Introduction to AUTO using the slow flow of Duffing's equation as an example system.

by Richard Rand
Originally written Oct. 2002
This version prepared Nov. 2005

Duffing's eq is:
x" + x + e c x' + e alpha x^3 = e F cos wt
Expand
w=1+k1 e
The slow flow is
2A' = - cA -2 k1 B + (3/4) alpha B (A^2+B^2)
2B' = -cB +2 k1 A - (3/4) alpha A (A^2+B^2) + F
We set c=.1 and alpha =1 throughout.
We vary k1 and F.

To begin with we use macsyma to obtain a nontrivial equilibrium:
We find F=.1, k=-.1, A=.31399145187, B=.110886466805

All of this stuff goes into the file "duf.c", a modified version of demo ab, file ab.c.
The relevant parts look like this:

```
int func (integer ndim, const doublereal *u, const integer *icp,
          const doublereal *par, integer ijac,
          doublereal *f, doublereal *dfdu, doublereal *dfdp) {
  doublereal a,b,k1,F;

  a = u[0];
  b = u[1];
  k1=par[0];
  F=par[1];


  f[0] = -.1*a-2*k1*b+.75*b*(a*a+b*b);
  f[1] = -.1*b+2*k1*a-.75*a*(a*a+b*b)+F;
  return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int stpnt (integer ndim, doublereal t,
           doublereal *u, doublereal *par) {

  /* Initialize the equation parameters */

  par[0] = (doublereal)-0.1;
  par[1] = (doublereal)0.1;

  /* Initialize the solution */
  u[0] = (doublereal)0.31399145187;
  u[1] = (doublereal)0.110886466805;

  return 0;
```

In addition to duf.c we prepare 4 files of constants (all modified versions of demo ab files):
c.du1 (varies k1 as k1 increases)
c.du2 (varies k1 as k1 decreases)
c.du3 (follows fold in k1 and F as k1 increases)

c.du4 (follows fold in k1 and F as k1 decreases)

The commands are:
cp c.du1 c.duf
@r duf
@sv duf
cp c.du2 c.duf
@r duf
@ap duf
cp c.du3 c.duf
@r duf
@sv foo
cp c.du4 c.duf
@r duf
@ap foo

This produces two graphic output files called duf and foo.
To view duf, type @p duf
Then in the graphic environment, type:
d1
bd0 (it defaults to k1 on the x axis and L2 norm of A,B on y axis)
sav
duf
Then return to unix and type @ps duf to get duf.ps.

To view foo, type @p foo
Then in the graphic environment, type:
ax
1 5 (this tells it to plot k1 on the x axis and F on the y axis)
d1
bd0
sav
foo
Then return to unix and type @ps foo to get foo.ps.

The file c.du1 looks like:

```
2 1 0 1           NDIM,IPS,IRS,ILP
1 0           NICP,(ICP(I),I=1 NICP)
50 4 3 1 1 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
0.01 0.005 0.05 1          DS,DSMIN,DSMAX,IADS
```

```
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

It produces the following screen display:

```
  BR    PT  TY LAB    PAR(0)        L2-NORM         U(1)          U(2)
   1     1  EP   1 -1.000000E-01  3.329962E-01  3.139915E-01  1.108865E-01
   1    94  LP   2  3.766742E-01  9.977447E-01 -6.697124E-02  9.954946E-01
   1   100       3  3.708405E-01  9.811430E-01 -1.896387E-01  9.626415E-01
   1   143  LP   4  1.741869E-01  4.187665E-01 -3.802793E-01  1.753654E-01
   1   166  EP   5  5.002012E-01  1.002111E-01 -9.970662E-02  1.004226E-02
```

The file c.du2 looks exactly like c.du1 except DS has the opposite sign. It gives:

```
  BR    PT  TY LAB    PAR(0)        L2-NORM         U(1)          U(2)
   2     1  EP   6 -1.000000E-01  3.329962E-01  3.139915E-01  1.108865E-01
   2    32  EP   7 -5.032141E-01  9.817611E-02  9.770183E-02  9.638549E-03
```

The file c.du3 looks like:

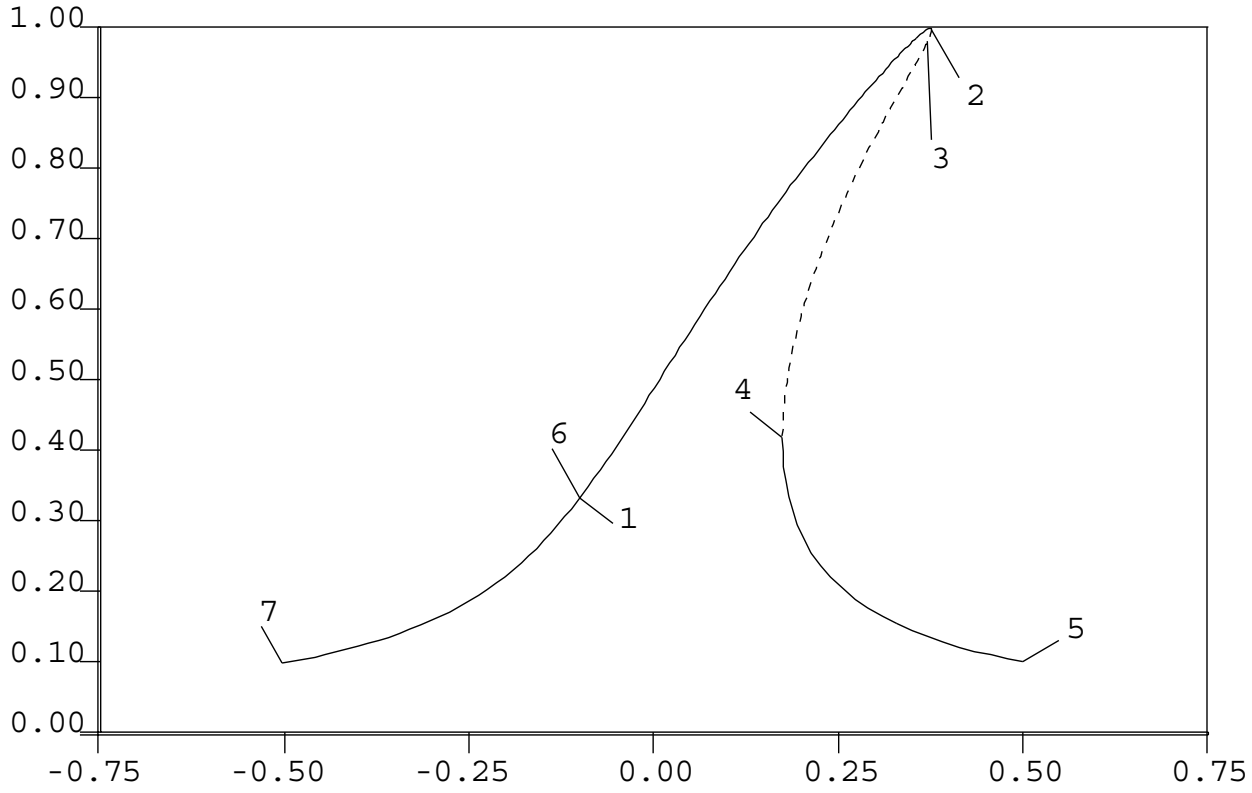
```
2 1 2 1           NDIM,IPS,IRS,ILP
2 0 1           NICP,(ICP(I),I=1 NICP)
50 4 3 0 2 0 0 0           NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0           NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0           NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001           EPSL,EPSU,EPSS
0.01 0.005 0.5 1           DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

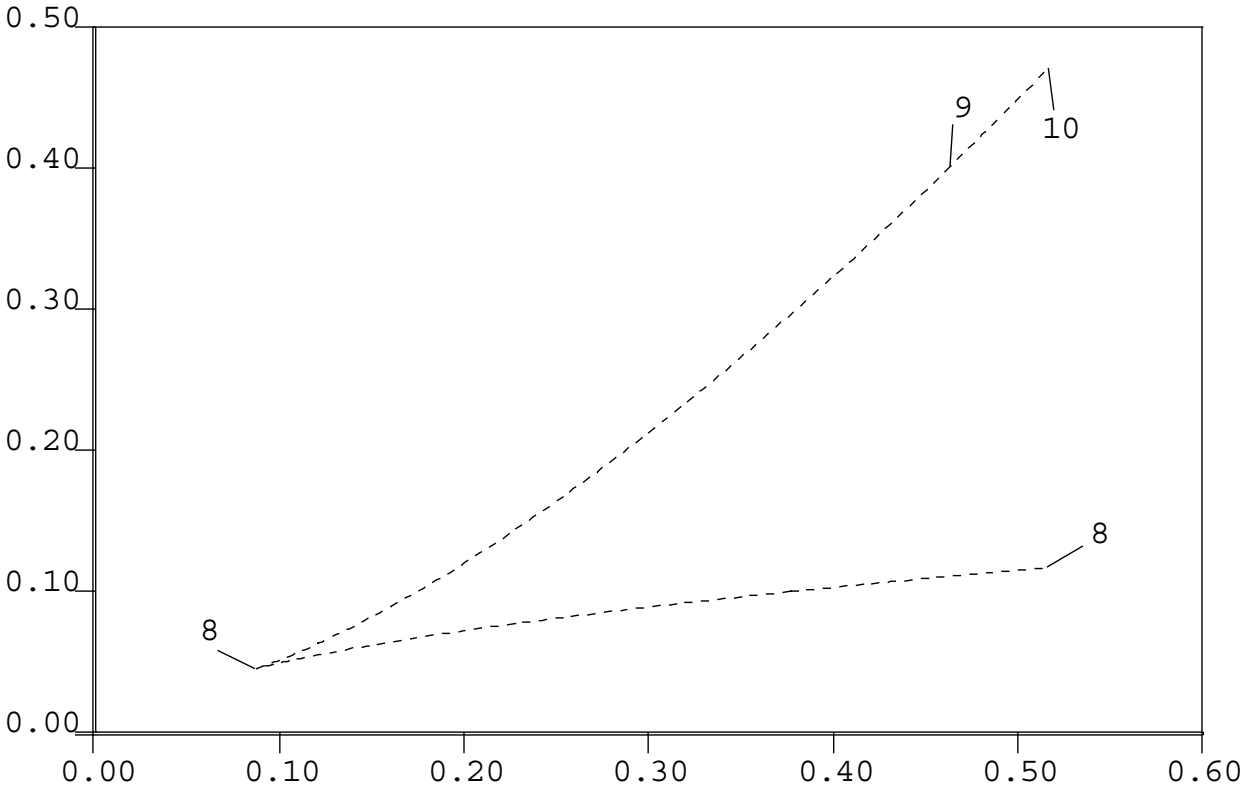
It produces the following screen display:

```
  BR    PT  TY LAB    PAR(0)        L2-NORM         U(1)          U(2)        PAR(1)
   2     9  EP   8  5.158720E-01  1.170107E+00 -5.704266E-02  1.168716E+00
1.171500E-01
```

The file c.du4 looks exactly like c.du3 except DS has the opposite sign. It gives:

```
  BR    PT  TY LAB    PAR(0)        L2-NORM         U(1)          U(2)        PAR(1)
   2    40  LP   8  8.660254E-02  3.923733E-01 -1.961929E-01  3.398017E-01
4.530785E-02
   2   100       9  4.629314E-01  6.471160E-01 -6.386409E-01  1.043883E-01
4.011551E-01
   2   103  EP  10  5.163022E-01  6.822298E-01 -6.750439E-01  9.875901E-02
4.712862E-01
```


1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
1
2
3
4
5
6
7


0.50
0.40
0.30
0.20
0.10
0.00
0.00
0.10
0.20
0.30
0.40
0.50
0.60
9
10
8
8

Setting up the constants files:

The file c.du1 looks like:

```
2 1 0 1           NDIM,IPS,IRS,ILP
1 0           NICP,(ICP(I),I=1 NICP)
50 4 3 1 1 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
0.01 0.005 0.05 1          DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

NDIM=2 is the no. of eq.'s to be solved (dim of phase space)
IPS=1 defines problem type. IPS=1 is for equilibria of ode's incl Hopfs.
IRS=0 means this is a new problem. IRS=m (m an integer) means restart at point labeled m.
ILP=1 means folds are to be detected. (ILP=0 for no detection of folds).
NICP=1 is the no. of parameters which are to be varied.
ICP=0 tells which are the parameters to be varied. Here it is par[0].
NTST,NCOL and IAD have to do with mesh discretization. Ignore for now.
ISP=1 means look for branch points of equilibria but not for periodic motions.
(ISP=2 means look for branch points of p.m.'s, including torus bifns.)
ISW=1 controls branch switching. ISW=2 is used to find locus of folds, Hopfs, etc.
IPLT=0 means use L2 norm. Changing this parameter redefines how the soln is measured.
NBC=0 is the no. of boundary conditions.
NINT=0 is the no. of integral constraints.
NMX=400 is the maximum of steps to be taken along any branch.
RL0=-0.5 lower bound for the prinicipal continuation parameter (first one in the ICP list).
RL1=0.5 upper bound for the prinicipal continuation parameter.
A0=0.0 lower bound on solution measure (= L2 norm if IPLT=0).
A1=100.0 upper bound on solution measure.
NPR=100 means print a labeled point out every 100 steps (even if it's not a fold, or Hopf, etc.)
to fort.8. (To have no such [meaningless] labels on your final plot, set NPR=0.)
MXBF=10 max. no. of bifns to be treated.
IID=2 regulates what information is printed in fort.9.
ITMX=8 max no. of iterations for some internal solver.
ITNW=5 ditto.
NWTN=3 ditto.
JAC=0 means obtain Jacobian b differencing. JAC=1 means user supplies derivatives.
EPSL,EPSU,EPSS convergence tolerances.

DS=0.01 stepsize along a branch. Automatically adapted if IADS>0 (see below).
DSMIN=0.005 minimum allowed DS.
DSMAX=0.05 max allowed DS.
IADS=1 no. of steps along branch after which DS is to be updated.
NTHL=1 no. of parameters not to be included in updating DS. To avoid mess ups due to infinte periods, type 10 0.0 on next line as shown. This means weight the period (=par[10]) as 0.0.
NTHU=0 same thing for state variables.
NUZR=0 if non-zero it controls which parameter values cause a point to be printed to fort.8.

WOW! These are alot of parameters. However, most of them need not be messed with unless there is a reason to do so, such as trouble in a run. The above file is basically the demo file ab.c.1 with some minor changes. One change is that AUTO 97 used to number parameters from 1 to 11, whereas AUTO 2000 numbers them 0 to 10. But the AUTO 2000 manual doesn't seem to know about this. So in line 2 of the above file we read 1 0, whereas it says 1 1 on p.61 of the manual. Similarly 3rd line from bottom says 10 0.0 whereas manual p.61 says 11 0. Another change is in line 4. I chose 400 points instead of 50 because I was not getting far enough along the branch. Naturally the limits RL0 and RL1 will be problem dependent.

Running the above constants is the first step to getting a bifurcation diagram.
The second step requires changing a few parameters in a predictable fashion.

The file c.du3 looks like:

```
2 1 2 1           NDIM,IPS,IRS,ILP
2 0 1           NICP,(ICP(I),I=1 NICP)
50 4 3 0 2 0 0 0           NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0           NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0           NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001           EPSL,EPSU,EPSS
0.01 0.005 0.5 1           DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

This file is based on the demo file ab.c.3.
What has been changed?
IRS=2 chosen because the point labeled 2 was a fold (=LP), and we want a locus of folds.
NICP=2 means 2 parameters are to be varied.
ICP= 0 1 means they are par[0] and par[1].
ISP=0 "This setting disables the detection of branch points..." OK, whatever.
ISW=2 In order to get locus of folds. See above.

If you were at a Hopf point (HB) and you wanted to get the locus of Hopf bifns, the demo file ab.c.5 would be a good one to use. The main changes required would be:
IRS=... use the label of the HB point.
NICP=2 as above.
ICP=m n where par[m] and par[n] are to be varied.

c.du1:

```
2 1 0 1           NDIM,IPS,IRS,ILP
1 0           NICP,(ICP(I),I=1 NICP)
50 4 3 1 1 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
0.01 0.005 0.05 1          DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

---

c.du2:

```
2 1 0 1           NDIM,IPS,IRS,ILP
1 0           NICP,(ICP(I),I=1 NICP)
50 4 3 1 1 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
-0.01 0.005 0.05 1          DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

---

c.du3:

```
2 1 2 1           NDIM,IPS,IRS,ILP
2 0 1          NICP,(ICP(I),I=1 NICP)
50 4 3 0 2 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
0.01 0.005 0.5 1          DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

---

c.du4:

```
2 1 2 1           NDIM,IPS,IRS,ILP
2 0 1          NICP,(ICP(I),I=1 NICP)
50 4 3 0 2 0 0 0          NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
400 -0.5 0.5 0.0 100.0          NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0          NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-06 1e-06 0.0001          EPSL,EPSU,EPSS
-0.01 0.005 0.5 1          DS,DSMIN,DSMAX,IADS
1          NTHL,(/,I,THL(I)),I=1,NTHL)
10 0.0
0          NTHU,(/,I,THU(I)),I=1,NTHU)
0          NUZR,(/,I,PAR(I)),I=1,NUZR)
```

The file duf.c:

```
#include "auto_f2c.h"
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/*   Duffing eq. slow flow  */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int func (integer ndim, const doublereal *u, const integer *icp,
          const doublereal *par, integer ijac,
          doublereal *f, doublereal *dfdu, doublereal *dfdp) {
  doublereal a,b,k1,F;

  /* Evaluates the algebraic equations or ODE right hand side */

  /* Input arguments : */
  /*      ndim   :   Dimension of the ODE system */
  /*      u      :   State variables */
  /*      icp    :   Array indicating the free parameter(s) */
  /*      par    :   Equation parameters */

  /* Values to be returned : */
  /*      f      :   ODE right hand side values */

  /* Normally unused Jacobian arguments : IJAC, DFDU, DFDP (see manual) */

  a = u[0];
  b = u[1];
  k1=par[0];
  F=par[1];


  f[0] = -.1*a-2*k1*b+.75*b*(a*a+b*b);
  f[1] = -.1*b+2*k1*a-.75*a*(a*a+b*b)+F;
  return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int stpnt (integer ndim, doublereal t,
           doublereal *u, doublereal *par) {
  /* Input arguments : */
  /*      ndim   :   Dimension of the ODE system */

  /* Values to be returned : */
  /*      u      :   A starting solution vector */
  /*      par    :   The corresponding equation-parameter values */
```

```
  /* Initialize the equation parameters */

  par[0] = (doublereal)-0.1;
  par[1] = (doublereal)0.1;

  /* Initialize the solution */
  u[0] = (doublereal)0.31399145187;
  u[1] = (doublereal)0.110886466805;

  return 0;
}
/* The following subroutines are not used here, */
/* but they must be supplied as dummy routines */

/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int bcnd (integer ndim, const doublereal *par, const integer *icp,
          integer nbc, const doublereal *u0, const doublereal *u1, integer ijac,
          doublereal *fb, doublereal *dbc) {
  return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int icnd (integer ndim, const doublereal *par, const integer *icp,
          integer nint, const doublereal *u, const doublereal *uold,
          const doublereal *udot, const doublereal *upold, integer ijac,
          doublereal *fi, doublereal *dint) {
    return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int fopt (integer ndim, const doublereal *u, const integer *icp,
          const doublereal *par, integer ijac,
          doublereal *fs, doublereal *dfdu, doublereal *dfdp) {
    return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
int pvls (integer ndim, const doublereal *u,
          doublereal *par) {
    return 0;
}
/* ---------------------------------------------------------------------- */
/* ---------------------------------------------------------------------- */
```