LyapXool

Manual file to autonomous usage

Carlos Argáez
Science Institute, University of Iceland

June 12, 2019

This is code to construct Complete Lyapunov functions (CLFs) for dynamical systems expressed as autonomous ordinary differential equations of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \tag{0.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $n \in \mathbb{N}$.

So far, the code has the next different libraries:

- `instructions.hpp`
- `problem.cpp`
- `oddsystem.cpp` and `oddsystem.hpp`
- `generalities.cpp` and `generalities.hpp`
- `chainrecurrentsets.cpp` and `chainrecurrentsets.hpp`
- `RBF.cpp` and `RBF.hpp`
- `wendland.cpp` and `wendland.hpp`

1 General approach

LyapXool uses the Armadillo library and it requires it order to compile.

The general approach to use the code is as follows.

- Write the dynamical system in the file: `odesystem.cpp`. To do that, use the c++ syntax, e.g.

$$\mathbf{f}(x,y) = \begin{pmatrix} -y \\ -x \end{pmatrix}. \tag{1.1}$$

  This dynamical system in the c++ syntax will be written as follows:

$$\begin{aligned} f(0) &= -1.0 * x(1); \\ f(1) &= -1.0 * x(0); \end{aligned} \tag{1.2}$$

  because $x(0)$ represents x and $x(1)$ represents y. For a 3 dimensional case, $x(2)$ represents z. Likewise, $f(0)$ represents $\dot{x}$, $f(1)$ represents $\dot{y}$ and in case of a 3 dimensional case, $f(2)$ represents $\dot{z}$.

  NOTICE: You must provide a name to your dynamical system in the syntax of `case Problema::NameYouWant:`. At the end of the dynamical system you need to include: `break;` to close the case.

- Once you have written your dynamical system, you need to include the name you have given to it, i.e. `NameYouWant`, as used in the point above, into the file `instructions.hpp` under the `enum Problema {` as well as `char const probnames[][11]=`, as shown below:

```
enum Problema {
        TWOORBITS,VDP,HOMOCLINIC,DECREASING,
        TD1,TD2,TD3,SIMPLE3D,LORENZ, NameYouWant
    };

    char const probnames[][11]={"TWOORBITS","VDP","HOMOCLINIC",
    "DECREASING","TD1","TD2","TD3","SIMPLE3D","LORENZ","NameYouWant"};
```

  NOTICE: The dynamical systems `TWOORBITS,VDP,HOMOCLINIC,DECREASING`, `TD1,TD2,TD3,SIMPLE3D,LORENZ` are examples included in LyapXool by default.

- Now, you just need to choose the parameters to run.
  - The problem to be ran, `problem=glovar::NameYouWant;`
  - The dimension of the problem, `variable`
  - The number of critical values (OPTIONAL), `ncritical`
  - `maxnegative,maxpositive`, sets the collection of points in cartesian form required to construct the RBF. It is recommended to use a collection as big as possible. Default values ± 450
  - The collocation grid: Hexagonal or Cartesian, `cartesian`

- The size of the α parameter for the grid. The higher the grid, the less points it contains, `alpha`
- The Boundaries of the collocation grid, `maxmaxx,minminx,maxmaxy`, `minminy,maxmaxz,minminz`. NOTICE: The max and min values in a given axis are not necessarily required to be equal. That depends on your problem
- Type of evaluation grid: Circular or directional (aligned to the flux). For 3 dimensional cases, the circular case becomes spherical, `gridtoeval`
  * For the circular case, the number of concentric circumferences you want to use, `circles`
  * The total amount of points you want in the circumference / sphere. For the directional case, `angles` does not represent the amount of point but half and a fourth of them, respectively, `angles`
  * `spherical` is a boolean variable to fix the 3 dimensional case to a sphere. In case it is false, then the points are distributed over to the x, y and z axes [5]
- `constante` controls that the Lyapunov equation $A\beta = \alpha$ is solved for an α vector whose all entries are -1. Then it is set to false, then it represents that the equation $A\beta = \alpha$ is set for an α vector whose all entries are $-\|f(\mathbf{x})\|$ for each corresponding $\mathbf{x}$ point in the collocation, [2]
- `normal` controls whether the algorithm to be used is the quasi-normalized method as in [6]
- `defcase` is a numerical variable whose options can only be: $1, 2$ or 3. It controls who the α vector will be used for iterations over previous computed CLFs. When set to 1, it follows the approach of 0 and -1 for the values the colocation points of the chai-recurrent set and the gradient-like flow must have when solving $A\beta = \alpha$ in a new iteration. For more, please look [4]. Case 2 considers the exponential decay to solve $A\beta = \alpha$. Please, refer to citepaper2. Finally, case 3 considers the averaging method introduced in [7].
- `eigenvaluesjudge` is a boolean variable. It is completely OPTIONAL function and to use it, you need to introduced the critical values in line 75 of `odesystem.cpp`. An example is given next for three critical points:

```
criticalpoints.resize(ncritical,variable);
criticalpoints<<0.0 << 0.0 << endr
<<0.0 << 0.5 << endr
<<0.0 << 1.0 << endr;
```

- `critval`, this is the tolerance parameter γ introduced in [4]
- `radio`, this is the radio parameter introduced in [4–7]

- `totaliterations`, this sets the total amount of iterations to be taken
- `l,k,c`, these are the Wendland function parameters [8]
- `OMP_NUM_THREADS`, this variable controls the total number of processors to be used for the computation
- `printing` is a boolean variable to control if you want the results printed in a file.
- `fextension`, in case of printing the results to a file, which extension should that file have? By default the extension is set for Matlab

2 Different evaluation grids

2.1 2 dimensional cases

2.1.1 Directional

The points are aligned to the flux of the ODE.

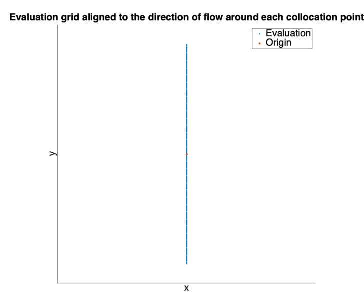

Figure 1: Directional gird, 2 dimensional case

```
const std::string gridtoeval="directional";
const int circles=2;
const int angles=100;
```

NOTICE: under "directional" there is not need to take care of the value assigned to "circles" for it will be automatically set to 2 regardless of the number introduced in `instructions.hpp`.

2.1.2 Circular

The points are displayed in concentric circumferences centred to each collocation point.

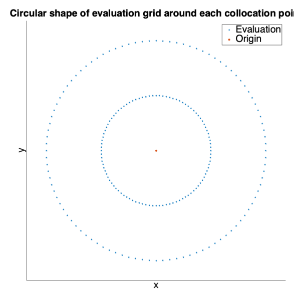

Figure 2: Circular gird, 2 dimensional case

```
const std::string gridtoeval="circular";
const int circles=2
const int angles=100;
```

2.2 3 dimensional cases

2.2.1 Directional

The points are aligned to the flux of the ODE.

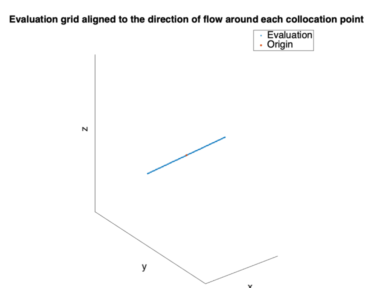

Figure 3: Directional gird, 3 dimensional case

```
const std::string gridtoeval="directional";
const int circles=2
const int angles=100;
```

NOTICE: under ”directional” there is not need to take care of the value assigned to ”circles” for it will be automatically set to 2 regardless of the number introduced in `instructions.hpp`.

2.2.2 Circular

The points are displayed in concentric circumferences centred to each collocation point.

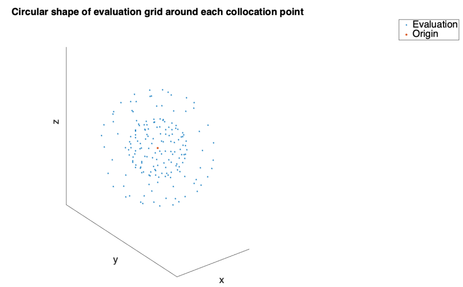

Figure 4: Circular gird, 3 dimensional case

```
const std::string gridtoeval="circular";
const int circles=2
const int angles=100;
const bool spherical=true;
```

2.2.3 Circular

The points are displayed in concentric circumferences centred to each collocation point.

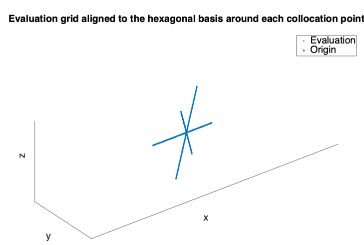

Figure 5: Circular gird, 3 dimensional case

```
const std::string gridtoeval="circular";
const int circles=2
const int angles=100;
const bool spherical=false;
```

References

[1] Peter Giesl,*Construction of a local and global Lyapunov function for discrete dynamical systems using radial basis functions, Lecture Notes in Mathematics, Vol.* ***1904****, 2007, Springer-Verlag Berlin Heidelberg*

[2] Jóhann Björnsson, Skuli Gudmundsson and Sigurdur Hafstein, Class library in C++ to compute Lyapunov functions for nonlinear systems, IFAC-PapersOnLine, Vol, ***48****, 2015, 778 783 (No. 11)*

[3] Jóhann Björnsson, Peter Giesl and Sigurdur Hafstein, Algorithmic verification of approximations to complete Lyapunov functions, Proceedings of the 21st International Symposium on Mathematical Theory of Networks and Systems, Groningen, The Netherlands, (2014), 1181-1188 (no. 0180)

[4] Argáez, C., Giesl, P., and Hafstein, S. (2017a). Analysing dynamical systems towards computing complete Lyapunov functions. In Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), pages 134-144. Madrid, Spain.

[5] Argáez,C.,Giesl,P.,andHafstein,S.(2018a). Computation of complete Lyapunov functions for three-dimensional systems. In Proceedings IEEE Conference on Decision and Control (CDC), 2018, pages 4059-4064. Miami Beach, FL, USA.

[6] Argáez, C., Giesl, P., and Hafstein, S. (2018b). Computational approach for complete Lyapunov functions. In Dynamical Systems in Theoretical Perspective. Springer Proceedings in Mathematics & Statistics. ed. Awrejcewicz J. (eds)., volume 248.

[7] Argáez, C., Giesl, P., and Hafstein, S. (2018c). Iterative construction of complete Lyapunov functions. In Proceedings of the 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH). Porto, Portugal.

[8] Argáez, C., Hafstein, S., and Giesl, P. (2017b). Wendland functions a C++ code to compute them. In Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), pages 323-330. Madrid, Spain.