A guide to using AUTO

Nathan L. Sanford

Applied Math
Northwestern University

May 17, 2018

NORTHWESTERN
UNIVERSITY

Table of Contents

History

- Created and maintained by Eusebius Doedel
- Most recent version from 2007 (original 1976)
- Resources used in this tutorial
  - AUTO manual (updated in 2012)
  - Björn Sandstede's (Brown) minicourse from
  - Papers by Doedel, Keller, and Kernevez, 1991 (I) and (II)

Theory and Capabilities

1. Based around pseudoarclength continuation with Newton iteration
2. Problem types
   - Algebraic - $f(u,p) = 0$
   - ODEs - $u'(t) = f(u(t),p)$
   - Parabolic PDEs - $u_t = Du_{xx} + f(u,p)$
   - Optimization
3. Systems can be conditioned with boundary, initial, and integral conditions

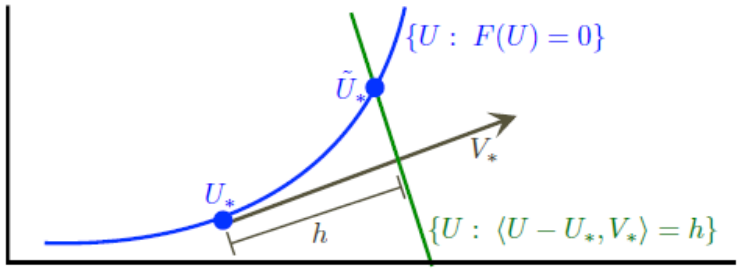


Figure: Geometric description of the pseudoarclength setup. From Sandstede's tutorial.

Pros and cons

Pros

- Powerful:
  - Can capture a lot of behavior
    - ★ Folds and bifurcations
    - ★ Equilibria and time-dep. solutions
    - ★ Multi-paramater continuation
    - ★ Homoclinics (HomCont)
  - Numerically robust
- Customizable:
  - Scripting gives complete control in parameter space
  - Scalable to higher dimensions
  - Lots of available examples/documentation

Cons

- Hard to use:
  - Relatively expensive to program
  - Old and deprecated/buggy
    - ★ Python implementation
    - ★ Plotting utilities
    - ★ Non-linux OS difficult
- Difficult to learn:
  - Documentation is opaque
  - Hard to debug scripts

Table of Contents

Pseudoarclength setup - handling folds

- Consider $f(u, \lambda) = 0, u \in \mathbb{R}^n, \lambda \in \mathbb{R}, f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ with a particular solution at u_0, λ_0 known (and a nonsingular Jacobian for f there).
- The *pseudoarclength continuation* system is

$$f(u_1, \lambda_1) = 0, \quad (u_1 - u_0)^T \dot{u}_0 + (\lambda_1 - \lambda_0)\dot{\lambda}_0 - \Delta s = 0, \qquad (\star)$$

- Δs is the steplength, $\dot{u}_0, \dot{\lambda}_0$ are the normalized directions at u_0, λ_0.
- These directions are computed using the full Jacobian of f (relying on the implicit function theorem) and normalizing.
- This extended system can handle folds, provided Δs is sufficiently small.

Pseudoarclength step

- $(\star)$ solved with Newton's method

$$\begin{pmatrix} (f_u^1)^{(\nu)} & (f_\lambda^1)^{(\nu)} \\ \dot{u}_0^T & \dot{\lambda}_0 \end{pmatrix} \begin{pmatrix} \Delta u_1^{(\nu)} \\ \Delta \lambda_1^{(\nu)} \end{pmatrix} = - \begin{pmatrix} f(u_1^{(\nu)}, \lambda_1^{(\nu)}) \\ (u_1^{(\nu)} - u_0)^T \dot{u}_0 + (\lambda_1^{(\nu)} - \lambda_0)\dot{\lambda}_0 - \Delta s \end{pmatrix} \quad (\dagger)$$

- Matrix solves accomplished via the *bordering algorithm*
  - Compute the LU factorization (with pivoting) of $(f_u^1)^{(\nu)}$ and then deal with the remaining blocks separately.
  - Able to still solve when $(f_u^1)^{(\nu)}$ is singular but overall system is not by computing left and right null vectors of $(f_u^1)^{(\nu)}$ (used for folds)
- Chord method sometimes used instead of full-Newton (Derivatives not updated between ν-iterates)
- Convergence criteria: $\frac{|\Delta\lambda|}{1+|\lambda|} < \epsilon_\lambda, \quad \frac{||\Delta u||_\infty}{1+||u||_\infty} < \epsilon_u$

Getting directions

- Direction vectors for next step, $(\dot{u}_1, \dot{\lambda}_1)$, computed by one additional solve (same matrix as in last iteration of $(\dagger)$)

$$\begin{pmatrix} (f_u^1)^{(\nu)} & (f_\lambda^1)^{(\nu)} \\ \dot{u}_0^T & \dot{\lambda}_0 \end{pmatrix} \begin{pmatrix} \dot{u}_1 \\ \dot{\lambda}_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

  and then by normalizing so that $||\dot{u}_1||^2 + \dot{\lambda}_1^2 = 1$

- Initial direction vector computed by solving $\begin{pmatrix} f_u^0 & f_\lambda^0 \\ 0^T & 0 \end{pmatrix} \begin{pmatrix} \dot{u}_0 \\ \dot{\lambda}_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

  with normalization $||\dot{u}_0||^2 + \dot{\lambda}_0^2 = 1$

Additional notes

- Step size handled adaptively: increased if convergence rapid, halved if convergence slow/failed
- If Jacobians aren't given, approximated with finite differences

$$f_u^j \approx \frac{f(u_j + \epsilon, \lambda_j) - f(u_j - \epsilon, \lambda_j)}{2\epsilon} \qquad \text{(and/or)}$$
$$f_\lambda^j \approx \frac{f(u_j, \lambda_j + \epsilon) - f(u_j, \lambda_j)}{\epsilon}$$

- Initial guess for Newton provided by $\begin{pmatrix} u_1^{(0)} \\ \lambda_1^{(0)} \end{pmatrix} = \begin{pmatrix} u_0 + \Delta s\, \dot{u}_0 \\ \lambda_0 + \Delta s\, \dot{\lambda}_0 \end{pmatrix}$

Branch points

- Again consider $f(u, \lambda) = 0$ but now with (u_0, λ_0) a *simple singular point* where either

$$\begin{aligned} &\text{(i)} \ \dim \mathcal{N}(f_u^0) = 1, \quad f_\lambda^0 \in \mathcal{R}(f_u^0), && \text{or} \\ &\text{(ii)} \ \dim \mathcal{N}(f_u^0) = 2, \quad f_\lambda^0 \notin \mathcal{R}(f_u^0). \end{aligned}$$

- Letting $x = (u, \lambda) \in \mathbb{R}^{n+1}$, we have $\mathcal{N}(f_x^0) = \text{Span}\{\phi_1, \phi_2\}$ and $\mathcal{N}((f_x^0)^T) = \text{Span}\{\psi\}$, as by above, $\text{Rank}(f_x^0) = n - 1$ in either case
- Two directions through the singular point x_0 can be constructed in the form $\alpha\phi_1 + \beta\phi_2$
- The constants α, β come from the *algebraic bifurcation equation*

$$c_{11}\alpha^2 + 2c_{12}\alpha\beta + c_{22}\beta^2 = 0, \quad c_{ij} = \psi^T f_{xx}^0 \phi_i \phi_j, i, j = 1, 2. \qquad (\diamond)$$

Branch switching

- The direction of the given branch is in $\mathcal{N}(f_x^0)$ so choose $\phi_1 = \dot{x}_0$
- Let $F(x; s) \equiv \begin{pmatrix} f(x) \\ (x - x_0)^T \dot{x}_0 - s \end{pmatrix}$, the pseudoarclength system
- Then $F_x^0 = \begin{pmatrix} f_x^0 \\ \dot{x}_0^T \end{pmatrix} = \begin{pmatrix} f_x^0 \\ \phi_1^T \end{pmatrix}$ has a one-dimensional null space
- Choosing ϕ_2, the second null vector of f_x^0, such that $\phi_2 \perp \phi_1$ also implies $\phi_2 \in \mathcal{N}(F_x^0)$
- Motivates choosing $x_0' = \phi_2$ for direction of bifurcating branch
- This *orthogonal direction method* is an approximation to using $(\diamond)$ that "works well in most practical applications" [Doedel, 1991 (I)]

Tracking bifurcations

- A scalar function $q(s)$ is tracked along the computed branch $x(s)$, and sign changes in it trigger a secant iteration to find the precise location of the bifurcation:

$$s^{\nu+1} = s^{\nu} - \frac{s^{\nu} - s^{\nu-1}}{q(s^{\nu}) - q(s^{\nu-1})} q(s^{\nu})$$

- $q(s) = \det(F_x)$ for simple bifurcations (from analysis of $(\diamond)$)
  - Relatively cheap to get from already computed LU decompositions
  - Analysis due to [Keller, 1978; 1987]
- $q(s) = \dot{\lambda}(s)$ for folds

Brief description of method for ODEs

$$
\begin{aligned}
u'(t) &= f(u(t), \lambda), \quad && t \in [0,1], u(\cdot), f(\cdot) \in \mathbb{R}^n \\
b(u(0), u(1), \lambda) &= 0, && b(\cdot) \in \mathbb{R}^{n_b} \\
\int_0^1 q(u(t), \lambda)\, dt &= 0, && q(\cdot) \in \mathbb{R}^{n_q}
\end{aligned}
$$

- The system is discretized using *orthogonal collocation*
- Piecewise continuous polynomials defined on m points between each point of an N-point mesh of $[0,1]$ using Lagrange interpolation

$$\{l_{j,i}(t)\},\ j = 1, \ldots, N,\ i = 1, \ldots, m$$

- Then seek to find

$$p_j(t) = \sum_{i=0}^{m} l_{j,i}(t) u_{j-(i/m)}, \text{ where } u_{j-(i/m)} \approx u(t_j - \frac{i}{m}\Delta t_j)$$

System discretization

- The pseudoarclength condition is now

$$\int_0^1 (u(t) - u_0(t))^T \dot{u}_0(t)\, dt + (\lambda - \lambda_0)\dot{\lambda}_0 - \Delta s = 0$$

- Integrals discretized with a Lagrange quadrature (Lagrange interpolation with appropriate weights)
- The Jacobian matrix to be inverted is then a $mnN + n_b + n_q + 1$ square matrix
- Local updates between mesh points dealt with using *condensation of parameters*
- Parts of the system can then be decoupled and solved efficiently

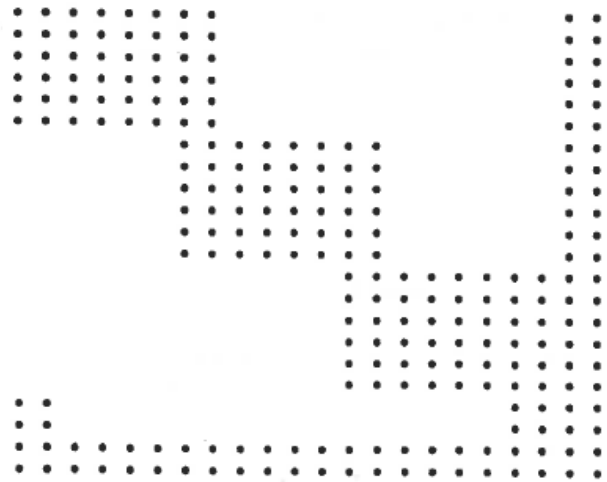


Table of Contents

Installation Basics

- Requires a Unix shell with a Fortran 90 compiler and Python to be installed on the system
  - Only Python 2.x versions supported (Newer than 2.4 sometimes throws warnings)
  - Need NumPy and Matplotlib (unclear if AUTO plays nicely with full Python distributions like Anaconda etc.)
- Unzipped AUTO file includes the manual and 'demos'
- After unzipping, need to configure and compile AUTO before running
- Easiest on Linux (instructions for Fedora, Ubuntu, Debian in manual)
- Unnecessary to install utilities for PLAUT04 and the AUTO GUI
- Balance AUTO manual recommendations with those from Sandstede's tutorial

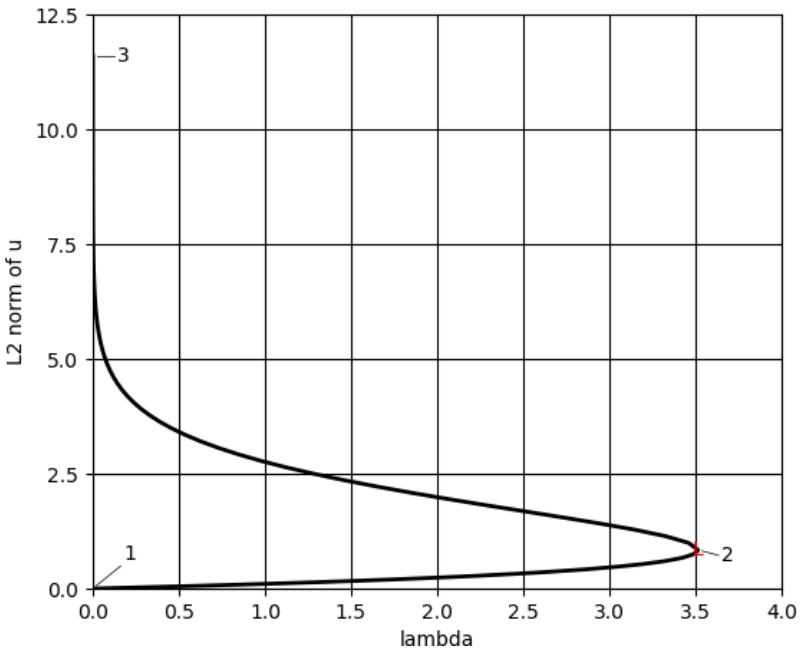
Comments on Windows Installation

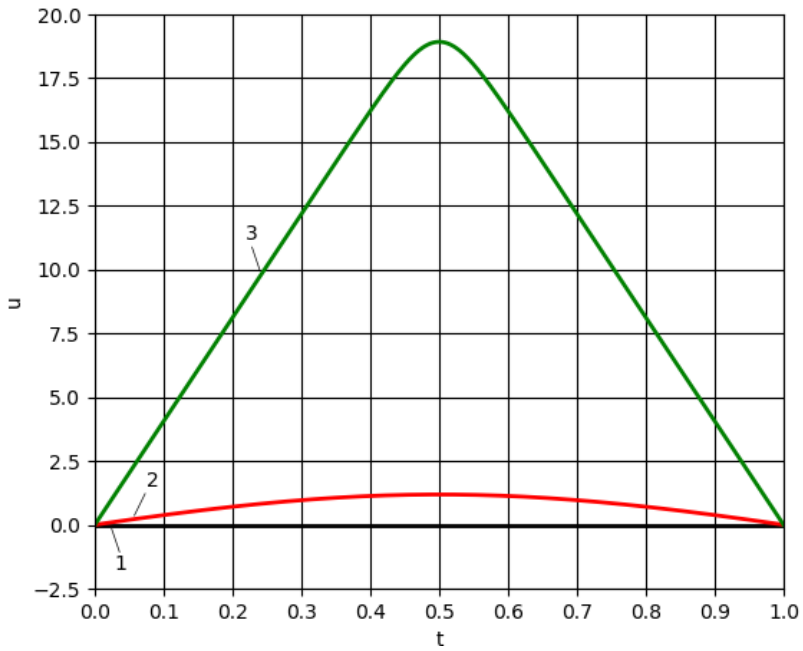
- Sandstede's tutorial suggests installing a Linux partition
- My more lightweight build:
  - MinGW and MSYS (Compilers and CLUI with shell commands)
  - Python 2.7 with SciPy and Matplotlib installed using pip
  - Notepad++ file editor
  - AUTO installed in the MinGW home directory (Python in the normal place)
- The configure step can sometimes trigger anti-virus software. Turn off the anti-virus software and then reattempt.

Table of Contents

Tutorial example - Bratu's problem

$$u'' + \lambda e^u = 0$$
$$u(0) = u(1) = 0$$





(Also available in AUTO demos **exp**, **pvl**, and **ezp**)

Table of Contents

Equations file

- Fortran or C file
- Supply portions of the problem statement in a specialized format
- Sometimes necessary to reformulate problem (autonomous, first degree system)
- Called x.f90 or x.f (i.e. bratu.f90)

Bratu problem equations file

1. Formulate as first order system in **func** (Jacobian unnecessary)
2. Initialize continuation in **stpnt**
3. Continuation parameters and solutions can be initialized here
4. Can also start from solutions given in column-formatted files

```
5
6   subroutine func(ndim,u,icp,par,ijac,f,dfdu,dfdp) ! give the system
7
8       implicit none
9       integer, intent(in) :: ndim, ijac, icp(*) ! constants for size and type of problem
10      double precision, intent(in) :: u(ndim), par(*) ! solution and parameter vectors
11      double precision, intent(out) :: f(ndim) ! function
12      double precision, intent(inout) :: dfdu(ndim,*), dfdp(ndim,*) ! jacobian and parameter derivative(s)
13
14      double precision lambda
15      ! not necessary, but elucidates that the first parameter is lambda
16      lambda = par(1);
17
18  !   u=(u,u')
19  !   Define the function components
20      f(1) = u(2)
21      f(2) = -lambda*EXP(u(1))
22
23  end subroutine func
24
25  !----------------------------------------------------------------------
26
27  subroutine stpnt(ndim,u,par,t) ! starting point for the continuation
28
29      implicit none
30      integer, intent(in) :: ndim
31      double precision, intent(inout) :: u(ndim), par(*)
32      double precision, intent(in) :: t
33
34
35  !   initialize the continuation parameter
36      par(1)  = 0.0           ! lambda
37
38  !   Initialize the solution
39      u(1)=0.0
40      u(2)=0.0
41  !   Can initialize the solution as a function of t
42
43
44  end subroutine stpnt
45
```

Figure: A portion of bratu.f90

Bratu problem equations file - 2

1. **bcnd** contains BCs ($u0$ at left and $u1$ at right)
2. **icnd** and **fopt** unused
3. **pvls** allows for tracking of solution measures as parameters

```
47
48  subroutine bcnd(ndim,par,icp,nbc,u0,ul,fb,ijac,dbc) ! boundary conditions
49
50      implicit none
51      integer, intent(in) :: ndim, icp(*), nbc, ijac ! constants for size and type of problem
52      double precision, intent(in) :: par(*), u0(ndim), ul(ndim) ! parameters and left/right conditions
53      double precision, intent(out) :: fb(nbc)
54      double precision, intent(inout) :: dbc(nbc,*)
55
56  !   dirichlet bcs at left and right
57      fb(1) = u0(1)
58      fb(2) = ul(1)
59
60  end subroutine bcnd
61
62   !---------------------------------------------------------------------
63
64  subroutine icnd ! integral conditions (unused)
65  end subroutine icnd
66
67   !---------------------------------------------------------------------
68
69  subroutine pvls(ndim,u,par) ! track solution measures as additional parameters
70      implicit none
71      integer, intent(in) :: ndim
72      double precision, intent(in) :: u(ndim)
73      double precision, intent(inout) :: par(*)
74
75      double precision, external :: GETP
76
77      ! getp is an AUTO function that can get a variety of measures of solution components
78      par(2)=GETP('NRM',1,u) ! L2 norm of u
79      ! PAR(2)=GETP('MAX',1,u) ! max value of u
80
81  end subroutine pvls
82
83  subroutine fopt ! optimization (unused)
84  end subroutine fopt
```

Figure: A portion of bratu.f90

Table of Contents

Constants file

- Called c.x (i.e. c.bratu)
- Can have multiple in same directory (c.bratu, c.bratu.1, c.bratu.2,...)
- Gives some constants that should be changed and some that shouldn't
- Two main formats:

```
1  parnames = {1:'lambda',2:'L2 norm of u'}
2  unames = {1:'u',2:'uprime'}
3  2 4 0 1                          NDIM,IPS,IRS,ILP
4  2 1 2                            NICP,(ICP(I),I=1 NICP)
5  100 4 3 2 1 0 2 0                NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
6  100 -1.e+8 1.e+8 -1.e+8 1.e+8    NMX,RL0,RL1,A0,A1
7  100 10 3 8 5 3 0                 NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
8  1.e-7 1.e-7 1.e-5                EPSL,EPSU,EPSS
9  0.05 0.001 1.0 1                 DS,DSMIN,DSMAX,IADS
10 0                                NTHL,(/,I,THL(I)),I=1,NTHL)
11 0                                NTHU,(/,I,THU(I)),I=1,NTHU)
12 0                                NUZR,(/,I,PAR(I)),I=1,NUZR)
```

Figure: c.bratu

```
1  parnames = {1:'lambda',2:'L2 norm of u'}
2  unames = {1:'u',2:'uprime'}
3  NDIM=   2, IPS =   4, IRS =   0, ILP =   1
4  ICP =  [1, 2]
5  NTST=   100, NCOL=   4, IAD =   3, ISP =   2, ISW = 1, IPLT= 0, NBC= 2, NINT= 0
6  NMX=  50, NPR= 50, MXBF=  10, IID =   3, ITMX= 8, ITNW= 5, NWTN= 3, JAC= 0
7  EPSL= 1e-07, EPSU = 1e-07, EPSS =1e-05
8  DS  =  0.05, DSMIN= 0.001, DSMAX= 1.0, IADS=   1
9  NPAR = 2, THL =  {}, THU =  {}
10 UZR =  {1: 2.0}
```

Figure: c.bratu.1

Bratu problem constants file - 1

"Problem" Constants

- Problem type (IPS)
- Number of dimensions (NDIM)
- Numbers of conditions (NBC, NINT)
- Number of parameters (NPAR/NICP)
- Bifurcations to look for (ILP, ISP)
- Whether Jacobian given (JAC)

```
1   parnames = {1:'lambda',2:'L2 norm of u'}
2   unames = {1:'u',2:'uprime'}
3   2 4 0 1                              NDIM,IPS,IRS,ILP
4   2 1 2                                NICP,(ICP(I),I=1 NICP)
5   100 4 3 2 1 0 2 0                    NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
6   100 -1.e+8 1.e+8 -1.e+8 1.e+8        NMX,RL0,RL1,A0,A1
7   100 10 3 8 5 3 0                     NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
8   1.e-7 1.e-7 1.e-5                    EPSL,EPSU,EPSS
9   0.05 0.001 1.0 1                     DS,DSMIN,DSMAX,IADS
10  0                                    NTHL,(/,I,THL(I)),I=1,NTHL)
11  0                                    NTHU,(/,I,THU(I)),I=1,NTHU)
12  0                                    NUZR,(/,I,PAR(I)),I=1,NUZR)
```

Bratu problem constants file - 2

"Numerical" Constants

- Tolerances (blue)
- Solver specifications (red)
- Solution mesh info (NTST, NCOL, IAD)

```
parnames = {1:'lambda',2:'L2 norm of u'}
unames = {1:'u',2:'uprime'}
2 4 0 1                                 NDIM,IPS,IRS,ILP
2 1 2                                   NICP,(ICP(I),I=1 NICP)
100 4 3 2 1 0 2 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
100 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
0.05 0.001 1.0 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)
```

- Newton solver information (ITMX, ITNW, NWTN)
- Max ITMX iterations used to converge to special points
- Max ITNW Newton iterations used to converge to regular points
- Performs full Newton for NWTN iterations (chord after)

Bratu problem constants file - 3

"Navigation" Constants

- Cont. step size (green)
- Initizialization and diagnostic info (pink)
- Cont. steps (NMX)
- Branch switching (ISW)
- Continuation parameter (ICP)
- Output formatting (parnames, unames)
- Solution saving (NPR, UZR)
- Stopping criteria - not included (SP, STOP, UZSTOP)

```
1  parnames = {1:'lambda',2:'L2 norm of u'}
2  unames = {1:'u',2:'uprime'}
3  2 4 0 1                          NDIM,IPS,IRS,ILP
4  2 1 2                            NICP,(ICP(I),I=1 NICP)
5  100 4 3 2 1 0 2 0                NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
6  100 -1.e+8 1.e+8 -1.e+8 1.e+8    NMX,RL0,RL1,A0,A1
7  100 10 3 8 5 3 0                 NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
8  1.e-7 1.e-7 1.e-5                EPSL,EPSU,EPSS
9  0.05 0.001 1.0 1                 DS,DSMIN,DSMAX,IADS
10 0                                NTHL,(/,I,THL(I)),I=1,NTHL)
11 0                                NTHU,(/,I,THU(I)),I=1,NTHU)
12 0                                NUZR,(/,I,PAR(I)),I=1,NUZR)
```

Table of Contents

Scripting capabilities overview

- AUTO can be run from command line with commands of the form "@x" (complete list of commands in the manual)
- AUTO can be run from Python scripts - "x.auto" (preferred by me)
- Scripts can load data (equations, constants), perform continuation runs, manage continuation data, plot data, and save data
- Constants can be changed in scripts (used in laser example especially)
- Scripts run with "auto x.auto"
- Some expert scripts available for data management (i.e. to_matlab.xauto)
- CLUI commands sometimes useful for data management as well

Bratu script

- Loads Bratu equations and constants
- Performs a continuation run
- Plots results in GUI
- Can use usual Python commands (i.e. track execution time)

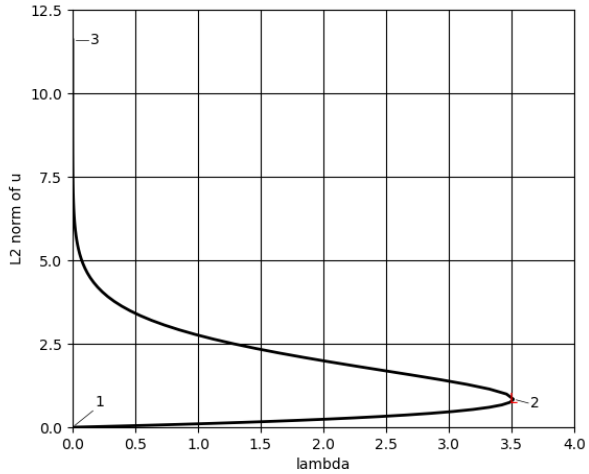
```
1  from datetime import datetime
2  startTime=datetime.now()
3  ## Begin AUTO commands
4  mycont=load(e='bratu')
5  mycont=load(mycont,c='bratu')
6  mycont=run(mycont)
7  ######### print execution time ##################
8  print "\n ***This run took ", datetime.now() - startTime ,"***"
9  ################################################
10 myplot=plot(mycont)
11 myplot.config(bifurcation_x=(0,),bifurcation_y=(4,))
12 wait()
```

Figure: bratu.auto

Table of Contents

Plotting using PyPLAUT

- Allows for plotting of bifurcation diagrams and solutions in 2D and 3D with GUI
- Special and requested solutions labeled in bif. diagram
- Special and requested solutions plottable
- Plotting GUI closes at end of script so should follow plotting commands with "wait()"
- Plots can be modified in scripts and in the GUI
- Save plots from the GUI



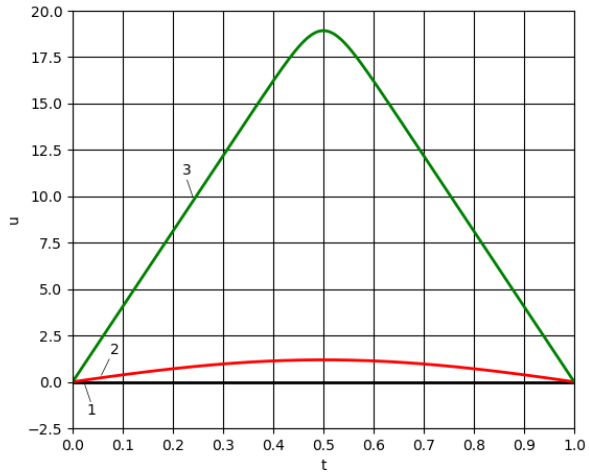


Table of Contents

AUTO output overview

- AUTO produces 3 files with each continuation run fort.7, fort.8, fort.9
- These files overwritten if not saved
- Only last continuation run shown in fort.x files after script execution
- fort.7 shows bifurcation diagram information (saved into b.x file types)
- fort.8 shows solution information from special and requested solutions (saved into s.x file types)
- fort.9 is diagnostic info from the solver (not typically saved)

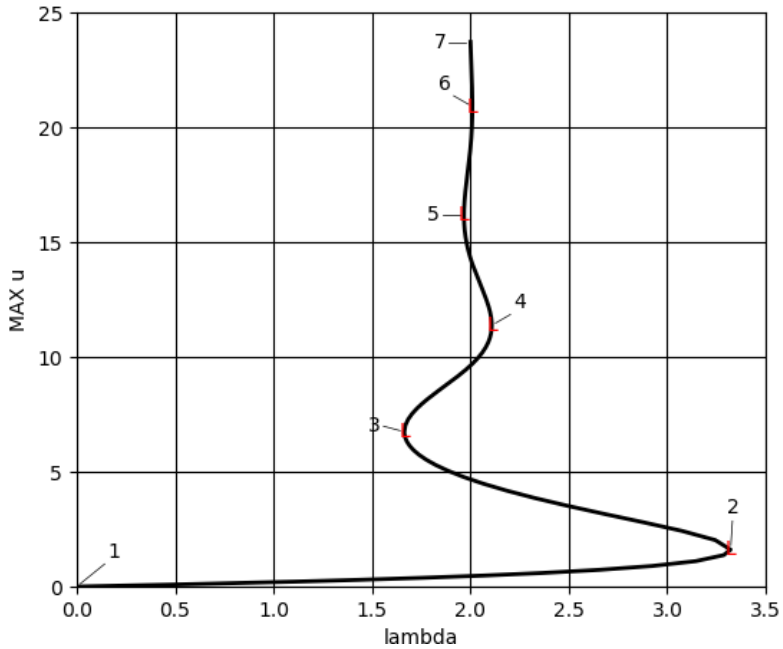
Bratu fort.7 output

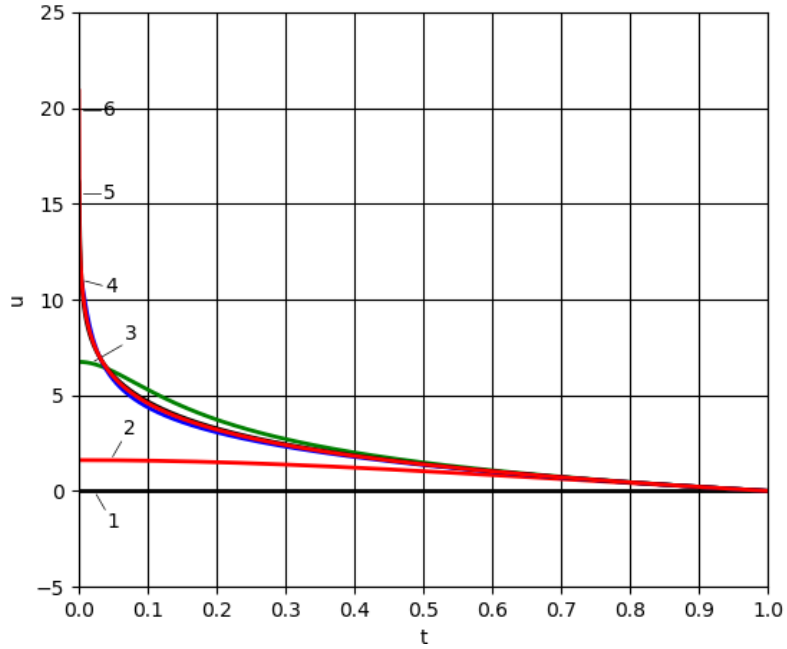
```
 1   0  -1.0000E+08  1.0000E+08 -1.0000E+08  1.0000E+08
 2   0   EPSL= 1.0000E-07  EPSU = 1.0000E-07  EPSS = 1.0000E-05
 3   0   DS  = 2.0000E-01  DSMIN= 1.0000E-03  DSMAX= 1.0000E+00
 4   0   NDIM=   2   IPS =   4   IRS =   0   ILP =   1
 5   0   NTST= 100   NCOL=   4   IAD =   3   ISP =   2
 6   0   ISW =   1   IPLT=   0   NBC =   2   NINT=   0
 7   0   NMX=  100   NPR=  100   MXBF=  10   IID =   3   IADS=   1
 8   0   ITMX=   8   ITNW=   5   NWTN=   3   JAC =   0   NUZR=   0
 9   0   NPAR=  36   THL = {}    THU = {}
10   0   e = 'bratu' s = '/'
11   0   parnames = {1: 'lambda', 2: 'L2 norm of u'}
12   0   unames   = {1: 'u', 2: 'uprime'}
13   0   User-specified parameters:         1   2
14   0   Active continuation parameter:   'lambda'
15   0
16   0    PT  TY  LAB     lambda             L2-NORM             MAX u              MAX uprime          L2 norm of u
17   1     1   9    1   0.0000000000E+00   0.0000000000E+00   0.0000000000E+00   0.0000000000E+00   0.0000000000E+00
18   1     2   0    0   1.9110258157E-01   5.9000152533E-02   2.4378109320E-02   9.7118168013E-02   1.7792929912E-02
19   1     3   0    0   4.7645716820E-01   1.5169422409E-01   6.2755329732E-02   2.4842851472E-01   4.5761700064E-02
20   1     4   0    0   7.5995369186E-01   2.5000342099E-01   1.0355895707E-01   4.0723428657E-01   7.5443415241E-02
21   1     5   0    0   1.0411699546E+00   3.5470738604E-01   1.4713142443E-01   5.7451335468E-01   1.0707588187E-01
22   1     6   0    0   1.3195659757E+00   4.6676068033E-01   1.9389135891E-01   7.5145419893E-01   1.4095021796E-01
23   1     7   0    0   1.5944166137E+00   5.8733745263E-01   2.4435634245E-01   9.3950360145E-01   1.7742447536E-01
24   1     8   0    0   1.8647279797E+00   7.1789468769E-01   2.9916685354E-01   1.1404346457E+00   2.1694290686E-01
25   1     9   0    0   2.1291080181E+00   8.6025175265E-01   3.5912862325E-01   1.3564273630E+00   2.6006019034E-01
26   1    10   0    0   2.3855640271E+00   1.0166817880E+00   4.2524768576E-01   1.5901506766E+00   3.0746924380E-01
27   1    11   0    0   2.6311875712E+00   1.1899910063E+00   4.9878386196E-01   1.8448058551E+00   3.6002531844E-01
28   1    12   0    0   2.8616846147E+00   1.3835084344E+00   5.8123353079E-01   2.1240196974E+00   4.1874272935E-01
29   1    13   0    0   3.0707668337E+00   1.6007829759E+00   6.7422333108E-01   2.4313191190E+00   4.8470226528E-01
30   1    14   0    0   3.2496892527E+00   1.8445753517E+00   7.7906591853E-01   2.7686907091E+00   5.5874351583E-01
31   1    15   0    0   3.3879063018E+00   2.1147285266E+00   8.9584885896E-01   3.1338518417E+00   6.4081521434E-01
32   1    16   0    0   3.4763420618E+00   2.4057501976E+00   1.0223392878E+00   3.5176172263E+00   7.2923785165E-01
33   1    17   0    0   3.5124563416E+00   2.7074192490E+00   1.1541676537E+00   3.9056097969E+00   8.2088830466E-01
34   1    18   5    2   3.5138307191E+00   2.7819433962E+00   1.1868424623E+00   4.0000027829E+00   8.4352622342E-01
35   1    19   0    0   3.4696267166E+00   3.2302144452E+00   1.3842475581E+00   4.5564280976E+00   9.7964652320E-01
36   1    20   0    0   3.3569070279E+00   3.6692852153E+00   1.5789599596E+00   5.0840552318E+00   1.1128536573E+00
```

Table of Contents

Bratu's problem with spherical symmetry

$$u'' + \frac{2}{r} + \lambda e^u = 0$$
$$u'(0) = u(1) = 0$$





Bratu's problem with spherical symmetry - implementation

Write the problem as an equivalent first-order autonomous system and initialize the system appropriately:

$$
\begin{aligned}
u_1' &= u_2 \\
u_2' &= -\frac{2}{u_3} - \lambda e^{u_1} \\
u_3' &= 1 \\
u_1(1) &= 0 \\
u_2(0) &= 0 \\
u_3(0) &= 0
\end{aligned}
$$

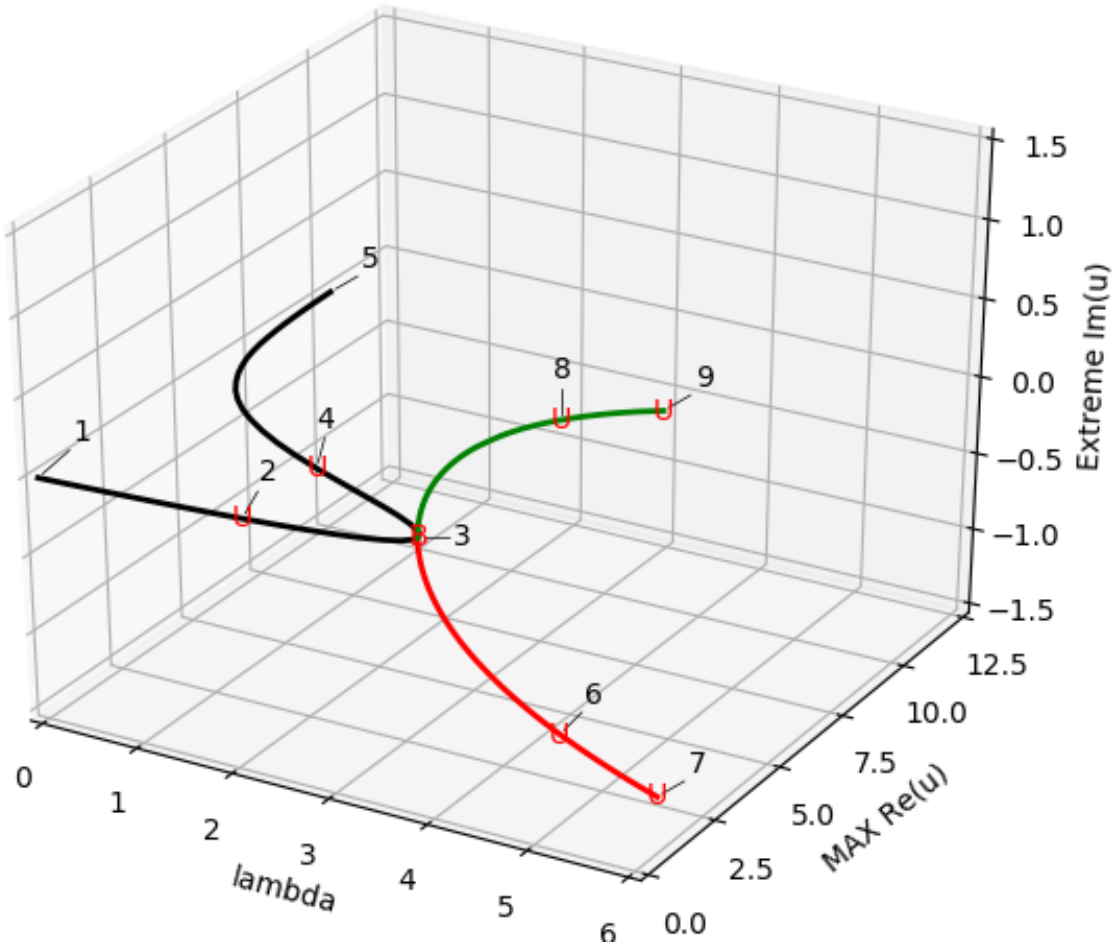
```
25  !-----------------------------------------------------------------------
26
27  subroutine stpnt(ndim,u,par,t)
28
29      implicit none
30      integer, intent(in) :: ndim
31      double precision, intent(inout) :: u(ndim), par(*)
32      double precision, intent(in) :: t
33
34  !   initialize the continuation parameter
35      par(1) = 0.0            ! lambda
36
37  !   Initialize the solution
38      u(1)=0.0
39      u(2)=0.0
40  !   The third component is the independent variable
41      u(3)=t
42
43
44  end subroutine stpnt
45
46  !-----------------------------------------------------------------------
```

Figure: A portion of rbratu.f90

Table of Contents

Bratu's problem with a complex dependent variable

$$u'' + \lambda e^u = 0, \quad u \in \mathbb{C}, \lambda \in \mathbb{R}$$
$$u'(0) = u(1) = 0$$



Bratu's problem with a complex dependent variable - scripting

See AUTO demo **ezp**. Multiple continuation runs performed, switching branches and direction:

```
1
2   #==============
3   # AUTO Demo ezp with modification by Nathan Sanford
4   #==============
5
6   print "\n***Compute a solution family***"
7   ezp=run(e='ezp',c='ezp')
8
9   print "\n***Compute one leg of the bifurcating family***"
10  ezp=ezp+run(ezp('BP1'),ISW=-1)
11
12  print "\n***Compute the other leg of the bifurcating family***"
13  ezp=ezp+run(ezp('BP1'),ISW=-1,DS='-')
14  # save(ezp,'ezp')
15
16  print "\n***Plot the results***"
17  ezplot=plot(ezp)
18  ezplot.config(bifurcation_x=(0,),bifurcation_y=(2,),bifurcation_z=(6,))
19  wait()
20
21
```

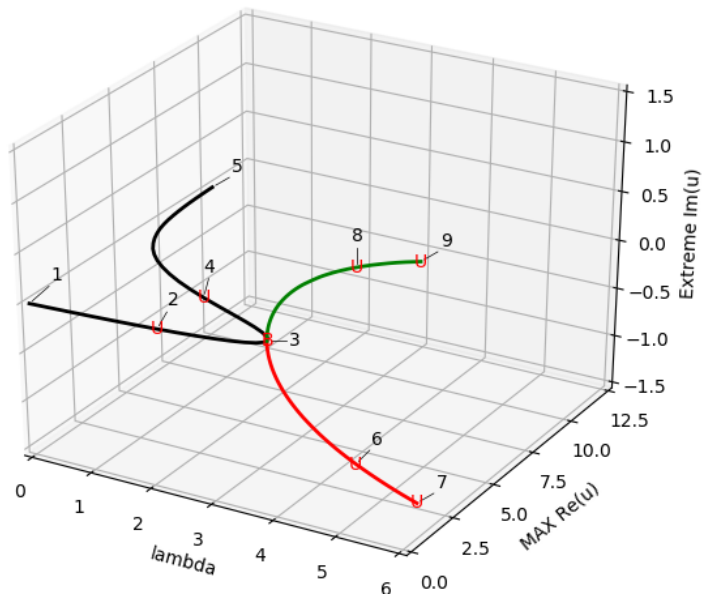


Table of Contents

Laser ODEs

This multipoint bvp describes likeliest error paths in a laser model (parameters in red):

$$\frac{d\eta_\Omega}{dz} = -A\eta_\Omega - \frac{\beta D}{C}\eta_T \qquad \frac{d\eta_T}{dz} = -\frac{B\omega C}{D}\eta_\Omega \cos(\omega T)$$

$$\frac{d\Omega}{dz} = A\Omega + B\sin(\omega T) + \eta_\Omega \qquad \frac{dT}{dz} = \beta\Omega + \eta_T$$

with BCs $T(0) = T_0$, $\Omega(0) = \Omega_0$, $T(z_L) = \hat{T}$, and $\eta_\Omega(z_L) = 0$.

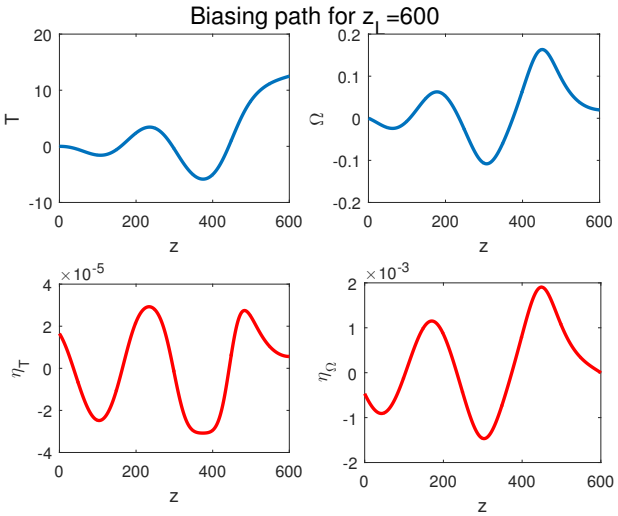


Figure: Solution with $T_0 = 0$, $\Omega_0 = 0$, and $z_L = 600$.

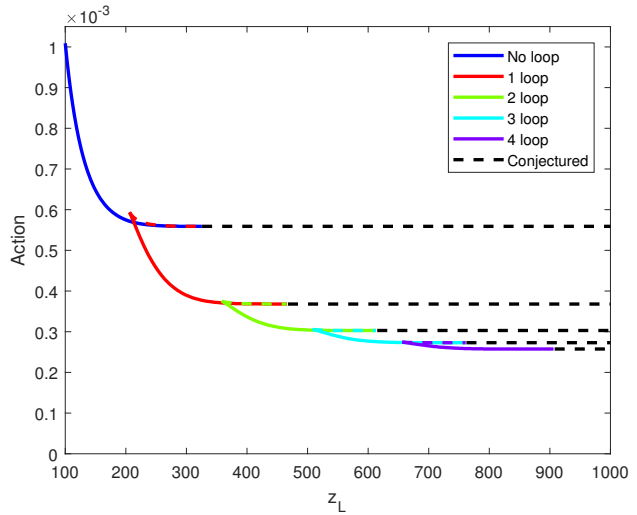


Figure: Bifurcation diagram varying z_L with $T_0 = 0$, $\Omega_0 = 0$.

AUTO usage

- Compute solutions on Cartesian grid in (T_0, Ω_0, z_L) parameter space
- Folds occur - discount solutions above (past) fold points for physical reasons
- Use 1 parameter continuation, switching parameters, to sweep the grid
- Many continuation runs performed, constants modified in "run" commands in script

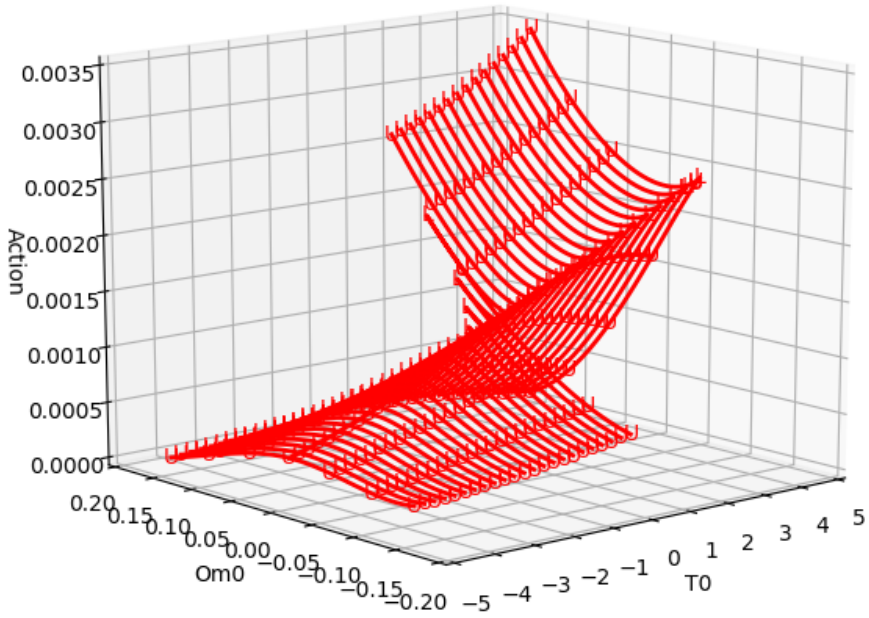


Figure: Curves traced by performing (T_0, Ω_0) continuation with $z_L = 210$.

AUTO implementation

- Modify ICP, changing first parameter to change continuation direction
- Use UZSTOP to stop runs at specific points
- Use UZR to save solutions at specific points
- Used "SP=[LP1]" to stop runs at first fold encounter (used in full code, but not here)

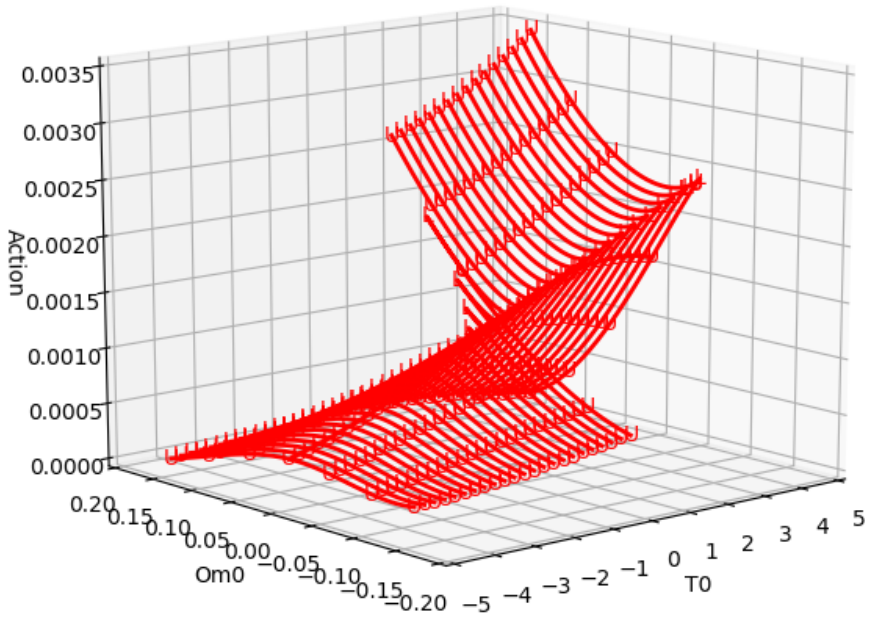


Figure: Curves traced by performing (T_0, Ω_0) continuation with $z_L = 210$.

Thanks! Any Questions?