Numerical Analysis project in ODEs for undergraduate students

Sigurdur Hafstein[0000−0003−0073−2765]

Science Institute, University of Iceland, Dunhagi 3, 107 Reykjavík, Iceland
shafstein@hi.is

Abstract. Designing good projects involving programming in numerical analysis for large groups of students with different backgrounds is a challenging task. The assignment has to be manageable for the average student, but to additionally inspire the better students it is preferable that it has some depth and leads to them to think about the subject. We describe a project that was assigned to the students of an introductory Numerical Analysis course at the University of Iceland. The assignment is to numerically compute the length of solution trajectories of a system of ordinary differential equations with a stable equilibrium point. While not difficult to do, the results are somewhat surprising and got the better students to get interested in what was happening. We describe the project, its solution using Matlab, and the underlying mathematics in some detail. Further, we discuss the pedagogical aspects of the project and the results in terms of its success and shortcomings.

Keywords: scientific computing project, ordinary differential equations, numerical integration, Lyapunov functions

1 Background

The project we describe in this paper was assigned to the undergraduate students of Numerical Analysis, a 6 ECTS unit course at the University of Iceland with approximately 150 students. The responsibility for this course is within the Faculty of Physical Sciences in the School of Engineering and Natural Sciences and it is mandatory for all BSc. students of Mechanical-, Industrial-, Chemical-, and Civil and Environmental Engineering as well as for all students of Physics, Engineering Physics, Mathematics, Applied Mathematics, and Mathematics and Mathematical Education. It is an elective course for students of Geophysics, Computer Science, Software Engineering, Electrical and Computer Engineering, and Chemistry.

As can be seen from this long lists the preparation and interests of the enrolled students vary considerably. Usually, the students enroll in the course in the fourth semester of six in total to complete a bachelors degree. Prerequisites for Numerical Analysis are one course in Computer Science (programming in Matlab or Python), one course in Linear Algebra, and two courses (three recommended) in Calculus. All of the students are thus familiar with applying linear algebra and calculus, but the mathematics students have also studied the theoretical aspects of these disciplines in some detail in

the framework of metric spaces, ring theory, etc. Students of Mathematics and Applied Mathematics are required to enroll simultaneously in the course Theoretical Numerical Analysis (2 ECTS units), where rigid mathematical proofs of most of the material covered in the Numerical Analysis course are studied.

The **Course Description** is:

> Fundamental concepts on approximation and error estimates. Solutions of systems of linear and non-linear equations. PLU decomposition. Interpolating polynomials, spline interpolation and regression. Numerical differentiation and integration. Extrapolation. Numerical solutions of initial value problems of systems of ordinary differential equations. Multistep methods. Numerical solutions to boundary value problems for ordinary differential equations.

and the **Learning Outcomes** are:

> Knowledge and understanding: To complete this course the student should be able to
>
> 1. define, explain and give examples of the main concepts of the course, such as error, matrix factorization, interpolating polynomial, and finite differences,
> 2. state and explain the main results of the course, for example by stating Newton's Method and the Secant Method and estimate the errors, state algorithms to solve boundary value problems using finite differences and verify the degree of the approximation.
>
> Skills: To complete this course the student should be able to
>
> 1. formulate a simple mathematical problem as a numerical problem, implement it on a computer, and compute an approximate solution,
> 2. estimate the error of numerical solutions,
> 3. use computer software, such as the Anaconda Python platform or Matlab, for programming, computing, and performing numerical experiments,
> 4. validate the results of numerical computations,
> 5. use the concepts and the results of the course to develop and advance algorithms for simple problems the student has not seen before.

In the course two larger group projects count for 30% of the final grade. The assignment we describe here was the third and last part of the second project. The other two parts were to

1. implement adaptive integration using the trapezoidal- and the Simpson's rule and test it for some integrals and
2. implement the shooting method and use it to solve a few boundary value problems.

Adaptive integration and the shooting method are discussed in sufficient detail in the lectures and in the textbook used in the class [20] to make them rather easy to do.

2 The Project and its Solution

In the project the time-reversed van der Pol oscillator

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}), \quad \text{where} \quad \mathbf{f}(x,y) = \begin{pmatrix} -y \\ -4(1-x^2)y + x \end{pmatrix} \tag{1}$$

is considered. It has an exponentially stable equilibrium at the origin and an unstable periodic orbit around it; see below for more details. The project has three objectives, which we describe below together with its solution using Matlab.

2.1 Objective I

The first objective of the assignment was to analyze the system (1) by drawing solution trajectories, both forward and backwards in time. This is easily achieved by first defining in `f.m`

```
function y=f(t,x)
    mu=4.0;
    y(1)=-x(2);
    y(2)=-mu*(1-x(1)^2)*x(2)+x(1);
    y=y';
end
```

and then typing in the command window, here using the initial value $(2,0)^T$ for $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ and $(1,1)^T$ for $\mathbf{x}' = -\mathbf{f}(\mathbf{x})$ and in both cases integrating over the time-interval $[0,20]$.

```
>> [t,w]=ode45(@f,[0,20],[2,0]);
>> plot(w(:,1),w(:,2))
>> [t,w]=ode45(@(t,x) -f(t,x),[0,20],[1,1]);
>> plot(w(:,1),w(:,2))
>> xlabel('X');ylabel('Y');
```

The plots produced are drawn in Figure 1. A few comments are in order. Usually, one defines the function `f` as a function of both time `t` and space `x`, even though the system is autonomous, i.e. $\mathbf{f}$ does not depend explicitly on time. We follow this tradition here, for otherwise we could not use the Matlab solver `ode45` directly. The system $\mathbf{x}' = -\mathbf{f}(\mathbf{x})$ is a time-reversion of the time-reversed van der Pol oscillator from (1), i.e. it is the van der Pol oscillator. It is well known that the van der Pol oscillator has a stable periodic orbit and it can be clearly seen in Fig. 1 (right). Since the periodic orbit is stable all "normal" initial values, except the unstable equilibrium point $(0,0)^T$, will converge to the orbit. The system $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ is the time-reversed van der Pol oscillator and the stable periodic orbit of the van der Pol oscillator becomes an unstable orbit and the unstable equilibrium point at the origin becomes stable. Thus, any initial value inside of the periodic orbit will converge to the equilibrium and outside of the orbit will diverge. Note that the Matlab solver `ode45` (and others) reports problems, i.e. "Unable to meet

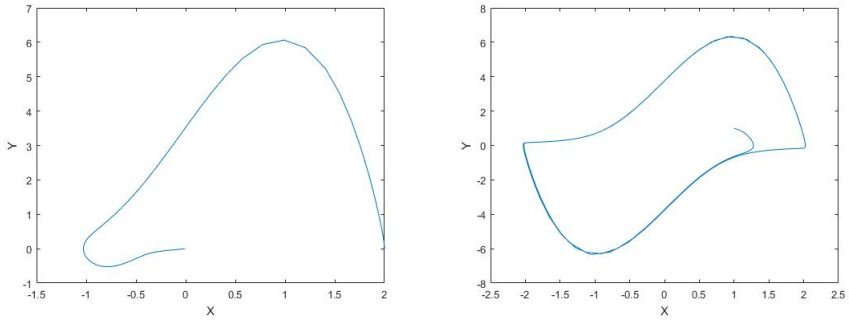

Fig. 1. Left: A solution trajectory of system $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ with $\mathbf{f}$ from (1), starting at $(2,0)^T$ and integrated numerically over the time-interval $[0,20]$ using Matlab's `ode45`. Right: A solution trajectory of system $\mathbf{x}' = -\mathbf{f}(\mathbf{x})$ with $\mathbf{f}$ from (1), starting at $(1,1)^T$ and integrated numerically over the time-interval $[0,20]$ using Matlab's `ode45`.

integration tolerances without reducing the step size below the smallest value allowed" in the integration for various initial values and time-intervals. This can be overcome by using a solver with fixed time-steps, to be delivered in the next objective.

2.2 Objective II

The second objective of the project was to program the standard Runge-Kutta method of order 4, commonly abbreviated RK4. In more detail, an implementation for the function

```
function [t,w]=RK4(f,xi,a,b,n)
```

should be given, where `f` is the function $\mathbf{f}$ in $\mathbf{x}' = \mathbf{f}(t,\mathbf{x})$, `xi` is the initial value at time `a`, and `[a,b]` is the time-interval over which the solution is computed, and `n` is the number of time-steps used. The output `[t,w]` should be the transposes of the outputs of the Matlab solver `ode45` (column vector notation). This is an easy task whose solution is given in the Appendix in Program I. Using `RK4` to plot solution trajectories as in Objective I now becomes, using 500 time-steps:

```
>> [t,w]=RK4(@f,[2,0]',0,20,500);
>> plot(w(1,:),w(2,:))
>> [t,w]=RK4(@(t,x) -f(t,x),[1,1],0,20,500);
>> plot(w(1,:),w(2,:))
>> xlabel('X');ylabel('Y');
```

This delivers trajectories comparable to the ones in Fig. 1. Note, that much fewer time-steps, e.g. 100, results in much less accurate results, see Fig. 2. We now move to the main objective of the problem.

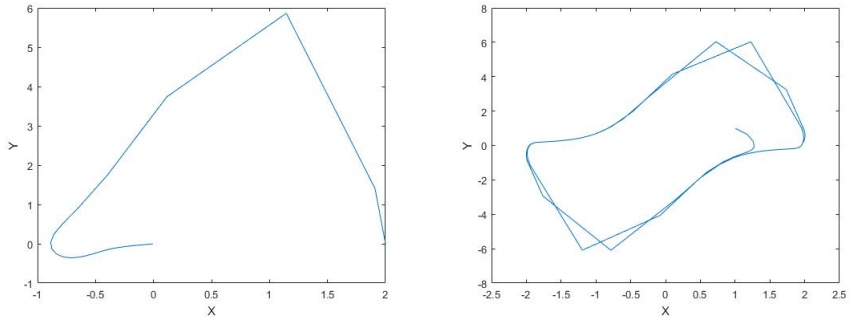

Fig. 2. Left: A solution trajectory of system $\mathbf{x}' = \mathbf{f}(\mathbf{x})$ with $\mathbf{f}$ from (1), starting at $(2,0)^T$ and integrated numerically over the time-interval $[0,20]$ using `RK4` and 100 time-steps (step-size 20/100=0.2). Right: A solution trajectory of system $\mathbf{x}' = -\mathbf{f}(\mathbf{x})$ with $\mathbf{f}$ from (1), starting at $(1,1)^T$ and integrated numerically over the time-interval $[0,20]$ again using `RK4` and 100 time-steps (step-size 20/100=0.2).

2.3 Objective III

On a uniform 101×101 grid on $[-3,3] \times [-8,8]$, compute the length of the solution trajectories to (1) integrated over a time-interval of length 4 and using `RK4` with 100 time-steps. Note that 100 time-steps for a time-interval of length 4 corresponds to the 500 time-steps for a time-interval of length 20 used above, because both have step-size 0.04. The solution to system (1), starting at $\xi \in \mathbb{R}^2$ at time zero, is a function

$$t \mapsto \phi(t,\xi) \text{ fulfilling } \phi(0,\xi) = \xi \text{ and } \phi'(t,\xi) = \frac{d}{dt}\phi(t,\xi) = \mathbf{f}(\phi(t,\xi)) \qquad (2)$$

for all t in the definition domain of $\phi(\cdot,\xi)$ (dependant of ξ either $\mathbb{R}$ or the maximum domain before ϕ becomes infinite). The length of the trajectory $t \mapsto \phi(t,\xi)$ on the time-interval $[0,T]$ is well known to be defined as

$$V(\xi) := \int_0^T \|\phi'(\tau,\xi)\| d\tau,$$

where $\|\cdot\|$ denotes the Euclidian norm on $\mathbb{R}^2$. Because of (2) we can substitute $\mathbf{f}(\phi(\tau,\xi))$ for $\phi'(\tau,\xi)$ and the formula for $V(\xi)$ becomes

$$V(\xi) := \int_0^T \|\mathbf{f}(\phi(\tau,\xi))\| d\tau. \qquad (3)$$

Since `[t,w]=RK4(@f,xi,0,T,n)` delivers $\phi(t_i,\xi)$ in its ith column, where `xi`$=\xi$ and $t_i = (i-1)$`T/n` for $i = 1,2,\ldots,$`n+1`, the integral in (3) can easily be approximated using numerical integration. In the project it was suggested to use the composite Simpson's Rule

$$\int_0^T g(\tau) d\tau \approx \frac{h}{3}\left(g(t_1) + g(t_{n+1}) + 4\sum_{i=1}^{n/2} g(t_{2i}) + 2\sum_{i=1}^{n/2-1} g(t_{2i+1}) \right),$$

where $h = T/n$ is the length of the time-steps. Note that for this formula we need that n is an even number. Since many of the trajectories will be very long, indeed so long that the numerical solver will fail to deliver a numerical value, it is useful to substitute e.g. $V(\xi) \leftarrow \min\{4T, V(\xi)\}$ (Matlab's `min` interprets `NaN` as larger than any number). The implementation is now simple and is given in the Appendix in Program II. Typing

```
>> TraLengths(@f,-3,3,-8,8,4,100)
```

in the command window, where `TraLengths` is the program from the Appendix, `f` is the function from `f.m`, $[-3,3]\times[-8,8]$ is the the area in the plane where a function is computed and plotted, `T=4` is the interval of integration in (3), and `n=100` is the number of time-steps used in the numerical integration, now delivers after a short time (12.5 sec. on a computer with an i7-7700K CPU) Fig. 3. After having produced this

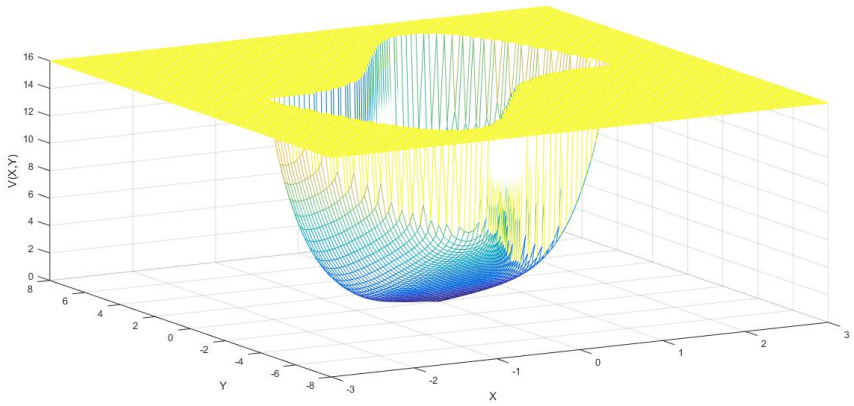

Fig. 3. The length of the trajectories of system (1) plotted as a function of starting position (x,y).

figure the less interested students have finished the project. The more interested ones, however, now might start to ask themselves and the instructor why this function looks like it does? Indeed, many of them did ask the instructor.

3 Study of the results

Let us discuss the results from Objective III and the function V computed in more detail.

Because the origin is a stable equilibrium, solutions $t \mapsto \phi(t,\xi)$ slow down close to it. Indeed, they become so slow that the limit $\phi(t,\xi) \to 0$ when $t \to \infty$ is never reached. The reason for this is that since $\mathbf{f}$ is continuous and $\mathbf{f}(0) = 0$, $\mathbf{f}$ and then also $\mathbf{x}'$ are small close to the origin. Therefore trajectories that start close $\xi \approx 0$ to the origin are short and $V(\xi) \approx 0$.

The function V is a so-called Lyapunov function for the system. This means that it has a local minimum at the stable equilibrium at the origin and that it is decreasing along all solution trajectories in a neighbourhood of the equilibrium. The local minimum is intuitively clear and the decrease can be seen from the following calculations:

$$\begin{aligned}\left.\frac{d}{dt}V(\phi(t,\xi))\right|_{t=0} &= \lim_{h\to 0}\frac{V(\phi(h,\xi))-V(\xi)}{h} \\ &= \lim_{h\to 0}\frac{1}{h}\left[\int_0^T \|\phi(\tau,\phi(h,\xi))\|d\tau - V(\xi)\right]\end{aligned} \tag{4}$$

and because solution trajectories are unique, i.e. $\phi(\tau,\phi(h,\xi)) = \phi(\tau+h,\xi)$, we get

$$\begin{aligned}\int_0^T \|\phi(\tau,\phi(h,\xi))\|d\tau &= \int_0^T \|\phi(\tau+h,\xi)\|d\tau \\ &= \int_h^{T+h} \|\phi(s,\xi)\|ds \\ &= \int_0^T \|\phi(s,\xi)\|ds + \int_T^{T+h} \|\phi(s,\xi)\|ds - \int_0^h \|\phi(s,\xi)\|ds \\ &= V(\xi) + \int_T^{T+h} \|\phi(s,\xi)\|ds - \int_0^h \|\phi(s,\xi)\|ds.\end{aligned}$$

Thus, by (4) and the Fundamental Theorem of Calculus,

$$\begin{aligned}\left.\frac{d}{dt}V(\phi(t,\xi))\right|_{t=0} &= \lim_{h\to 0}\frac{V(\phi(h,\xi))-V(\xi)}{h} \\ &= \lim_{h\to 0}\frac{1}{h}\left[\int_T^{T+h} \|\phi(s,\xi)\|ds - \int_0^h \|\phi(s,\xi)\|ds\right] \\ &= \|\phi(T,\xi)\| - \|\phi(0,\xi)\| \\ &= \|\phi(T,\xi)\| - \|\xi\|.\end{aligned}$$

Since $\lim_{t\to\infty}\|\phi(t,\xi)\| = 0$ for all ξ in a neighbourhood of the origin, we have for all such ξ that $\|\phi(T,\xi)\| < \|\xi\|$ for sufficiently large $T > 0$ and $\xi \neq 0$. Thus

$$V'(\xi) := \left.\frac{d}{dt}V(\phi(t,\xi))\right|_{t=0} < 0$$

and V is decreasing along solution trajectories in a neighbourhood of the stable equilibrium at the origin, i.e. the mapping $t \mapsto V(\phi(t,\xi))$ is decreasing.

The stability theory of Lyapunov and Lyapunov functions are covered in most textbooks on dynamical systems and/or control theory, e.g. [16, 19, 21], and the interested student can be pointed to them for additional information. For an attractor, like our stable equilibrium at the origin for the time-reversed van der Pol system, the sublevel sets of a Lyapunov function serve as "traps". Once inside the component of a sublevel set, that includes the origin and does not extend to the boundary of the domain of the Lyapunov function, the solution cannot escape. This comes because the solution is decreasing along solution trajectories and therefore cannot climb over the edge/boundary of the sublevel set.

The theory of complete Lyapunov functions even tells us that every system given by an ODE possesses a *complete Lyapunov function* that goes a long way in characterizing the qualitative behaviour of the system. Indeed, this holds true for very general dynamical systems. A complete Lyapunov function is a scalar-values function from the whole state-space that is non-increasing along along all solution trajectories and strictly decreasing where possible. Note that, e.g. for a periodic orbit, it cannot be strictly decreasing. In general it is strictly decreasing along all solution trajectories on the part of the state-space where the flow is gradient-like and constant on every transitive component of the *chain-recurrent set*. The theory of complete Lyapunov functions was developed by Auslander, Conley, and Hurley [1, 6, 13–15], see also [18].

Computing Lyapunov functions by using results from converse theorems in dynamical systems, i.e. theorems guarantying the existence of certain kinds of Lyapunov functions for systems with particular stability properties, using integrals or sums over solutions trajectories has been studied in numerous publications [2, 4, 3, 5, 17, 10–12, 7–9]. A central issue is the verification of the properties of a Lyapunov function for the function computed. One way to do this is to interpolate the values computed over the simplices of a triangulation. Essentially, one demands $\nabla V(\mathbf{x})\cdot\mathbf{f}(\mathbf{x})+E_{\mathbf{x}}\|\nabla V(\mathbf{x})\|_1<0$ at all vertices of the triangulation, where the error $E_{\mathbf{x}}\geq 0$ assures that not only

$$\frac{d}{dt}V(\phi(t,\mathbf{x}))\bigg|_{t=0}=\nabla V(\mathbf{x})\cdot\mathbf{f}(\mathbf{x})<0$$

at the vertices, but for any $\mathbf{x}$ in the domain of the triangulation. Here $\|\mathbf{x}\|_1=|x_1|+|x_2|$ and the function V is defined on the simplices by using convex interpolation of the values at the vertices over the whole simplex. A detailed discussion of this method is beyond the scope of this paper, but the interested reader can have a look at [11, 12].

By adding to the code for the function `TraLengths` as described in Program III in the Appendix, a verification of the interpolation of the values as described in [11, 12] is carried out. If one want to implement this for a different two-dimensional system than (1), only the function `f` in `f.m` and the function `B` in Program III have to be modified accordingly. The modification of `f` is obvious because it is the right-hand side of(1), and the return value for `B(r,s,xm,ym)` should be an upper bound, preferably close, on

$$\max_{\substack{i=1,2\\ |x_1|\leq \mathtt{xm},|x_2|\leq \mathtt{ym}}}\left|\frac{\partial^2 f_i}{\partial x_{\mathtt{r}}\partial x_{\mathtt{s}}}(x_1,x_2)\right|.$$

The results of this verification for our system (1) with the same parameters as in Objective III is plotted in Fig.4 (left) and with a higher spatial resolution, i.e. `xres=301; yres=301;`, on the right. By using larger values for the parameters `xres,yres,T,n`, the area within the periodic orbit where the orbital derivative fails to be negative is effectively reduced to an arbitrary small neighbourhood of the equilibrium at the origin, cf. [12, Figs. 1,2]. Using a compiled programming language like C++ also makes these computations very fast.

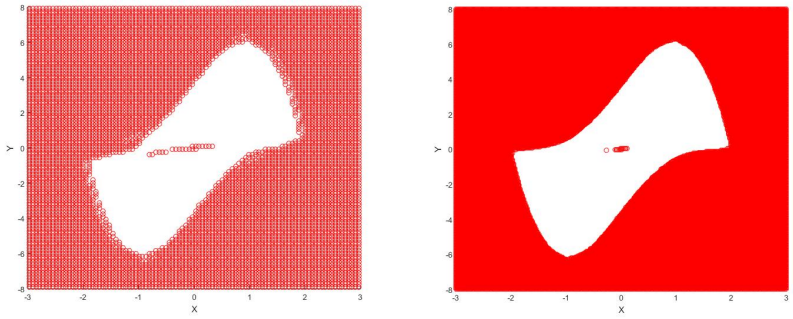

Fig. 4. Verification of the negativity of the orbital derivative of the function V computed in Objective III (left). Areas where it is not negative are plotted in red. Not only is it not negative outside of the periodic orbit, but also in a strip within it. By increasing the spatial resolution the area where the orbital derivative fails to be negative inside of the orbit becomes smaller (right).

4 Conclusion

We presented a new project for undergraduate students in Numerical Analysis. It is not difficult to solve the problem and thus the average student should be able to do so without too much difficulties. The solution, i.e. the function plotted, is somewhat surprising and intended to awaken the interest of the better students. The solution to the project is given and the results and the underlying theory are discussed in some detail. Matlab code is given for all objectives of the project together with code for a more sophisticated verification of the results. Further, numerous references to the underlying theory are given.

Let us compare the presented project, in terms of pedagogical value, to the larger projects from the textbook [20] of the course. The textbook contains, apart from numerous Exercises and Computer Problems, eleven so-called Reality Checks, which are larger project of comparable scale and complexity to the project presented. These Reality-Checks are very well designed and include topics such as the kinematics of the Steward Platform frequently used in flight-simulators, positioning using GPS, motion control in computer-aided modeling, and a simple audio codec. These Reality Checks have been used as assignments for the larger projects in the course discussed in the paper with a great success. The difference, however, between those Reality Checks and the project described, is that the Reality Checks inspire the better students to become interested in the technology behind the project, e.g. flight simulators, GPS, audio codecs, but not in mathematics. The teacher of the course has had lively discussion with students about these technological topics during and after the Reality Checks projects, but rarely about mathematics. A possible drawback of the proposed project is that the theory behind the project has little direct connection to technology. While a project on, say the Stewart Platform, might invoke the interest of an engineering student in numerical root-finding of nonlinear equations because she/he finds the steering of a flight simulator

fascinating, she/he will just routinely solve the project proposed in this paper without gaining any interest in numerical analysis at all.

The advantage of the proposed project is that it inspires some of the mathematics students to get interested in the theory of dynamical systems, a topic that they had only known indirectly from studying differential equations. Further, as Lyapunov functions are a mathematical extension of the concept of dissipative energy from physics, some of the physics students were particularly interested as well. This allowed to the teacher to discuss attractors, repellers, chaos, and complete Lyapunov functions with interested students.

References

1. Auslander, J.: Generalized recurrence in dynamical systems. Contr. to Diff. Equ. **3**, 65–74 (1964)
2. Björnsson, J., Giesl, P., Hafstein, S.: Algorithmic verification of approximations to complete Lyapunov functions. In: Proceedings of the 21st International Symposium on Mathematical Theory of Networks and Systems. pp. 1181–1188 (no. 0180). Groningen, The Netherlands (2014)
3. Björnsson, J., Giesl, P., Hafstein, S., Kellett, C., Li, H.: Computation of continuous and piecewise affine Lyapunov functions by numerical approximations of the Massera construction. In: Proceedings of the CDC, 53rd IEEE Conference on Decision and Control. Los Angeles (CA), USA (2014)
4. Björnsson, J., Giesl, P., Hafstein, S., Kellett, C., Li, H.: Computation of Lyapunov functions for systems with multiple attractors. Discrete Contin. Dyn. Syst. Ser. A **35**(9), 4019–4039 (2015)
5. Björnsson, J., Hafstein, S.: Efficient Lyapunov function computation for systems with multiple exponentially stable equilibria. Procedia Computer Science **108**, 655–664 (2017), proceedings of the International Conference on Computational Science (ICCS), Zurich, Switzerland, 2017.
6. Conley, C.: Isolated Invariant Sets and the Morse Index. CBMS Regional Conference Series no. 38, American Mathematical Society (1978)
7. Doban, A.: Stability domains computation and stabilization of nonlinear systems: implications for biological systems. PhD thesis: Eindhoven University of Technology (2016)
8. Doban, A., Lazar, M.: Computation of Lyapunov functions for nonlinear differential equations via a Yoshizawa-type construction. IFAC-PapersOnLine **49**(18), 29 – 34 (2016)
9. Doban, A., Lazar, M.: Computation of Lyapunov functions fornonlinear differential equations via a Massera-type construction. IEEE Trans. Automat. Control **63**(5), 1259–1272 (2018)
10. Hafstein, S., Kellett, C., Li, H.: Computing continuous and piecewise affine Lyapunov functions for nonlinear systems. Journal of Computational Dynamics **2**(2), 227 – 246 (2015)
11. Hafstein, S., Valfells, A.: Study of dynamical systems by fast numerical computation of Lyapunov functions. In: Proceedings of the 14th International Conference on Dynamical Systems: Theory and Applications (DSTA). vol. Mathematical and Numerical Aspects of Dynamical System Analysis, pp. 220–240 (2017)
12. Hafstein, S., Valfells, A.: Efficient computation of Lyapunov functions for nonlinear systems by integrating numerical solutions. Nonlinear Dynamics (To be published 2019)
13. Hurley, M.: Chain recurrence and attraction in non-compact spaces. Ergod. Th. & Dynam. Sys **11**, 709–729 (1991)

14. Hurley, M.: Chain recurrence, semiflows, and gradients. J Dyn Diff Equat **7**(3), 437–456 (1995)
15. Hurley, M.: Lyapunov functions and attractors in arbitrary metric spaces. Proc. Amer. Math. Soc. **126**, 245–256 (1998)
16. Khalil, H.: Nonlinear systems. Pear, 3. edn. (2002)
17. Li, H., Hafstein, S., Kellett, C.: Computation of continuous and piecewise affine Lyapunov functions for discrete-time systems. J. Difference Equ. Appl. **21**(6), 486–511 (2015)
18. Patrão, M.: Existence of complete Lyapunov functions for semiflows on separable metric spaces. Far East Journal of Dynamical Systems **17**(1), 49–54 (2011)
19. Sastry, S.: Nonlinear Systems: Analysis, Stability, and Control. Springer (1999)
20. Sauer, T.: Numerical Analysis. Pearson, 2nd edn. (2012)
21. Vidyasagar, M.: Nonlinear System Analysis. Classics in applied mathematics, SIAM, 2. edn. (2002)

Appendix

Below is an implementation in Matlab to a solution of the project.
Program I:

```
function [t,w]=RK4(f,xi,a,b,n)
    h=(b-a)/n;
    t=a:h:b;
    w=zeros(length(xi),n+1);
    w(:,1)=xi;
    for i=1:n
        wi=w(:,i);
        s1=f(t(i),wi);
        s2=f(t(i)+h/2,wi+h/2*s1);
        s3=f(t(i)+h/2,wi+h/2*s2);
        s4=f(t(i)+h,wi+h*s3);
        w(:,i+1)=wi+h/6*(s1+2*s2+2*s3+s4);
    end
end
```

Program II:

```
function TraLengths(f,ax,bx,ay,by,T,n)
  xres=101; yres=101;
  V=zeros(xres,yres);
  vx=zeros(n+1,1);
  x=linspace(ax,bx,xres);
  y=linspace(ay,by,yres);
  for ix=1:xres
    for jy=1:yres
      xi=[x(ix),y(jy)]';
      [t,w]=RK4(f,xi,0,T,n);
      for ivx=1:n+1
```

```
12          vx(ivx)=norm(f(t,w(:,ivx)));
13        end
14        V(ix,jy)=T/(3*n)*(vx(1)+vx(n+1)+4*sum(vx(2:2:n))...
15                  +2*sum(vx(3:2:n-1)));
16      end
17    end
18    V=min(V,4*T*ones(size(V)));
19    mesh(x,y,V')
20    xlabel('X');ylabel('Y');zlabel('V(X,Y)');
21  end
```

Program III: Modification of `TraLengths` to additionally verify the validity of the conditions for a Lyapunov function (decrease of the orbital derivative).

Add the following code to the function `TraLengths` between lines 19 and 20.

```
1     % verify decrease of orbital derivative
2     hx=(bx-ax)/(xres-1);
3     hy=(by-ay)/(yres-1);
4     As1(1,1)=hx; As1(1,2)=hx; As1(2,1)=0; As1(2,2)=hy;
5     As2(1,1)=0; As2(1,2)=hx; As2(2,1)=hy; As2(2,2)=hy;
6     k=1;
7     for ix=1:xres-1
8       for jy=1:yres-1
9         xm=max(abs(x(ix)),abs(x(ix+1)));
10        ym=max(abs(y(jy)),abs(y(jy+1)));
11        % sigma = ()
12        gV=[(V(ix+1,jy)-V(ix,jy))/hx, ...
13            (V(ix+1,jy+1)-V(ix+1,jy))/hy];
14        gVn=norm(gV,1);
15        err1=0;
16        c1=~(dot(gV,f(0,[x(ix),y(jy)]))+err1*gVn < 0);
17        err2=E(1,xm,ym,As1);
18        c2=~(dot(gV,f(0,[x(ix+1),y(jy)]))+err2*gVn < 0);
19        err3=E(2,xm,ym,As1);
20        c3=~(dot(gV,f(0,[x(ix+1),y(jy+1)]))+err3*gVn < 0);
21        % sigma = (1 2)
22        gV=[(V(ix+1,jy+1)-V(ix,jy+1))/hx, ...
23            (V(ix,jy+1)-V(ix,jy))/hy];
24        gVn=norm(gV,1);
25        err4=0;
26        c4=~(dot(gV,f(0,[x(ix),y(jy)]))+err4*gVn < 0);
27        err5=E(1,xm,ym,As2);
28        c5=~(dot(gV,f(0,[x(ix),y(jy+1)]))+err5*gVn < 0);
29        err6=E(2,xm,ym,As2);
30        c6=~(dot(gV,f(0,[x(ix+1),y(jy+1)]))+err6*gVn < 0);
31        % check if ~(orbital deririvative < 0)
```

```
      if c1 || c2 || c3 || c4 || c5 || c6
        NotNegx(k)=x(ix)+hx/2;
        NotNegy(k)=y(jy)+hy/2;
        k=k+1;
        continue
      end
    end
  end
  hold on
  plot3(NotNegx,NotNegy,zeros(k-1,1),'ro')
```

Further, add the following functions after the code for the function `TraLengths`

```
function Bval=B(r,s,xm,ym)
  if r==1 && s==1
    Bval=8*ym;
  elseif r==2 && s==2
    Bval=0;
  else
    Bval=8*xm;
  end
end

function Eval=E(i,x,y,A)
  Eval=0;
  for r=1:2
    for s=1:2
      Eval=Eval+B(r,s,x,y)*A(r,i)*(A(s,i)+A(s,2));
    end
  end
  Eval=0.5*Eval;
end
```