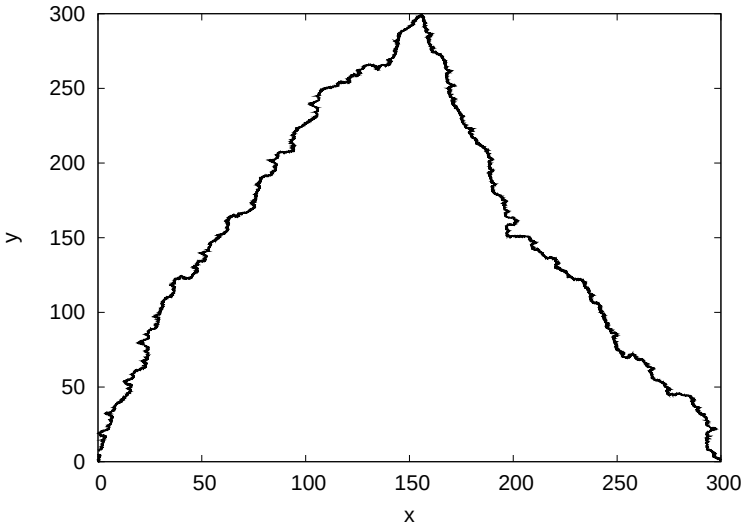
Lecture VIII : Fractals and fractal dimension

In this lab, we shall write a function that calculates thre fractal dimension of a curve given its x and y coordinates. Although recursively generated fractals are esthetically pleasing, fractals in nature occur as a result of random processes making the analytical determination of their dimension impossible. We will, however, use a method that is similar to the one we used in determining the dimension of the Koch fractal in lecture.

Consider the following randomly generated fractal curve.

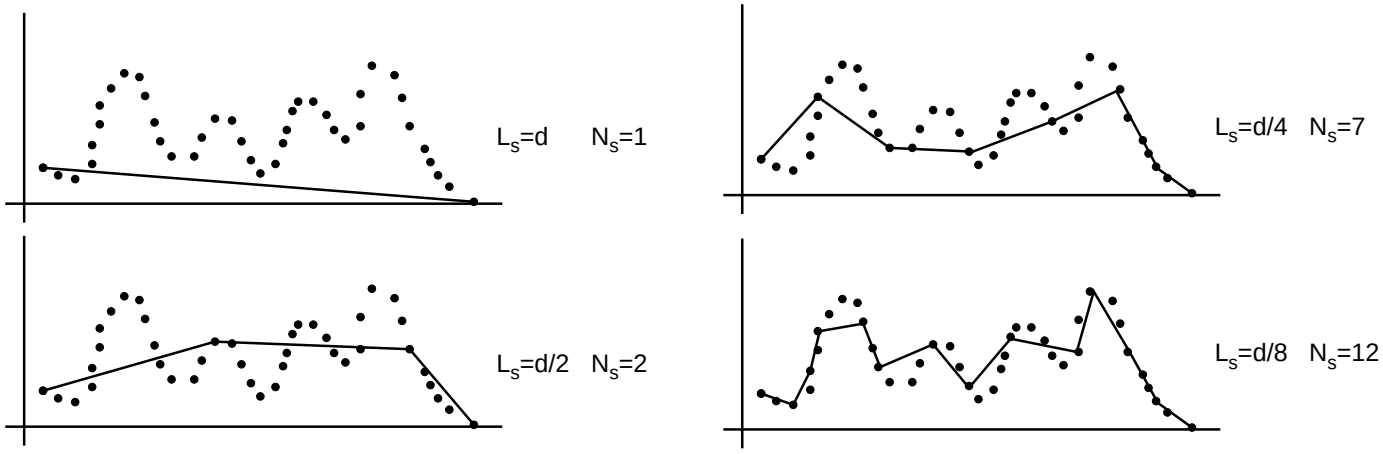


This is in fact a Koch curve generated by inserting a triangle between pairs of points at different levels. This time, however, the angles at the corners of the triangles and their side lengths have been chosen at random. Notice that this graph has a strong resemblance to a coastline on a map. Coastlines are in fact also fractals.

We are going to find the dimension of such curves by successively decreasing the step size and counting how many steps are needed to reach from one end to the other. Each time we decrease the step size by, say, two, the number of steps needed to reach the end should increase by a factor that is often larger than two as more and more of the underlying structure of the fractal is revealed. At this point, we also remember that the fractal dimension is given by

$$\log N_s = -d_f \log L_s \tag{1}$$

Thus the negative slope of the linear fit to the $\log N_s$ versus $\log L_s$ curve will provide the fractal dimension. In order to understand the procedure described above, let's consider a cartoon of the algorithm. Be warned that the distances of line segments in the plot below are not drawn to scale, so don't try to make sense of the number of steps.



1. Start by determining the linear distance between the first and the last points of the curve. The step length at this level L_s is thus equal to this distance, d and the number of steps is N_s.

2. At each level decrease the step size by a factor of 2, setting $L_s = d/2^n$ and initialize N_s to zero.

3. Start at the first point and we calculate the distance between the first point and consecutive points until we reach a distance of L_s. (Of course we are never going to get exactly L_s but we can decide to stop just before we reach L_s or just after. If the number of points on the curve is large enough and the step sizes eventually get down to small enough values, it shouldn't make much of a difference.)

4. When you find the point which is a distance L_s or greater away from the initial point stop and increment N_s by one.
5. Start from the same point where you stopped at the last step. Measure the difference between this point and consecutive points until you reach a distance of at least L_s. Increment N_s
6. Keep incrementing N_s until you come to the end of the curve. At the end, there's going to be a little segment whose length is smaller than L_s. Ignore that segment. (Again for large number of points, it shouldn't change the result).
7. Record N_s for this level.
8. Come back to the beginning of the curve and move onto the next level by halving the step size.
9. Repeat the above procedure for a predetermined number of levels and record N_s for each level (or equivalently for each L_s).
10. Make a log-log plot of N_s versus L_s and calculate the slope.

Exercise 1 : Fit and plot
We'll first start with a simple function which takes in the x and y coordinates of a curve and fits a polynomial to it. As we have seen in previous lectures, there already exists such a function called `polyfit` in `Octave`. The new function, `fit`, that we are about to write adds an additional functionality to this function, which is plotting the data and the fit together so as to assess visually the quality of the fit. Such programs that modify the use or output of an already-existing function or program are called *wrapper*s.

```
## Function that fits a given set of data point to a polynomial of
## specified degree, producing at the same time a superimposed plot of the
## data points and the fit.
## Usage p=fit(x,y,n)

function p=fit(x,y,n)

  p=polyfit(x,y,n);
  xx=linspace(min(x),max(x),100);
  plot(x,y,'b*;Data;',x,polyval(p,xx),'r-;Fit;');

endfunction
```

- The `polyfit` function is called in the usual way, returning the polynomial coefficients as an array, `p`.
- Then we define an additional array that spans the same interval as `x` but in smaller steps (Assuming `x` has fewer than 100 elements). This is just to obtain a smooth curve for the fit in case the number of elements in `x` are too few.
- Plot the data as blue points (`b*`) and the fit as a red curve (`r-`) giving them appropriate legends.
- In plotting the fit, use the `polyval` function to evaluate the polynomial at the elements of `xx`. This is equivalent to doing the following

  ```
  p(1)*x.^n+p(2)*x.^(n-1)+....+p(n-1)*x+p(n)
  ```

  Pay attention to the fact that the coefficient of the highest order term of the polynomial is the first element.

We'll use this function for determining the slope of the log-log plot in the second example.

Exercise 2 : Determining the fractal dimension
In this part, we cast into `Octave` code the algorithm discussed above. For this exercise, we need three nested loops :

- A `for` loop that goes over levels and decreases the step length at every level.

- A `while` loop that enables us to count the number of steps along the curve for a given level until we reach the last segment whose length is smaller than L_s.
- A second, innermost `while` loop that calculates the distances between a given point and the consecutive points until the distance is greater than L_s.

For the `for` loop incrementing the index (let's call it `n`) that goes over levels is not a problem because it gets incremented automatically. However, for the `while` loops we need to increment or advance the indices ourselves. Each of the `while` loops is associated with a different index. Let's call these `c1` and `c2`. Let `c1` represent the index for the fixed point and gets advanced in the outer `while` loop while `c2` is the index of the consecutive points and gets incremented in the innermost `while` loop.

In addition, we need to count the number of steps for each level. For this, we set up a counter, `Ns` and increment it each time the innermost `while` loop is completed.

Let's call our function `fractal_dimension.m`.

```
## Program that calculates the fractal dimensionality of an arbitrary
## curve (could also be smooth)
## Usage : df=fractal_dimension(x,y,Nlevels)
##         df : fractal dimension
##         x,y : coordinates of the fractal
function df=fractal_dimension(x,y,Nlevels)

  ## Calculate the distance between the initial and final point of the
  ## curve.
  l=length(x);  ## Gets used several times, so assign to variable
  d = sqrt( (x(l)-x(1))^2+(y(l)-y(1))^2 );
  Lsf=[]; Nsf=[];  ##

  for n = 1:Nlevels
    n          ## For display, no semicolons
    Ls  = d/2^n;   Lsf = [Lsf;Ls];
    Ns  = 0;   ## Counter for number of steps
    c1  = 1;   ## Counter for the reference point
    di_to_end = d+eps;  ## Distance of c1 to the end

    while ( di_to_end>Ls )

      c2 = c1+1;   ## Consecutive point
      di = 0;  ## Initialize the distance between c1 and c2

      while ( di < Ls )
        di = sqrt( (x(c2)-x(c1))^2 + (y(c2)-y(c1))^2 );
        c2++;
      endwhile

      c1=c2;
      di_to_end = sqrt( (x(c1)-x(l))^2 + (y(c1)-y(l))^2 );
      Ns++;   ## Increment the number of steps

    endwhile

    Nsf=[Nsf;Ns];

  endfor

  p=fit(log(Lsf),log(Nsf),1);
  df=-p(1);

endfunction
```

- Because we need to make a log-log plot of N_s versus L_s, we need to keep record of the total number steps at every level as an array. `Nsf` and `Lsf` are such arrays, to which we add `Ls` and the final `Ns` at every step of the `for` loop.
- `di_to_end` is a variable that measures the distance between the point designated by `c1` and the endpoint. The outer `while` loop terminates when this distance is smaller than `Ls`.
- For each `c1`, the first `c2` designates the adjacent point. Therefore, at the beginning of the outer `while` loop, `c2` is initialized to `c1+1`.
- After the innermost `while` loop is finished for a given `c1`, `c1` needs to be reset to begin the next step. The point where it should be reset is the final point where the last `c2` was found. We thus set `c1` equal to `c2` at the end of the innermost `while` loop.
- At the end, just before terminating the function, we use the `fit` function from the previous exercise to make a linear fit and extract the fractal dimension from the slope.

If you use this function with a smooth curve such as a line or a low-order polynomial, the slopw should give something that is close to one. If you use it for the random Koch fractal in the first figure, it should be something that is close to the regular Koch curve, about 1.1.