Jon Newman
Sept 6 2008

The following is an implementation of the multipoint shooting algorithm for continuos flows applied to the Rossler Flow. This code is good for finding periodic orbits of that are 2-periodic or greater.

Function 1: CycleEstimateRossler

This function takes a set of initial guesses on a specified Poincaré section and runs each of them for one return to that section. It simultaneously records the deformation of a sphere surrounding each initial guess during its time evolution. This provides the MPS algorithm with an initial guess and information about how that a change to that guess will affect the resulting end point after a single return.

In[289]:=
```
cycleEstimateRossler[a_, b_, c_, guess_] :=
  Module[{},

   (*state equations: Rossler Flow*)
   dxdt = x'[t] == -y[t] - z[t];
   dydt = y'[t] == x[t] + a * y[t];
   dzdt = z'[t] == b + z[t] * (x[t] - c);

   (*evolution operators of fundamental matrix: J̊=AJ*)
   dm11 = m11'[t] == -m21[t] - m31[t];
   dm12 = m12'[t] == -m22[t] - m32[t];
   dm13 = m13'[t] == -m23[t] - m33[t];

   dm21 = m21'[t] == m11[t] + a * m21[t];
   dm22 = m22'[t] == m12[t] + a * m22[t];
   dm23 = m23'[t] == m13[t] + a * m23[t];

   dm31 = m31'[t] == z[t] * m11[t] + (x[t] - c) * m31[t];
   dm32 = m32'[t] == z[t] * m12[t] + (x[t] - c) * m32[t];
   dm33 = m33'[t] == z[t] * m13[t] + (x[t] - c) * m33[t];

   (*storage: points from the poincare section*)
   sect = {};
   (*storage: Fundemental matrix at time of poincare section crossing*)
   Jmatrix = {};
   (*Here we test the performance of the intial guesses by numerically
    integrating each for a single return.  This gives us a set of errors
    that can be simultaneiously minimized in the MPSM newton routine.*)
   Do[
    {xinit, yinit, zinit} = j;
    sol = NDSolve[(*RK4*)
      {dxdt, dydt, dzdt,
       dm11, dm12, dm13,
       dm21, dm22, dm23,
       dm31, dm32, dm33,
       x[0] == xinit, y[0] == yinit, z[0] == zinit,
       m11[0] == 1, m13[0] == 0, m12[0] == 0,
       m21[0] == 0, m22[0] == 1, m23[0] == 0,
       m31[0] == 0, m32[0] == 0, m33[0] == 1},
      {x, y, z, m11, m12, m13, m21, m22, m23, m31, m32, m33},
      {t, 0, 1000},
```

```
        MaxSteps → Infinity,
        MaxStepSize → 0.01,
        (*EventLocator is Mathematica's event finding system for numerical integration*)
        Method → {"EventLocator",
          (*poincare section: x[t]=0*)
          "Event" → x[t],
          (*Orientation condition: only record section point for ∂tx>0 at crossing*)
          Direction → -1,
          (*EvenAction says what to do when even criteria are met*)
          EventAction :→ Throw[{AppendTo[sect, {x[t], y[t], z[t]}]}, "StopIntegration"]}];
      (*find the total time of the run up to the even*)
      tOfRun = (x /. sol[[1, 1]])[[1, 1, 2]];
      (*find the total time of the run up to the even*)
      AppendTo[Jmatrix, Flatten[{{m11[tOfRun], m12[tOfRun], m13[tOfRun]},
          {m21[tOfRun], m22[tOfRun], m23[tOfRun]},
          {m31[tOfRun], m32[tOfRun], m33[tOfRun]}} /. sol, 1]],
      {j, guess}]; (*repeat one period of for every j ∈ initial guess*)
    Return[{Jmatrix, sect}];
   ];
```

##

From here on code should be easily generalizable to any continuous flow

##

Function 2 : MConstructRossler

This function constructs a matrix, M, that essentially allows a single newton method to be run for each periodic portion of an n-periodic cycle. The logic is this: the end point at the section after time evolution of a $j_1 \in \{\text{initial guess}\} = j_2$, the end point of $j_2 \in \{\text{initial guess}\} = j_3$...the end point of $j_n \in \{\text{initial guess}\} = j_1$. Each of these allows us to correct an error that is the dimension using a separate Newton routine. The amount of change to each initial guess is a function of the deformation matrices calculated in the previous function.

In[290]:=
```
(*Function to create M Matrix by constructing its four blocks and the concatinating*)
MConstructRossler[n_, JM_, guess_, aMat_, a_, b_, c_] :=
  (*cycle-length,J, guess placement,A,a,b,c*)
  Module[{},
   nullSeed = ConstantArray[0, {3, (n - 2) * 3}];
    upperleft = {};
   Do[
    piece = RotateRight[Join[-JM[[i]], IdentityMatrix[3], nullSeed, 2], {0, (3 * i) - 3}];
    AppendTo[upperleft, piece],
    {i, 1, n - 1}];
   PrependTo[upperleft, Join[IdentityMatrix[3], nullSeed, -JM[[n]], 2]];
   UL = Flatten[upperleft, 1];
   lowerleft = {};
   (*In the yellow text I create the lower left block that is a diagonal block of unit
     vectors perpendular to the section so that a*(x'-x) = 0 can be imposed*)
   Do[
    piece = RotateRight[Join[aMat, ConstantArray[0, (n - 1) * 3]], (3 * i) - 3];
    AppendTo[lowerleft, piece],
    {i, 1, n}];
   LL = lowerleft;
   upperright = {};
   (*In the orange text, I create to upper right block of the matrix,
   a diagonal block of instaneous velociites of the each 'guess' on the poincare section*)
   Do[
    piece = RotateRight[Join[{{-guess[[i, 2]] - guess[[i, 3]]},
        {guess[[i, 1]] + a * guess[[i, 2]]},
        {b + guess[[i, 3]] * (guess[[i, 1]] - c)}},
       ConstantArray[0, {3, (n - 1)}], 2], {0, i - 1}];
    AppendTo[upperright, piece],
    {i, 1, n}];
   UR = Flatten[upperright, 1];
   LR = ConstantArray[0, {n, n}];
   M = Join[UL, UR, 2] ~ Join ~ Join[LL, LR, 2];
   Return[M];
  ];
```

Function 3 : FConstructRossler

This is a 1 X n*d matrix that calculates the error between the Poincaré crossing point of evolution of guess[i] and the location of guess[i+1] for each of the d dimensions.

In[291]:=
```
(*Function to create F matrix*)
FConstructRossler[guess_, sec_, n_] :=
  (*array of guesses, the numerically calculated section, cycle-length*)
  Module[{},
   F = {guess[[1]] - sec[[n]]};
   Do[
    diff = (guess[[i]] - sec[[i - 1]]);
    AppendTo[F, diff],
    {i, 2, n}
   ];
   AppendTo[F, ConstantArray[0, n]];
   Return[F = Flatten[F]];
  ];
```

Function 4 : VarConstruct

Make an array of variables to solve for. Because this is Mathematica I can leave these variables in symbol form.

In[292]:=
```
VarConstruct[n_] :=
  Module[{},
   (*Here I create the variables to be solved for, notice the extra +
    n accounting for the dummy dt's that allow us to solve n*d+n equations*)
   DM = ToExpression["d" <> ToString[#]] & /@ Range[3 * n + n];
   Return[DM];
  ];
```

Function 5 : MPSA

This is the fully implemented algorithm for the Rossler flow:

1. create section from initial guesses. Record deformation under the flow of the initial guesses.
2. Construct M matrix
3.Construct F (error) matrix
4. Construct variable matrix
5. Create n*d equations
6. Solve for n*d variables to find the change in initial guess. Use guess+=change as your new guess and repeat until convergence.

In[293]:=
```
(*Full Multipoint Shooting Algorithm*)
(*input is guess: initial guess, cl: cycle length, n: num interations*
  *normvec: vector normal to section, a, b and c: parameters for Rossler flow*)
MPSA[guess_, cl_, n_, damp_, normvec_, a_, b_, c_] := (*guess, cycle length,
  damping factorm,number of revisions,unit vector normal to the section,a,b,c*)
  Module[{},
   g = guess;
   Do[
    {jm, section} = cycleEstimateRossler[a, b, c, g];
    Mmat = MConstructRossler[cl, jm, g, normvec, a, b, c];
    Fmat = FConstructRossler[g, section, cl];
    Dmat = VarConstruct[cl];
    dotProd = Mmat.Dmat;
    eqs = Table[dotProd[[i]] == -Fmat[[i]], {i, Length[dotProd]}];
    deltaPos = Solve[eqs, DM]; (*this line is why Mathematica is nice.  I have
     used its symbolic capabilites to solve this set of equations.  In matlab
     one would have to due some actual inversions and stuff. see the book*)
    g += damp * Partition[Drop[Flatten[DM /. deltaPos], -cl], 3];
    Print[g],
    {n}];
  ];
```