractal Boundary Polynomial Approximation')

% Use built-in polyfit function to polynomial fit the boundary curve
p = polyfit(xBoundary, yBoundary, POLYNOMIAL_ORDER);
hold on
plot(xBoundary, polyval(p, xBoundary), '-b');
```



Integrating along the curve

```
% Use custom polyLen method to calculate the length of the polynomial curve
% between the specified boundaries, and output the result to the console
mandelbrotLength = polyLen(p, xBoundary(1), xBoundary(end));
fprintf('Length of Mandelbrot perimeter by polynomial fitting: %f\n',
mandelbrotLength);
```

Length of Mandelbrot perimeter by polynomial fitting: 2.837169

Helper functions

```
% Takes single complex number; returns -1 if c converges and the number of
% iterations if it diverges
function iterations = fractalIterationsToDivergence(c)
    CUTOFF_ITER = 100; % maximum number of iterations before declaring
convergence
    FRACTAL_BOUND = 2; % value of the complex number indicating divergence
    iterations = 0;
    z = c;
    while iterations < CUTOFF_ITER && abs(z)^2 < FRACTAL_BOUND
        z = z^2 + c;
        iterations = iterations + 1;
    end
    if iterations == CUTOFF_ITER
```

```
        iterations = -1;
    end
end

% Indicator function for a particular column provided as the input parameter
% Anonymous function returns -1 if complex number formed by the X and Y
% values is divergent and 1 if convergent
function fn = indicatorFunctionForCol(xval)
    fn = @(yval) (fractalIterationsToDivergence(xval + 1i * yval) >= 0) * 2
- 1;
end


% Function that uses a binary search algorithm structure to find the point
% at which complex values switch from being outside to inside the
% Mandelbrot set on a particular X-value - SENSITIVE TO N=1000
function m = bisection(fn_f, s, e)
    m = e;
    m0 = s;
    mt = 0;
    while fn_f(m) == fn_f(m+1/1000) % not at boundary, 1/N
        mt = m;
        if fn_f(m) < 0 % diverges
            m = m + 0.5*(abs(m0 - m));
        elseif m < 1/1000 % close enough to 0
            m = 0;
            break;
        else % converges
            m = m - 0.5*(abs(m - m0));
        end
        m0 = mt;
    end
end


% Function to determine the length of a polynomial curve p from a left and
% right bound s and e. Returns a scalar value l representing the length of
% the curve between the specified bounds.
function l = polyLen(p, s, e)
    % Prepare for creation of ds by calculating polynomial derivative and
set of
    % numbers to use as exponents
    dp = polyder(p);
    % 15 should be changed to POLYNOMIAL_ORDER
    exps = (15 - 1):-1:0;

    % %sanity checks plotting the p and dp
    % hold on
    % plot(x_boundary, polyval(p, x_boundary), 'b-');

    % Anonymous function for dp/dx
    ds = @(xvalue) sqrt(1+sum((xvalue .^ exps) .* dp)^2);

    % integration with ArrayValued set to true so the arrays play nice
```

```
    l = integral(ds, s, e, 'ArrayValued', true);
end
```

Published with MATLAB® R2024a