Polygon Generator

Standard 2D Plotting

3D Scatter Plot

Enhanced Primitives

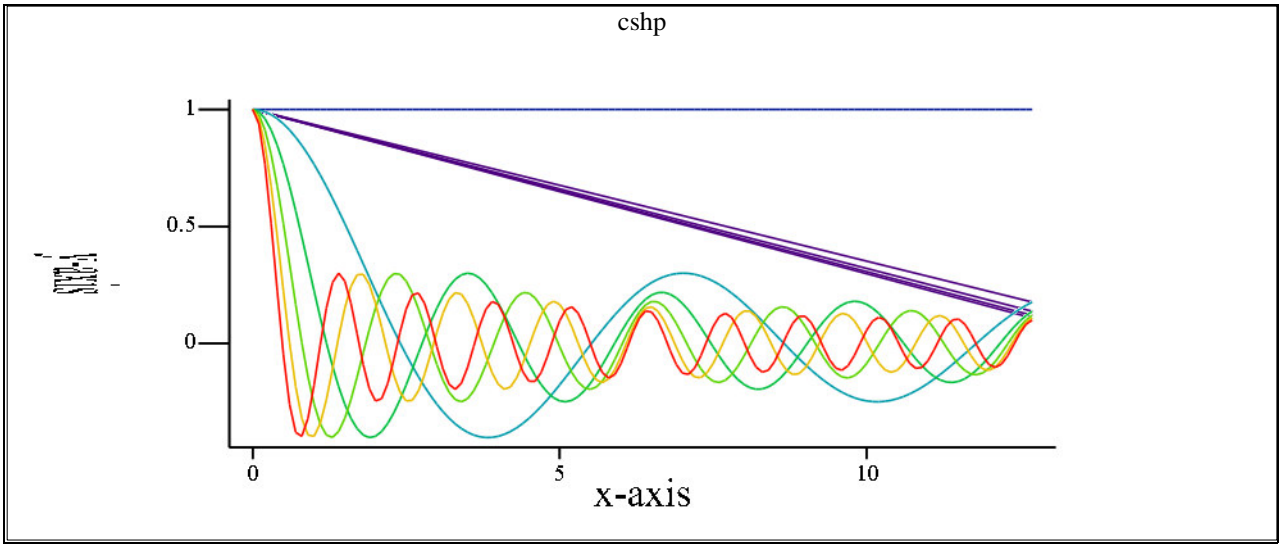
Hiding Lines by colormaps

Plotting array data by columns

2D Array multiplot functions v2

Basic 3D Plot Support Functions

$t0 := time(0)$



Basic 3D Plot Support Function Examples

3D Plot Examples

Example 1 - basic plot

$ftitle := "cshp"$ think of a graph title

$gattr := gSetTitle(f3DScatter, ftitle)$ set it

$gattr := gSetAxisAutoplot(f3DScatter, 0, -1)$ set autoscaling for the x axis

$gattr := gSetAxisAutoplot(f3DScatter, 1, -1)$ set autoscaling for the y axis

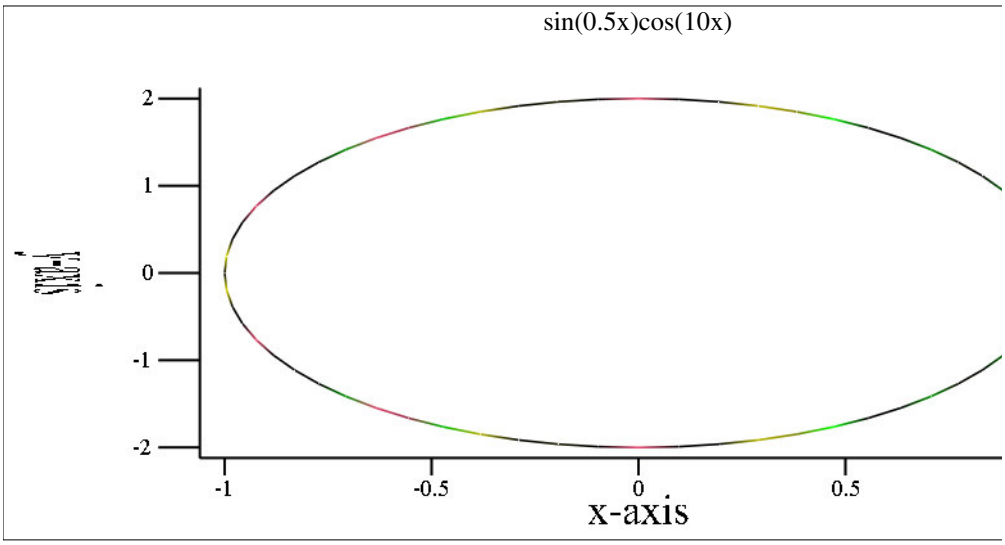
$gattr := f3DScatter(cshp_0, 0)$ plot the x,y,z data contained in the 3-vector $cshp_0$

Attributes returned in a 5-vector

```
1st item : graph title and location
2nd item : x-axis data - title, show title, autoscale, minval, maxval, datamin, datamax
3rd item : y-axis data
4th item : z-axis data
5th item : colormap data - cmap index, line cmap, point cmap, polygon cmap
```

create new functional plot

f3DScatter(array, attributes) :=



f(i, j) := sin(0.5·i)·cos(j)

(array attributes)

Example 2 - basic animation

```
gattr := gSetTitle(f3DScatter, "Animated Sine")
         gSetAxisTitle(f3DScatter, 0, "x-axis")
         gSetAxisRange(f3DScatter, 0, 0, 2π)
         gSetAxisTitle(f3DScatter, 1, "y-axis")
         gSetAxisRange(f3DScatter, 1, −1, 1)
         n ← 64
         scl ← 2·π / n
         for i ∈ 0..n
             θ_i ← scl·i
             x_i ← cos(θ_i)
             x_i ← θ_i
             y_i ← sin(2 θ_i)
             z_i ← f(θ_i, θ_i)
             M ← (x y z)^T
             f3DScatter(M, 0)
```

Example 3 - basic animation

overload previous functional plot

```
a := gSetAxisAutoplot(f3DScatter, 0, −1)
     gSetAxisRange(f3DScatter, 1, −2, 2)
     for j ∈ 0..10
         n ← 64
         scl ← 2·π / n
         for i ∈ 0..n
             θ_i ← scl·i
             x_i ← cos(θ_i)
             y_i ← sin(θ_i)·(0.1·j + 1)
             z_i ← f(θ_i, θ_i·j)
         M ← (x y z)^T
         s ← concat("sin(0.5x)cos(", num2str(j), "x)")
         concat(s, " : npts = ", num2str(n))
         gSetTitle(f3DScatter, s)
         f3DScatter(M, 0)
         wait(0.3)
```

Example 4 - basic animation - ODE

We define a function, ODESolverSS, which will take an integration function, g, and solve an ODE. The input parameters are identical to the rkfixed parameters, with the addition of g and fplot as the last 2 parameters. The format of the results is different to rkfixed, in that it returns a nested vector where each element is one of the 'columns' of the rkfixed result.

The major difference is that that solver shell outputs intermediate results to the 3D plot function

```
ODESolverSS(x0,t0,t1,n,D,g,fplot) :=
    h ← (t1 − t0)/n
    t ← t0
    x ← x0
    τ ← t
    ξ ← x^T
    for i ∈ 1..n
        t ← t + h
        τ ← stack(τ,t)
        x ← g(t,x,D,h)
        ξ ← stack(ξ,x^T)
        τξ_0 ← τ
        for k ∈ 0..cols(ξ) − 1
            τξ_k ← ξ^⟨k⟩
        fplot(τξ,0)
    τξ
```

Ideally, there should be variants of the plot component that handles ODE solutions directly or plot multiple columns.

Runge-Kutta 4th Order Integrator

The differential at t for the Runge-Kutta integrator is calculated by estimating the slope of the curve at 4 different points and averaging them as follows (Note that its first slope term is the entire Euler integrator). The value of x at the end of the step h is then given by the integration function rk4:

```
rk4(t,x,f,h) :=
    f1 ← h·f(t,x)                          <-----Runge-Kutta 1st term
    f2 ← h·f(t + h/2, x + (1/2)·f1)        <-----Runge-Kutta 2nd term
    f3 ← h·f(t + h/2, x + (1/2)·f2)        <-----Runge-Kutta 3rd term
    f4 ← h·f(t + h, x + f3)                <-----Runge-Kutta 4t term
    dx ← (1/6)·(f1 + 2·f2 + 2·f3 + f4)     <---- Runge-Kutta differential
    x + dx
```

leading to the animated integrator ***RK4***:

```
RK4(x0,t0,t1,n,D,fplot) := ODESolverSS(x0,t0,t1,n,D,rk4,fplot)   <---- Runge-Kutta Integrator
```

Basic ODE

```
D1(t,x) := x_1

D2(t,x) := (x_5·cos(x_6·t) − x_3·x_1 − x_4·(x_0)^3) / x_2

D(t,x) := (D1(t,x)  D2(t,x)  0  0  0  0  0)^T
```

Duffing Oscillator ODE

$$m\cdot\frac{d^2}{dt^2}x + c\cdot\frac{d}{dt}x + k\cdot x^3 = a\cdot\cos(\omega\cdot t)$$

Initial conditions

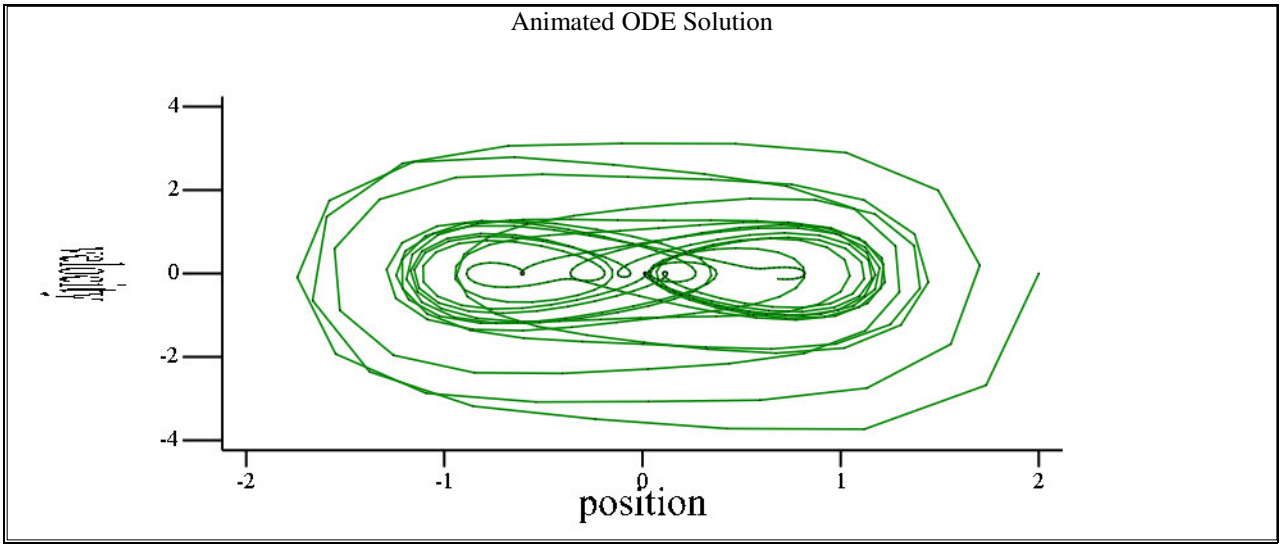
mass-spring system: $m := 1.0$ $c := 0.1$ $k := 1.0$

forcing oscillator: $a := 2.25$ $\omega := 0.54$

initial conditions: $x_0 := 2$ $x'_0 := 0$

$X := \left(x_0 \;\; x'_0 \;\; m \;\; c \;\; a \;\; k \;\; \omega\right)^T$ $D(0, X)^T = (0 \;\; -17 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0)$

create new functional plot

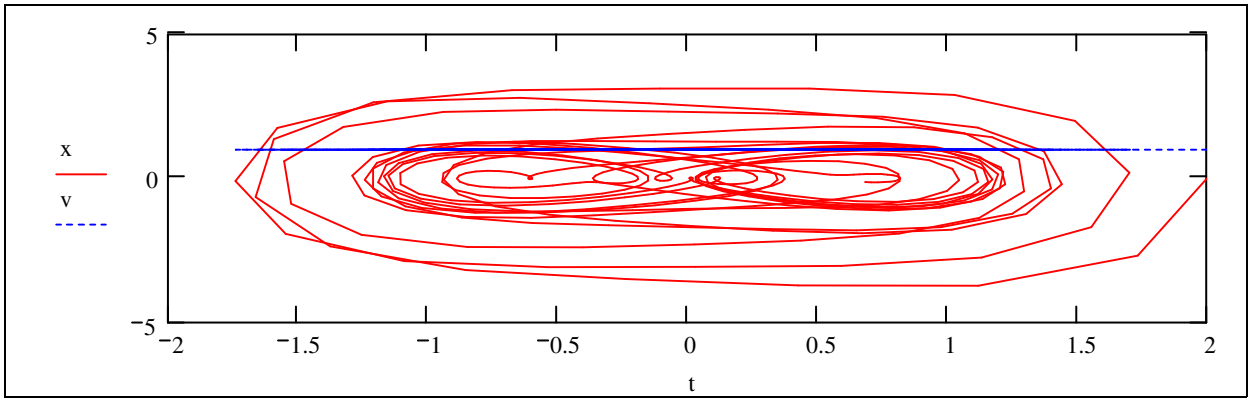


Solve ODE

$t_0 := 0$ $t_1 := 30\pi$ $N := 512$

```
txv := gSetTitle(f3DScatter, "Animated ODE Solution")
       gSetAxisTitle(f3DScatter, 0, "position")
       gSetAxisRange(f3DScatter, 0, −2, 2)
       gSetAxisTitle(f3DScatter, 1, "velocity")
       gSetAxisRange(f3DScatter, 1, −4, 4)
       RK4(X, t0, t1, N, D, f3DScatter)
```

$t := txv_0$ $x := txv_1$ $v := txv_2$

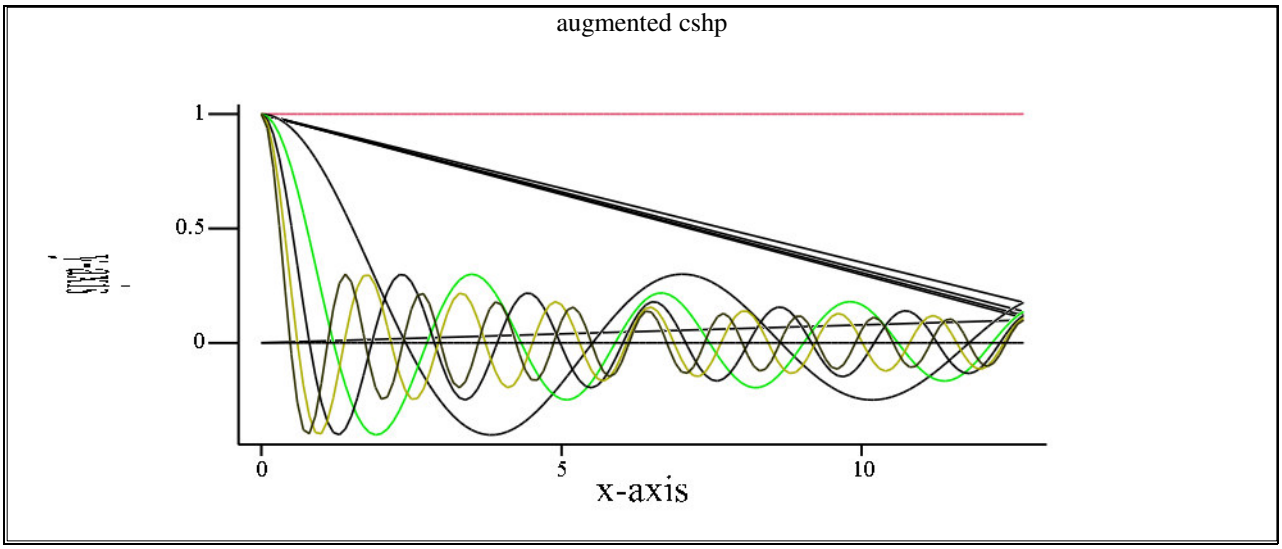


An alternative method of plotting array columns

The graph below allows a user to directly plot an array by columns. It uses the same method of hiding the 'return' lines by invisible colours, but makes this process internal to the plot component. Part of this functionality is handled by a Mathcad program on one of the input parameters and the rest by the component itself.

The function, f3DPlotCols, expects a single array with the first column being the x values.

Obviously, there is a lot of tailoring that could be done to enhance the interface, and it is possible to rewrite it to give each column its own plot tab.



$cplt := augment(xC, C)$ create the data array

$gattr :=$ plot it

```
ftitle ← "augmented cshp"
gSetTitle(f3DPlotCols, ftitle)
gSetAxisAutoplot(f3DPlotCols, 0, −1)
gSetAxisAutoplot(f3DPlotCols, 1, −1)
f3DPlotCols(cplt, 0)
```

VBScript development component

$time(0) - t0 = 3.985$