```
> restart : with(LinearAlgebra) : with(ListTools) : with(VectorCalculus) : with(combinat) :
    with(ArrayTools) : with(Grid) :
```

Computing all steady states for second-order coupling

Setting up the system. We set the parameters to their standard values in the paper.
The only assumption is that s is a quartic polynomial with a positive leading coefficient.
For all N from 1 to Nmax, the necessary arrays are set up. tol denotes a tolerance; anything smaller is considered as zero.

```
> r := 0; o1 := 1; p1 := −1; q1 := 0; o2 := 1; p2 := −2; q2 := 3;
```

$$r := 0$$
$$o1 := 1$$
$$p1 := -1$$
$$q1 := 0$$
$$o2 := 1$$
$$p2 := -2$$
$$q2 := 3 \quad \textbf{(1)}$$

```
> Nmax := 10; Nprint := Nmax; μc := 3.5; γc := 1; tol := 1e−8;
  vars := [x[j] $ j = 1 ..Nmax];
  s := (x)→(o1·x^2 + p1·x + q1 − μc)·(o2·x^2 + p2·x + q2 − μc) :
  collect(s(x), x);
```

$$Nmax := 10$$
$$Nprint := 10$$
$$\mu_c := 3.5$$
$$\gamma_c := 1$$
$$tol := 1. \times 10^{-8}$$
$$vars := [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}]$$
$$1.75 + x^4 - 3\,x^3 - 2.0\,x^2 + 7.5\,x \quad \textbf{(2)}$$

```
> σ := Array(1 ..Nmax, 0) :
  sys := Array(1 ..Nmax, 0) :
  J := Array(1 ..Nmax, 0) :
  steadyNew := Array(1 ..Nmax, 0) : steadyTot := Array(1 ..Nmax, 0) :
  statNew := Array(1 ..Nmax, 0) : statTot := Array(1 ..Nmax, 0) :
  parts4 := Array(1 ..Nmax, 0) :
  perms4 := Array(1 ..Nmax, 0) :
```

A first auxiliary procedure for producing extended distributions of length 2 and 4.

```
> ext4 := proc(l)
  # extend smaller splits to splits with two or four entries by adding zeros
  # all possibilities are returned as a sequence
  local nl, res;
  nl := nops(l);
```

```
    if nl = 4 then res := l;
    else res := [op(l), 0 $ 4 - nl];
    end if;
    if nl = 2 then res := res, l;
    elif nl = 1 then res := res, [l[1], 0];
    end if;
    return res;
    end proc:
```

We produce all possible partitions of each N with up to four summands and each partition is permuted in all possible ways.

```
> for N from 1 to Nmax do
    σ[N] := add(vars[1..N]);
    sys[N] := [ -x[j]·s(x[j]) + γ_c·x[j]·σ[N] $ j = 1..N ];
    J[N] := expand(Jacobian(sys[N], vars[1..N]));
    parts4[N] := map(ext4, select(l→evalb(nops(l) ≤ 4), partition(N)));
    perms4[N] := map(op@permute, parts4[N]);
    print(N, nops(parts4[N]), "partitions", nops(perms4[N]), "permutations");
  end do:
```

1, 2, "partitions", 6, "permutations"

2, 4, "partitions", 13, "permutations"

3, 5, "partitions", 24, "permutations"

4, 8, "partitions", 40, "permutations"

5, 9, "partitions", 62, "permutations"

6, 13, "partitions", 91, "permutations"

7, 15, "partitions", 128, "permutations"

8, 20, "partitions", 174, "permutations"

9, 23, "partitions", 230, "permutations"

10, 29, "partitions", 297, "permutations" **(3)**

```
> print("Total: ", add(map(nops, parts4)), "partitions", add(map(nops, perms4)),
      "permutations");
```

"Total: ", 128, "partitions", 1065, "permutations" **(4)**

Analysing the form of the graph of s

Depending on the value of μ, there can be one or three critical points: one or two minima and zero or one maxima. We compute all critical points and determine which minimum is the global one (we only need the ordinates of the extrema).

```
> s1 := D(s)
```

$$s1 := x \mapsto (2\cdot x - 1)\cdot\left(x^2 - 2\cdot x - 0.5\right) + \left(x^2 - x - 3.5\right)\cdot(2\cdot x - 2) \quad \textbf{(5)}$$

```
> cp := [fsolve(s1(x), x)];
  if nops(cp) = 1 then
    cmin1 := s(cp[1]);
    cmin2 := FAIL;
    cmax := FAIL;
  else
```

```
    cv := map(s, cp);
    cmin1 := min(cv[1], cv[3]);
    cmin2 := max(cv[1], cv[3]);
    cmax := cv[2];
  end if;
```

$$cp := [-0.9393732160,\ 0.8550829284,\ 2.334290288]$$

$$cv := [-3.794703657,\ 5.359768465,\ -0.1080335549]$$

$$cmin1 := -3.794703657$$

$$cmin2 := -0.1080335549$$

$$cmax := 5.359768465 \qquad \textbf{(6)}$$

Routines for numerically computing steady states exploiting the residual symmetry and analysing their stability behaviour

The first procedure evaluates the condition for a steady state as described in the paper. The use of second-order coupling is hard coded into this procedure.

```
> symeq := proc(c, sp)
  # evaluates condition for steady state with residual symmetry determined by a given split
  # sp desired split
  # c height at which graph of s is intersected,
  # c defines steady state corresponding to split, if return value 0
  option hfloat;
  local ys, x;
  global s, μc, γc;
   ys := map(rhs, RootFinding:-Isolate(s(x) - c, x));
   if nops(ys) ≠ nops(sp) then
      error("split not suitable");
   else
      c - γc·DotProduct(Vector(sp), Vector(ys));
  end if;
  end proc:
```

The second procedure searches for critical values of c in a prescribed range. Using the NextZero option, we compute iteratively all critical values (in most cases, only one exists, but one can observe a few exceptions). Returned are not the critical values, but the corresponding preimages computed using Isolate.

```
> findy := proc(c1, c2, sp)
  # search for critical values of c corresponding to steady states
  # computing corresponding coordinates
  option hfloat;
  local c, x, ys, res;
  global s, tol;
  res := [ ];
  c := fsolve(x→symeq(x, sp), c1 ..c2, method = NextZero);
  while type(c,'float') do
     ys := map(rhs, RootFinding:-Isolate(s(x) - c, x));
     res := [op(res), remove[indices](i→evalb(sp[i] = 0), ys)];
     c := fsolve(x→symeq(x, sp), (c + tol) ..c2, method = NextZero);
```

```
 end do;
 return res;
end proc:
```

The third procedure deals with distributions of length 2. It uses the previous procedure to search for critical values of c and to collect the corresponding real preimages. It uses the above computed information about the shape of the graph of s in form of the global variables cmin1, cmin2 and cmax and decides on their basis in which regions findy should search for critical values.

```
> relroots2 := proc(sp)
  # determines the real preimages corresponding to critical values of c for splits of length 2
  # cmin1 global minimum, cmin2 local minimum (or FAIL), cmax local maximum (or FAIL)
  # sp desired split of length 2,
  # returns list of corresponding preimages or FAIL
  # a list of lists is returned with one entry for each critical value
  option hfloat;
  local res;
  global cmin1, cmin2, cmax;
  if cmax = FAIL then
     res := findy(cmin1, max(25, 10·abs(cmin1)), sp);
  else      # must search in two regions!
   res := [op(findy(cmin1, cmin2, sp)), op(findy(cmax, max(25, 10·abs(cmax)), sp))];
  end if;
  if res = [ ] then FAIL else res end if;
  end proc:
```

The fourth procedure is analogous to the last one, but handles distributions of length 4.

```
> relroots4 := proc(sp)
  # determines the real preimages corresponding to critical values of c for splits of length 4
  # cmin1 global minimum, cmin2 local minimum (or FAIL), cmax local maximum (or FAIL)
  # sp desired split of length 4,
  # returns list of corresponding preimages or FAIL
  # a list of lists is returned with one entry for each critical value
  option hfloat;
  local res;
  global cmin1, cmin2, cmax;
  if cmax = FAIL then error "no region with four preimages" end if;
  res := findy(cmin2, cmax, sp);
  if res = [ ] then FAIL else res end if;
  end proc:
```

The following auxiliary procedure modifies the built-in function sign by including zero as a possible result. The global variable tol says up to which absolute value a number is considered as zero.

```
> realsign := proc(x)
  # determine sign of real part of potentially complex number with tolerance tol around 0
  # (opposed to the built-in sign function, we distinguish zero from positive or negative)
  local r;
  global tol;
  r := Re(x);
  if abs(r) < tol then 0
  else sign(r)
  end if;
```

```
end proc:
```

The sixth procedure performs a stability analysis of a family of steady states. Given an extended distribution and the corresponding preimages, it evaluates for one steady state in the family the Jacobian (all other steady states possess the same stability). It returns the number of eigenvalues with positive, negative or zero real part (we never observed a zero real part).

```
> stabroot := proc(sp, yb)
  # stability analysis of steady state corresponding to split sp and preimages yb
  # returns numbers (np,nm,n0) of eigenvalues of Jacobian with positive, negative or zero real part
  local spb, JJ, xx, i, j, k, l, evas;
  global N, J;
  spb := remove(x->x=0, sp);
  xx := [ 0 $ N ];
  k := 1;
  for i from 1 to nops(spb) do
    for j from 1 to spb[i] do
      xx[k] := yb[i];
      k := k + 1;
    end do;
  end do;
  JJ := subs(seq(x[l]=xx[l] $ l=1..N), J[N]);
  evas := map(realsign, Eigenvalues(JJ));
  return nops([SearchAll(1, evas)]), nops([SearchAll(-1, evas)]), nops([SearchAll(0,
    evas)]);
  end proc:
```

Computing for each split a representative steady state and determining its stability

The following auxiliary procedure uses the information computed by stabroot for a classification of steady states. In the case of stable nodes, we further distinguish whether they have more or less than three levels. In the case of saddle points, we distinguish separating saddle points having a hypersurface als invariant manifold. Furthermore, it determines how many members the corresponding family has.

```
> anapart2 := proc(par, ys)
  # analyse for the split par the stability of a representative of the family of steady states given by ys
  # returns a list [type, num, ys, par] with
  # - the type of the steady state (3=stable node (3 or 4 split), 4=stable node (1 or 2 split),
  #                                 5=unstable node, 6=separating saddle, 7=non-separating saddle,
  #                                 8=non-hyperbolic)
  # - the num(ber) of elements in the family
  # - the different coordinates ys of the steady states
  # - the partition par
  local np, nm, n0, mnc;
  global N;
  mnc := multinomial(N, op(par));
  np, nm, n0 := stabroot(par, ys);
  if n0 > 0 then
    #print(par, ys, mnc, "not hyperbolic");
    return [8, mnc, ys, par];
  elif nm=N then
    if nops(select(x->evalb(x <> 0), par)) > 2 then
```

```
      #print(par, ys, mnc, "stable node - 3/4 split");
      return [3, mnc, ys, par];
    else
      #print(par, ys, mnc, "stable node - 1/2 split");
      return [4, mnc, ys, par];
    end if;
  elif np = N then
    #print(par, ys, mnc, "unstable node");
    return [5, mnc, ys, par];
  elif (nm = N − 1) or (np = N − 1) then
    #print(par, ys, mnc, "separating saddle");
    return [6, mnc, ys, par];
  else
    #print(par, ys, mnc, "non-separating saddle");
    return [7, mnc, ys, par];
  end if;
  end proc:
```

The next procedure is the top level one. Depending on the size of the extended distribution, it determines via relroots2 and relroots4, respectively, the existence of steady states. If some exist, they are analysed by anapart2.

```
> anapart := proc(par)
  # analyse for the split par the stability of a representative of each arising family of steady states
  # returns a sequence with one result for each family
  local yss, ys, res;
  if nops(par) = 2 then
    yss := relroots2(par);
  else
    yss := relroots4(par);
  end if;
  if yss = FAIL then    # no steady states exist
    #print(par, ys);
    return NULL;
  else
    res := NULL;
    for ys in yss do
      res := res, anapart2(par, ys);
    end do;
    return res;
  end if;
  end proc:
```

The following auxiliary procedure takes lower-dimensional steady states without zero levels and lifts them to the current dimension by adding zero levels. Both the old, lower-dimensional steady states and the new ones of the current dimension are managed in global arrays.

```
> lift := proc( )
  # "promote" all steady states for lower N's without zero entries
  # to steady states for current N with zero entries
  # and analyse their stability
  local n, z, ys, par, res, resl;
```

```
global N, steadyNew, steadyTot;
resl := NULL;
for n from N − 1 by −1 to 1 do
  for z in steadyNew[n] do
    if z[3] = [0.] then
      ys := z[3];
      par := [N];
    else
      ys := [op(z[3]), 0];
      par := [op(z[4]), N − n];
    end if;
    res := anapart2(par, ys);
    #if res[1] ≠ z[1] then
    #   print("change in stability in lift", par, ys);
    #end if;
    resl := resl, res;
  end do;
end do;
steadyTot[N] := [op(steadyNew[N]), resl];
return NULL;
end proc:
```

The final auxiliary procedure performs some statistics on the found types of steady states by adding up the results of previous computations. It returns an array with eight components. The first entry is the number of families of steady states and the second entry the total number of steady states. The remaining entries count the different types of steady states using the type code from anapart2.

```
> count := proc(steady)
  # count the different types of steady states
  local z, stat;
  stat := Array(1..8, fill = 0);
  stat[1] := nops(steady);
  for z in steady do
    if z ≠ 0 then
      stat[2] := stat[2] + z[2];
      stat[z[1]] := stat[z[1]] + z[2];
    end if;
  end do;
  return stat;
  end proc:
```

```
> stt := time[real]( ) :
```

Treating the special case N=1 to start induction

For N=1, we simply compute directly the steady states of the scalar differential equation using Isolate. The results are written to a data file. In a separate data file, we start to accumulate the total statistics of the steady states and their types.

```
> fnameN := FileTools:-JoinPath([ "Maple", "MathBiol", "Speciation" , "SteadyStatesN1.txt"],
      base = homedir) :
  fdN := fopen(fnameN, WRITE) :
  fprintf (fdN, "Steady states for N= 1 computed by SteadyStatesSecondOrder.mw\n") :
```

```
fprintf(fdN,
    "o1=%.1f, p1=%.1f, q1=%.1f, o2=%.1f, p2=%.1f, q2=%.1f, r=%.3f, N=%2d, gamma=%.2f,
    mu=%.2f\n", o1, p1, q1, o2, p2, q2, r, 1, γc, μc) :
fprintf(fdN, "type\t mult\t\t levels\t\t dist\n") :
fnameS := FileTools:-JoinPath([ "Maple", "MathBiol", "Speciation",
    "SteadyStatesStatistics.txt"], base = homedir) :
fdS := fopen(fnameS, WRITE) :
fprintf(fdS,
    "Statistics about steady states up to N=%2d computed by SteadyStatesSecondOrder.mw\n",
    Nmax) :
fprintf(fdS,
    "o1=%.1f, p1=%.1f, q1=%.1f, o2=%.1f, p2=%.1f, q2=%.1f, r=%.3f, gamma=%.2f, mu=
    %.2f\n", o1, p1, q1, o2, p2, q2, r, γc, μc) :
fprintf(fdS,
    " N   perm\t new fam\t new\t\t total fam\t total\t\t stable3\t stable\t\t unstable\t sep saddle\t
    saddle\t\t non-hyp\n") :
> N := 1 :
ysl := map(rhs, RootFinding:-Isolate(sys[1][1], vars[1])):
J1 := map(y→subs(vars[1]=y, J[1])[1, 1], ysl):
steadyNew[N] := Array(1..nops(ysl), 0) :
for i from 1 to nops(ysl) do
  if abs(J1[i]) < tol then
    steadyNew[1][i] := [7, 1, [ysl[i]], [1]];
  elif J1[i] < 0 then
    steadyNew[1][i] := [3, 1, [ysl[i]], [1]];
  else
    steadyNew[1][i] := [4, 1, [ysl[i]], [1]];
  end if;
  fprintf(fdN, "%1d\t %12d\t [%12.8f]\t [%2d]\n",
      steadyNew[1][i][1], steadyNew[1][i][2], convert(steadyNew[1][i][3], Array),
    convert(steadyNew[1][i][4], Array)) :
end do:
steadyTot[N] := steadyNew[N] :
statNew[N] := count(steadyNew[N]):
statTot[N] := statNew[N];
fprintf(fdS, "%2d   %3d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\n", 1,
    nops(perms4[1]), statNew[1][1], statNew[1][2], op(convert(statTot[1], list))) :
fclose(fdN) :
```

$$ysl := [-1.347717443, -0.2573915659, 0., 1.795655704, 2.809453304]$$

$$J1 := [-19.20211076, 1.767019690, -1.75, 11.74814279, -36.31305175]$$

$$statNew_1 := \begin{bmatrix} 4 & 5 & 3 & 2 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$statTot_1 := \begin{bmatrix} 4 & 5 & 3 & 2 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(7)}$$

Running over all higher values of N until Nmax

In the finite induction to Nmax, we treat all permutations for the current dimension in parallel. Some statistics on computation times and the number of found steady states of the different types are printed. For each dimension N<=Nprint a separate data file is produced containing the coordinates, the multiplicity and the type of each family of steady states. For larger dimensions the file sizes would be too large.

```
> Set('r','o1','p1','q1','o2','p2','q2','μc','γc','tol','s','J','sys','cmin1','cmin2','cmax','symeq','findy','relroots2
  ','relroots4','realsign','stabroot','anapart2','anapart'):
> for N from 2 to Nmax do
    Set('N');
    st := time[real]( ): steadyNew[N] := Map(anapart, perms4[N]): st := time[real]( ) - st:
    print("N=", N, "used time for searching for new steady states", st);
    statNew[N] := count(steadyNew[N]): print("statistics (new)", statNew[N]);
    st := time[real]( ): lift( ): st := time[real]( ) - st:
    print("N=", N, "used time for lifting old steady states", st);
    statTot[N] := count(steadyTot[N]): print("statistics (total)", statTot[N]);
    fprintf(fdS, "%2d    %3d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\t%8d\n", N,
     nops(perms4[N]), statNew[N][1], statNew[N][2], op(convert(statTot[N], list))):
    if N ≤ Nprint then
      fnameN := FileTools:-JoinPath([ "Maple", "MathBiol", "Speciation",
     cat("SteadyStatesN", N, ".txt")], base=homedir):
      fdN := fopen(fnameN, WRITE):
      fprintf(fdN, "Steady states for N=%2d computed by SteadyStatesSecondOrder.mw\n", N):
      fprintf(fdN,
     "o1=%.1f, p1=%.1f, q1=%.1f, o2=%.1f, p2=%.1f, q2=%.1f, r=%.3f, N=%2d, gamma=%.2f,
     mu=%.2f\n", o1, p1, q1, o2, p2, q2, r, N, γc, μc):
      fprintf(fdN, "type\t\t mult\t\t levels\t\t dist\n"):
      for st in steadyTot[N] do
        fprintf(fdN, "%1d\t %12d\t [%12.8f]\t [%2d]\n",
                st[1], st[2], convert(st[3], Array), convert(st[4], Array)):
      end do:
      fclose(fdN);
    end if:
  end do:
  fclose(fdS):
  stt := time[real]( ) - stt:
  print("total time", stt);
```

"N=", 2, "used time for searching for new steady states", 6.105

"statistics (new)", [10 16 0 6 2 8 0 0]

"N=", 2, "used time for lifting old steady states", 0.013

"statistics (total)", [15 25 0 9 4 12 0 0]

"N=", 3, "used time for searching for new steady states", 10.323

"statistics (new)", [16 52 0 8 2 42 0 0]

"N=", 3, "used time for lifting old steady states", 0.010

"statistics (total)", [31 113 6 15 8 84 0 0]

"N=", 4, "used time for searching for new steady states", 16.997

"statistics (new)", [24 168 12 16 2 84 54 0]

"N=", 4, "used time for lifting old steady states", 0.020

"statistics (total)", [55 489 24 31 16 244 174 0]

"N=", 5, "used time for searching for new steady states", 30.254

"statistics (new)", [25 512 30 26 1 145 310 0]

"N=", 5, "used time for lifting old steady states", 0.036

"statistics (total)", [80 2053 50 57 31 675 1240 0]

"N=", 6, "used time for searching for new steady states", 38.532

"statistics (new)", [34 2095 90 42 0 468 1495 0]

"N=", 6, "used time for lifting old steady states", 0.056

"statistics (total)", [114 8992 180 99 62 1776 6875 0]

"N=", 7, "used time for searching for new steady states", 55.042

"statistics (new)", [40 7254 350 99 0 1520 5285 0]

"N=", 7, "used time for lifting old steady states", 0.078

"statistics (total)", [154 40736 602 198 119 5230 34587 0]

"N=", 8, "used time for searching for new steady states", 76.072

"statistics (new)", [45 20082 700 219 0 2453 16710 0]

"N=", 8, "used time for lifting old steady states", 0.120

"statistics (total)",

[199 180599 1316 382 218 13565 165118 0]

"N=", 9, "used time for searching for new steady states", 98.210

"statistics (new)", [54 83030 1260 382 0 8137 73251 0]

"N=", 9, "used time for lifting old steady states", 0.164

"statistics (total)",

$$\begin{bmatrix} 253 & 791553 & 3096 & 638 & 381 & 34678 & 752760 & 0 \end{bmatrix}$$

"N=", 10, "used time for searching for new steady states", 128.027

"statistics (new)", $\begin{bmatrix} 67 & 307186 & 5460 & 848 & 0 & 32801 & 268077 & 0 \end{bmatrix}$

"N=", 10, "used time for lifting old steady states", 0.256

"statistics (total)",

$$\begin{bmatrix} 320 & 3522911 & 10590 & 1234 & 637 & 93281 & 3417169 & 0 \end{bmatrix}$$

"total time", 463.062 **(8)**

>

Statistical analysis of the growth of the total number of steady states in dependency on the dimension

We perform an exponential fit and compare its results with the actual values. For analysing the accuracy of the obtained model, we look at the relative errors. As they are randomly distributed and fairly small, the model seems to be a very good fit as also indicated by an R^2 value of 1.

> *with*(*plots*) : *with*(*Statistics*) :

> *NN* := ['*n*' $ '*n*'= 3 ..*Nmax*];
FSt := ['*statTot*[*n*][1]' $ '*n*'= 3 ..*Nmax*];
StSt := ['*statTot*[*n*][2]' $ '*n*'= 3 ..*Nmax*];
PP := ['*nops*(*perms4*[*n*])' $ '*n*'= 3 ..*Nmax*];

$$NN := [3, 4, 5, 6, 7, 8, 9, 10]$$

$$FSt := \left[(statTot_3)_1, (statTot_4)_1, (statTot_5)_1, (statTot_6)_1, (statTot_7)_1, (statTot_8)_1, (statTot_9)_1, (statTot_{10})_1 \right]$$

$$StSt := \left[(statTot_3)_2, (statTot_4)_2, (statTot_5)_2, (statTot_6)_2, (statTot_7)_2, (statTot_8)_2, (statTot_9)_2, (statTot_{10})_2 \right]$$

$$PP := [1, 1, 1, 1, 1, 1, 1, 1] \quad \textbf{(9)}$$

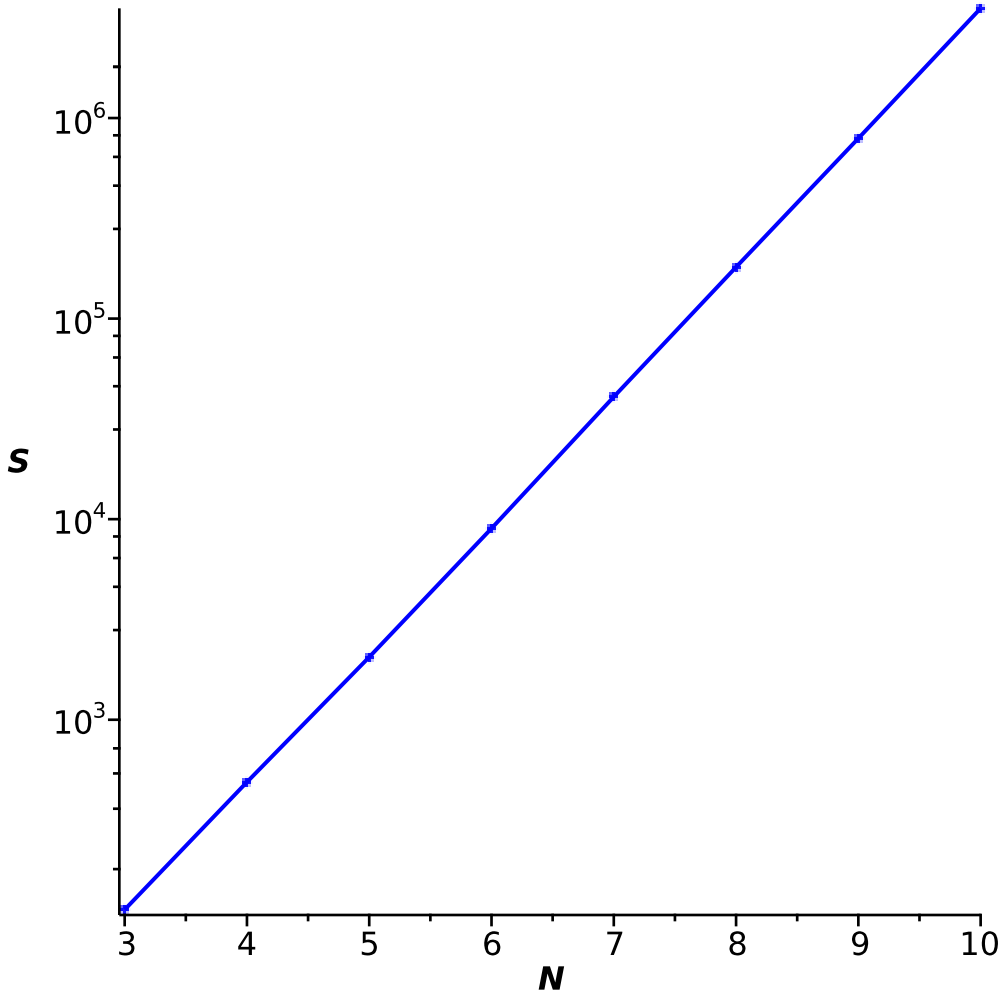
> *smodel* := *ExponentialFit*(*NN*, *StSt*,'*z*', *summarize* = *true*)

```
Summary:
----------------
Model: 1.2946662*exp(1.4800063*z)
----------------
Coefficients:
              Estimate  Std. Error  t-value   P(>|t|)
Parameter 1    0.2583    0.0258      10.0274    0.0001
Parameter 2    1.4800    0.0037      396.0533   0.0000
----------------
R-squared: 1.0000, Adjusted R-squared: 1.0000
```

(10)

$$smodel := 1.29466620910786\, e^{1.48000629282264\, z} \quad \textbf{(10)}$$

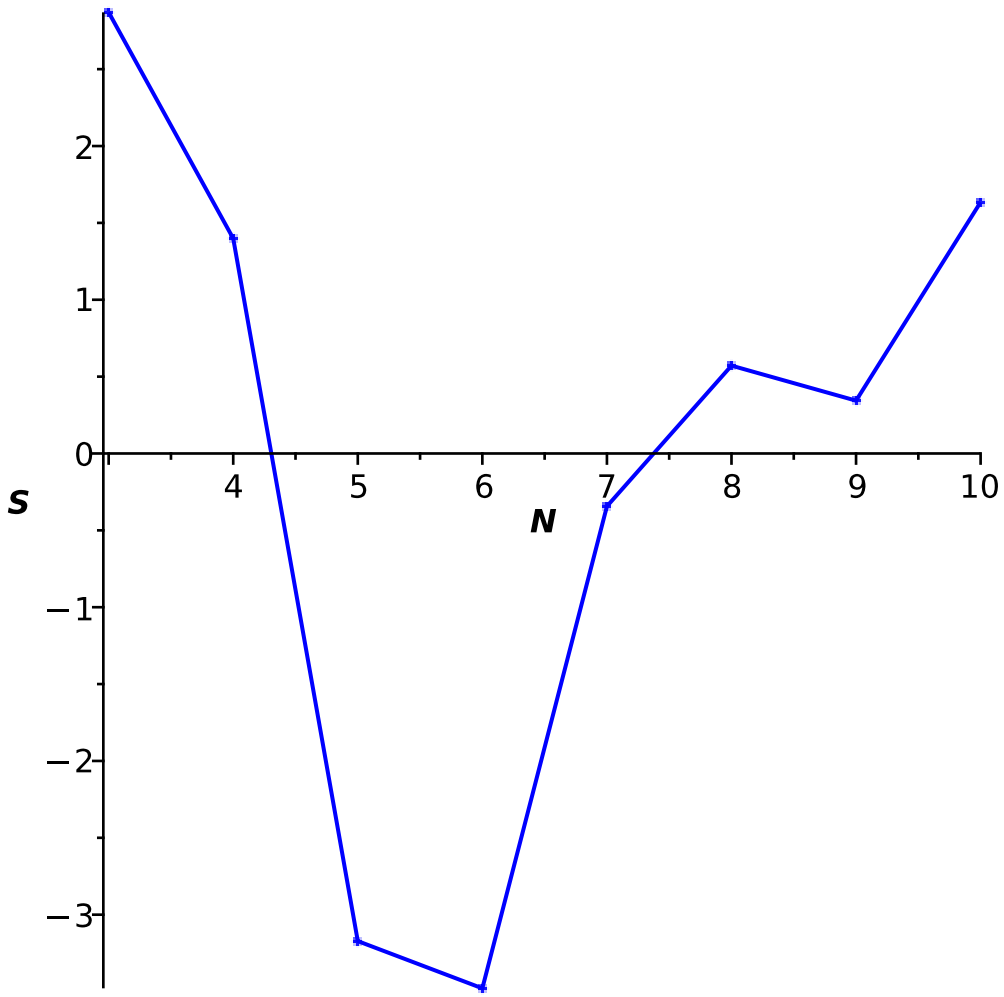
```
> pointplot(NN, StSt, style=pointline, symbol=soliddiamond, symbolsize=7, axis[2]=[mode
    =log],
          labels=['N','S'], labelfont=[Helvetica, bold, 12], font=[Helvetica, roman, 12], color
    =blue);
```




```
> StErr := [100·(StSt['n'-2]-evalf(subs(z='n', smodel)))/StSt['n'-2] $ 'n'=3..Nmax];
```

$$StErr := [2.86962442941164,\ 1.39856885617962,\ -3.17196316970508,\ -3.47916407282562,\ -0.343534405899064,\ 0.571474031983991,\ 0.343665491737400,\ 1.63472670377920] \quad \textbf{(11)}$$

```
> pointplot(NN, StErr, style=pointline, symbol=soliddiamond, symbolsize=7,
          labels=['N','S'], labelfont=[Helvetica, bold, 12], font=[Helvetica, roman, 12], color
    =blue);
```



> $\frac{add(map(\mathrm{abs}, StErr))}{nops(StErr)}$

1.72659014519020 **(12)**

>

> $p3model := PolynomialFit(3, NN, FSt,'z', summarize = true)$

```
Summary:
----------------
Model: -26.952381+18.832612*z-.43939394*z^2+.20202020*z^3
----------------
Coefficients:
              Estimate    Std. Error    t-value   P(>|t|)
Parameter 1   -26.9524     11.7590     -2.2921    0.0837
Parameter 2    18.8326     6.2205      3.0275    0.0389
Parameter 3   -0.4394      1.0149     -0.4329    0.6874
Parameter 4    0.2020      0.0518      3.8996    0.0175
----------------
R-squared: 0.9999, Adjusted R-squared: 0.9998
```

$p3model := -26.9523809523806 + 18.8326118326117\,z - 0.439393939393914\,z^2 + 0.202020202020201\,z^3$ **(13)**

> $p4model := PolynomialFit(4, NN, FSt,'z', summarize = true)$

```
Summary:
----------------
```

```
Model: 6.5476190-5.8870851*z+5.9431818*z^2-.48737374*
z^3+.26515152e-1*z^4
----------------
Coefficients:
              Estimate   Std. Error    t-value   P(>|t|)
Parameter 1    6.5476     40.5921       0.1613    0.8821
Parameter 2   -5.8871     29.2938      -0.2010    0.8536
Parameter 3    5.9432     7.4535        0.7974    0.4835
Parameter 4   -0.4874     0.7989       -0.6101    0.5849
Parameter 5    0.0265     0.0307        0.8649    0.4507
----------------
R-squared: 0.9999, Adjusted R-squared: 0.9998
```

$$p4model := 6.54761904761585 - 5.88708513708277\,z + 5.94318181818121\,z^2 - 0.487373737373673\,z^3 + 0.0265151515151491\,z^4 \quad (14)$$

```
> emodel := ExponentialFit( NN, FSt,'z', summarize = true)
Summary:
----------------
Model: 14.647220*exp(.32149056*z)
----------------
Coefficients:
              Estimate   Std. Error   t-value   P(>|t|)
Parameter 1    2.6843     0.1384      19.3879    0.0000
Parameter 2    0.3215     0.0201      16.0038    0.0000
----------------
R-squared: 0.9995, Adjusted R-squared: 0.9993
```

$$emodel := 14.6472199523716\,e^{0.321490556914364\,z} \quad (15)$$

$$> FStErrP3 := \left[100 \cdot \frac{(FSt['n'-2] - evalf(subs(z='n', p3model)))}{FSt['n'-2]}\ \$\ 'n'=3..Nmax\right];$$

$$FStErrP4 := \left[100 \cdot \frac{(FSt['n'-2] - evalf(subs(z='n', p4model)))}{FSt['n'-2]}\ \$\ 'n'=3..Nmax\right];$$

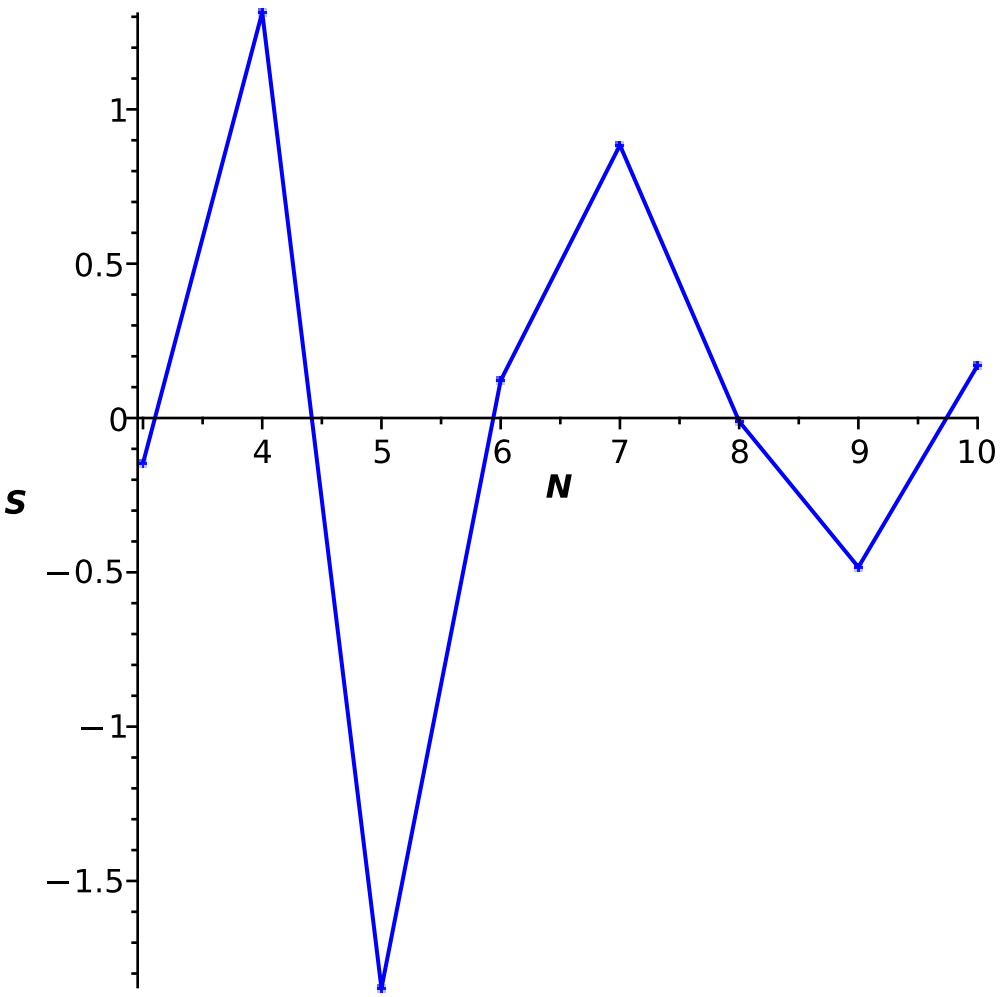
$$FStErrP3 := [-0.146627565982566, 1.31444313262491, -1.84794372294374, 0.121515910989607, 0.884072637319387, -0.0108768952990021, -0.483376965590388, 0.170454545454559]$$

$$FStErrP4 := [-1.17302052785922, 2.38882329791407, -1.67748917748921, -0.237335763651521, 0.618429189857814, 0.0576475450847439, -0.249816060092745, 0.0710227272727160] \quad (16)$$

$$> FStErrE := \left[100 \cdot \frac{(FSt['n'-2] - evalf(subs(z='n', emodel)))}{FSt['n'-2]}\ \$\ 'n'=3..Nmax\right]$$

$$FStErrE := [-23.9533401104249, 3.64398885732060, 8.63642381724819, 11.5740364186341, 9.72125640555998, 3.64489573988900, -4.52708899057941, -13.9779846284067] \quad (17)$$

```
> pointplot(NN, FStErrP3, style = pointline, symbol = soliddiamond, symbolsize = 7,
          labels = ['N','S'], labelfont = [Helvetica, bold, 12], font = [Helvetica, roman, 12], color
    = blue);
```


1
0.5
0
−0.5
−1
−1.5
4
5
6
7
8
9
10
N
S

>