Bifurcation diagram for damped, driven pendulum.

Author: David Craig <http://web.lemoyne.edu/~craigda/>
Version: 1.0 [5-2-06]

To do:
(i) Speed up bifurcation diagram?
(ii) Improve procedure for resetting angle x mod 2Pi – in particular, eliminate "connectives" at jumps [plot as multiple plots over [-Pi+delta,+Pi-delta], say?]
(iii) Improve titles/legends

```
> restart;
  with(plots):
```

```
Warning, the name changecoords has been redefined
```

```
> Digits := 15;    # numerical precision [default value is 10]; larger is
  slower
```

$$Digits := 15$$

> Damped driven oscillator parameters:

```
# default values as in Taylor, chapter 12
w0 := (3/2)*w;     # natural frequency;
b := w0/4;         # damping parameter  [Taylor's beta]
g := 1.07;          # driving parameter  [Taylor's gamma]
w := 2*Pi;         # driving frequency
T := 2*Pi/w;       # driving period
```

x - angle; y - angular velocity; z - wt
Equations of motion in first order form; note eq2 requires specification of driving parameter g as in eq2(g) :

```
eq1 := diff( x(t),t ) = y(t);
eq2 := g -> diff( y(t),t ) = -2*b*y(t) - w0^2*sin(x(t)) +
g*w0^2*cos(z(t));
eq3 := diff( z(t),t ) = w;
```

Initial conditions

```
x0 := -Pi/2;
y0 := 0;
z0 := 0;
ic := x(0)=x0,y(0)=y0,z(0)=z0;
```

list of parameter values for title/legend

```
plist :=  cat("w=",convert(w,string),"    ", "T=",convert(T,string),"
",
              "w0=",convert(w0,string),"    ", "b=",convert(b,string),"
",
              "g=",convert(g,string),  "\n",
```

```
                "x0=",convert(x0,string),"    ", "y0=",convert(y0,string) )
 ;
# options for appearance of plots
plotopts := 'axes=BOXED,title=plist,titlefont=[HELVETICA,16]' ;
```

Numerical solution with initial conditions ic as a function of driving parameter g [consider method=gear at some point]
(The option maxfun=0 eliminates the errors that throw when you try to evaluate to large t [> 90, often]. See Help for dsolve,maxfun.)

```
# Making g an argument seems to slow down other plots a bit?
# Only use this approach for bifurcation diagram, in which we need to vary
g at each step.
numsol := g -> dsolve(
[eq1,eq2(g),eq3,ic],[x(t),y(t),z(t)],numeric,maxfun=0 );
```

$$w0 := 3\,\pi$$

$$b := \frac{3}{4}\,\pi$$

$$g := 1.07$$

$$w := 2\,\pi$$

$$T := 1$$

$$eq1 := \frac{d}{dt}\,x(t) = y(t)$$

$$eq2 := g \rightarrow \frac{d}{dt}\,y(t) = -2\,b\,y(t) - w0^{2}\,\sin(x(t)) + g\,w0^{2}\,\cos(z(t))$$

$$eq3 := \frac{d}{dt}\,z(t) = 2\,\pi$$

$$x0 := -\frac{1}{2}\,\pi$$

$$y0 := 0$$

$$z0 := 0$$

$$ic := x(0) = -\frac{1}{2}\,\pi,\ y(0) = 0,\ z(0) = 0$$

$$plist := "w=2*Pi \quad T=1 \quad w0=3*Pi \quad b=3/4*Pi \quad g=1.07 \quad x0=-1/2*Pi \quad y0=0"$$

$$plotopts := axes = BOXED,\ title = plist,\ titlefont = [HELVETICA,\ 16]$$

$$numsol := g \rightarrow dsolve([eq1,\ eq2(g),\ eq3,\ ic],\ [x(t),\ y(t),\ z(t)],\ numeric,\ maxfun = 0)$$

> Output of dsolve is a procedure that numerically evaluates the functions [x(t),y(t),z(t)] at the time specified by its argument

```
numsol(g)(1);     # for example, this is a list of the values of those
functions at t=1 for the current value of g
numsol(g)(1)[3];  # this is how you access the list results individually -
```

```
note each list entry is an EQUATION
# (see the help for "dsolve,numeric" for details and instructions on
options to change this)
# this is how you get the actual numerical value of each result - rhs()
extracts the right side of an equation
rhs( numsol(g)(1)[3] ) ;
```

> Resets x to remain in the range -Pi..Pi [i.e. x mod 2Pi]

```
# subtracts/adds the appropriate multiple of 2Pi when x increases past
Pi/decreases past -Pi
mod2Pi :=  x -> x - 2*Pi*floor( (x+Pi)/(2*Pi) )  :
```

> Angle vs. time

```
odeplot( numsol(g), [t,x(t)],50..60,plotopts,numpoints=400);
# same plot but with x mod 2Pi
# odeplot( numsol(g), [t,mod2Pi(x(t))],25..30,plotopts,numpoints=400);  #
plots x mod2Pi
```

> Animate trajectory ; only execute when you need it as this is resource intensive

```
odeplot( numsol(g), [t,x(t)],0..15,plotopts,frames=400);
```

> Phase-space trajectory

```
odeplot( numsol(g), [x(t),y(t)],0..6,plotopts,numpoints=400);
# same plot but with x mod 2Pi
# odeplot( numsol(g), [mod2Pi(x(t)),y(t)],10..30,plotopts,numpoints=400);
```

> Animate phase trajectory ; only execute when you need it as this is QUITE resource intensive - and sometimes twitchy

```
odeplot( numsol(g), [x(t),y(t)],0..30,plotopts,frames=400);
```

> phase-space trajectory in time [3D]

```
odeplot( numsol(g), [x(t),y(t),z(t)],0..10,plotopts,numpoints=400);
```

> Poincare section

[NB: Maple also has its own built-in tools for Poincare sections!]

```
N := 30;                # number of periods to plot
ni := 10;               # starting period of section
nf := ni + N-1;         # ending period of section
ti := ni*T ;            # starting time of section
tf := nf*T;             # ending time of section
wx := 25;               # square plot window bounds – x
wv := 25;               # square plot window bounds – v
pwsq := view=[-v..v,-v..v] ;        # square plot window
pwrct := view=[-wx..wx,-wv..wv] ;   # rectangular plot window
pwot := view=[-.5..0.5,-15..25] ;       # other plot window
# numpoints = N ensures a point is plotted just once per driving period
odeplot( numsol(g),
[x(t),y(t)],ti..tf,plotopts,style=point,symbol=circle,symbolsize=5,numpoin
```

```
ts=N,pwot);
# same plot but with x mod 2Pi
# odeplot( numsol(g),
[mod2Pi(x(t)),y(t)],ti..tf,plotopts,style=point,symbol=circle,symbolsize=5
,numpoints=N,pwot);
```

> Animate Poincare section ; only execute when you need it as this is resource intensive – and twitchy?

```
# animation shows additional points at early times I don't yet understand
odeplot( numsol(g),
[x(t),y(t)],ti..tf,plotopts,style=point,symbol=circle,symbolsize=5,frames=
N,pwrct);
```

> Poincare section – brute force version

```
# may be better if more elegant approach complains about maxfun
N := 30;                # number of periods to plot
ni := 20;               # starting period of section
# sequence of points consisting of [x,y] pairs at times ni*T to (ni+N)*T;
output suppressed
sectionpoints := { seq( [ rhs(numsol(g)(ni*T + i*T)[2]) ,
rhs(numsol(g)(ni*T + i*T)[3]) ], i=0..N-1) }:
# same sequence but with x mod 2Pi
# sectionpoints := { seq( [ mod2Pi(rhs(numsol(g)(ni*T + i*T)[2])) ,
rhs(numsol(g)(ni*T + i*T)[3]) ], i=0..N-1) }:
pointplot( sectionpoints ,symbol=circle,symbolsize=3, plotopts, pwrct);
```

> Bifurcation diagram – brute force version

```
# Requires maxfun=0 option in numsol in order to generate a result without
complaints.
N := 100;               # number of periods to plot at each value of g
ni := 500;               # starting period
gf := 1.0870;           # final g
gi := 1.06;             # initial g
d := .0001;             # increment
pwg := view=[1.06..1.09,-0.7..0.0] ;   # bifurcation diagram plot window;
[1.06..1.09,-0.7..0.0] works for g=1.07
gvals := seq( gi+i*d, i=0..270) :    # sequence of values of g
# sequence of points consisting of [g,x(t)] pairs at times ni*T to
(ni+N)*T; output suppressed
bfpoints := { seq(  seq( [g, rhs(numsol(g)(ni*T + i*T)[2])], g=gvals),
i=0..N-1) }:
# same sequence but with x mod 2Pi
# bfpoints := { seq(  seq( [g, mod2Pi(rhs(numsol(g)(ni*T + i*T)[2]))],
g=gvals),  i=0..N-1) }:
pointplot( bfpoints ,symbol=circle,symbolsize=1, plotopts, pwg);
```

$$N := 100$$

$$ni := 40$$

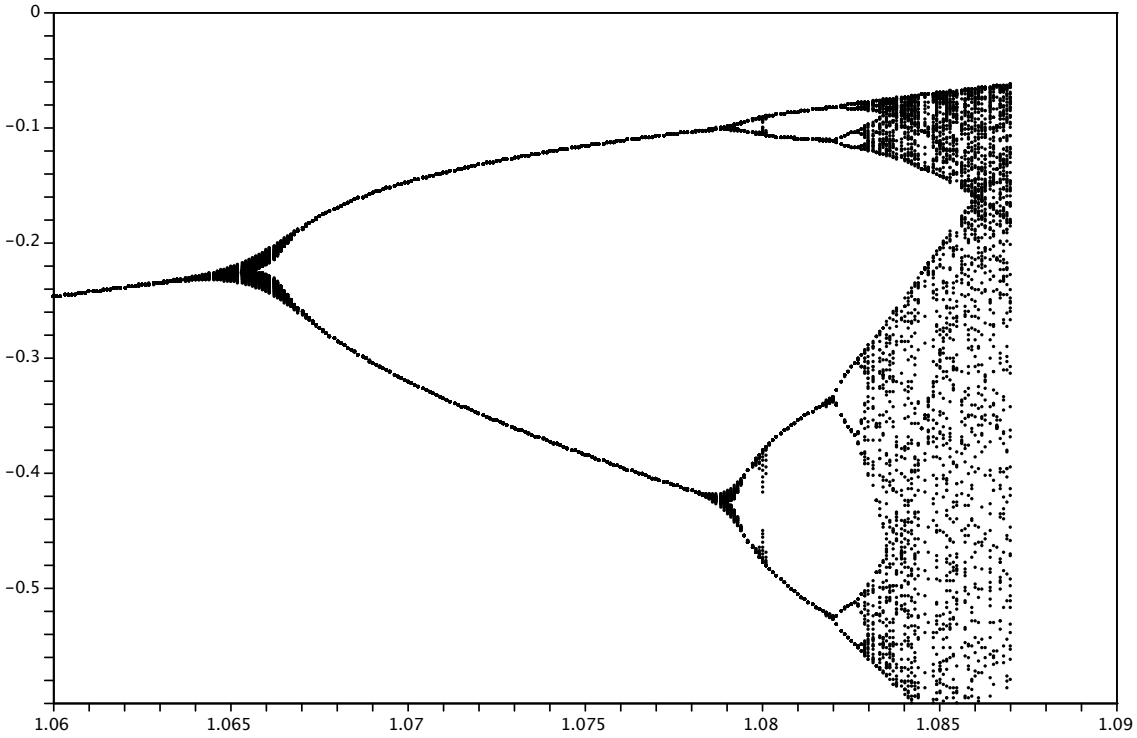
$$gf := 1.0870$$

$$gi := 1.06$$

$d := 0.0001$

$pwg := view = [1.06 .. 1.09, -0.6 .. 0.]$

w=2*Pi T=1 w0=3*Pi b=3/4*Pi g=1.07
x0=-1/2*Pi y0=0



>