Square Wave ODE Example 2 (odesolve):

This example file is meant to show how to drive a mass, spring, damper system with a square wave. For comparison purposes, you can also use a cosine wave. u1 is the cosine wave, u2 is the square wave. The odesolve solver is used to solve the second order ordinary equation. It has the benefit of making more useful output, to compute velocity, acceleration, and kinetic energy. The state space solver is a little faster, however, it's output seems to be problematic when wanting to calculate the aforementioned information. There is a seperate example that uses the state space solver.

Second Order ODE that describes the physics:

$$m\cdot\frac{d^2}{d\tau^2}x(\tau) + b\cdot\frac{d}{d\tau}x(\tau) + k\cdot x(\tau) = F_0\cdot\cos(\omega_F\cdot\tau)$$

Note; $A_0 = \frac{F_0}{m}$ $\left(\omega_0\right)^2 = \frac{k}{m}$

$\zeta := .01$ Damping Ratio

When the damping is low and the forcing function oscillates at the natural frequency, resonance will occur.

Defining Inputs:

$m := 60$ $k := 1\cdot 10^1$ $\omega_0 := \sqrt{\frac{k}{m}} = 0.408248$ $b := \zeta\cdot(2\cdot m\cdot\omega_0)$ $b = 0.489898$

$\omega_f := \omega_0\cdot 1.0$ $F_0 := 1$ $init := \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $t1 := 0$

$n := 10$ Number of Cycles $t2 := n\cdot\left(\frac{2\cdot\pi}{\omega_f}\right)$

$npoints := 100\cdot n = 1\times 10^3$ $intvls := npoints - 1 = 999$

$Fs := \frac{\omega_f}{2\cdot\pi} = 0.064975$ $tmax := t2 = 153.90598$

$tstep := \frac{tmax}{intvls} = 0.15406$ $d := 10\%$ Duty Cycle $A := F_0 = 1$

$t := 0, tstep .. tmax$

$u1(\tau) := F_0\cdot\cos(\omega_f\tau)$ Cosine Wave

$u2(\tau) := A\cdot\left(\mathrm{mod}\left(\tau, \frac{1}{Fs}\right)\cdot Fs \le d\right)\cdot(\tau < tmax)$ Square Wave

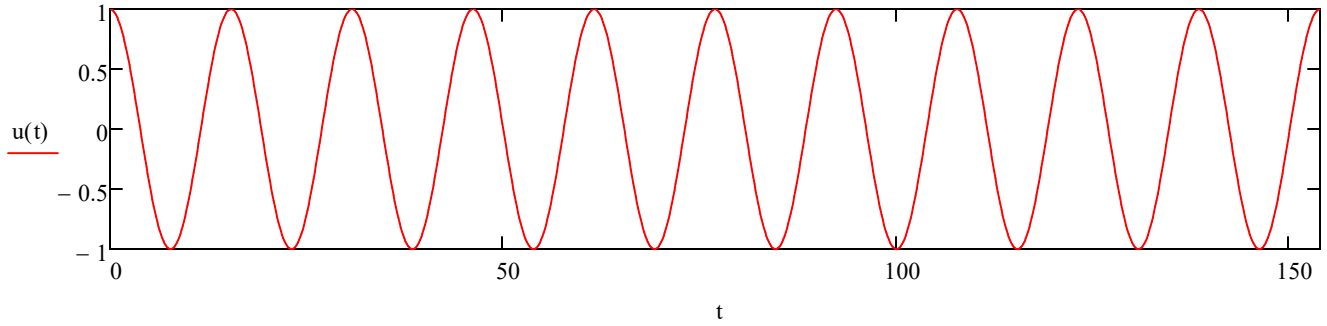
$u(\tau) := u1(\tau)$ Enter u1 or u2 as the forcing function

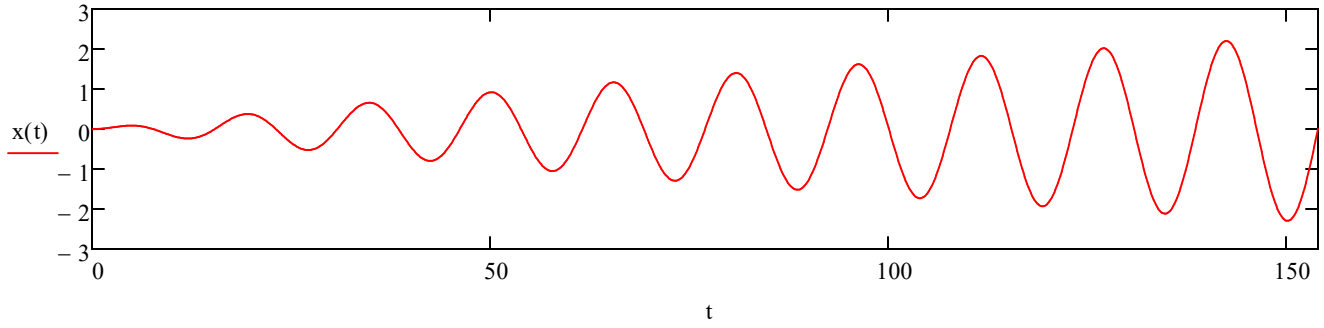
Given

$$m \cdot \frac{d^2}{d\tau^2} x(\tau) + b \cdot \frac{d}{d\tau} x(\tau) + k \cdot x(\tau) = u(\tau)$$

$x(0) = 0 \qquad x'(0) = 0$

$x := \text{Odesolve}(\tau, tmax, intvls)$





Using odesolve makes calculating the amplification factor more difficult. You have to dump the odesolve 'function' output into matrices, so that the max displacement can be calculated.

$i := 0 .. intvls \qquad j := 1 .. intvls - 1 \qquad p := 2 .. intvls - 2$

$tvec_i := tstep \cdot i$

$xvec_i := \overrightarrow{x(tvec_i)} \qquad uvec_i := \overrightarrow{u(tvec_i)}$

$$vvec_j := \frac{xvec_{j+1} - xvec_{j-1}}{tvec_{j+1} - tvec_{j-1}}$$

$$kevec_j := .5 \cdot m \cdot (vvec_j)^2$$

$$avec_p := \frac{vvec_{p+1} - vvec_{p-1}}{tvec_{p+1} - tvec_{p-1}}$$

$max(xvec) = 2.203604$

$$\frac{max(xvec)}{\left(\frac{F_0}{k}\right)} = 22.036045$$ Amplification Factor

When the damping ratio is .01, classical solutions state that the above should equal 50. Moreover, the dynamic displacement should be 50x the static spring displacement. The above is used to serve as a check. Notice what happens when you switch to a square wave. Also note what happens when you change the duty cycle of the square wave. You will find that classical solutions come with a lot of 'ands, ifs, and buts'. Therefore, it's better to just solve the ODE rather than use classical simplifications. Solving the ODE was very difficult in the old days 'pre-computer'. Therefore, classical solutions gained wide spread use.

Note how the number of cycles affects this as well. You need about 300 cycles to get to the classical solution. This makes viewing the wave forms impossible. So you can switch to something like 5 cycles, to see the waves.

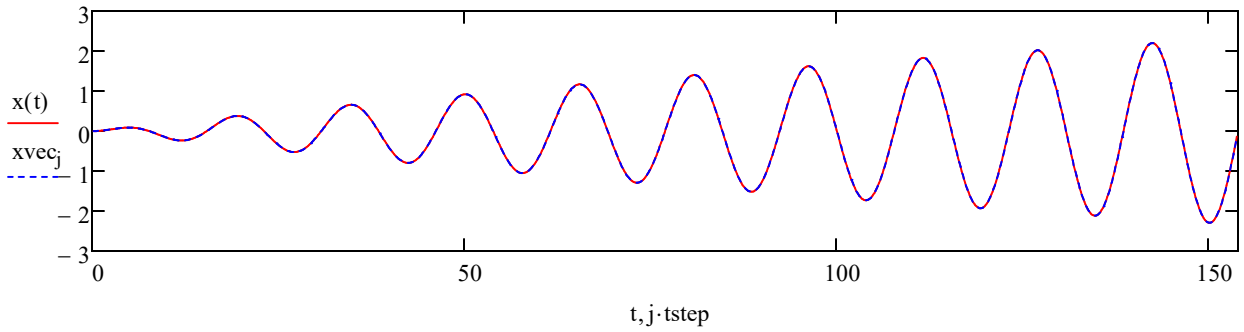
Using odesolve, rather than state space, makes the following calculations simpiler. However, they are much slower and seem to have more error.

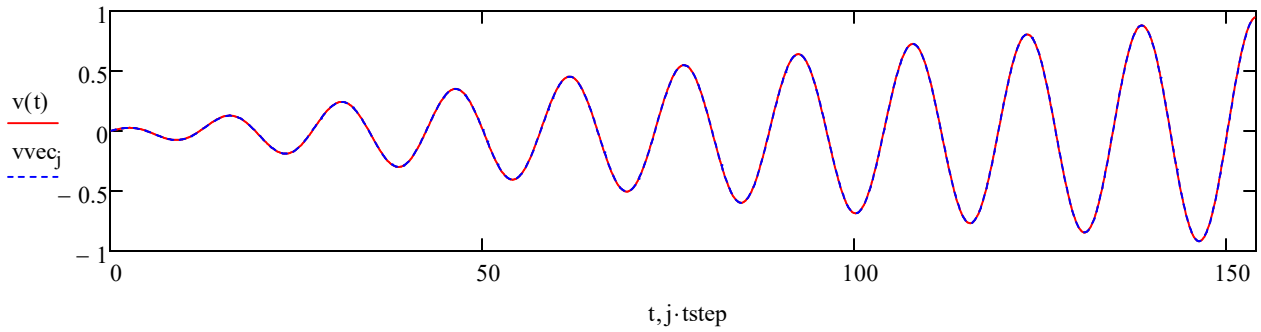
$t := tstep, 2 \cdot tstep .. tmax - tstep$

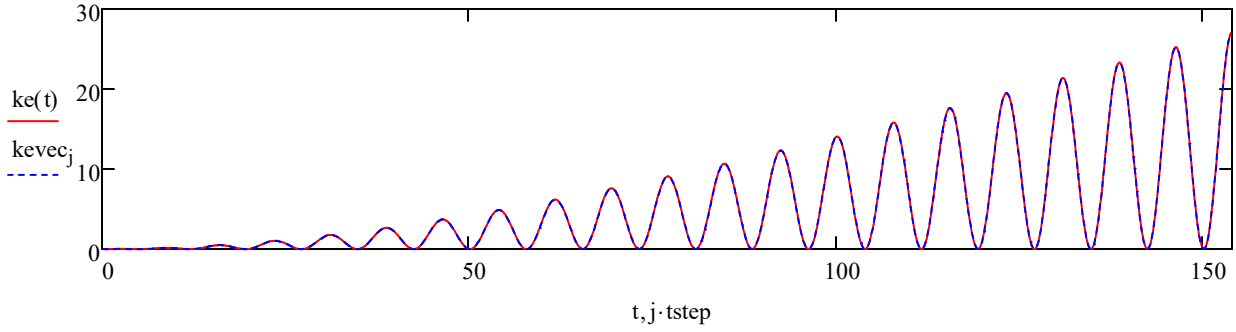
$v(t) := \frac{d}{dt} x(t)$ Calculating mass velocity versus time

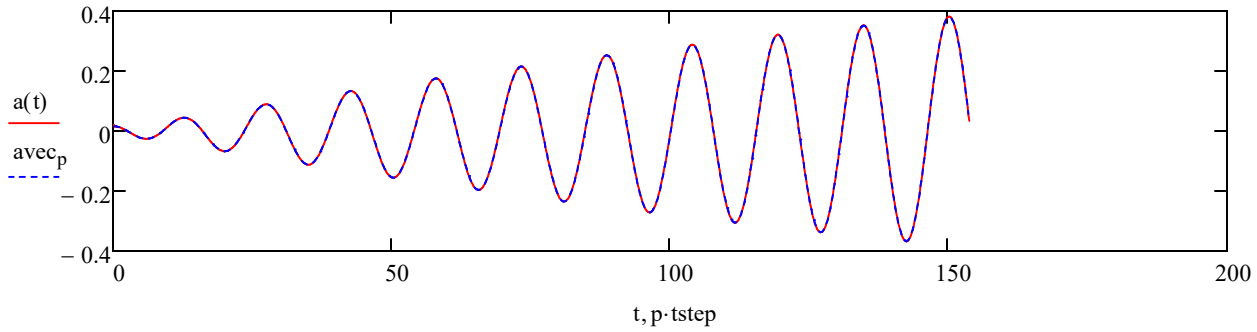
$ke(t) := .5 \cdot m \cdot (v(t))^2$ Calculating mass kinetic energy versus time

$a(t) := \frac{d^2}{dt^2} x(t)$ Calculating mass acceleration versus time









Reaction Forces:

$rf\_mass(t) := m \cdot a(t)$ $rf\_damping(t) := b \cdot v(t)$ $rf\_spring(t) := k \cdot x(t)$

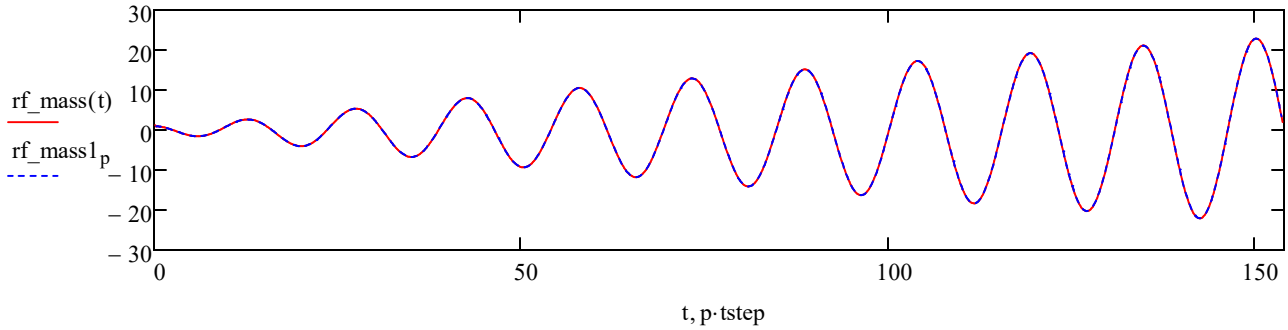
$rf\_mass1_p := m \cdot avec_p$ $rf\_damping1_p := b \cdot vvec_p$ $rf\_spring1_p := k \cdot xvec_p$

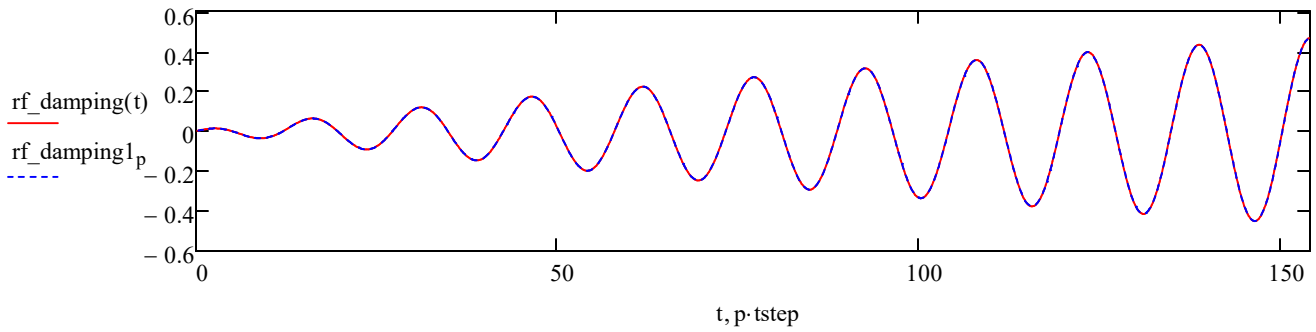
$rf\_total(t) := rf\_mass(t) + rf\_damping(t) + rf\_spring(t)$

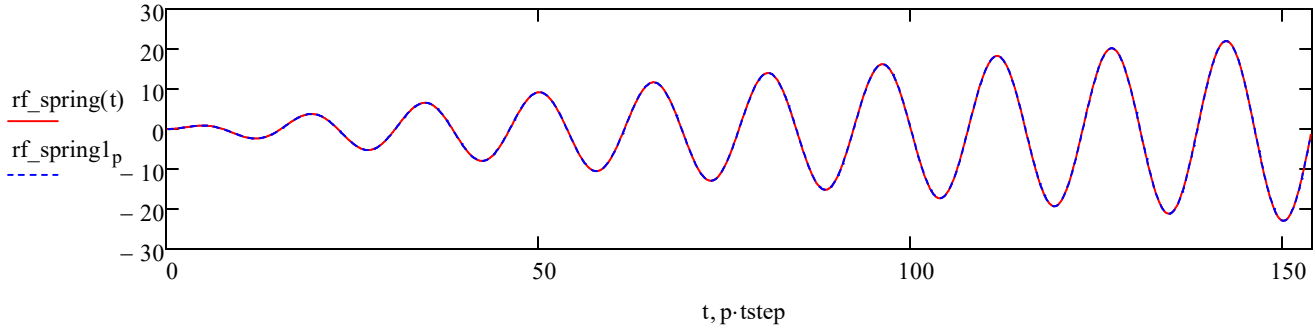
$rf\_total1_p := rf\_mass1_p + rf\_damping1_p + rf\_spring1_p$

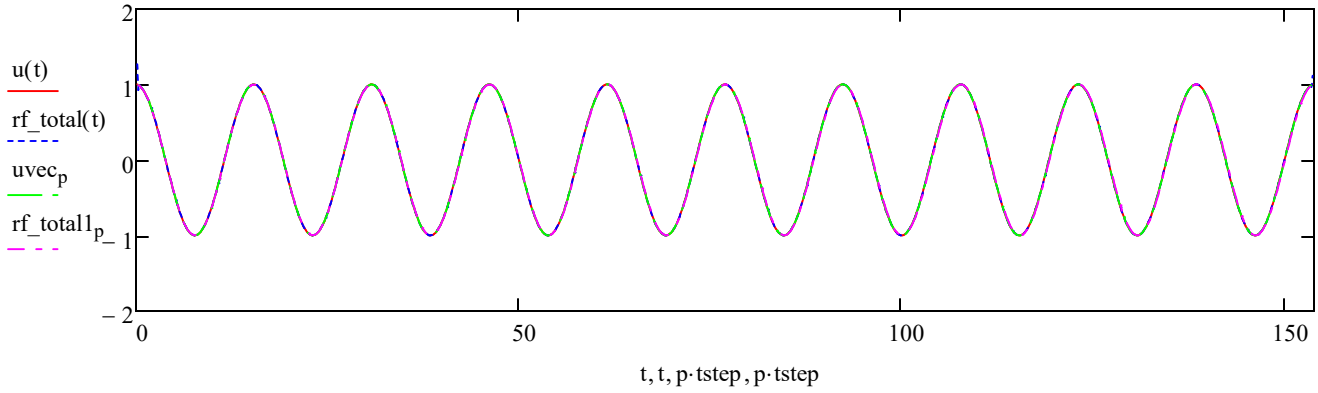
$check(t) := u(t) - rf\_total(t)$ The check should equal zero, due to Newton's third law

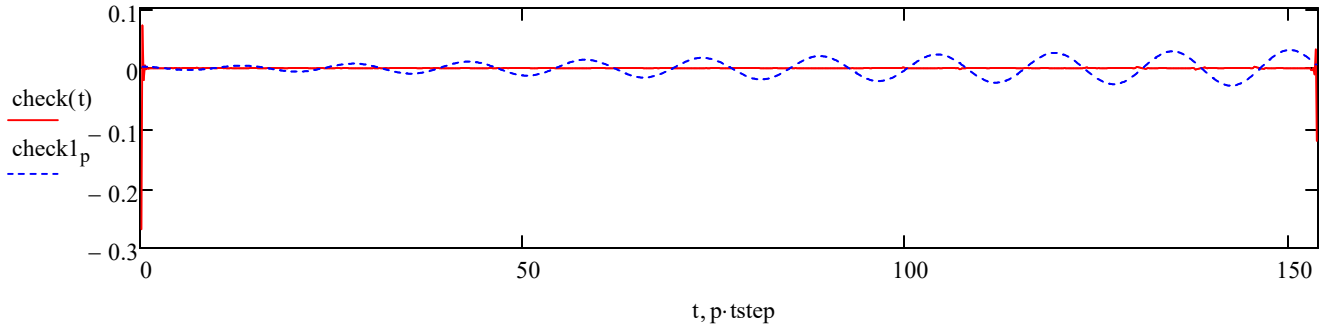
$check1_p := uvec_p - rf\_total1_p$

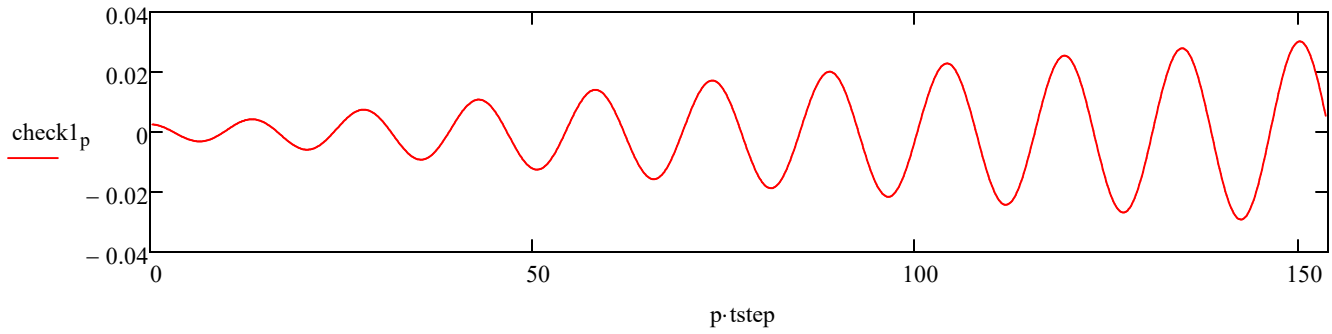



rf_damping(t)
rf_damping1p
0.6
0.4
0.2
0
– 0.2
– 0.4
– 0.6
0
50
100
150
t, p·tstep


rf_spring(t)
rf_spring1p
30
20
10
0
– 10
– 20
– 30
0
50
100
150
t, p·tstep


u(t)
rf_total(t)
uvecp
rf_total1p
2
1
0
– 1
– 2
0
50
100
150
t, t, p·tstep, p·tstep


check(t)
check1p
0.1
0
– 0.1
– 0.2
– 0.3
0
50
100
150
t, p·tstep



$$checkvec_j := check\left(tvec_j\right)$$

$$\frac{\max(checkvec)}{F_0} = 7.116793 \cdot \%$$

$$\frac{\max(check1)}{F_0} = 3.026125 \cdot \%$$

$$\frac{\min(checkvec)}{F_0} = -26.86829 \cdot \%$$

$$\frac{\min(check1)}{F_0} = -2.912631 \cdot \%$$