

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 8-8: Julia Set

Julia sets are famous fractal shapes in the complex plane. They are constructed from the basins of attraction of zeros of complex functions when the Newton method is applied to find them.

This problem treats Newton's method in 2D and is related to [Lecture → Section 8.5]

In the space $\mathbb{C}$ of complex numbers the equation

$$z^3 = 1 \quad (8.8.1)$$

has three solutions:

$$z_1 = 1, \quad z_2 = -\frac{1}{2} + \frac{1}{2}\sqrt{3}\mathbf{i}, \quad z_3 = -\frac{1}{2} - \frac{1}{2}\sqrt{3}\mathbf{i}, \quad (8.8.2)$$

the cubic roots of unity.

(8-8.a) ☐ (15 min.) As you know from the analysis course, the complex plane $\mathbb{C}$ can be identified with $\mathbb{R}^2$ via $(x, y) \mapsto z = x + \mathbf{i}y$. Using this identification, convert Eq. (8.8.1) into a system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ for a suitable function $\mathbf{F} : \mathbb{R}^2 \mapsto \mathbb{R}^2$.

SOLUTION of (8-8.a):

We have

$$z^3 - 1 = (x + \mathbf{i}y)^3 - 1 = x^3 + 3\mathbf{i}x^2y - 3xy^2 - \mathbf{i}y^3 - 1 = x^3 - 3xy^2 - 1 + \mathbf{i}(3x^2y - y^3) = 0.$$

Thus, Eq. (8.8.1) is equivalent to the 2×2 non-linear system of equations

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x^3 - 3xy^2 - 1 \\ 3x^2y - y^3 \end{bmatrix} = \mathbf{0}, \quad \mathbf{x} := \begin{bmatrix} x \\ y \end{bmatrix}. \quad (8.8.3)$$

▲

(8-8.b) ☐ (15 min.) [depends on Sub-problem (8-8.a)]

Formulate the **Newton iteration** [Lecture → Eq. (8.5.1.6)] for the non-linear equation $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ with $\mathbf{x} = [x, y]^T$ and $\mathbf{F}$ from Sub-problem (8-8.a).

SOLUTION of (8-8.b):

The Newton iteration in $\mathbb{R}^n$ reads

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{D}\mathbf{F}(\mathbf{x}^{(k)}))^{-1}\mathbf{F}(\mathbf{x}^{(k)}). \quad (8.8.4)$$

see [Lecture → Eq. (8.5.1.6)], where $\mathbf{D}\mathbf{F}$ is the Jacobian

$$\mathbf{D}\mathbf{F}(\mathbf{x}) = \begin{bmatrix} 3x^2 - 3y^2 & -6xy \\ 6xy & 3x^2 - 3y^2 \end{bmatrix}. \quad (8.8.5)$$

▲

(8-8.c) 🕒 (15 min.) [depends on Sub-problem (8-8.a)]

In the file `julia.hpp` implement two C++ functions

```
Eigen::Vector2d juliaF(const Eigen::Vector2d& x);
Eigen::Matrix2d juliaDF(const Eigen::Vector2d& x);
```

that return $F(\mathbf{x})$ and the Jacobian $DF(\mathbf{x})$ for the function $F: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ found in Sub-problem (8-8.a).

SOLUTION of (8-8.c):

We use the initialization of matrices via the `<<` operator offered by EIGEN, see [Lecture → § 1.2.1.3].

C++ code 8.8.6: Evaluation of F

```
2 Eigen::Vector2d juliaF(const Eigen::Vector2d &z) {
3   Eigen::Vector2d Fz = Eigen::Vector2d::Zero();
4   // TODO: (9-8.c) Implement the function  $F: \mathbb{R}^2 \rightarrow \mathbb{R}^2$  such that  $F(z)=0$  is
5   // equivalent to  $z^3 = 1$  (interpreting  $z$  as a complex number).
6   // START
7   Fz << z(0) * z(0) * z(0) - 3.0 * z(0) * z(1) * z(1) - 1.0,
8         3.0 * z(0) * z(0) * z(1) - z(1) * z(1) * z(1);
9   // END
10  return Fz;
11 }
```

Get it on 🐙 GitLab (`julia.hpp`).

C++ code 8.8.7: Evaluation of Jacobian DF

```
2 Eigen::Matrix2d juliaDF(const Eigen::Vector2d &z) {
3   Eigen::Matrix2d DFz = Eigen::Matrix2d::Zero();
4   // TODO: (9-8.c) Implement the Jacobian of  $F$  at  $z$ .
5   // START
6   const double a = 3.0 * (z(0) * z(0) - z(1) * z(1));
7   const double b = 6.0 * z(0) * z(1);
8   DFz << a, -b, b, a;
9   // END
10  return DFz;
11 }
```

Get it on 🐙 GitLab (`julia.hpp`).

**(8-8.d)** 🕒 (60 min.) [depends on Sub-problem (8-8.b)]

Denote by $\mathbf{x}^{(k)}$ the iterates produced by Newton method from the previous subproblem with some initial vector $\mathbf{x}^{(0)} \in \mathbb{R}^2$. Depending on $\mathbf{x}^{(0)}$, the sequence $\mathbf{x}^{(k)}$ will either diverge or converge to one of the three cubic roots of unity.

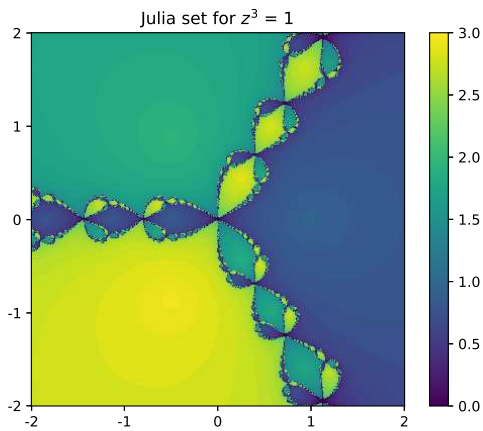
Analyze the behavior of Newton iterations by means of a C++ code using the following approach:

- Use equally spaced points on the domain $[-2, 2]^2 \subset \mathbb{R}^2$ as starting points of Newton iterations.
- Color the starting points differently depending on which of the three roots is the limit of the sequence $\mathbf{x}^{(k)}$.

Concretely complete the implementation of the C++ function

```
void julia(void);
```

supplied in `julia.hpp`. It creates and saves a plot in `julia.png`.



◁ The Julia set for $z^3 - 1 = 0$ on a mesh containing 780×780 points, $N_{it}=20$.

This is how your final plot should look like.

Remark. The boundary of the three so-called domain of attraction of the three different zeros is a **fractal set**: At whatever scale you look at it, it displays the same complex shape.

Fig. 92

HINT 1 for (8-8.d): You may stop the iteration once you are closer in distance to one of the three roots of unity than 10^{-3} .

The three (non connected) sets of points whose iterations converge to the different z_i are called Fatou domains: their boundaries are the Julia sets.

SOLUTION of (8-8.d):

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

```
2  void julia(void) {
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::MatrixXcd C(res, res);
21
22     // TODO: (9-8.d) Fill C with real numbers, such that each entry C(i,j)
23     // corresponds to
24     // which of the roots (z1, z2, or z3) Newton's method converges
25     // (if it converges), when using Z(i,j) as a starting point.
26     // The values in C will be interpreted as colors below.
27     // Additional: You may choose different shades of each color to indicate
28     // how many iterations were needed to reach the tolerance.
29     // 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);
58 for (unsigned int i = 0; i < 5; ++i) {
59     ticks[i] = i * (res - 1) / 4;
60 }
61 std::vector<std::string> labels{"-2", "-1", "0", "1", "2"};
62
63 plt::figure();
64 // Need to transpose C because plt::imshow() uses
65 // first index for y axis and second index for x axis.
66 // Different colormaps can be found at
67 // https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
68 plt::imshow(C.transpose(), {"cmap", "viridis"}, {"origin", "lower"});
69 plt::colorbar();
70 plt::xticks(ticks, labels);
71 plt::yticks(ticks, labels);
72 plt::title("Julia set for  $z^3 = 1$ ");
73 plt::savefig("./cx_out/julia.png");
74 }

```

Get it on  GitLab (julia.hpp).

For each starting point at most N_it iterations are carried out. For a given starting point $\mathbf{x}^{(0)}$, as soon as the condition $|x_1^{(k)} + ix_2^{(k)} - z_\ell| < 10^{-3}$, $\ell \in \{1, 2, 3\}$, is satisfied, we assign a color depending on the root z_ℓ and on the number of iterations done. Each attractor z_ℓ , $\ell \in \{1, 2, 3\}$ is associated to blue, green, or yellow, respectively; lighter colors correspond to points with faster convergence. Starting points for which the iteration does not converge within N_it iterations are dark blue.



End Problem 8-8 , 105 min.