1

Maple Programs

M.1 Maple Programs for Semisimple Cases

The Maple programs (source code and sample input files) for computing the normal forms of semisimple cases are listed here (see Theorem 2.2).

M.1.1 Maple source code

The Maple code can be used for computing the normal form and nonlinear transformation of a given system associated with semisimple cases: the Jacobian of the system includes the combination of zero and pairs of pure imaginary eigenvalues. The Maple source code (named *program4*) and the input file (named *input4*) can be downloaded from the website *http://pyu1.apmaths.uwo.ca/~pyu/pub/software*.

```
#      This symbolic Maple program was developed to find the
#      Normal Forms of ordinary differential equations
#      associated with semi-simple cases: the Jacobian of the
#      system includes the combination of zero and pairs of pure
#      imaginary eigenvalues. The theory and methodology are
#      described in paper "Symbolic computation for normal forms
#      of differential equations" by Q. Bi and P. Yu, published in
#      the Journal of Computational and Applied Mathematics.


###    DEFINITION OF VARIABLES    ###
#      Eign =  the eigenvalues of the Jacobian
#      x    =  the variables of the original system in real form
#      v    =  the variables of the system in complex form
#      u    =  the variables of the normal form
#      d    =  the Frechet derivative
#      C    =  the coefficients of normal forms
#      F    =  normal forms
#      H    =  the coefficients of nonlinear transformations
#      T    =  nonlinear transformations
#      G    =  the jth order vector field
#      B    =  the coefficients of G
#      NF   =  output of normal form
#      NT   =  output of nonlinear transformation
with(linalg):
```

```
read input:                          # READ A PREPARED INPUT FILE
###   TRANSFORM THE REAL JORDAN FORM TO COMPLEX JORDAN FORM   ###
if  m1 <> 0 then
    for i from 1 to m1 do
        x[i]    := v[i]:
        f[i]    := f[i]:
        Eign[i] := 0:
    od:
fi:
if  m2 <> 0 then
    mm2 := m1 + 2*m2 - 1:
    for i from m1+1 by 2 to mm2 do
        x[i]      := (v[i] + v[i+1])/2:
        x[i+1]    := (v[i] - v[i+1])/2/I:
        temp      :=  f[i] + f[i+1]*I:
        f[i+1]    :=  f[i] - f[i+1]*I:
        f[i]      :=  temp:
        Eign[i]   :=  diff(f[i], v[i]):
        Eign[i+1] :=  diff(f[i+1], v[i+1]):
    od:
fi:
if  m3 <> 0 then
    for i from m1+2*m2+1 to m1+2*m2+m3 do
        x[i]    := v[i]:
        Eign[i] := diff(f[i], v[i]):
    od:
fi:
if  m4 <> 0 then
    for i from m1+2*m2+m3+1 by 2 to n-1 do
        x[i]      :=  (v[i] + v[i+1])/2:
        x[i+1]    :=  (v[i] - v[i+1])/2/I:
        temp      :=   f[i] + f[i+1]*I:
        f[i+1]    :=   f[i] - f[i+1]*I:
        f[i]      :=   temp:
        Eign[i]   :=   diff(f[i], v[i]):
        Eign[i+1] :=   diff(f[i+1], v[i+1]):
    od:
fi:
for i from 1 to n do
    for k from 1 to n do
        Eign[i] := subs(v[k]=0, Eign[i]):
    od:
od:
#### INITIALIZATION  #####
cm := m1+2*m2:
for i from 1 to n do
    T[i] := 0:
    F[i] := 0:
    d[i] := 0:
od:
#### RECURSIVE PROCEDURE TO COMPUTE THE jth ORDER VECTOR FIELD  ####
for j from 2 to norder do
    for k from 1 to n do
        G[k] := f[k]:
        for m from 1 to n do
            if  m < cm+1 then
                G[k] := subs(v[m]=u[m]+T[m], G[k]):
            else
                G[k] := subs(v[m]=T[m], G[k]):
            fi:
        od:
        G[k] := G[k] - d[k]:
        for m from 1 to cm do
            G[k] := subs(u[m]=epsilon*u[m],G[k]):
        od:
        G[k] := subs(epsilon=0,diff(G[k],epsilon$j)/j!):
        G[k] := expand(G[k]):
```

```
if  cm = 2 then
    for l from 0 to j do
        B[k,l,j-l] := coeff(G[k],u[1],l):
        B[k,l,j-l] := coeff(B[k,l,j-l],u[2],j-l):
        B[k,l,j-l] := subs(u[1]=0,u[2]=0,B[k,l,j-l]):
    od:
elif cm = 3 then
    for l from 0 to j do
        for m from 0 to j-l do
            J := j-l-m:
            B[k,l,m,J] := coeff(G[k],u[1],l):
            B[k,l,m,J] := coeff(B[k,l,m,J],u[2],m):
            B[k,l,m,J] := coeff(B[k,l,m,J],u[3],J):
            B[k,l,m,J] := subs(u[1]=0,u[2]=0,u[3]=0,B[k,l,m,J]):
        od:
    od:
elif cm = 4 then
    for l from 0 to j do
        for m from 0 to j-l do
            for p from 0 to j-l-m do
                J := j-l-m-p:
                B[k,l,m,p,J] := coeff(G[k],u[1],l):
                B[k,l,m,p,J] := coeff(B[k,l,m,p,J],u[2],m):
                B[k,l,m,p,J] := coeff(B[k,l,m,p,J],u[3],p):
                B[k,l,m,p,J] := coeff(B[k,l,m,p,J],u[4],J):
                B[k,l,m,p,J] := subs(u[1]=0,u[2]=0,u[3]=0,u[4]=0,
                                     B[k,l,m,p,J]):
            od:
        od:
    od:
elif cm = 5 then
    for l from 0 to j do
        for m from 0 to j-l do
            for p from 0 to j-l-m do
                for q from 0 to j-l-m do
                    J := j-l-m-p-q:
                    B[k,l,m,p,q,J] := coeff(G[k],u[1],l):
                    B[k,l,m,p,q,J] := coeff(B[k,l,m,p,q,J],u[2],m):
                    B[k,l,m,p,q,J] := coeff(B[k,l,m,p,q,J],u[3],p):
                    B[k,l,m,p,q,J] := coeff(B[k,l,m,p,q,J],u[4],q):
                    B[k,l,m,p,q,J] := coeff(B[k,l,m,p,q,J],u[5],J):
                    B[k,l,m,p,q,J] := subs(u[1]=0,u[2]=0,u[3]=0,u[4]=0,
                                           u[5]=0, B[k,l,m,p,q,J]):
                od:
            od:
        od:
    od:
elif cm = 6 then
    for l from 0 to j do
        for m from 0 to j-l do
            for p from 0 to j-l-m do
                for q from 0 to j-l-m do
                    for s from 0 to j-l-m do
                       J := j-l-m-p-q-s:
                       B[k,l,m,p,q,s,J]:= coeff(G[k],u[1],l):
                       B[k,l,m,p,q,s,J]:=coeff(B[k,l,m,p,q,s,J],u[2],m):
                       B[k,l,m,p,q,s,J]:=coeff(B[k,l,m,p,q,s,J],u[3],p):
                       B[k,l,m,p,q,s,J]:=coeff(B[k,l,m,p,q,s,J],u[4],q):
                       B[k,l,m,p,q,s,J]:=coeff(B[k,l,m,p,q,s,J],u[5],s):
                       B[k,l,m,p,q,s,J]:=coeff(B[k,l,m,p,q,s,J],u[6],J):
                       B[k,l,m,p,q,s,J]:=subs(u[1]=0,u[2]=0,u[3]=0,
                                 u[4]=0,u[5]=0,u[6]=0,B[k,l,m,p,q,s,J]):
                    od:
                od:
            od:
        od:
    od:
```

```
        fi:
    od:
##### COMPUTE THE jth ORDER NORMAL FORM AND NONLINEAR TRANSFORMATION  ####
    if  cm = 2 then
        for l from 0 to j do
            for k from 1 to cm do
                lambda0 := l*Eign[1]+(j-l)*Eign[2]-Eign[k]:
                if  lambda0 = 0 then
                    C[k,l,j-l] := B[k,l,j-l]:
                    H[k,l,j-l] := 0:
                else
                    H[k,l,j-l] := B[k,l,j-l]/lambda0:
                    C[k,l,j-l] := 0:
                fi:
            od:
            for k from cm+1 to n do
                lambda0    := l*Eign[1] + (j-l)*Eign[2] - Eign[k]:
                H[k,l,j-l] := B[k,l,j-l]/lambda0:
                C[k,l,j-l] := 0:
            od:
        od:
    elif cm = 3 then
        for l from 0 to j do
            for m from 0 to j-l do
                J := j-l-m:
                for k from 1 to cm do
                    lambda0 := l*Eign[1] + m*Eign[2] + J*Eign[3] - Eign[k]:
                    if  lambda0 = 0 then
                        C[k,l,m,J] := B[k,l,m,J]:
                        H[k,l,m,J] := 0:
                    else
                        H[k,l,m,J] := B[k,l,m,J]/lambda0:
                        C[k,l,m,J] := 0:
                    fi:
                od:
                for k from cm+1 to n do
                    lambda0    := l*Eign[1]+m*Eign[2]+J*Eign[3]-Eign[k]:
                    H[k,l,m,J] := B[k,l,m,J]/lambda0:
                    C[k,l,m,J] := 0:
                od:
            od:
        od:
    elif cm = 4 then
        for l from 0 to j do
            for m from 0 to j-l do
                for p from 0 to j-l-m do
                    J := j-l-m-p:
                    for k from 1 to cm do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                                  + J*Eign[4] - Eign[k]:
                        if  lambda0 = 0 then
                            C[k,l,m,p,J] := B[k,l,m,p,J]:
                            H[k,l,m,p,J] := 0:
                        else
                            H[k,l,m,p,J] := B[k,l,m,p,J]/lambda0:
                            C[k,l,m,p,J] := 0:
                        fi:
                    od:
                    for k from cm+1 to n do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                                  + J*Eign[4] - Eign[k]:
                        H[k,l,m,p,J] := B[k,l,m,p,J]/lambda0:
                        C[k,l,m,p,J] := 0:
                    od:
                od:
            od:
        od:
```

```
elif  cm = 5 then
    for l from 0 to j do
        for m from 0 to j-l do
            for p from 0 to j-l-m do
                for q from 0 to j-l-m-p do
                    J := j-l-m-p-q:
                    for k from 1 to cm do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                                  + q*Eign[4] + J*Eign[5]- Eign[k]:
                        if  lambda0 = 0 then
                            C[k,l,m,p,q,J] := B[k,l,m,p,q,J]:
                            H[k,l,m,p,q,J] := 0:
                        else
                            H[k,l,m,p,q,J] := B[k,l,m,p,q,J]/lambda0:
                            C[k,l,m,p,q,J] := 0:
                        fi:
                    od:
                    for k from cm+1 to n do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                                  + q*Eign[4] + J*Eign[5]- Eign[k]:
                        H[k,l,m,p,q,J] := B[k,l,m,p,q,J]/lambda0:
                        C[k,l,m,p,q,J] := 0:
                    od:
                od:
            od:
        od:
    od:
elif  cm = 6 then
    for l from 0 to j do
        for m from 0 to j-l do
            for p from 0 to j-l-m do
                for q from 0 to j-l-m-p do
                for r from 0 to j-l-m-p-q do
                    J := j-l-m-p-q-r:
                    for k from 1 to cm do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                          + q*Eign[4] + r*Eign[5] +J*Eign[6]- Eign[k]:
                        if  lambda0 = 0 then
                            C[k,l,m,p,q,r,J] := B[k,l,m,p,q,r,J]:
                            H[k,l,m,p,q,r,J] := 0:
                        else
                            H[k,l,m,p,q,r,J]:= B[k,l,m,p,q,r,J]/lambda0:
                            C[k,l,m,p,q,r,J]:= 0:
                        fi:
                    od:
                    if cm < n then
                    for k from cm+1 to n do
                        lambda0 :=  l*Eign[1] + m*Eign[2] + p*Eign[3]
                          + q*Eign[4] + r*Eign[5] +J*Eign[6]- Eign[k]:
                        H[k,l,m,p,q,r,J] := B[k,l,m,p,q,r,J]/lambda0:
                        C[k,l,m,p,q,r,J] := 0:
                    od:
                    fi:
                od:
                od:
            od:
        od:
    od:
fi:
for k from 1 to n do
    if  cm = 2 then
        for l from 0 to j do
            xs   := u[1]^l*u[2]^(j-l):
            T[k] := T[k] + H[k,l,j-l]*xs:
            F[k] := F[k] + C[k,l,j-l]*xs:
        od:
    elif cm = 3 then
```

```
            for l from 0 to j do
                for m from 0 to j-l do
                    J    := j-l-m:
                    xs   := u[1]^l*u[2]^m*u[3]^J:
                    T[k] := T[k] + H[k,l,m,J]*xs:
                    F[k] := F[k] + C[k,l,m,J]*xs:
                od:
            od:
        elif cm = 4 then
            for l from 0 to j do
                for m from 0 to j-l do
                    for p from 0 to j-l-m do
                        J    := j-l-m-p:
                        xs   := u[1]^l*u[2]^m*u[3]^p*u[4]^J:
                        T[k] := T[k] + H[k,l,m,p,J]*xs:
                        F[k] := F[k] + C[k,l,m,p,J]*xs:
                    od:
                od:
            od:
        elif cm = 5 then
            for l from 0 to j do
                for m from 0 to j-l do
                    for p from 0 to j-l-m do
                        for q from 0 to j-l-m-p do
                            J    := j-l-m-p-q:
                            xs   := u[1]^l*u[2]^m*u[3]^p*u[4]^q*u[5]^J:
                            T[k] := T[k] + H[k,l,m,p,q,J]*xs:
                            F[k] := F[k] + C[k,l,m,p,q,J]*xs:
                        od:
                    od:
                od:
            od:
        elif cm = 6 then
            for l from 0 to j do
                for m from 0 to j-l do
                    for p from 0 to j-l-m do
                        for q from 0 to j-l-m-p do
                            for r from 0 to j-l-m-p-q do
                                J := j-l-m-p-q-r:
                                xs:= u[1]^l*u[2]^m*u[3]^p*u[4]^q*u[5]^r*u[6]^J:
                                T[k] := T[k] + H[k,l,m,p,q,r,J]*xs:
                                F[k] := F[k] + C[k,l,m,p,q,r,J]*xs:
                            od:
                        od:
                    od:
                od:
            od:
        fi:
    od:
#### TO COMPUTE THE FRECHET DERIVATIVES  ####
    for i from 1 to n do
        d[i] := 0:
        if  i < cm+1 then
            TT[i] := u[i] + T[i]:
        else
            TT[i] := T[i]:
        fi:
        for l from 1 to cm do
            d[i]:= d[i] + diff(TT[i],u[l])*F[l]:
        od:
    od:
od:
#####  TRANSFORM BACK TO SYSTEM IN REAL FORM  ####
for i from 1 to n do
    if  i < cm+1 then
        F[i] := Eign[i]*u[i] + F[i]:
        T[i] := u[i] + T[i]:
```

```
    else
        T[i] :=T[i]:
    fi:
    TT[i] := T[i]:
od:
if  m1 <> 0 then
    for i from 1 to m1 do
        u[i] := y[i]:
    od:
fi:
if  m2 <> 0 then
    for i from m1+1 by 2 to mm2 do
        u[i]   := y[i] + I*y[i+1]:
        u[i+1] := y[i] - I*y[i+1]:
    od:
    for i from m1+1 by 2 to mm2 do
        temp1  := (F[i] + F[i+1])/2:
        temp2  := (T[i] + T[i+1])/2:
        F[i+1] := (F[i] - F[i+1])/2/I:
        T[i+1] := (T[i] - T[i+1])/2/I:
        F[i]   := temp1:
        T[i]   := temp2:
    od:
fi:
if  m1 <> 0 then
    for i from 1 to m1 do
        T[i] := T[i]:
        F[i] := F[i]:
    od:
fi:
if  m3 <> 0 then
    for i from m1+2*m2+1 + 2 to m1+2*m2+m3 do
        T[i] := T[i]:
    od:
fi:
if m4 <> 0 then
    for i from m1+2*m2+m3+1 by 2 to n-1 do
        temp   := (T[i] + T[i+1])/2:
        T[i+1] := (T[i] - T[i+1])/2/I:
        T[i]   := temp:
    od:
fi:
F := simplify(F):
save F,`NF`;              #  OUTPUT OF NORMAL FORM
save T,`NT`;              #  OUTPUT OF NONLINEAR TRANSFORMATION
```

M.1.2 Sample input file

A sample input file is given below. The readers can prepare their own input file by modifying the values of several parameters and the vector field. The modification is straightforward.

```
#             d x_1 / dt = f_1
#             d x_2 / dt = f_2
#                      ...
#             d x_n / dt = f_n
#
#      The functions, f_i, must be put in the form such that the
#      Jacobian of the system is in block real Jordan canonical form:
#
#                 [     |     |     ]
#                 [ J_0 |     |     ]
#                 [ - - | - - | - - ]
#          J  =   [     | J_1 |     ]
```

```
#                [ - - | - - | - - ]
#                [     |     | J_2 ]
#                [     |     |     ]
#
#      where  J_0  represents the part of eigenvalues with zero real parts.
#      J_1  includes the part of real eigenvalues, and  J_2  has the
#      part of complex conjugate eigenvalues, which must appear in pairs.
#      For the following examples,
#
#                                      [ -1  1 ]
#          J_1  =  [ -1 ]      J_2  =  [       ]
#                                      [ -1 -1 ]
###    DEFINITION OF VARIABLES    ###
#      m1     = number of zero eigenvalues
#      m2     = number of pairs of purely imaginay eigenvalues
#      m3     = number of non-zero real eigenvalues
#      m4     = number of pairs of complex conjugate eigenvalues
#      n      = the dimension of the system
#      norder = the order of normal forms to be computed
#      cc     = the dimension of center manifold
#      f[i]   = the ith component of the vector field of the system
###    INPUT DATA FILE STARTS HERE   ###
m1     :=  0:
m2     :=  3:
m3     :=  1:
m4     :=  1:
n      :=  m1+2*m2+m3+2*m4:
norder :=  5:
cc     :=  m1+2*m2:
####   A purely imaginary pair   ###
if   cc = 2  then
     f[1] :=   x[2] + x[1]^2 - x[1]*x[3]:
     f[2] := - x[1] + x[2]^2 + x[1]*x[4] + x[2]^3:
     f[3] := - x[3] + x[1]^2:
     f[4] := - x[4] + x[5] + x[1]^2:
     f[5] := - x[4] - x[5] + x[2]^2:

####   A simple zero and a purely imaginary pair   ###
elif cc = 3  then
     f[1] := - (x[1] -  x[2] - x[4])^2:
     f[2] :=   x[3] - (x[1] - x[2] + x[5] )^2:
     f[3] := - x[2] - (x[2] - x[3] + x[4] )^2:
     f[4] := - x[4] + (x[1] - x[5])^2:
     f[5] := - x[5] + x[6] + (x[1] - x[4] )^2:
     f[6] := - x[5] - x[6] + (x[2] + x[5] )^2:
####   Two purely imaginary pairs (non-resonance)   ###
elif cc = 4  then
     f[1] :=   x[2] + x[1]^3 - x[1]^2*x[5] + x[1]^2*x[7]:
     f[2] := - x[1] - 2*x[1]*x[3]^2 :
     f[3] :=   sqrt(2)*x[4] + x[1]^2*x[3] - 4*x[5]^3:
     f[4] := - sqrt(2)*x[3]:
     f[5] := - x[5] + (x[1]-x[5])^2:
     f[6] := - x[6] + x[7] + (x[1] - x[4] )^2:
     f[7] := - x[6] - x[7] + (x[2] - x[6] )^2:
####   Two purely imaginary pairs (resonance)   ###
elif cc = 4  then
     f[1] :=   x[2] + x[1]^3 - 2*x[1]*x[3]^2 - x[1]^2*x[5] + 5*x[2]*x[6]^2:
     f[2] := - x[1] + x[3]^3 - 2*x[2]^2*x[4] + x[1]*x[5]*x[6]:
     f[3] :=   x[4] + x[1]^2*x[3] + 3*x[1]^2*x[4]:
     f[4] := - x[3] + 4*x[3]^3  - x[3]^2*x[4]:
     f[5] := - x[5] + (x[1]-x[5])^2:
     f[6] := - x[6] + x[7] + (x[1] - x[4] )^2:
     f[7] := - x[6] - x[7] + (x[2] - x[6] )^2:
####   Three purely imaginary pairs (resonance)   ###
elif cc = 6  then
     f[1] :=   x[2]:
     f[2] := - x[1] + x[7]^2 +  x[1]*x[5]:
```

```
      f[3] :=   2*x[4]:
      f[4] := - 2*x[3] + (x[1] - x[5])^2 + x[1]^2*x[7]:
      f[5] :=   3*x[6]:
      f[6] := - 3*x[5] + (x[1] - x[5])^2 - x[3]^2*x[9]:
      f[7] := - x[7] + (x[1] - x[5])^2:
      f[8] := - x[8] + x[9] + x[1]^2:
      f[9] := - x[8] - x[9] + x[2]^2:
fi:
```

M.2 Maple Programs for Hopf Bifurcation

The Maple programs (source code and sample input files) for computing the normal forms associated with Hopf and generalized Hopf bifurcations are given below.

M.2.1 Maple source code

The Maple code and sample input file can be downloaded from the web site *http://pyu1.apmaths.uwo.ca/~pyu/pub/software*. File names are *program 1* and *input 1*.

```
#  This symbolic Maple program was developed to find the Normal Forms
#  of Hopf bifurcations of an n-dim. vector filed, based on
#  the theory and method described in paper
#  "Computation of normal forms via a perturbation technique" by P. Yu,
#  published in J. Sound & Vib., v. 211(1), pp. 19-38, 1998.
with(linalg):
read input:            # Read a prepared input file.
# The procedure "solution" calculates the coefficients of the normal forms
# and periodic solutions at each perturbation step.
Solve := proc(k,Y,X)
   local XX:
   if  k=1 then
       if  coeff(Y,X)<>0 then
           XX := solve(Y,X):
       else
           XX := 0:
       fi:
   else
       XX := Y:
   fi:
   XX := rationalize(XX):
   XX := expand(XX):
   X  := XX:
end:
solution := proc(n)
   local  i, j, k, A, B, C, D, E, f, myA, myb, temp:
   global Ord, func, F1, F2, FF, X, Dr, Dphi, DDr:
# Solve for solution  x_n1.
   for i from 1 to N do
       f[i] := 0:
       for j from 1 to n do
           f[i] := f[i]
   + combine(subs(eps=0,diff(FF[i,j],eps$(n+1-j))/(n+1-j)!),trig):
       od:
       f[i] := f[i]+ combine(subs(eps=0,FF[i,n+1]),trig):
   od:
   for i from 1 to N do
```

```
        for j from 1 to n do
            f[i] := f[i]-(r^(j+1)*Dr[j]*diff(X[i,n-j],r)
                + r^j*Dphi[j]*combine(diff(X[i,n-j],theta),trig)):
        od:
    od:
    X[1,n] := A[0]:
    for i from 2 to n+1 do
        X[1,n] := X[1,n] + A[i]*cos(i*theta) + B[i]*sin(i*theta):
    od:
    X[1,n] := r^(n+1)*X[1,n]:
    F1 := diff(X[1,n],theta$2) + X[1,n] - diff(f[1],theta) - f[2]:
# Solve for the coefficients of the normal forms, D_n a  and  D_n phi.
    Solve(1, coeff(F1, sin(theta)), Dr[n]):
    Solve(1, coeff(F1, cos(theta)), Dphi[n]):
    for i from 2 to n+1 do
        Solve(1, coeff(F1, cos(i*theta)), A[i]):
        Solve(1, coeff(F1, sin(i*theta)), B[i]):
    od:
    Solve(1, F1, A[0]):
# Find solution x_n2.
    X[1,n] := X[1,n]:
    X[2,n] := diff(X[1,n], theta) - f[1]:
# Solve for solutions  x_np, p = 3, 4, ... M1+2.
    if M1 <> 0 then
       for i from 3 to M1+2 do
           A := 'A':
           B := 'B':
           X[i,n] := A[0]:
           for j from 1 to n+1 do
               X[i,n] := X[i,n] + A[j]*cos(j*theta) + B[j]*sin(j*theta):
           od:
           F1 := combine(diff(X[i,n], theta) - f[i], trig):
           for j from 1 to n+1 do
               for k from 1 to 2 do
                   if   k=1 then
                        temp := coeff(F1, cos(j*theta)):
                   elif k=2 then
                        temp := coeff(F1, sin(j*theta)):
                   fi:
                   C[k,1] := coeff(temp, A[j]):
                   C[k,2] := coeff(temp, B[j]):
                   C[k,3] := C[k,1]*A[j] + C[k,2]*B[j] - temp:
               od:
               myA  := array([[C[1,1], C[1,2]], [C[2,1], C[2,2]]]):
               myb  := array( [C[1,3], C[2,3]]):
               myb  := linsolve(myA, myb):
               Solve(2, myb[1], A[j]):
               Solve(2, myb[2], B[j]):
           od:
           F1 := combine(F1, trig):
           if F1 <> 0 then
              Solve(1, F1, A[0]):
           fi:
           X[i,n] := X[i,n]:
       od:
    fi:
# Solve for solutions  x_nq  and  x_n(q+1), q = M1+3, M1+5, ... N-1.
    if M2 <> 0 then
       for i from M1+3 by 2 to N do
           A := 'A':
           B := 'B':
           X[i,n]   := A[0]:
           X[i+1,n] := D[0]:
           for j from 1 to n+1 do
               X[i,n]   := X[i,n]   + A[j]*cos(j*theta) + B[j]*sin(j*theta):
               X[i+1,n] := X[i+1,n] + D[j]*cos(j*theta) + E[j]*sin(j*theta):
           od:
```

```
          F1:= combine(diff(X[i,n],theta) - f[i], trig):
          F2:= combine(diff(X[i+1,n],theta) - f[i+1], trig):
          for j from 1 to n+1 do
              for k from 1 to 4 do
                  if   k=1 then
                       temp := coeff(F1, cos(j*theta)):
                  elif k=2 then
                       temp := coeff(F1, sin(j*theta)):
                  elif k=3 then
                       temp := coeff(F2, cos(j*theta)):
                  elif k=4 then
                       temp := coeff(F2, sin(j*theta)):
                  fi:
                  C[k,1] := coeff(temp, A[j]):
                  C[k,2] := coeff(temp, B[j]):
                  C[k,3] := coeff(temp, D[j]):
                  C[k,4] := coeff(temp, E[j]):
                  C[k,5] := C[k,1]*A[j] + C[k,2]*B[j]
                            +C[k,3]*D[j] + C[k,4]*E[j] - temp:
              od:
              myA  := array([[C[1,1], C[1,2], C[1,3], C[1,4]],
                             [C[2,1], C[2,2], C[2,3], C[2,4]],
                             [C[3,1], C[3,2], C[3,3], C[3,4]],
                             [C[4,1], C[4,2], C[4,3], C[4,4]]]):
              myb  := array( [C[1,5], C[2,5], C[3,5], C[4,5]] ):
              myb  := linsolve(myA, myb):
              Solve(2, myb[1], A[j]):
              Solve(2, myb[2], B[j]):
              Solve(2, myb[3], D[j]):
              Solve(2, myb[4], E[j]):
          od:
          F1 := combine(F1, trig):
          if F1 <> 0 then
             Solve(1, F1, A[0]):
          fi:
          F2 := combine(F2, trig):
          if F2 <> 0 then
             Solve(1, F2, D[0]):
          fi:
      od:
   fi:
end:
for i from 1 to N do
    func[i] := expand(func[i]/omega):
od:
func[1] := func[1]-x[2]:
func[2] := func[2]+x[1]:
for i from 1 to N do
    for j from 1 to N do
        func[i] := subs(x[j]=eps*x[j],func[i]):
    od:
    for j from 1 to Ord+1 do
        FF[i,j] := subs(eps=0,diff(func[i],eps$j)/j!):
    od:
od:
# Main program; write the formal ordered perturbation solution  x_i.
for i from 1 to N do
    x[i] := 0:
    for j from 0 to Ord do
        x[i] := x[i] + X[i,j]*eps^j:
    od:
od:
# Zero order solution  x_0.
X[1,0] :=   r*cos(theta);
X[2,0] := - r*sin(theta);
if N > 2 then
   for i from 3 to N do
```

```
       X[i,0] := 0:
   od:
fi:
# Call procedure "solution" to obtain the asymptotic solutions and the
# coefficients of the normal forms for the i-step perturbation equations.
for i from 1 to Ord do
    solution(i):
    print(` Order = `, i):
    for j from 1 to i do
        DDr[j]  := simplify(subs(r=1,omega*Dr[j])):
        Dphi[j] := simplify(subs(r=1,omega*Dphi[j])):
    od:
od:
for i from 1 to N do
    x[i] := x[i]:
od:
save Dr, Dphi, output:       # Store results in the file "output"
quit:
```

M.2.2 Sample input file

Two input files are listed here, one for Example 2.6 and the one for the double pendulum example discussed in Section 5.2 (see Eq. (5.20)).

For Example 2.6:

```
M1      :=   1:                 #  No. of non-zero real eigenvalues
M2      :=   1:                 #  No. of complex conjugate eigenvalues
N       :=   2 + M1 + M2*2:
Ord     :=   5:
omega   := 1:
func[1] := x[2]+x[1]^2-x[1]*x[3];
func[2] :=-x[1]+x[2]^2+x[1]*x[4]+x[2]^3;
func[3] :=-x[3]+x[1]^2;
func[4] :=-x[4]+x[5]+x[1]^2;
func[5] :=-x[4]-x[5]+x[2]^2;
```

For the Double Pendulum Example (see Eq. (5.20)):

```
M1      := 2:                #  No. of non-zero real eigenvalues
M2      := 0:                #  No. of complex conjugate eigenvalues
N       := 2 + M1 + M2*2:
Ord     := 5:
omega   := 1:
func[1] := x[2]-2659/3250*x[1]*x[2]*x[4]-157/325*x[1]*x[2]*x[3]
          +4543/1300*x[2]*x[3]*x[4]-1713/650*x[1]*x[3]*x[4]-89/2600*x[1]^2*x[3]
          -439/520*x[1]*x[3]^2+153/130*x[2]*x[3]^2-53/325*x[2]^2*x[1]
          +168/325*x[2]^2*x[3]+3996/1625*x[2]*x[4]^2+3/650*x[1]^2*x[2]
          +87/1625*x[1]^2*x[4]+3503/520*x[3]^2*x[4]-12723/6500*x[1]*x[4]^2
          +9019/13000*x[2]^2*x[4]+2383/260*x[3]*x[4]^2+253/7800*x[1]^3
          +67/975*x[2]^3+503/312*x[3]^3+48919/12000*x[4]^3;
func[2] :=-x[1]+2097/3250*x[1]*x[2]*x[4]+151/325*x[1]*x[2]*x[3]
          -2869/1300*x[2]*x[3]*x[4]+379/650*x[1]*x[3]*x[4]-1173/2600*x[1]^2*x[3]
          +197/520*x[1]*x[3]^2-119/130*x[2]*x[3]^2+47/325*x[2]^2*x[1]
          -174/325*x[2]^2*x[3]-2133/1625*x[2]*x[4]^2+119/650*x[1]^2*x[2]
          -1021/1625*x[1]^2*x[4]+2331/520*x[3]^2*x[4]+1229/6500*x[1]*x[4]^2
          -8777/13000*x[2]^2*x[4]+1567/260*x[3]*x[4]^2+1577/7800*x[1]^3
          -11/195*x[2]^3+331/312*x[3]^3+31403/12000*x[4]^3;
func[3] :=-x[3]+881/250*x[1]*x[2]*x[4]+47/25*x[1]*x[2]*x[3]
          -1637/100*x[2]*x[3]*x[4]+747/50*x[1]*x[3]*x[4]+259/200*x[1]^2*x[3]
          +173/40*x[1]*x[3]^2-51/10*x[2]*x[3]^2+83/125*x[2]^2*x[1]-48/25*x[2]^2*x[3]
```

```
          -1512/125*x[2]*x[4]^2-117/250*x[1]^2*x[2]+147/125*x[1]^2*x[4]
          -2101/40*x[3]^2*x[4]+5841/500*x[1]*x[4]^2-2681/1000*x[2]^2*x[4]
          -7129/100*x[3]*x[4]^2-2059/3000*x[1]^3-109/375*x[2]^3
          -301/24*x[3]^3-75733/2400*x[4]^3;
func[4] :=-3/2*x[4]-1176/325*x[1]*x[2]*x[4]-128/65*x[1]*x[2]*x[3]
          +1076/65*x[2]*x[3]*x[4]-952/65*x[1]*x[3]*x[4]-72/65*x[1]^2*x[3]
          -56/13*x[1]*x[3]^2+68/13*x[2]*x[3]^2-224/325*x[2]^2*x[1]+132/65*x[2]^2*x[3]
          +3942/325*x[2]*x[4]^2+128/325*x[1]^2*x[2]-304/325*x[1]^2*x[4]
          +642/13*x[3]^2*x[4]-3694/325*x[1]*x[4]^2+914/325*x[2]^2*x[4]
          +4358/65*x[3]*x[4]^2+584/975*x[1]^3+292/975*x[2]^3
          +460/39*x[3]^3+2227/75*x[4]^3;
```