

C++ code 8.8.8: Plotting of attractors of complex roots

```
2 void julia() {
3     // Exact solutions of  $z^3 = 1$  for complex  $z$ .
4     // Use 2-dimensional real vectors to represent complex numbers.
5     Eigen::Vector2d z1, z2, z3;
6     z1 << 1, 0;
7     z2 << -0.5, 0.5 * std::sqrt(3);
8     z3 << -0.5, -0.5 * std::sqrt(3);
9
10    // Tolerance and maximum number of iterations for Newton's method.
11    constexpr double tol = 1e-4;
12    constexpr unsigned int N_it = 20;
13
14    // The image will have a resolution of res*res.
15    constexpr unsigned int res = 780;
16    // We imagine that we have a res*res grid Z on the square [-2,2] with
17    // Z(0,0)=(-2,-2), Z(0,res-1)=(-2,2),
18    // Z(res-1,0)=(2,-2), Z(res-1,res-1)=(2,2), etc.
19    // C(i,j) is the color assigned to the (i,j)-th point on the grid.
20    Eigen::MatrixXd C(res, res);
21
22    // TODO: (8-8.d) Fill C with real numbers, such that each entry C(i,j)
23    // corresponds to which of the roots (z1, z2, or z3) Newton's method converges
24    // (if it converges), when using Z(i,j) as a starting point.
25    // The values in C will be interpreted as colors below.
26    // Additional: You may choose different shades of each color to indicate
27    // how many iterations were needed to reach the tolerance.
28    // Hint: To speed up runtimes, start with a low value for the parameter res.
29    // START
30    C.setZero();
31    for (unsigned int i = 0; i < res; i++) {
32        for (unsigned int j = 0; j < res; j++) {
33            // Newton iteration starting at Z(i,j)=z=(x,y).
34            const double x = -2 + 4.0 * i / (res - 1);
35            const double y = -2 + 4.0 * j / (res - 1);
36            Eigen::Vector2d z(x, y);
37            for (unsigned int k = 0; k < N_it; k++) {
38                z -= juliaDF(z).lu().solve(juliaF(z));
39                // Check if we are sufficiently close to any of the three roots:
40                if ((z - z1).norm() < tol) {
41                    C(i, j) = 1.0 - 1.0 * k / N_it;
42                    break;
43                }
44                if ((z - z2).norm() < tol) {
45                    C(i, j) = 2.0 - 1.0 * k / N_it;
46                    break;
47                }
48                if ((z - z3).norm() < tol) {
49                    C(i, j) = 3.0 - 1.0 * k / N_it;
50                    break;
51                }
52            }
53        }
54    }
55    // END
56    // Axis labels
57    std::vector<double> ticks(5);
```