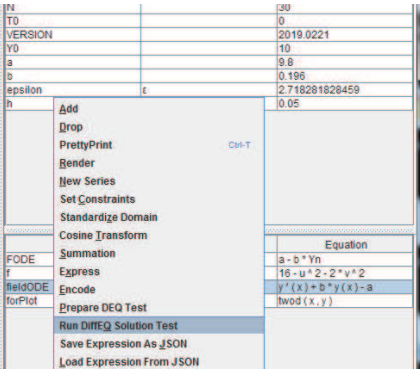
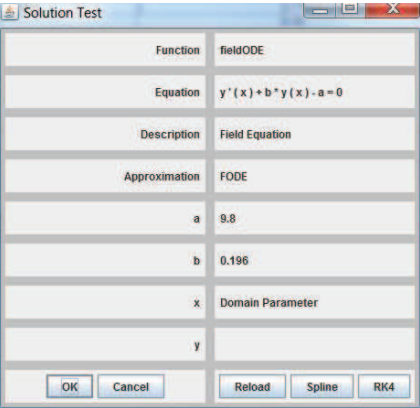
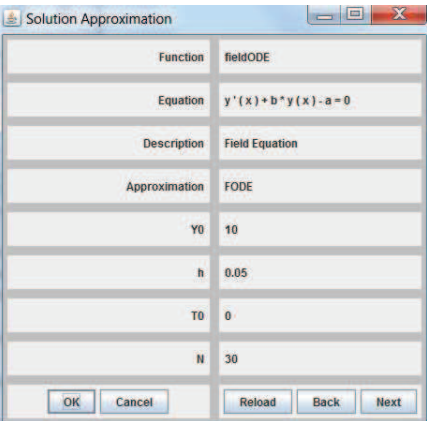
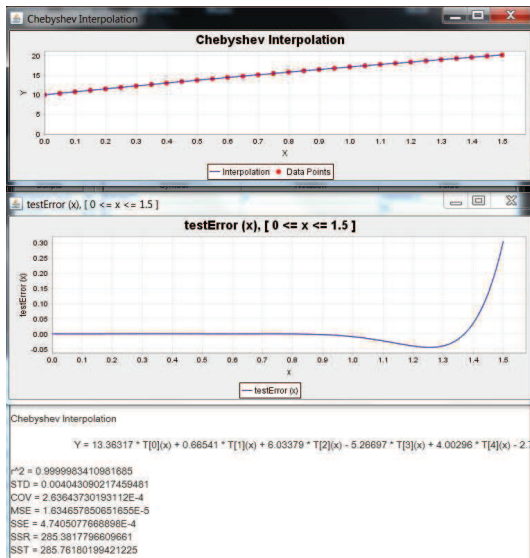


Differential Equations

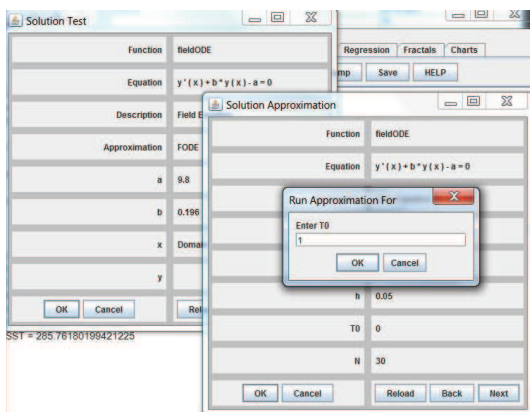
	Select an equation “Run DiffEQ Solution Test” From the Function drop-down menu
	The “Solution Test” tool is shown The information about the equation is shown The required parameters are shown with their values The RK4 button starts the Runge-Kutta tool
	The “Solution Approximation” screen shows the set values of the RK4 formula parameters Press the OK button to run the T0 approximation



The Interpolation displays show the quality of the best fit attempt made for the approximation

In addition to the interpolation displays a “testError” plot is shown

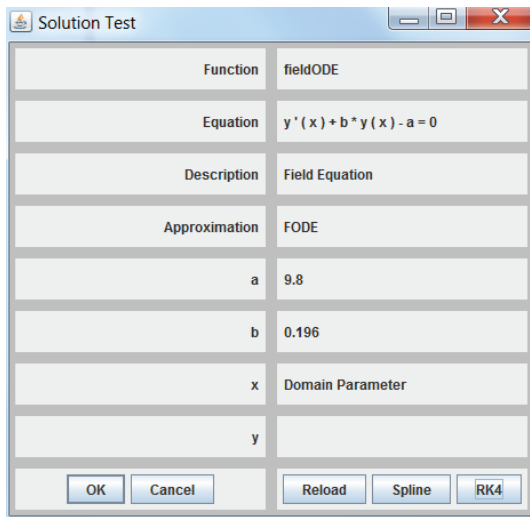
From the error plot the knot point should be chosen



Use the Next button on the RK4 Solution Approximation screen which will present a dialog requesting the T0 for the knot

This will present the next Interpolation

This repeats for as many knots as are required



Now the Solution Test “Spline” button will generate the spline for the solution approximation

CALCLIB Environment			
Scripts	Symbol	Notation	Value
ADSpineTwoD	INFINITY	∞	1000000.0
AirCalcTests	N		30
AirODE	SOLUTION_INTERPOLATION_C		(9.464487513303952, 8.1667006)
AirDeclaration	T		(1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.4)
AirErrorPlots	T0		1
AirFunctions	Tn		2.499999999999998
AirInterp	VERSION		2019.0221
AirRenderers	Y		(17.118872471442053, 17.43953)
AirSpline	Y0		17.118872471442053
AirSplineTests	Yn		25.49296084518933
ArrayAsFunction	a		9.8
AssignFieldId	b		0.198
AtanIterator	epsilon	ϵ	2.718281828459
AtanSeriesCom	h		0.05
BarsAndPies	i		30.0
BernoulliODE	it		2.499999999999998
BernoulliPlots	jt		1.0
	kt		0.005000000000000001
Data	Function	Parameters	Equation
airVtots.TDF	FODE	Tn, Yn	$a \cdot b \cdot Yn$
AirCmplxSeq.T	f	Tn, Yn	FODE (Tn, Yn)
BesselHValues	RelODE	x	$y'(x) = b \cdot y(x) - a$
bessel.TDF	forPlot	x, y	twod (x, y)
besselKValues	testError	x	$y'(x) - f(x, y(x))$
data.TDF	y	x	Function Spline
expr.json	y'	x	Function Derivative Spline
JO-fractions.TDF	y''	x	Second Derivative Spline
JO-fractions.TDF			

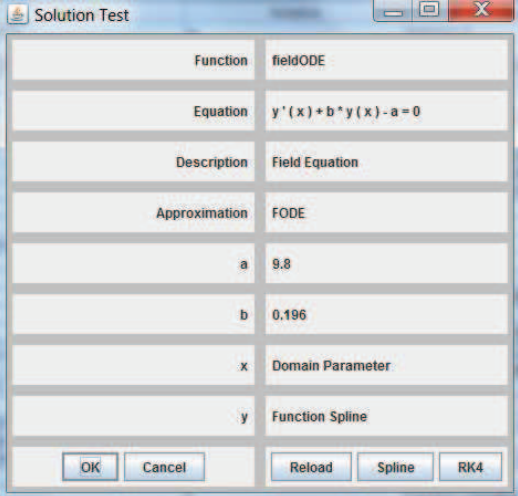
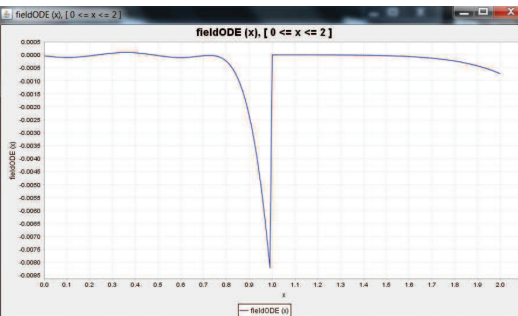
Now the solution function “y” can be found in the functions list along with the derivatives y’ and y’’

CALCLIB Environment			
Scripts	Symbol	Notation	Value
ADSpineTwoD	INFINITY	∞	1000000.0
AirCalcTests	N		30
AirODE	SOLUTION_INTERPOLATION_C		(9.464487513303952, 8.1667006)
AirDeclaration	T		(1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3, 1.4)
AirErrorPlots	T0		1
AirFunctions	Tn		2.499999999999998
AirInterp	VERSION		2019.0221
AirRenderers	Y		(17.118872471442053, 17.43953)
AirSpline	Y0		17.118872471442053
AirSplineTests	Yn		25.49296084518933
ArrayAsFunction	a		9.8
AssignFieldId	b		0.198
AtanIterator	epsilon	ϵ	2.718281828459
AtanSeriesCom	h		0.05
BarsAndPies	i		30.0
BernoulliODE	it		2.499999999999998
BernoulliPlots	jt		1.0
	kt		0.005000000000000001
Data	Function	Parameters	Equation
airVtots.TDF	FODE	Tn, Yn	$a \cdot b \cdot Yn$
AirCmplxSeq.T	f	Tn, Yn	FODE (Tn, Yn)
BesselHValues	RelODE	x	$y'(x) = b \cdot y(x) - a$
bessel.TDF	forPlot	x, y	twod (x, y)
besselKValues	testError	x	$y'(x) - f(x, y(x))$
data.TDF	y	x	Function Spline
expr.json	y'	x	Function Derivative Spline
JO-fractions.TDF	y''	x	Second Derivative Spline
JO-fractions.TDF			

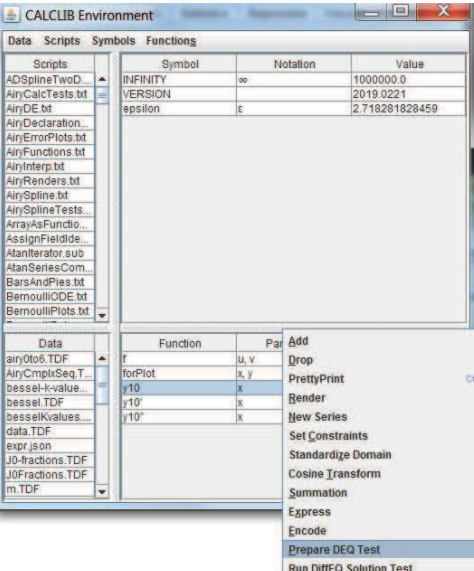
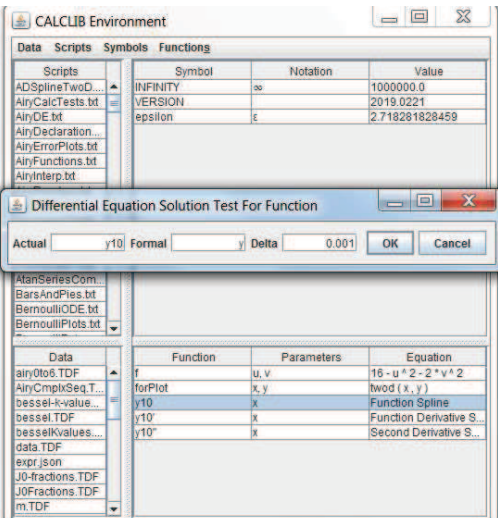
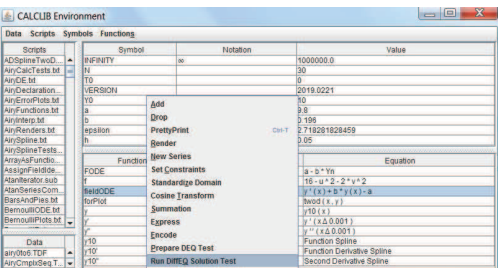
The function can be saved as a JSON expression for later import as a common singleton function

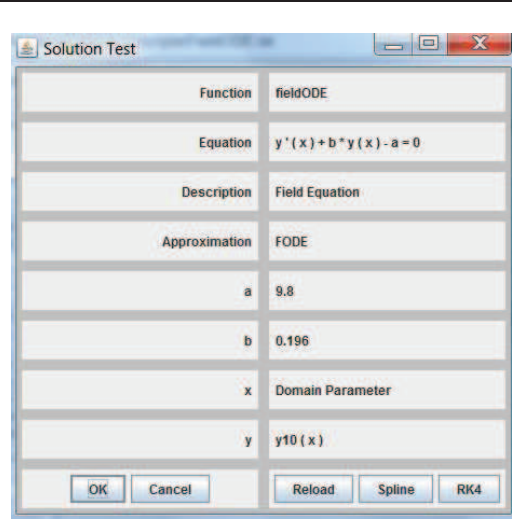
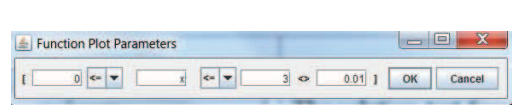
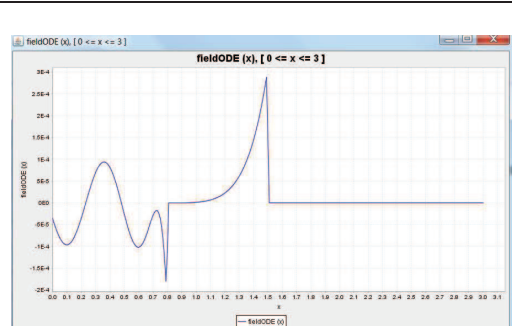
Hence the solution is available as a function and the accuracy across the domain is know from the error plot

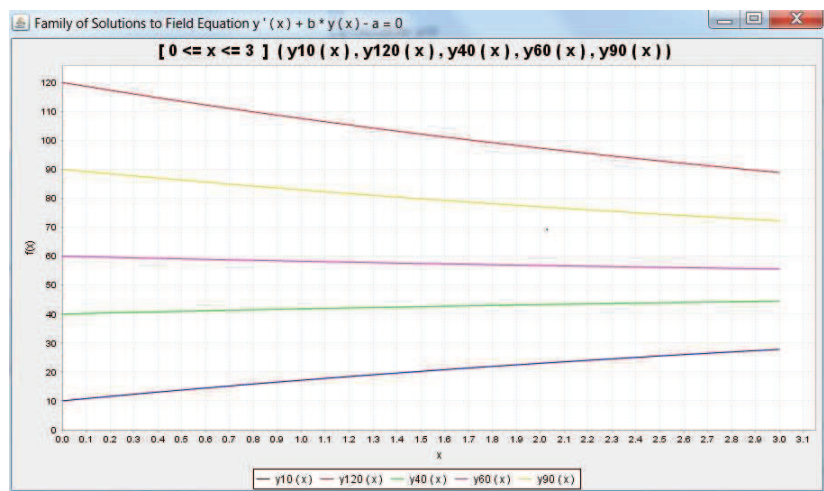
Test the Solution Quality

 The Solution Test dialog box contains the following fields: <table border="1"><thead><tr><th>Field</th><th>Value</th></tr></thead><tbody><tr><td>Function</td><td>fieldODE</td></tr><tr><td>Equation</td><td>$y'(x) + b \cdot y(x) - a = 0$</td></tr><tr><td>Description</td><td>Field Equation</td></tr><tr><td>Approximation</td><td>FODE</td></tr><tr><td>a</td><td>9.8</td></tr><tr><td>b</td><td>0.196</td></tr><tr><td>x</td><td>Domain Parameter</td></tr><tr><td>y</td><td>Function Spline</td></tr></tbody></table> Buttons: OK, Cancel, Reload, Spline, RK4	Field	Value	Function	fieldODE	Equation	$y'(x) + b \cdot y(x) - a = 0$	Description	Field Equation	Approximation	FODE	a	9.8	b	0.196	x	Domain Parameter	y	Function Spline	The Solution Test screen now shows the solution “y” has a “Function Spline” available for test
Field	Value																		
Function	fieldODE																		
Equation	$y'(x) + b \cdot y(x) - a = 0$																		
Description	Field Equation																		
Approximation	FODE																		
a	9.8																		
b	0.196																		
x	Domain Parameter																		
y	Function Spline																		
 The Function Plot Parameters dialog box shows the following settings: <table border="1"><thead><tr><th>Parameter</th><th>Value</th></tr></thead><tbody><tr><td>Start</td><td>0</td></tr><tr><td>End</td><td>2</td></tr><tr><td>Step</td><td>0.01</td></tr></tbody></table> Buttons: OK, Cancel	Parameter	Value	Start	0	End	2	Step	0.01	The OK button show a form for entering domain parameters										
Parameter	Value																		
Start	0																		
End	2																		
Step	0.01																		
 The plot shows the solution error fieldODE(x) versus x. The x-axis ranges from 0.0 to 2.0, and the y-axis ranges from -0.0005 to 0.0005. The error is near zero for most of the domain, but there is a sharp negative spike at x = 1.0, reaching approximately -0.00045. The legend indicates the plot is for fieldODE(x).	A plot of the solution error for the domain is shown An error spike is typically seen at each knot																		

Test function as Solution

	Select function “Prepare DEQ Test” from function menu
	A dialog will request the formal parameter name for the function within the differential equation
	Select the differential equation “Run DiffeQ Solution Test” from the Function drop-down menu

 The solution test form comes up Now the formal parameter is shown to reference the selected actual parameter Function $y = y10(x)$ Press OK button	The solution test form comes up Now the formal parameter is shown to reference the selected actual parameter Function $y = y10(x)$ Press OK button
 Enter domain description	Enter domain description
 The error plot is displayed	The error plot is displayed



RungeKutta.txt

Use Runge-Kutta approximation and check the error computed against solution vector

```
// Runge-Kutta error test
```

```
// generate interpolation
```

```
T = [ 0 <= t <= N ] ( T0 + t * h )
```

```
solution = CHEBYSHEV (T, Y)
```

```
// generate alias for solution and for derivative
```

```
!! y (x) = solution @*^ x
```

```
!! y' (x) = solution @*^' x
```

```
// improve polynomial use efficiency
```

```
OPTIMIZE y; OPTIMIZE y'
```

```
// post error test
```

```
!! testError (x) = y' (x) - f ( x, y(x) )
```

```
// plot error against domain described by test parameters
```

```
PLOTF testError [ T0 <= x <= T0 + h * N <> h/10 ]
```

```
SIDEBYSIDE "Regression Versus Error Plot"
```

RungeKuttaRiccati.txt

A simple example of use of RungeKutta.txt

// Riccati

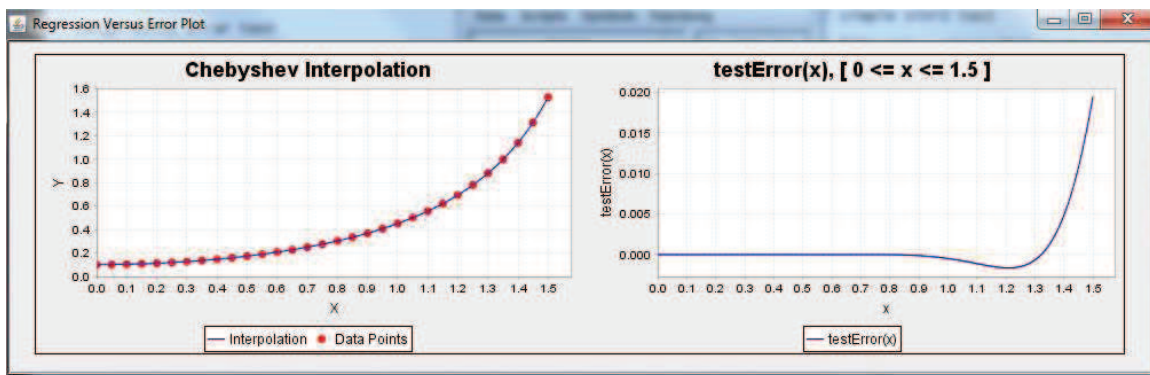
!! $u(x) = x / 2$

$Y_0 = 0.1$; $T_0 = 0$; $h = 0.05$; $N = 30$

!! $f(T_n, Y_n) = Y_n^2 + T_n * u(T_n) * Y_n + u(T_n)$

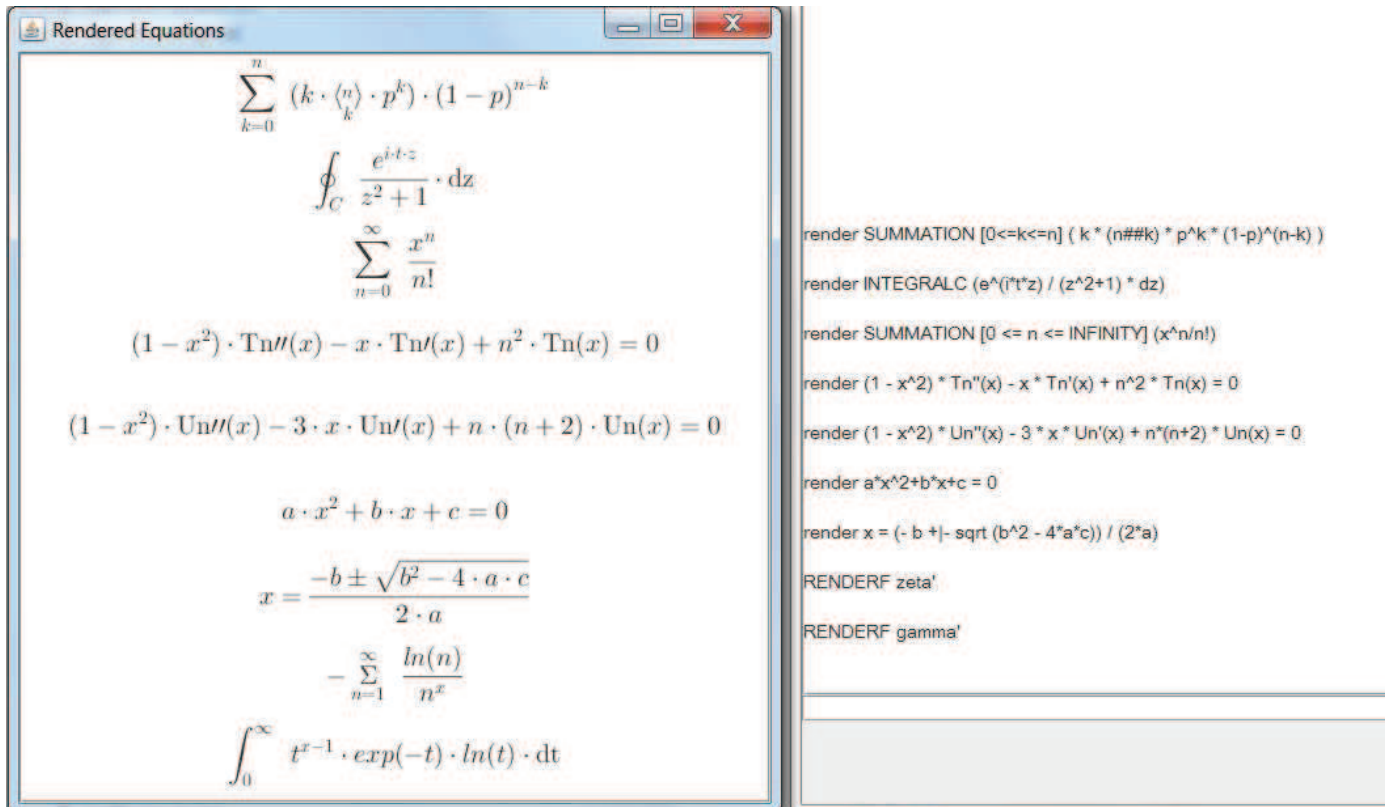
$Y = (0.1, 0.10113171857626335, 0.10354424073431215, 0.10726542307837783,$
 $0.11232678123700574, 0.11876514779425473, 0.12662446126207821,$
 $0.1359577239358084, 0.14682917975976403, 0.15931677951018194,$
 $0.1735150208399999, 0.18953827665792933, 0.20752475923532693,$
 $0.22764131262151488, 0.250089287107206, 0.2751118334633071,$
 $0.303003071635968, 0.3341197537209428, 0.36889627761184174,$
 $0.40786425183380254, 0.45167832094096894, 0.5011507269701952,$
 $0.5572982587771222, 0.6214070869571133, 0.6951239492603989,$
 $0.7805870513163335, 0.8806183876003942, 0.999013878270224,$
 $1.1409946276569765, 1.3139341820639356, 1.5285808552014561)$

READ RungeKutta.txt



Equation Renders

RENDER commands are available to supply MathML / LaTeX type renders of sophisticated equations.



The screenshot shows a window titled "Rendered Equations" containing several mathematical equations. To the right of the window, the corresponding RENDER commands are listed in a plain text font.

Equations shown in the window:

$$\sum_{k=0}^n (k \cdot \binom{n}{k} \cdot p^k) \cdot (1-p)^{n-k}$$

$$\oint_C \frac{e^{i \cdot t \cdot z}}{z^2 + 1} \cdot dz$$

$$\sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$(1 - x^2) \cdot T_{nn}(x) - x \cdot T_{n'}(x) + n^2 \cdot T_n(x) = 0$$

$$(1 - x^2) \cdot U_{nn}(x) - 3 \cdot x \cdot U_{n'}(x) + n \cdot (n + 2) \cdot U_n(x) = 0$$

$$a \cdot x^2 + b \cdot x + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

$$- \sum_{n=1}^{\infty} \frac{\ln(n)}{n^x}$$

$$\int_0^{\infty} t^{x-1} \cdot \exp(-t) \cdot \ln(t) \cdot dt$$

RENDER commands shown to the right:

```
render SUMMATION [0<=k<=n] ( k * (n##k) * p^k * (1-p)^(n-k) )
render INTEGRALC (e^(i*t*z) / (z^2+1) * dz)
render SUMMATION [0 <= n <= INFINITY] (x^n/n!)
render (1 - x^2) * Tn''(x) - x * Tn'(x) + n^2 * Tn(x) = 0
render (1 - x^2) * Un''(x) - 3 * x * Un'(x) + n*(n+2) * Un(x) = 0
render a*x^2+b*x+c = 0
render x = (- b +/- sqrt (b^2 - 4*a*c)) / (2*a)
RENDERF zeta'
RENDERF gamma'
```

Functions found in the Functions list can be rendered using:

RENDERF function-name

As seen above equations can be rendered directly from the command line e.g.

RENDER (1 - x^2) * Tn''(x) - x * Tn'(x) + n^2 * Tn(x) = 0