SymOrb

Symmetric Periodic Action-Minimizing Orbits in GAP
(Preliminary version)

by

Davide L. Ferrario
Dipartimento di Matematica e Applicazioni – Università di Milano–Bicocca

Contents

Introduction

The purpose of the package is to find numerically local minima of the Lagrangian action restricted to the space of symmetric loops, according to the results of [FT2003] and [zz]. In order to obtain a good approximation of the symmetric orbits, two steps are necessary: first, given an integer $n \geq 3$ and a dimension $d \geq 2$ one has to list all possible (finite) symmetry groups for the n-body problem in $\mathbb{R}^d$. Then, it is necessary communicate the data about the symmetry group to a custom optimization package, that interactively parses the input and output the supposed minimizer in data format. At the end, some post-processing is done in order to visualize an animation of the orbit found and compute some of its invariants (energy, angular momentum, and so on).

This package thus consist of two piaces: first a GAP package that interactively allows to classify and define symmetry groups; then, an optimization package performs numerical optimization, via a custom user interface, allowing interactive manipulation and visualization of the orbit, which planar or spatial.

At the end of this short manual we will show a sample interactive session and list some of the most used commands.

Milano, July 2005

Davide L. Ferrario

1 Symorb Manual

The main object of the package is the Lagrange symmetry group (in short LGS). In the first section we show the usage of the main commands, then some helpful utilities, and then the methods, properties and attributes of LGS's.

1.1 Main commands

1 ▶ `LagSymmetryGroup(`*action_type*`,`*NOB*`, `*kern*`, `*rotV*`, `*rotS*`, `*refV*`, `*refS*`)`

The arguments are: *action_type* is the action type (restricted: 0 if cyclic and anything else otherwise), *NOB* is the number of bodies, *kern* is the kernel of τ, *rotV* is the matrix of the cyclic generator acting on the Euclidean space, *rotS* is the permutation corresponding to the cyclic generator, *refV* is the reflection in the space and *refS* is the reflection in the index set (if the action type is cyclic they are ignored).

2 ▶ `TrivialKerTau( `*dim*` )`

Returns a trivial kernel of τ for a space of dimension *dim*; it can be used with `LagSymmetryGroup`.

3 ▶ `MinorbInitString(`*LSG*`)`

Build the init string. *LSG* is a symmetry group, as defined above. The init string is the parseable string necessary to pipe to the minimization engine.

4 ▶ `MinorbInfoString(`*LSG*`)`

Build the info string. *LSG* is a symmetry group, as defined above. The info string is a human-readable string that describes the LSG and some of its properties.

5 ▶ `MakeMinorbSymFile(`*basefilename*`,`*LSG*`)`

Create *basefilename*`.init` and *basefilename*`.info`, where it is stored the init symfile and the info file, produced by 1.1.3 and 1.1.4.

6 ▶ `AllLSGTK (`*action_type*`,`*NOB*`,`*dim*`)`

It returns a list of all LGS's with trivial kernel of τ, action type equal to `action_type`, `NOB` bodies and dimension `dim`.

1.2 Utilities, properties and attributes

1 ▶ `RotationMatrixDim2(`*order*`)`

A rotation `2x2` matrix of order *order*.

2 ▶ `RotationMatrixDim3(`*order*`)`

A rotation `3x3` matrix of order *order*. It fixed the third coordinate.

3 ▶ `IsTypeRDirection` (*LSG*, *dir*)

It retuns true or false according to whether the direction *dir* (in 1, 2, or 3) is a direction under which the symmetry group is of type R.

4 ▶ `IsTypeR` (*LSG*)

True if *LSG* is of type R.

5 ▶ `TransitiveDecomposition(` *LSG* `)`

It yields the transitive decomposition of the action of the group on the index set.

6 ▶ `IsTransitiveLSG(` *LSG* `)`

True if the group is transitive on the index set.

7 ▶ `IsValidLSG(` *LSG* `)`

It checks whether the *LSG* is well-defined or not.

8 ▶ `IsCoercive(` *LSG* `)`

True if *LSG* is coercive.

9 ▶ `HasAlwaysCollisions(` *LSG* `)`

If true, then *LSG* has always collisions. If false, mostly it has not. (To be improved).

10 ▶ `ActionType(` *LSG* `)`

It returns 0 if the action type is cyclic, 1 if it is brake and 2 if it is dihedral.

11 ▶ `IsRedundant(` *LSG* `)`

True if *LSG* is redundant.

12 ▶ `GroupOrder(` *LSG* `)`

The order of the group.

13 ▶ `KernelTauOrder(` *LSG* `)`

The order of the kernel of τ.

1.3 An Example Session

```
gap> RequirePackage("symorb");
true
gap> NOB:=3;dim:=2;
3
2
gap> rotV:=[[-1,0],[0,1]];
[ [ -1, 0 ], [ 0, 1 ] ]
gap> rotS:=(1,2,3);
(1,2,3)
gap> refV:=[[-1,0],[0,-1]];
[ [ -1, 0 ], [ 0, -1 ] ]
gap> refS:=(1,2);
(1,2)
gap> LSG:= LagSymmetryGroup(1,NOB, TrivialKerTau(2), rotV, rotS, refV, refS);
LagSymmetryGroup(NOB=3, dim=2, action_type=2, rotation=Tuple( [
```

```
[ [ -1, 0 ], [ 0, 1 ] ], (1,2,3) ] ), reflection=Tuple( [
[ [ -1, 0 ], [ 0, -1 ] ], (1,2) ] )
gap> s:=MinorbInfoString(LSG);;
gap> Print(s);
% SYMORB version : 4r2 fix8-0.9
% date        : Fri Mar 26 13:53:48 CET 2004
% on          : Linux i586 unknown
@ GroupOrder: 12
@ KernelTauOrder: 1
@ ActionType: 2
@ IsTypeR: false
@ IsCoercive: true
@ IsRedundant: false
@ HasAlwaysCollisions: false
@ TransitiveDecomposition: [ [ 1, 2, 3 ] ]
@ TypeRDirections: [  ]
x_1(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_2(t)
x_2(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_3(t)
x_3(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_1(t)
x_1(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_2(t)
x_2(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_1(t)
x_3(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_3(t)
gap> MakeMinorbSymFile("/tmp/eight",LSG);
file /tmp/eight.sym created!
file /tmp/eight.info created!
0
gap> IsTransitiveLSG(LSG);
true
gap> IsCoercive(LSG);
true
gap> HasAlwaysCollisions(LSG);
false
```

1.4 Example of generating file

```
RequirePackage("symorb");
NOB:=12;
dim:=3;
mat1:=[[-1,0,0],[0,-1,0],[0,0,1]];
mat2:=[[0,1,0],[0,0,1],[1,0,0]];
G:=GroupWithGenerators([mat1,mat2]);
hom:=ActionHomomorphism(G,G,OnRight);
s1:=Image(hom,mat1);
s2:=Image(hom,mat2);
kert:=GroupWithGenerators([ Tuple([mat1,s1]), Tuple([mat2,s2]) ] );
rotV:=[[-1,0,0],[0,-1,0],[0,0,-1]];
rotS:=();

LSG:=LagSymmetryGroup(0,NOB,kert, rotV,rotS,rotV,rotS);
MakeMinorbSymFile("try",LSG);
```

1.5 Generating LSGs

1 ► `LagSymmetryGroupCHARS(`*NOB*`,` *Tchar*`,` *Vchar*`,` *sigma*`)` Constructor for the new object with rec...

2 ► `MakeLSGfromCHARS(`*stru*`)` ;Where *stru* is build with —LagSymmetryGroup—

3 ► `MakeActions(`*maxorbs*`,`*group*`,`*n*`,`*dim*`)`

2 MinPath: an interactive optimization program

Now we assume that the files `/tmp/eight.sym` and `/tmp/eight.info` exist. We can interactively perform and visualize the minimization process as follows.

2.1 Main commands

1 ▶ `x=minpath(`*symfile*`)`

Define a MinPath object with symmetry group stored in the file *symfile*. If no *symfile* is present, a list of symfiles in the current directory is given. Otherwise, there are the following pre-built objects: `eight_c6`, 'eight_d6', 'eight_d12', 'trivial', 'line', 'isosceles', 'hill', 'choreography', 'lagrange', 'choreography_21' (from the list in [zz]).

2 ▶ `x.info()`

Show info file of the path object `x`.

3 ▶ `x.new(*SPMETHOD=`*SPMETHOD*`)`

Give coefficients to `x`. If SPMETHOD=*SPMETHOD* is present, use StartingPath Method *SPMETHOD*. For example, SPMETHOD=1 assigns random coefficients to the path.

4 ▶ `x.relax()`

Optimization subroutines. It gives the list of all available algorithms with a short description and their codes, as follows:

```
0:       Unconstrained Minimization with analytic Gradient
1:       Unconstrained Minimization with Analytic Hessian
2:       Unconstrained Minimization with finite-Difference
Hessian
3:       Unconstrained Minimization with Conjugate
Gradient and analytic Gradient
4:       Unconstrained Minimization without gradient
(nonsmooth)
5:       Linearly Constrained Minimization with Analytic
Gradient
6:       Step-Flow Descent
7:       Simple Conjugate Gradient
-----------------------------------------------------------------------
100:     Newton-Powell Finite-difference Jacobian
200:     Newton-Powell Analytic Jacobian
300:     Secant Broyden's Update and Finite-difference
Jacobian
400:     Secant Broyden's Update and Analytic Jacobian
```

5 ▶ `x.relax(`*relax_code* `+` *newton_code*`)`

Optimization: perform an optimization algorithm number *relax_code* and subsequently a newton root-finding with algorithm *newton_code*.

6 ▶ `x.newton(`*newton_code*`)`

Optimization: perform a newton root-finding with algorithm *newton_code*.

7 ▶ `x.view()`

View the path (GEOMVIEW has to be installed).

Set the rotation vector ω.

8 ▶ `x.reshape(`*STEPS*`)`

Reshape the coefficient matrix of `x`.

9 ▶ `x.load(`*filename*`)`

Load coefficients for `x` from *filename*.

10 ▶ `x.write(`*filename*`)`

Write coefficients for `x` to *filename*.

11 ▶ `x.printsol(`*filename*`)`

Write to the file *filename* the positions of the *NOB* bodies in time, in a gnuplot-like format. It is the format that can be visualized by `orbview`.

12 ▶ `x.dump()`

For debug only. Write to stdout everything.

13 ▶ `x.action()`

Compute the action of the loop `x`.

14 ▶ `x.howsol()`

Compute the norm of the gradient of the action of `x`.

15 ▶ `x.withcoll`

Set this variable to 1 if we want to use minorb avoiding collisions, or 0 otherwise.

2.2 A database of planar symmetry groups

▶ `all_minpaths(dim=`*dim*`,NOB=`*NOB*`,GroupOrder=`*group_order*`,KernelTauOrder=`*kernel_tau_order*`,ActionType=`*action_type*

All options are not necessary. It gives back the list of all minpath objects with the desired properties (all of them have to be fulfilled).

```
MinorbShell > len(all_minpaths(NOB=3,dim=2,IsCoercive=true,\
IsRedundant=false,HasAlwaysCollisions=false))
6
MinorbShell
```

2.3 An example session

First, it is necessary to run the interpreter. It is a subshell of `python`, so that all python syntax and modules are accessible insider `minpath`.

```
[unix_shell] $ minpath
minpath -- beginning at Fri Mar 26 14:37:51 CET 2004

symfiles:


MinorbShell >
```

Now we can use the interactive shell.

```
MinorbShell > x=lagrange
MinorbShell > x.new()
 starting new path...
MinorbShell > x.relax(203)
 relaxing...
 # using IMSL DUMCGG
 # Unconstrained Minimization with Conjugate Gradient and
analytic Gradient
 # using NONLINEAR DNEQNJ
 # Newton-Powell Analytic Jacobian
 ==> action:   4.7124; howsol: 1.5637e-15
MinorbShell > x.view()
MinorbShell > x.reshape(50)
<minpath object; NOB=3, dim=2, steps=50>
MinorbShell > x.newton(400)
 computing newton path...
 # using NONLINEAR DNEQBJ
 # Secant Broyden's Update and Analytic Jacobian
 ==> action:   4.7124; howsol: 1.5688e-15
MinorbShell > x.write('/tmp/lag1.myg')
```

In the following example we consider the symmetry file that we have built in the previous chapter, and compute its minimizer with finite-difference Hessian and Newton-Powell with analytic Jacobian.

```
MinorbShell > ei=minpath('/tmp/eight.sym')
MinorbShell > ei.info()
info on /tmp/eight:
----------------------------------------------------------------
@ GroupOrder: 12
@ KernelTauOrder: 1
@ ActionType: 2
@ IsTypeR: false
@ IsCoercive: true
@ IsRedundant: false
@ HasAlwaysCollisions: false
@ TransitiveDecomposition: [ [ 1, 2, 3 ] ]
@ TypeRDirections: [  ]
x_1(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_2(t)
x_2(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_3(t)
```

```
x_3(t+T/6) = [ [ -1.0, 0.0 ], [ 0.0, 1.0 ] ] * x_1(t)
x_1(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_2(t)
x_2(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_1(t)
x_3(-t) = [ [ -1.0, 0.0 ], [ 0.0, -1.0 ] ] * x_3(t)
----------------------------------------------------------------

MinorbShell > ei.new()
 starting new path...
MinorbShell > ei.relax(202)
 relaxing...
 # using IMSL DUMIDH
 # Unconstrained Minimization with finite-Difference
Hessian
 # using NONLINEAR DNEQNJ
 # Newton-Powell Analytic Jacobian
 ==> action:   3.6906; howsol: 9.3383e-16
MinorbShell > ei.view()
MinorbShell > ei.write('/tmp/eight.myg')
```

2.4 Parallel cluster

Again, make password-less access on a grid (SGE) available. Export the variable `REMOTE_MINORB=hostname`, then `minpath` will (probably) be able to connect remotely. Then the nice function `remjob` will be available:

```
MinorbShell > res=remjob(x,200,"new();relax(2);newton(300)" )
```

for example (the syntax is obvious).

The environment variable `SYMORBDIR` has to be set on the nodes to locate the proper directory.

```
. /share/SGE6.1/default/common/settings.sh
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/lf9562/lib
export PATH=${PATH}:${HOME}/local/bin
```

Other useful pieces of code:

```
rr=filter(lambda y: y.howsol() < 0.001, res)
rr2=[x.newton(200) for x in rr]
```

Bibliography

[BFT07] Vivina Barutello, Davide L. Ferrario, and Susanna Terracini. Symmetry groups of the planar 3-body problem and action–minimizing trajectories. *Arch. Rational Mech. Anal.*, 2007. to appear.

[FT04] Davide L. Ferrario and Susanna Terracini. On the existence of collisionless equivariant minimizers for the classical n-body problem. *Invent. Math.*, 155(2):305–362, 2004.