Using AUTO for Stability Problems

Björn Sandstede and David Lloyd

June 6, 2012

1 Installation and documentation material for AUTO

We have set up an AUTO website at http://www.dam.brown.edu/people/sandsted/auto.php where information and links about installing and using AUTO can be found. The source codes used in this tutorial are also available at this website. AUTO07P is written in Fortran95, and the codes we supplied are written in a mixture of Fortran and Fortran95. The supplied codes are documented and can be used as templates for other projects.

2 Newton's method

Newton's method provides a way of solving nonlinear equations numerically using an iterative scheme that starts from a given, sufficiently accurate initial guess of the solution. For definiteness, assume that we are interested in solving $G(U) = 0$ for a given smooth function $G : \mathbb{R}^N \to \mathbb{R}^N$. We assume that this equation has a regular zero: in other words, we assume that there is a $U_* \in \mathbb{R}^N$ with $G(U_*) = 0$ such that the Jacobian $G_U(U_*)$ is invertible. We are then interested in finding arbitrarily accurate approximations of U_* starting from a reasonable guess for U_*. Newton's method works as follows: there exists an $\epsilon > 0$ such that the sequence $U_n \in \mathbb{R}^N$, defined by

$$U_{n+1} := U_n - G_U(U_n)^{-1}U_n, \qquad n \geq 0, \tag{2.1}$$

converges to U_* as $n \to \infty$ for each given U_0 with $|U_* - U_0| < \epsilon$. Thus, given a sufficiently accurate initial guess U_0 of a regular zero U_* of G, Newton's method will converge to the solution U_*. If $N = 1$, the iterates U_n are obtained by approximating the nonlinear function G by its tangent at U_n and solving the associated linear problem exactly.

3 Continuation methods

AUTO is a continuation code: it finds branches (that is, curves) of solutions of systems of the form $F(U) = 0$, where $F : \mathbb{R}^{N+1} \to \mathbb{R}^N$ is a given smooth function. Note that the solution set $\{U : \ F(U) = 0\}$ consists typically

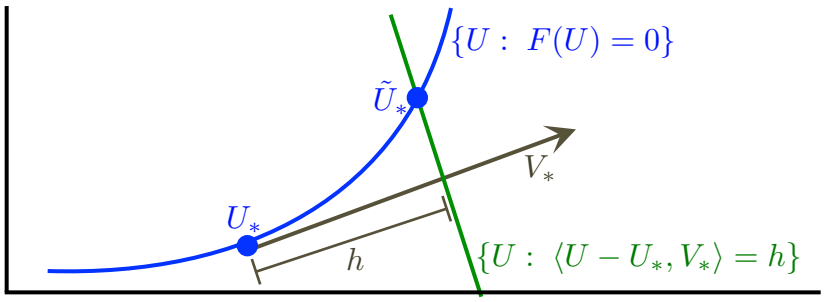

Figure 1: *A graphical illustration of the continuation algorithm used in AUTO.*

of one-dimensional smooth curves, and the issue of solving $F(U) = 0$ is equivalent to traces out these curves. Thus, given a solution U_* of $F(U) = 0$, we want to continue this solution to trace out a curve of solutions to this system as illustrated in Figure 1. The continuation algorithm implemented in AUTO works as follows:

- Assume that we found a solution U_* of $F(U_*) = 0$: then calculate a solution V_* with $|V_*| = 1$ of the linear problem $F_U(U_*)V = 0$ (such a V_* exists since $F_U(U_*) : \mathbb{R}^{N+1} \to \mathbb{R}^N$).
- Next, choose $0 < h \ll 1$ and solve

$$G(U) = \begin{pmatrix} F(U) \\ \langle U - U_*, V_* \rangle - h \end{pmatrix} = 0 \tag{3.1}$$

  using Newton's method with initial guess $U_0 = U_*$ or $U_0 = U_* + hV_*$ to obtain another solution $\tilde{U}_*$ of $F(U)$ close to U_*.
- Iterate this process by replacing U_* with the new solution $\tilde{U}_*$.

The geometric intuition behind this method is illustrated in Figure 1. Among many other enhancements, AUTO adds step-size and accuracy control to this basic continuation method. AUTO can also identify bifurcation points along the solution branch and allows branch-switching not new solution branches that emerge as bifurcation points.

The key requirements that assure that the continuation algorithm outlined above works are:

(i) A good (that is, sufficiently accurate) initial guess U_* of $F(U) = 0$.

(ii) The Jacobian $F_U(U) : \mathbb{R}^{N+1} \to \mathbb{R}^N$ is onto along the branch (or, equivalently, the null space of $F_U(U)$ is one-dimensional), except possibly at isolated points along the branch.

The first requirement (i) ensures that the first application of Newton's method can converge. The second requirement (ii) ensures that the branch is one-dimensional so that the system $F_U(U_*)V = 0$ has a unique solution, up to scalar multiples; the surjectivity of $F_U(U)$ also guarantees that the Jacobian $G_U(U)$ of the extended system $G(U) = 0$ defined in (3.1) is invertible, so that the individual Newton steps taken during the continuation algorithm are well defined.

Strategies for finding good initial guesses depend strongly on the underlying specific problem: often, finding good starting data is the main obstacle for using AUTO successfully. A few key strategies are:

- Matlab has an excellent built-in Newton trust-region solver (it is called FSOLVE and available as part of the optimization toolbox) that often converges even for very poor initial guesses; the solution obtained using Matlab can then be used in AUTO.
- Use continuation (or homotopy) from a simpler problem or from parameter values at which explicit solutions are known.
- AUTO can import column-formatted data files: this allows to import initial guesses obtained from initial-value problem solvers for ODEs or PDEs as appropriate.
- AUTO's built-in branch-switching facility can be used to switch branches at bifurcation points, for instance from equilibria to bifurcating periodic orbits at Hopf bifurcations.

4 Scope of AUTO's capabilities

AUTO can solve the following types of problems (this is not a complete list but suffices for our purposes):

- Algebraic problems:

$$F(U) = 0 \text{ with } F : \mathbb{R}^{N+1} \to \mathbb{R}^N. \tag{4.1}$$

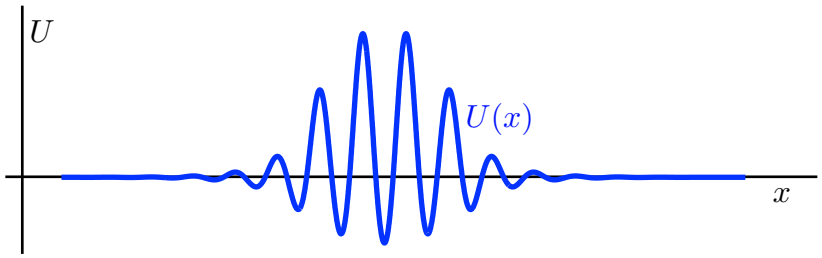

Figure 2: *A solution profile of (6.2) with $\mu = 0.2$ and $\nu = 1.6$ is plotted.*

- Boundary-value problems:

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}t} &= f(u,\mu), \qquad 0 < t < 1, \qquad (u,\mu) \in \mathbb{R}^n \times \mathbb{R}^p \\ g(u(0),u(1),\mu) &= 0, \qquad g \text{ maps into } \mathbb{R}^{n_{\mathrm{bc}}} \\ \int_0^1 h(u(t),\mu)\,\mathrm{d}t &= 0, \qquad h \text{ maps into } \mathbb{R}^{n_{\mathrm{int}}}. \end{aligned} \tag{4.2}$$

AUTO uses collocation with Lagrange polynomials, whose degree can be chosen by the user, to discretize the boundary-value problem (4.2) in the time variable t. To check whether the system can possibly satisfy the requirement (ii) above, calculate the following count, which gives a necessary condition for (ii) to hold:

$$\underbrace{(n+p)}_{\text{remaining variables after solving the ODE}} - \underbrace{(n_{\mathrm{bc}}+n_{\mathrm{int}})}_{\text{remaining equations}} \stackrel{!}{=} \underbrace{1}_{\text{one extra dimension for continuation}}$$

or

$$p = n_{\mathrm{bc}} + n_{\mathrm{int}} + 1 - n, \tag{4.3}$$

where p is the number of free parameters that AUTO adjusts during continuation.

5 Implementation

As input, AUTO requires two files. The first file, named `PROBLEM.f90` or `PROBLEM.f`, contains the description of the function F for algebraic problems or the right-hand side f of the ODE, the boundary conditions g and the integral conditions h for boundary-value problems. In the constants file `c.PROBLEM`, the user sets various constants that specify what problem AUTO should solve and how. Here, `PROBLEM` is a placeholder that the user can set to any name as desired.

The file `PROBLEM.f90` contains a number of subroutines whose purpose we now explain briefly:

- func: defines the right-hand side of the ODE (for boundary-value problems) or the function F for which we want to solve $F(U) = 0$ (for algebraic problems);
- bcnd: defines the boundary conditions;
- icnd: defines the integral conditions;
- stpnt: sets parameter values and initial guess for the start of the continuation;
- pvls: can be used to access solutions and parameter during continuation;
- fopt: used for optimization problems.

Example codes can be found in Appendix A.1 and will be explained later.

6 Case study: Localized rolls in the one-dimensional Swift–Hohenberg equation

Goal: We want to compute stationary localized roll solutions of the Swift–Hohenberg equation

$$U_t = -(1+\partial_x^2)^2 U - \mu U + \nu U^2 - U^3, \qquad x \in \mathbb{R}. \tag{6.1}$$

Such solutions satisfy the ODE

$$-(1+\partial_x^2)^2 U - \mu U + \nu U^2 - U^3 = 0, \qquad x \in \mathbb{R}, \tag{6.2}$$

and a typical solution profile is shown in Figure 2.

Problem formulation: There are two feasible approaches for finding such solutions in AUTO:

(a) Discretize in x (for instance, using centered finite differences) and solve the resulting algebraic problem in AUTO.

(b) Set up (6.2) as a boundary-value problem and let AUTO handle the discretization.

We follow the second strategy as it is much more robust: AUTO includes a very efficient mesh-adaptation algorithm that allows us to keep the number of required mesh points low. Here are the main steps involved in preparing (6.2) for implementation in AUTO:

- Fix $L \gg 1$ and consider $x \in (0, L)$ with appropriate boundary conditions at $x = 0, L$. We use Neumann conditions and require that $U_x(x) = U_{xxx}(x) = 0$ at $x = 0, L$ (other options are to use the Dirichlet conditions $U = U_{xx} = 0$ at $x = 0, L$ or periodic boundary conditions, which require that $(U, U_x, U_{xx}, U_{xxx})$ coincide at $x = 0, L$).
- Next, write the system (6.2) and the boundary conditions as the first-order system

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}x} &= f(u,\mu,\nu) := \begin{pmatrix} u_2 \\ u_3 \\ u_4 \\ -2u_3 - (1+\mu)u_1 + \nu u_1^2 - u_1^3 \end{pmatrix}, \qquad 0 < x < L \\ \begin{pmatrix} u_2 \\ u_4 \end{pmatrix} &= 0, \qquad \text{at } x = 0, L, \end{aligned} \tag{6.3}$$

  where $u = (U, U_x, U_{xx}, U_{xxx}) \in \mathbb{R}^4$.
- Rescale x by setting $x = Lt$ with $0 < t < 1$ to get

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}t} &= L f(u,\mu,\nu), \qquad 0 < t < 1 \\ \begin{pmatrix} u_2 \\ u_4 \end{pmatrix} &= 0, \qquad \text{at } t = 0, 1. \end{aligned} \tag{6.4}$$

It is instructive to consider this system more abstractly in the following form:

$$\begin{aligned} F: \quad & C^1([0,1],\mathbb{R}^4) \times \mathbb{R}^3 \longrightarrow C^0([0,1],\mathbb{R}^4) \times \mathbb{R}^4, \\ & (u,\mu,\nu,L) \longmapsto (\dot{u} - L f(u,\mu,\nu), u_2(0), u_4(0), u_2(1), u_4(1)). \end{aligned} \tag{6.5}$$

We have three free parameters, namely (μ, ν, L), in which we can continue. The count (4.3) gives

$$p = n_{\mathrm{bc}} + n_{\mathrm{int}} + 1 - n = 4 + 0 + 1 - 4 = 1;$$

hence we can continue in one of the three available parameters, while the others are kept constant: one option is to continue in μ for fixed (ν, L), another option is to continue in L for fixed (μ, ν).

Implementation in AUTO: The AUTO files for this problem are contained in the directory

`auto-tutorial/course-materials/sh-auto-bvp`

The relevant files are named `c.auto_SH_phase` and `auto_SH_phase.f90`: these are listed in Appendix A.1. Instructions for running AUTO on this example problem are provided in the file `README.txt` in the aforementioned directory and are also listed below.

Starting data: We obtained starting data by solving a finite-difference approximation of the Swift–Hohenberg equation (6.2) in Matlab using the built-in Matlab routine FSOLVE from a rough initial guess in form of a very approximate explicit function that resembles the profile we seek. The Matlab codes for generating this guess are contained in the directory

`auto-tutorial/course-materials/sh-matlab`

to which we refer for details. AUTO can start its continuation algorithm from a column-formatted data file such as the one supplied here as `auto_SH_phase.dat`.

Running AUTO: To run the AUTO code, type the following at the command line in a terminal (without the leading ">", which is used here to indicate the beginning of the command line in the terminal):

```
> @r auto_SH_phase
```

AUTO will print two lines: the first line marked with EP corresponds to the solution we start from. The second line contains an MX, which indicates that AUTO did not converge to a solution. Type

```
> ls -l
```

and you will see that AUTO generated the three files `fort.7`, `fort.8`, and `fort.9`. We can inspect their content by typing

```
> more fort.7
> more fort.8
> more fort.9
```

You see that `fort.7` contains information about each continuation step; `fort.8` contains the solution data in a format that is explained in Appendix A.3. Finally, `fort.9` contains auxiliary information about each continuation step and, in fact, about each individual Newton step during each continuation step; in particular, if the constant IID is set to 3 in the constants file, then AUTO lists the residuals for the reduced system of equations that it attempts to solve. The residuals are the remainders upon substituting the initial guess into the ODE and the boundary and integral conditions: they serve as an excellent indicator for (1) errors in the implementation of the right-hand sides and (2) the accuracy of the initial guess. Our residuals are

```
Residuals of reduced system:
-0.435E-02 -0.266E-02 -0.329E-02 0.385E-02 0.109E-20 0.153E-20 -0.154E-20 0.000E+00 0.812E-02
```

and therefore quite small. So, why does AUTO fail? The reason is that our system violates the requirement (ii), namely that the derivative of our system (6.4) in u, evaluated at the initial guess or a genuine solution, is invertible. Indeed, Figure 3 indicates that we can shift the initial guess in x without changing the values of the boundary conditions much; since the shifted function will still satisfy the ODE, we obtain an approximate one-parameter family of solutions to (6.4). This shows that the derivative $F_u(u_*, \mu_*, \nu_*, L_*)$ of (6.5) in u will have an eigenvalue close to zero, thus precluding invertibility: we can therefore not expect that Newton's method will converge. We therefore need to select a single element from the family of shifted functions to make the solution unique. A good selection criterion, often referred to as a phase condition as it selects the phase or shift of the translated functions, is the integral condition

$$\Phi(u) := \int_0^1 \langle \dot{u}_{\rm old}(t), u(t) - u_{\rm old}(t) \rangle \, {\rm d}t = 0, \tag{6.6}$$

where $u_{\rm old}$ refers to the solution at the previous continuation step, or the initial guess when AUTO begins continuation. Indeed, omitting the arguments (μ, ν, L) of F for the sake of clarity, we know that $F(u_*(\cdot + \epsilon))$

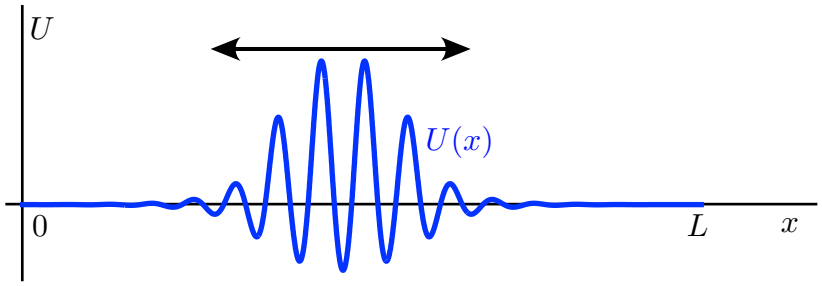

Figure 3: *The solution profile of (6.4) on the bounded interval $[0, L]$ can be shifted back and forth without a noticeable change of the boundary conditions: this indicates that the linearization of (6.4) in u will have an eigenvalue close to zero. In the case of periodic boundary conditions, we can, in fact, shift the profile arbitrarily in the x-direction: the shifted functions remain solutions, and the linearization of (6.4) in u will always have an eigenvalue at the origin.*

will be approximately equal to zero as ϵ varies near zero. Assuming for simplicity that $F(u_*(\cdot + \epsilon)) = 0$ for all ϵ near zero, we find that

$$0 = \frac{\mathrm{d}}{\mathrm{d}\epsilon} F(u_*(\cdot + \epsilon))\big|_{\epsilon=0} = F_\epsilon(u_*)\dot{u}_*$$

so that $\dot{u}_*$ lies in the null space of $F_u(u_*)$, and will, in fact, be a basis of the null space unless there are additional generacies in the system. If $\dot{u}_*$ spans the null space of $F_u(u_*)$, then the new map $\tilde{F} : u \mapsto (F(u), \Phi(u))$ has trivial null space since $F_u(u_*)v = 0$ implies $v = \dot{u}_*$, up to scalar multiples, while

$$\Phi_u(u_*)v = \langle \dot{u}_*, v\rangle \overset{v=\dot{u}_*}{=} \langle \dot{u}_*, \dot{u}_*\rangle \neq 0.$$

Thus, we should add the phase condition (6.6) to our system (6.4) and the AUTO code. However, since we add an equation, we need an additional parameter that compensates for the additional phase constraint we added. Here is how we can identify an appropriate parameter: The reason we have a one-parameter family of solutions is that the original PDE (6.1),

$$U_t = -(1 + \partial_x^2)^2 U - \mu U + \nu U^2 - U^3, \qquad x \in \mathbb{R},$$

is invariant under translations in x. This, in turn, implies that we can seek travelling-wave solutions of the form $U = U(x - ct)$ of (6.1). Our solutions are stationary, that is have $c = 0$, but we can add the wave speed c as a free parameter to our problem. The rationale for this choice is explained in detail, and in much more generality, in [2]: in a nutshell, the derivative with respect to c corresponds to taking the partial derivative ∂_x with respect to x which generates the semigroup of translations. Introducing the new moving coordinate $x \mapsto x - ct$ in (6.1) gives the PDE

$$U_t = -(1 + \partial_x^2)^2 U + cU_x - \mu U + \nu U^2 - U^3, \qquad x \in \mathbb{R}$$

with the additional term cU_X. Incorporating this additional term by going through the previous steps, we end up with the new right-hand side

$$f(u, \mu, \nu, c) := \begin{pmatrix} u_2 \\ u_3 \\ u_4 \\ cu_2 - 2u_3 - (1 + \mu)u_1 + \nu u_1^2 - u_1^3 \end{pmatrix}.$$

We implement this in AUTO by making the following changes in `auto_SH_phase.f90`:

- comment out line 25 and uncomment line 26, which contains the new right-hand side;
- uncomment line 79, which contains the phase condition (6.6)

and the following changes in the constants file `c.auto_SH_phase`:

- change the line for NICP to "2 1 3" to include the wave speed c =par(3) as free parameter;
- change NINT from 0 to 1 to tell AUTO that we added an integral condition.

Now, we can start AUTO again:

```
> @r auto_SH_phase
```

and see that AUTO successfully computes a branch of solutions. Note that the wave speed c is very close to zero as expected. We can plot the bifurcation diagram and the solution profiles using AUTO's built-in plotting program by typing

```
> @pp
```

We can save the data we just computed and which are currently stored in `fort.7`, `fort.8`, and `fort.9` in the new files `b.run1`, `d.run1`, and `s.run1`, respectively, by typing

```
> @sv run1
```

The solutions labelled LP and BP correspond to fold and branch points, respectively: the branch points correspond to pitchfork bifurcations at which asymmetric profiles emerge that are odd in x. AUTO allows us to switch branches from the curve of symmetric profiles to the bifurcating branch of asymmetric solutions: we pick the branch point corresponding to label 4 and therefore make the following changes in the constants file `c.sh_auto_phase`:

- set IRS=4 to indicate that we wish to restart from the solution with label 4;
- set ISW=-1 to tell AUTO that we want to switch branches;
- set NMX=20 so that we take only 20 steps along the new branch.

Our data are now stored in `*.run1`, and we therefore restart from these data files by typing

```
> @r auto_SH_phase run1
```

To save the data in the files `*.run2` and plot them, we type

```
> @sv run2
> @pp run2
```

Note that we indeed computed asymmetric solutions that bifurcate at the branch point from the symmetric localized rolls. AUTO can plot data from only one set of files: to plot symmetric and asymmetric branches together, we copy and append the data as follows in the new files `*.run_all`:

```
> @cp run1 run_all
> @ap run2 run_all
> @pp run_all
```

Finally, we discuss how we can compute the PDE spectra of the profiles we computed in Matlab. First, we re-compute the profiles: make the following changes in the constants file `c.sh_auto_phase`:

- set IRS=0, ISP=1, ILP=0, NPR=2, DSMAX=0.01.

and start AUTO:

```
> @r auto_SH_phase
> @sv run
```

Next, we convert the solution data to Matlab format by typing

```
> autox to_matlab_sh.xauto run run
```

This command generates the files `run_bifur`, `run_label`, and `run_solution_*`, which contain, respectively, the bifurcation diagram, the labels plus par(1), and the individual solutions. The file `to_matlab_sh.xauto` is a python script that extracts and reformats AUTO data. Next, open Matlab and run

```
> sh_spectrum
```

which computes and plots the spectra of the localized rolls we computed.

Outline of other methods for finding localized rolls:

- The directory

  `auto-tutorial/course-materials/sh-auto-bvp`

  also contains the AUTO code `auto_SH_half.f90` with the constants files `c.auto_SH_half`, which can be used to compute half of a symmetric roll structure. In this case, no phase condition is needed, but the code cannot detect the pitchfork bifurcations. Instructions for running this code are provided in the `README.txt` file.
- The directory

  `auto-tutorial/course-materials/sh-auto-finite-differences`

  contains AUTO codes that implement the computation of localized rolls using finite-difference approximations together with AUTO's capability to solve and continue solutions of algebraic problems. In the case of computing localized structures that are centered in the interior of the interval [0,L], we again need a phase condition, which is implemented as an additional equation. The advantage of these codes is that they provide information about the PDE spectrum; the disadvantage is that the underlying spatial mesh is fixed. *Caveat:* The eigenvalues that AUTO computes along a branch are those of the linearization of the algebraic problem that we solve; since we added a phase condition, it is not obvious (and in general wrong) that the eigenvalues AUTO computes agree with the eigenvalues of the PDE linearization we are interested in: For the self-adjoint problem considered here, it can be shown that they indeed agree.

7 Tips and tricks for using AUTO

- Debugging:
  - Always set IID=3 so that AUTO writes the residuals to `fort.9`. The residuals are extremely helpful in checking for errors in the code or the initial guess.
  - The error "A null space may be multi-dimensional" indicates that the requirement (ii) is violated: check that your solution is unique or modify your system to make the solution unique.
  - If you encounter MX errors, try to reduce the accuracy by making the EPS constants larger; often, it also helps to reduce or increase the step-size constant DS. For problems on large intervals, it often helps to make the interval smaller or larger. Also, vary the number NTST of mesh points. Over time, you will get a feeling for what might go wrong and which of these recipes works best in a given situation.
  - Try to make your system simpler: start with a reduced system if at all possible
- Newton's method: By default, AUTO's first continuation step will always be a genuine continuation step. If your initial data are not particularly accurate or if AUTO does not converge, try to continue in a dumb parameter, that is in a parameter that you actually do not use in your code. Continuing in a dummy parameter means that AUTO solves the underlying system using Newton's method.

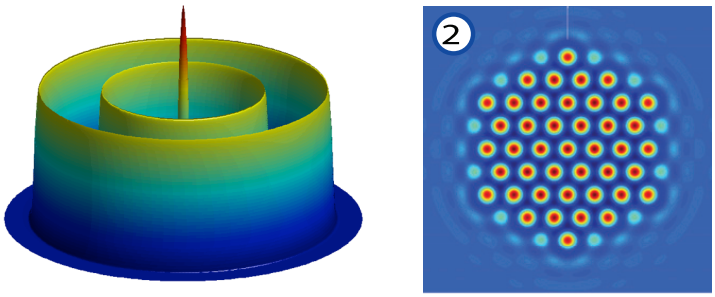

Figure 4: *Left: Shown is the spatial profile at a fixed time of a radially symmetric localized oscillon pattern that satisfies a certain planar four-component reaction-diffusion system. Right: Stationary localized hexagon patch of the planar Swift–Hohenberg equation.*

- Large-scale problems: For large problems, it helps to specify the Jacobian of the right-hand side and the boundary and integral conditions explicitly in the AUTO code. If the Jacobian is not specified, AUTO computes it internally by finite differences.
- Export and import to Matlab: A common issue is to combine AUTO data from two different problems to compute a solution to a larger problem. For instance, once we computed a localized roll profile, we might want to add the most unstable eigenvalue and its eigenfunction and set up a problem that computes simultaneously the profile and this eigenvalue with the associated eigenfunction as parameters vary. The best way of generating initial data in these circumstances is to export the original AUTO data using the command

  ```
  > autox to_matlab.xauto AUTO_DATA_FILENAME OUTPUT_FILENAME
  ```

  and manipulate the data in Matlab to generate new initial guesses that combine the two data sets. AUTO can then be restarted from the columnn-formatted data file as illustrated in our case study.
- If you need access to the independent variable x, we can add the equation $\dot{x} = L$ to the problem. To adjust the initial data, we need to add the values of x to the starting data, which can be done, for instance, as outlined under the previous bullet item.

8 Computation of multi-dimensional patterns

We outline briefly how AUTO can be used to compute multi-dimensional patterns that depends on $x \in \mathbb{R}^d$ with $d > 1$ and spatiotemporal time-periodic patterns that depend on (x, t). In all these cases, the only possible approach is to discretize in all but independent variables and to solve the resulting ODE in the remaining variable in AUTO as a boundary-value problem. Possible discretizations are finite-differences or spectral methods such as Fourier series or Chebyshev polynomials. In some cases, there will be a natural choice for which variable is best suited to be handled by AUTO; in other cases, many choices are possible. We focus on the two patterns shown in Figure 4.

The left image in Figure 4 is of an oscillon, a time-periodic radial localized solution of the form $U(x, t) = U_*(r, t)$, where $x \in \mathbb{R}^2$, $r = |x|$ and $U_*(\cdot, t+T) = U_*(\cdot, t)$ for all t for some period $T > 0$. Thus, the profile $U_*(r, t)$ depends on two independent variables. The oscillon shown in Figure 4 was computed by discretizing in r using centered finite differences and implementing the resulting problem in t as a boundary-value problem in AUTO. This allowed us to continue in external parameters, whilst keeping track of the temporal period. In addition, period-doubling bifurcations of oscillons could be detected efficiently using this approach.

The right image in Figure 4 shows a localized planar hexagon patch [3]. It was computed by writing the planar Swift–Hohenberg equation in polar coordinates, truncating the Fourier series in the angle φ, and formulating the resulting problem in the remaining radial variable r as a boundary-value problem in AUTO. This allowed us

to adjust the domain length during continuation and furthermore enabled us to exploit the symmetries of the pattern by restricting to those Fourier modes compatible with the underlying symmetry of the pattern.

Other examples of patterns computed with AUTO can be found in [1]

A.1 AUTO files for the case study `auto_SH_phase`

The right-hand sides are specified in the file `auto_SH_phase.f90`, We list the content of this file below with all declarations of variables removed for ease of reading:

```
!---------------------------------------------------------------------
subroutine func(ndim,u,icp,par,ijac,f,dfdu,dfdp)
  mu = par(1)
  nu = par(2)
  c  = par(3)

  f(1) = u(2);
  f(2) = u(3);
  f(3) = u(4);
  f(4) = -(1+mu)*u(1) + nu*u(1)*u(1) - u(1)*u(1)*u(1) - 2*u(3)
! f(4) = -(1+mu)*u(1) + nu*u(1)*u(1) - u(1)*u(1)*u(1) - 2*u(3) + c*u(2)

  do j=1,ndim
     f(j) = par(11)*f(j)
  end do
end subroutine func
!---------------------------------------------------------------------
subroutine stpnt(ndim,u,par,t)
  par(1)  = 0.2     ! mu
  par(2)  = 1.6     ! nu
  par(3)  = 0.0     ! c (wave speed: should be zero)
  par(11) = 100.0   ! interval length L
end subroutine stpnt
!---------------------------------------------------------------------
subroutine bcnd(ndim,par,icp,nbc,u0,u1,fb,ijac,dbc)
  ! Neumann boundary conditions
  fb(1) = u0(2)
  fb(2) = u0(4)
  fb(3) = u1(2)
  fb(4) = u1(4)
end subroutine bcnd
!---------------------------------------------------------------------
subroutine icnd(ndim,par,icp,nint,u,uold,udot,upold,fi,ijac,dint)
! fi(1) = upold(1)*(u(1)-uold(1))   ! phase condition
! fi(2) = u(1)*u(1)-par(4)          ! L^2 norm
end subroutine icnd
!---------------------------------------------------------------------
subroutine pvls
  end subroutine pvls
subroutine fopt
  end subroutine fopt
!---------------------------------------------------------------------
```

Various constants that AUTO needs to now about are specified in the file `c.auto_SH_phase`:

```
dat='auto_SH_phase.dat'
4 4 0 1                          NDIM,IPS,IRS,ILP
1 1                              NICP,(ICP(I),I=1 NICP)
400 4 3 2 1 0 4 0                NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
100 -1.e+8 1.e+8 -1.e+8 1.e+8    NMX,RL0,RL1,A0,A1
10 10 3 8 5 3 0                  NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1.e-7 1.e-7 1.e-5                EPSL,EPSU,EPSS
-0.01 0.001 0.1 1                DS,DSMIN,DSMAX,IADS
0                                NTHL,(/,I,THL(I)),I=1,NTHL)
0                                NTHU,(/,I,THU(I)),I=1,NTHU)
0                                NUZR,(/,I,PAR(I)),I=1,NUZR)
```

These constants are explained in Appendix A.2

A.2 Explanation of the constants that appear in the constants file

e, s, dat, sv	Define file names: equation prefix (.f,.f90,.c), restart solution suffix (s.), user data prefix (.dat), output suffix (b.,s.,d.)
unames, parnames	Dictionary (mapping) of U(*) and PAR(*) to user-defined names
NDIM	Problem dimension
IPS	Problem type; 0=AE, 1=FP(ODEs), -1=FP(maps), 2=PO, -2=IVP, 4=BVP, 7=BVP with Floquet multipliers, 5=algebraic optimization problem, 15=optimization of periodic solutions
IRS, TY	Start solution label, start solution type
ILP	Fold detection; 1=on, 0=off
ICP	Continuation parameters
NTST	# mesh intervals
NCOL	# collocation points
IAD	Mesh adaption every IAD steps; 0=off
ISP	Bifurcation detection; 0=off, 1=BP(FP), 3=BP(PO,BVP), 2=all
ISW	Branch switching; 1=normal, -1=switch branch (BP, HB, PD), 2=switch to two-parameter continuation (LP, BP, HB, TR) 3=switch to three-parameter continuation (BP)
IPLT	Select principal solution measure
NBC	# boundary conditions
NINT	# integral conditions
NMX	Maximum number of steps
RL0, RL1	Parameter interval $RL0 \leq \lambda \leq RL1$
A0, A1	Interval of principal solution measure $A0 \leq \|\cdot\| \leq A1$
NPR	Print and save restart data every NPR steps
MXBF	Automatic branch switching for the first MXBF bifurcation points if IPS=0, 1
IBR, LAB	Set initial branch and label number; 0=automatic
IIS	Control solution output of branch direction vector; 0=never, 3=always
IID	Control diagnostic output; 0=none, 1=little, 2=normal, 4=extensive
ITMX	Maximum # of iterations for locating special solutions/points
ITNW	Maximum # of correction steps
NWTN	Corrector uses full newton for NWTN steps
JAC	User defines derivatives; 0=no, 1=yes
EPSL, EPSU, EPSS	Convergence criterion: parameters, solution components, special points
DS	Start step size
DSMIN, DSMAX	Step size interval DSMIN $\leq h \leq$ DSMAX
IADS	Step size adaption every IADS steps; 0=off
NPAR	Maximum number of parameters
THL, THU	list of parameter and solution weights
UZR, UZSTOP	list of values for user defined output
SP, STOP	list of bifurcations to check and bifurcation stop conditions
NUNSTAB, NSTAB	HomCont: unstable and stable manifold dimensions
IEQUIB, ITWIST, ISTART	HomCont: control solution types adjoint, starting
IREV, IFIXED, IPSI	HomCont: control reversibility, fixed parameters, test functions

A.3 Format of AUTO output files

The files `fort.8` or `s.NAME` have the following format:

```
first identifying line:
   ibr   :   the index of the branch
   ntot  :   the index of the point
   itp   :   the type of point
   lab   :   the label of the point
   nfpr  :   the number of free parameters used in the computation
   isw   :   the value of isw used in the computation
   ntpl  :   the number of points in the time interval [0,1]
       for which solution data are written
   nar   :   the number of values written per point
       (nar=ndim+1, since t and u(i), i=1,..,ndim are written)
   nrowpr:   the number of lines printed following the identifying line
       and before the next data set or the end of the file
       (used for quickly skipping a data set when searching)
   ntst  :   the number of time intervals used in the discretization
   ncol  :   the number of collocation points used
   nparx :   the dimension of the array par
following this are ntpl lines containing
   t u_1(t) u_2(t) ... u_ndim(t)
following this is a line containing the indices of the free parameters
   icp(i) for i=1,...,nfpr
followed by a line containing the derivative of parameters wrt arclength
   rl_dot(i) for i=1,...,nfpr
following this are ntpl lines containing the derivative of the solution wrt arclength
   u_dot_1(t) u_dot_2(t) ... u_dot_ndim(t)
followed by the parameter values
   par(i) for i=1,...,nparx
```

If we set `IID=3`, then the files `fort.9` or `d.NAME` contain the residuals of the reduced system in the following format:

```
Residuals of reduced system:
   NDIM differential equations, NBC boundary conditions, NINT integral conditions,
   pseudo-arclength condition
```

A.4 AUTO-07P projects

The following three sample projects could be pursued by participants:

- Demo pp2: if you would like to use prepared AUTO files, try this demo to continue equilibria of the planar ODE, locate and continue saddle-node and Hopf bifurcations, and continue periodic orbits.
- Compute a circle using continuation: this problem allows you to implement an algebraic problem in AUTO.
- Compute and continue a pulse solution: this project is about the computation of travelling waves in AUTO via a boundary-value-problem formulation. Starting data come from an explicit solution.

The source files for the solutions of these projects are contained in the directory

`auto-tutorial/projects`

Other sample project are listed in the following section.

Predator-prey model: Consider the predator-prey model

$$\dot{u} = bu(1-u) - uv - a(1-\mathrm{e}^{-cu}), \qquad \dot{v} = -v + duv$$

where $u, v \geq 0$ and $a, b, c, d > 0$. This system is used for the AUTO demo `pp2` for which details are given in the AUTO manual. The code is set up in the directory `auto-pp2-demo`: go to this directory and follow the commands outlined in the `README` file.

Compute a circle with AUTO: Trace out the circle $x^2 + y^2 = 1$ numerically with AUTO using continuation. The solution can be found in the directory `auto-circle`.

Pulses in a bistable partial differential equation: Consider the bistable PDE

$$u_t = u_{xx} - u + au^3, \qquad x \in \mathbb{R},$$

which admits the standing pulse solution $u(x,t) = \sqrt{2}\,\mathrm{sech}(x)$ when $a = 1$. Compute and continue these pulses numerically in AUTO starting from the exact solution. Experiment with using different truncation intervals and different values of NTST. The solution can be found in the directory `auto-bistable`.

A.5 Additional AUTO-07P projects

Computation of real eigenvalues: Take your favorite $n \times n$ matrix A and determine its real eigenvalues numerically with AUTO via continuation:

- AUTO: solve the algebraic system $(A-\lambda)v = 0$ for v by continuation in $\lambda \in \mathbb{R}$ with starting data $(v, \lambda) = 0$. Enable detection of branch points. What do you find?
- Theory: investigate why this algorithm works.

Hopf bifurcations and branch switching: Consider the Brusselator

$$\dot{u} = a - (b+1)u + u^2 v, \qquad \dot{v} = bu - u^2 v$$

where $u, v \geq 0$ and $a, b > 0$. This system has the unique equilibrium $(u, v) = (a, b/a)$.

- AUTO: continue the equilibrium numerically and investigate its Hopf bifurcations. Trace out the curve of Hopf bifurcations in (a, b)-space, and compute the periodic orbits that bifurcate from the equilibrium. Check `fort.9` to see whether the periodic orbits are stable or unstable.
- Theory: compare the location of the Hopf bifurcation curve in (a, b)-space with the theoretical prediction.

Pitchfork bifurcations and branch switching: Consider the second-order equation

$$\ddot{u} = \sin(u)\left[\cos(u) - \frac{1}{\gamma}\right]$$

that describes a bead that slides on a wire hoop. This system has the equilibrium $u = 0$ for all values of γ. Investigate its bifurcations numerically using AUTO: branch switch onto any bifurcating solutions by (i) using the branch switching algorithm built into AUTO and (ii) adding a symmetry-breaking term.

Fronts in the Nagumo equation: Consider the Nagumo PDE

$$u_t = u_{xx} + u(u-a)(1-u), \qquad x \in \mathbb{R},$$

which admits the travelling fronts $u(x,t) = 1 + \tanh((x+ct)/2)$ with $c = (1-2a)/\sqrt{2}$ for $0 < a < 1$. Compute and continue these fronts numerically in AUTO starting from the exact solution. Experiment with using different truncation intervals and different values of NTST.

Finite-difference approximations of PDEs: Consider the Nagumo PDE

$$u_t = u_{xx} - cu_x + u(u-a)(1-u)$$

in a moving coordinate frame on a large interval $(-L, L)$ with Dirichlet boundary conditions at $x = \pm L$. Discretize this PDE in space using centered finite differences and implement the resulting large algebraic system in AUTO. Using appropriate values of L and the step size (you need to experiment), find the travelling waves $u(x) = 1 + \tanh(x/2)$ with $c = (1-2a)/\sqrt{2}$ in AUTO and determine the stability of the front by checking `fort.9`. Plot the solution profiles in MATLAB and compare them with the exact solution.

References

[1] D. Avitabile, D. J. B. Lloyd, J. Burke, E. Knobloch and B. Sandstede. To snake or not to snake in the planar Swift–Hohenberg equation. *SIAM Journal on Applied Dynamical Systems* **9** (2010) 704–733.

[2] A. R. Champneys and B. Sandstede. Numerical computation of coherent structures. In: *Numerical Continuation Methods for Dynamical Systems* (B. Krauskopf, H. M. Osinga and J. Galan-Vioque, eds.). Springer, 2007, 331–358.

[3] D. J. B. Lloyd, B. Sandstede, D. Avitabile and A. R. Champneys. Localized hexagon patterns of the planar Swift–Hohenberg equation. *SIAM J. Appl. Dynam. Syst.* **7** (2008) 1049–1100.