Lab # 3 Solutions

Sam Fleischer

Tues. Mar. 24, 2015

Contents

Problem 1

Write a Python function that takes as input the integer `n` *are returns a 1d* `numpy` *array of lenth* `n` *holding the* `n` *roots of unity; that is, the* `n` *roots of the polynomial* $z^n - 1 = 0$.

```
def roots_of_unity(n):
    zeros = np.zeros(n, complex)
    for i in xrange(0, n):
        zero = math.cos((2*math.pi/n)*i) + math.sin((2*math.pi/n)*i)*1j
        zeros[i] = zero

    return zeros
```

Listing 1: Roots of Unity FUnction

Problem 2

Write a Python function named `nfractal` *that takes as input four arguments:* `K`, `pixel`, `tol`, *and* `n`, *and*

1. *generates a mesh of the complex plane consisting of* `pixel` *points along the* x*-axis and* `0.75*pixel` *points along the* y*-axis, with* $x, y \in [-1.5, 1.5]$.

2. *applies the iteration* $z_{k+1} = z_k - a\left(\dfrac{p(z_k)}{p'(z_k)}\right)$ *to each* $z = x + iy$ *on the mesh* `K` *times and records the number of times that the iterated value is within* `tol` *distance to one of the roots of the polynomial* $p(z)$.

3. *colors each point in the mesh grid according to the number of times it is within a distance* `tol` *of one of the roots of the polynomial* $p(z)$.

```
def nfractal(K, pixel, tol, n):
#generates the newton fractal for a given polynomial

    r = 0.75
    a = alpha + beta*1j

    limit = 1.5

    x = np.linspace(-limit, limit, pixel)
    y = np.linspace(-limit, limit ,round(pixel*r))

    [Re, Im] = np.meshgrid(x,y)
    C = np.zeros([round(r*pixel), pixel], complex)
    C = Re + Im*1.0j

    B = np.zeros([round(r*pixel), pixel], float)
    Id = np.ones(B.shape)
```

```
18
19      zeros = roots_of_unity(n)
20      iteration = define_iterative_method(n, a, Id, zeros, tol)
21
22      Cn = C
23
24      for k in range(1,K):
25          (Cn, B) = iteration(Cn, B)
26          if k % 10 == 0:
27              print k
28
29      plt.pcolormesh(x, y, B, cmap = color) # bone, copper, summer
30      file_name = "fractal_%d_%d_%d.png" % (K, pixel, n)
31      plt.savefig(file_name, format="png", dpi = 150)
32      print file_name
```

Listing 2: Newton Fractal Function

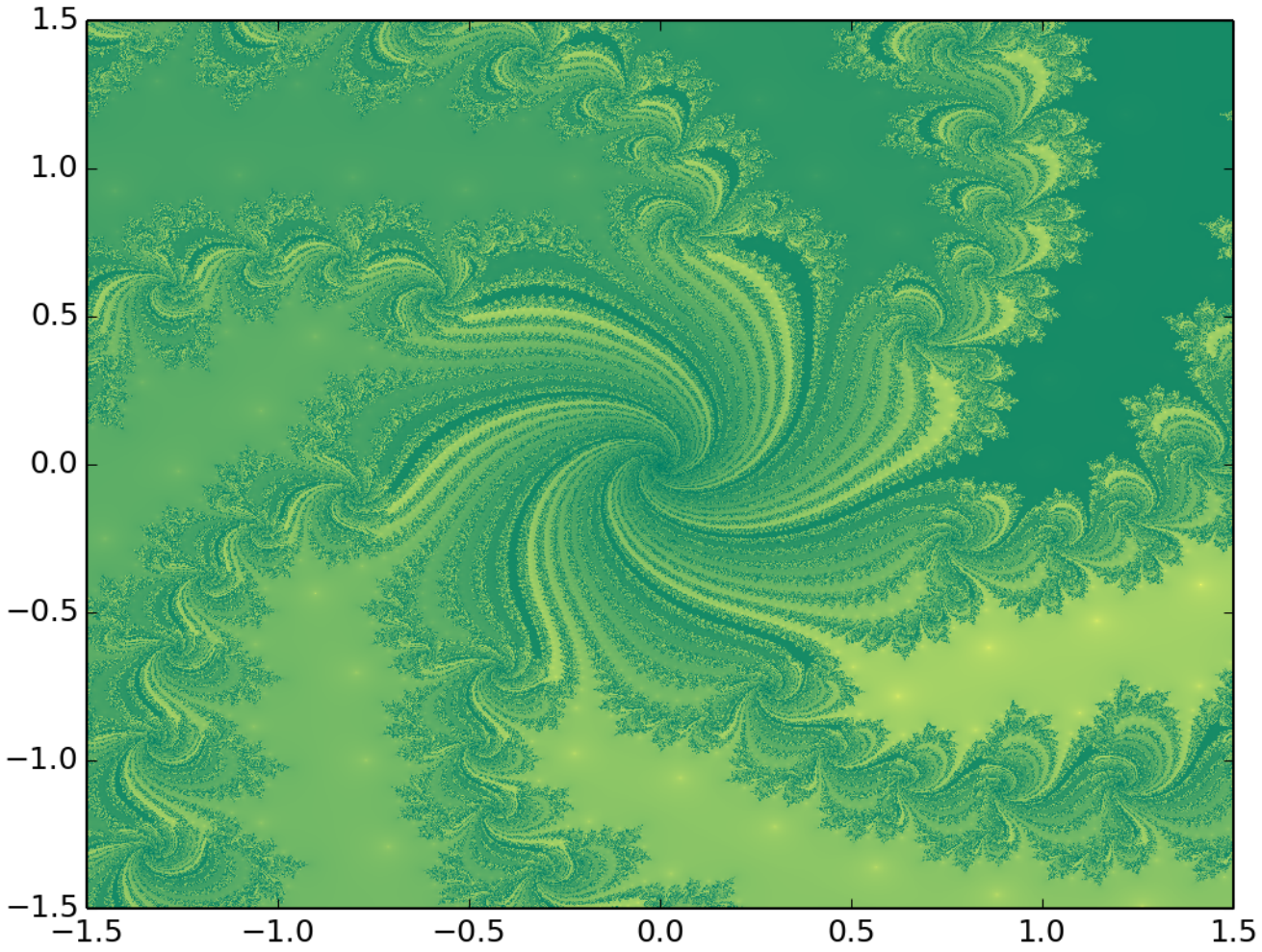
Line 20 defines the iterative method for line 25 by calling the following function:

```
1  def define_iterative_method(n, a, Id, zeros, tol):
2      def iteration(Cn, B):
3          Cn -= a*(Cn**n - Id) / (n*Cn**(n-1))
4          for i in xrange(0, n):
5              B += (i)*(np.abs(Cn - Id*zeros[i]) < tol)
6          return (Cn, B)
7      return iteration
```

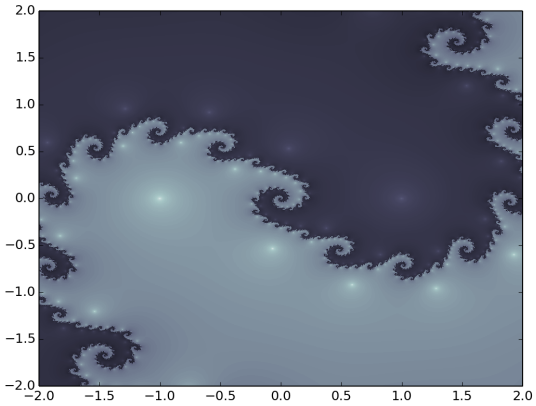
Listing 3: Definition of the Iterative Method

Apply the iteration for polynomials of the form $p(z) = z^n - 1$ *for values of* $n = 2, 3, 4$, *and different complex values of* a. *This will generate the Julia Set of the rational function* $z_{k+1} = z_k - a\left(\frac{p(z_k)}{p'(z_k)}\right)$ *for the polynomial* $p(z)$. *Repeat the calculation for different values of* `pixel`, `K`, *and* `tol`, *and try different* `matpoltlib` *color maps until you obtain a fractal you like.*

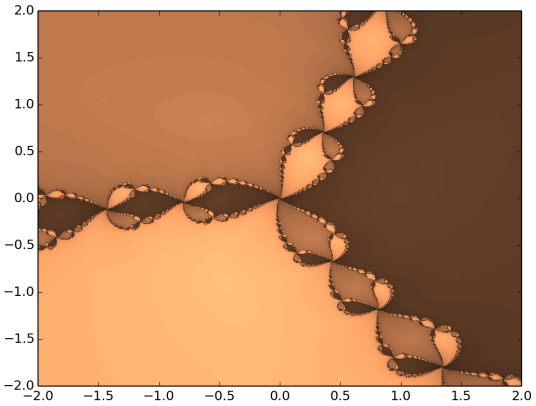
Repeat the images you created above but coloring the points on the grid according to the root they converge to. That is, if the k^{th} *iteration of the point* $z_{m,n} = x_m + iy_n$ *is closest to the* s^{th} *root, increase the counter of that point and not the others and making sure that points that converge to one root end up being colored differently than those converging to another root.*



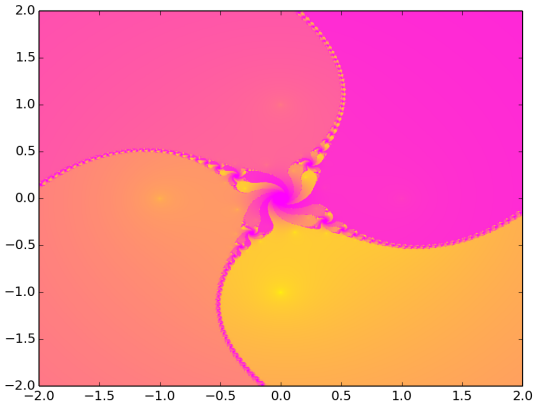
My favorite generated image: $z^7 - 1 = 0$, $a = 1 + \frac{19}{20}i$



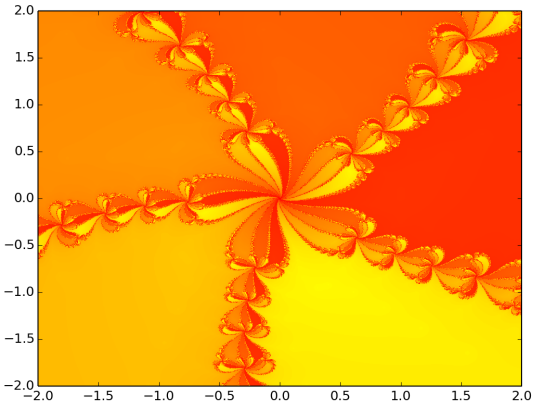
$z^2 - 1 = 0$, $a = \frac{1}{2} + \frac{3}{4}i$



$z^3 - 1 = 0$, $a = 1 + \frac{1}{20}i$



$z^4 - 1 = 0$, $a = \frac{1}{10} + \frac{1}{10}i$



$z^5 - 1 = 0$, $a = 1 + \frac{1}{5}i$