Appendix G: Relative Rotation Rate Package

The following *Mathematica* package calculates relative rotation rates and intertwining matrices by the methods described in section 5.5.4.

```
(*
   Relative Rotation Rates --- Mathematica Package
   Date: 10/04/90
   Last Modified:

Authors: Copyright 1990
A. Lorentz, N. Tufillaro and P. Melvin.
Departments of Mathematics and Physics
Bryn Mawr College, Bryn Mawr, PA 19010-2899 USA

Bugs:
   Many of these symbolic computations are exponential time algorithms,
so they are slow for long periodic orbits and templates with many
branches. A "C" version of these routines exists which runs considerably
faster. The algorithm for the calculation of the relative
rotation rate from a single word pair, however, is polynomial time.

About the Package:
This collection of routines automates the process for the symbolic
calculation of relative rotation rates, and the intertwining matrix
for an arbitrary template. The template is represented algebraically
by a framed braid matrix, which is specified by the global variable
"bm" in these routines.  This variable must be defined by the user
when these routines are entered.
   There are three major routines:

      RelRotRate[word pair], AllRelRotRates[word pair], and
      Intertwine[start row, stop row].

The input for the RelRotRate programs is a word pair, which is just
a list of lists. For example, a valid input for these programs is

      { {0,1,0,1,1,0}, {1,1,0} }

where the first word of the word pair is "010110" and the second word
is "110". The input for the Intertwine program is just two integers,
"start row" and "stop row". For instance, Intertwine[2,3] would produce
all relative rotation rates for all words of length between 2 and 3.
   In addition, the routine
```



```
      GetFunCyc[branches, period]

generates all fundamental cycles of length "period" for a template with
"b" branches or a symbolic alphabet of "b" letters.  This routine can
be useful for the cycle expansion techniques (see, R. Artuso, E. Aurell,
and P. Cvitanovic, Recycling of strange sets I and II, Nonlinearity
Vol 3., Num. 2, May 1990, p. 326.).

References:
[1] G. B. Mindlin, X.-J. Hou, H. G. Solari, R. Gilmore, and N. B.
Tufillaro, Classification of strange attractors by integers, Phys. Rev.
Lett. 64 (20), 2350 (1990).
[2] N.B. Tufillaro, H.G. Solari, and R. Gilmore, Relative rotation rates:
 fingerprints for strange attractors, Phys. Rev. A 41 (10), 5717 (1990).
[3] H. G. Solari and R. Gilmore, Organization of periodic orbits in the
    driven Duffing oscillator, Phys. Rev. A 38 (3), 1566 (1988).
[4] H. G. Solari and R. Gilmore, Relative rotation rates for driven
    dynamical systems, Phys. Rev. A 37 (8), 3096 (1988).
*)

(*
   The template braid matrix is a global variable that should be
   defined by the user before this package is used. Comment out the
   default setting for the braid matrix.
   Examples for the braid matrix are presented below.

   Global Variable Abbreviation.
   bm braid matrix --- algebraic description of template
*)

(* Braid Matrix Example: The Horseshoe Template *)
(*
bm = {
   { 0, 0 },
   { 0, 1 }
};
*)

(* Braid Matrix Example: Second Iterate of The Horseshoe Template *)
bm = {
   { 0, 0, 0, 0},
   { 0, 1, 1, 1},
   { 0, 1, 2, 1},
```

```
   { 0, 1, 1, 1}
};

(* Calculate the relative rotation rate for a pair of words.
  Variables local to RelRotRate.
  wordp     wordpair
  lcm       lowest common multiple of word pair lengths
  ewordp    expanded word pair
  top       top list
  bottom    bottom list
  rrr       relative rotation rate of word pair
*)

RelRotRate[wordp_List] :=
Block[
   {
    lcm,
    ewordp, top, bottom,
    rrr,
 },

If[wordp[[1]] == wordp[[2]], Return[0]];     (* Self rotation rate *)

  ewordp = ExpandWordPair[wordp];
  top = GetTop[ewordp];
  bottom = GetBottom[ewordp];
  lcm = Length[top];

  rrr    = (SumAll[top] + SumAll[bottom])/(2 lcm);
  Return[rrr]
]

(* Permute the word pair list to generate
   all possible relative rotation rates *)

AllRelRotRates[wordp_List] :=
Block[
   {i, pa, pb, lcm, wordstep, rrr,
   firstword, secondword, rrrs},

   firstword = wordp[[1]];
   secondword = wordp[[2]];
   pa = Length[firstword];
   pb = Length[secondword];
```



```
   lcm = LCM[pa, pb];
   wordstep = (lcm*lcm)/(pa*pb);
   rrrs = {};
   For[i = 0, i < lcm, i += wordstep,
      rrr = RelRotRate[{firstword, secondword}];
      rrrs = Append[rrrs, {firstword, secondword, rrr}];
      firstword = RotateRight[firstword, wordstep];
   ];
   Return[rrrs];
]

(* Calculate the Intertwining Matrix of all orbits of length "startrow"
   to length "stoprow", startrow <= stoprow. *)

Intertwine[startrow_Integer, stoprow_Integer] :=
Block[
   {b, i, j, k, rowsize, rsum, colsize, csum, mult,
    cycls = {}, rcycls = {}, ccycls = {}, rrr = {}, rrrs = {}},

   b = Length[bm];

   rowsize = 0; rsum = 0;
   For[i = startrow, i <= stoprow, ++i,
      cycls = GetFunCyc[b, i]; rowsize = Length[cycls];
      rsum += rowsize;
      For[j = 1, j <= rowsize, ++j,
         rcycls = Append[rcycls, cycls[[j]]];
      ];
   ];
   colsize = 0; csum = 0;
   For[i = 1, i <= stoprow, ++i,
      cycls = GetFunCyc[b, i]; colsize = Length[cycls];
      csum += colsize;
      For[j = 1, j <= colsize, ++j,
         ccycls = Append[ccycls, cycls[[j]]];
      ];
   ];
   rrr = {}; rrrs = {};
   For[i = 1, i <= rsum, ++i,
      For[j = 1, j <= csum, ++j,
         rrr = AllRelRotRates[{rcycls[[i]], ccycls[[j]]}];
         mult = Length[rrr];
         rrrs = {rrr[[1,1]], rrr[[1,2]]};
         For[k = 1, k <= mult, ++k,
```

```
            rrrs = Append[rrrs, rrr[[k,3]]];
          ];
          Print[rrrs];
          If[rcycls[[i]] == ccycls[[j]], Break[]];
        ];
    ];
]

(* Subroutines for major programs: RelRotRate, AllRelRotRates,
   Intertwine *)

ExpandWordPair[wp_List] :=
Block[

{i, lcm, firstword, secondword},

firstword = wp[[1]]; secondword = wp[[2]];
lcm = LCM[Length[firstword], Length[secondword]];
Return[
   { Flatten[ Table[firstword,  {i, lcm/Length[firstword]  }]],
  Flatten[ Table[secondword, {i, lcm/Length[secondword] }]] }
];
]

GetTop[ewp_List] :=
Block[
   {i, lcm},
   lcm = Length[ ewp[[1]] ];
   Return[
      Table[ bm[[ ewp[[1,i]]+1, ewp[[2,i]]+1]], {i, lcm} ]
   ];
]

GetBottom[ewp_List] :=
Block[
   {i=0, s=0, j=0, prevj=0, nextj=0, cnt=0, lcm=0, sgn=0,
   top={}, ewpd={}, b1={}, b2={}, b3={}, bot={}},

   lcm    = Length[ewp[[1]]];
   top    = GetTop[ewp];
   ewpd   = ewp[[1]] - ewp[[2]];

   (* initialize rows b1, b2, b3 *)
   For[i = 1, i < lcm+1, i++,
```



```
      AppendTo[b1,0]; AppendTo[b2,0]; AppendTo[b3,0];
   ];

   (* calculate b1 row *)
   For[i = 1, i < lcm + 1, i++,
      If[ewpd[[i]] != 0, b1[[i]] = Mod[top[[i]],2]];
   ];

   (* calculate b2 row *)
   For[s = 0, ewpd[[s+1]] == 0, ++s];
   For[i = 1, i < lcm + 1, i++,
      j = i + s; If[j > lcm, j -= lcm];
      prevj = j - 1; If[prevj < 1, prevj += lcm];
      If[0 == ewpd[[j]],
         cnt = 0;
         For[k = j, ewpd[[k]] == 0, k++,
            cnt = cnt + top[[k]];
            If[k == lcm, k = 0];
            i++;
         ];
         cnt = Mod[cnt, 2];
         b2[[prevj]] = cnt;
      ];
   ];

   (* calculate b3 row *)
   For[i = 1, i < lcm + 1, i++,
      j = i + s; If[j > lcm, j -= lcm];
      nextj = j + 1; If[nextj > lcm, nextj -= lcm];
      k = nextj;
      While[0 == ewpd[[nextj]],
         ++nextj; ++i;
         If[nextj > lcm, nextj -= lcm];
      ];
      If[Negative[ewpd[[j]]*ewpd[[nextj]]], sgn = 1, sgn = 0];
      b3[[j]] = sgn;
   ];

   bot = Mod[b1 + b2 + b3, 2];
   Return[bot];
]


SumAll[l_List] :=
```

```
Block[{i},
Return[Sum[l[[i]], {i, Length[l]}]]
]

(* Generates the fundamental cycles of length "period" for a template
   with "b" branches *)

GetFunCyc[b_Integer, period_Integer] :=
Block[
{cycles},
                                  (* get all cycles of length *)
cycles = GetAllCyc[b, period];    (* "period", and b "branches" *)
cycles = DelSubCyc[b, cycles];    (* delete nonfundamental subcycles *)
cycles = DelCycPerm[cycles];      (* delete cyclic permutations
                                     of fundamental cycles *)
Return[cycles];
]

(* Subroutines for GetFunCyc *)

GetAllCyc[branches_Integer, levels_Integer] :=
Block[
   {n, m, i, j,
    roots, tree, nextlevel, cycle, cycles},

   For[i = 0; roots = {}, i < branches, ++i,
    roots = Append[roots, i];
   ];

   (* creates full n-ary tree recursively *)
   tree = {roots};
   For[n = 1, n < levels, ++n,
      nextlevel = {};
     For[m = 1, m <= branches, ++m,
       nextlevel = Append[nextlevel, Last[tree]];
     ];
     tree = Append[tree, Flatten[nextlevel]];
   ];

   (* reads up each branch of tree to root, from left to right *)
   cycles = {};
   For[i = 1, i <= branches^levels, ++i,
     cycle = {};
        For[j = levels, j > 0, --j,
```



```
          k = Ceiling[i/(branches^(levels-j))];
          cycle = Prepend[cycle, tree[[j]][[k]]];
        ];
        cycles = Append[cycles, cycle];
   ];
   Return[cycles];
]

DelSubCyc[b_Integer, allcycls_List] :=
Block[
   {i, j, k, levels, period, numofdivs, numofsub, copies, wordpos,
    cycles = {}, div = {}, word = {}, droplist = {},
    subcycles = {}, subword = {}, nonfun = {}, funcycls = {}},

   (* Initializations and gets divisor list *)
   cycles = allcycls; levels = Length[ cycles[[1]] ];
   period = levels; div = Divisors[levels];

   (* Creates nonfundamental words from periodic orbits created from
      divisor list *)
   numofdivs = Length[div];
   For[i = 1, i < numofdivs, ++i,     (* go throw divisor list *)
      copies = period/div[[i]];
      subcycles = GetAllCyc[b, div[[i]]];
      numofsub = Length[subcycles];
      (* create subwords of lengths found in divisor list *)
      For[j = 1; subword = {}, j <= numofsub, ++j,
         subword = subcycles[[j]];
         (* expand subwords to length of periodic orbits *)
         For[k = 1; word = {}, k <= copies, ++k,
            word = Flatten[Append[word, subword]];
         ];
         (* find positions of nonfundamental cycles in all cycles *)
         wordpos = Flatten[Position[cycles, word]][[1]];
         droplist = Union[Append[droplist, wordpos]];
      ];
   ];

   (* this is a kludge to delete nonfundamental cycles *)
   For[i = 1; nonfun = {}, i <= Length[droplist], ++i,
      nonfun = Append[nonfun, cycles[[droplist[[i]]]]];
   ];
   funcycls = Complement[cycles, nonfun];
```

```
   Return[funcycls];
]

DelCycPerm[funcycls_List] :=
Block[
   {size, i, j, period,
    cycs, word },
   cycs = funcycls;
   size = Length[cycs]; period = Length[cycs[[1]]];
   For[i = 1, i < size, ++i,
      word = cycs[[i]];
      For[j = 1, j < period, ++j,
         word = RotateLeft[word];
         cycs = Complement[cycs, {word}];
         size = size-1;
      ];
   ];
   Return[cycs];
]
```