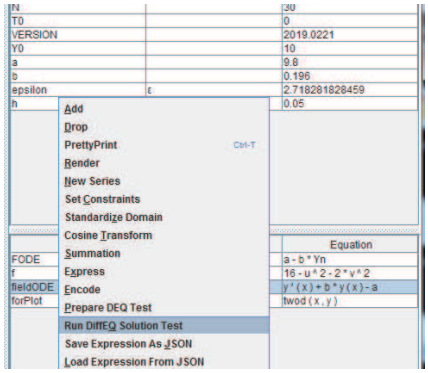
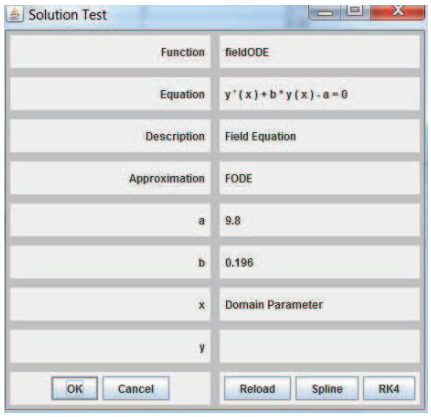
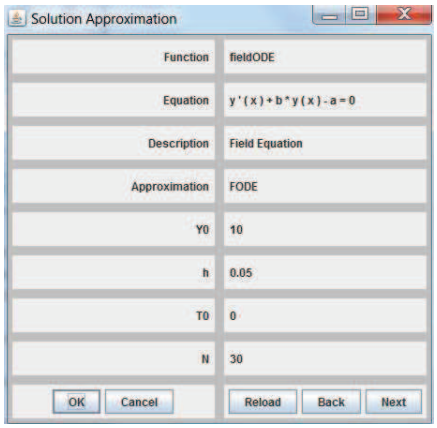
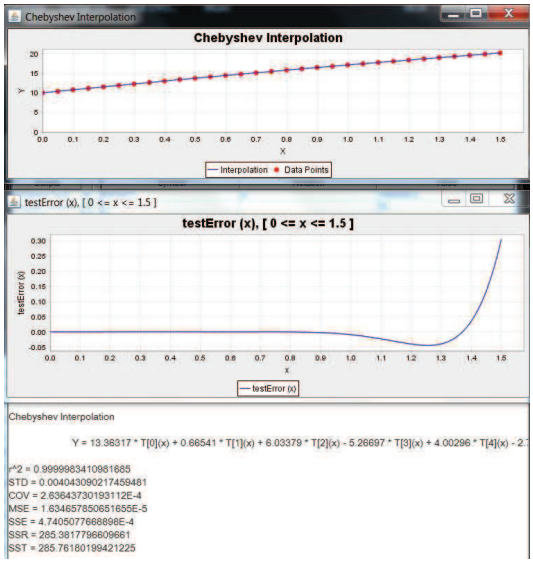
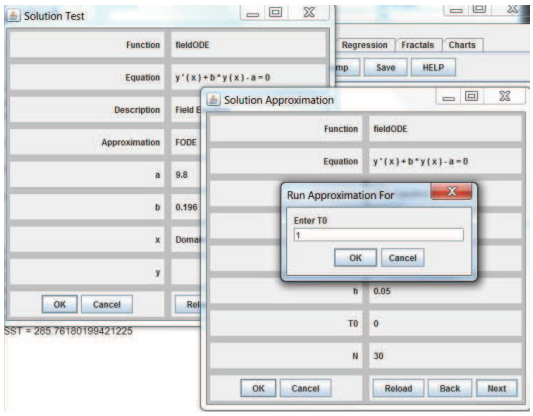
Differential Equations

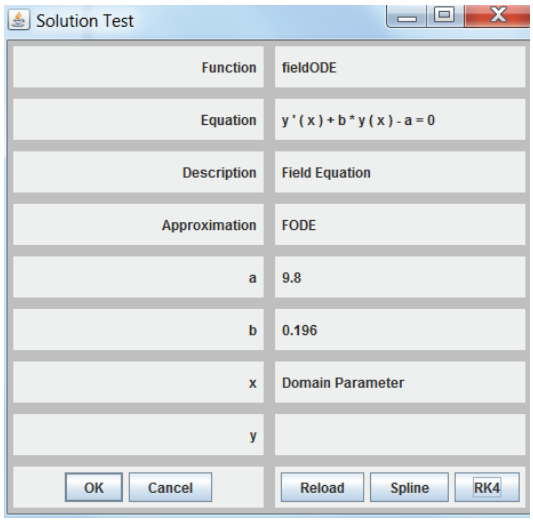
	Select an equation “Run DiffEQ Solution Test” From the Function drop-down menu
	The “Solution Test” tool is shown The information about the equation is shown The required parameters are shown with their values The RK4 button starts the Runge-Kutta tool
	The “Solution Approximation” screen shows the set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

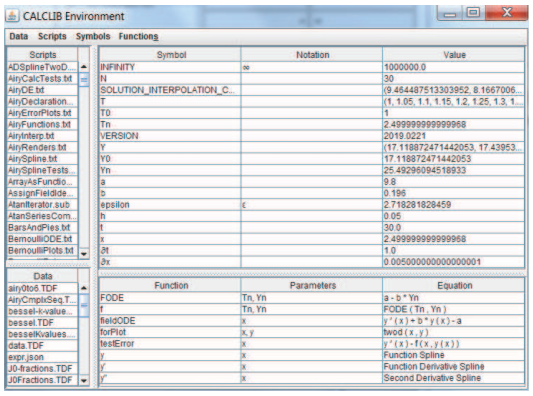
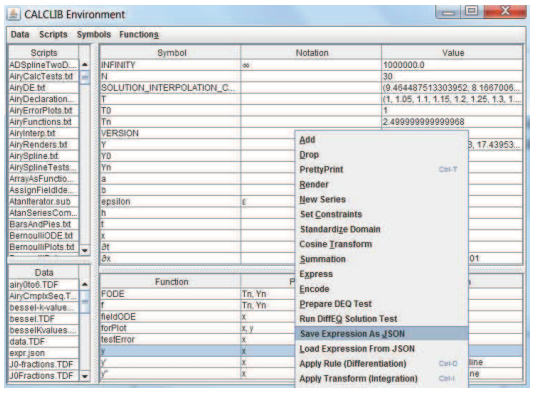
<table>
<tr>
<td> 
</td>
<td>The Interpolation displays show the quality of the best fit attempt made for the approximation

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

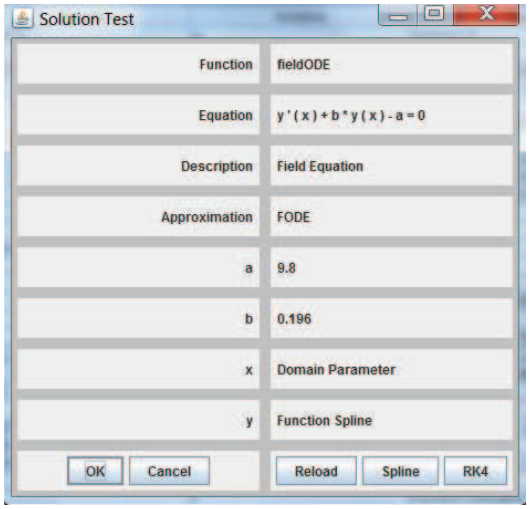
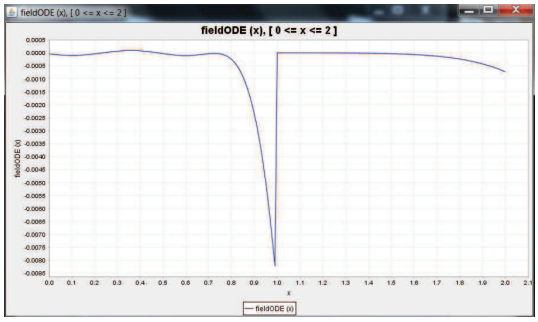
From the error plot the knot point should be chosen</td>
</tr>
<tr>
<td> 
</td>
<td>Use the Next button on the RK4 Solution Approximation screen which will present a dialog requesting the T0 for the knot

This will present the next Interpolation

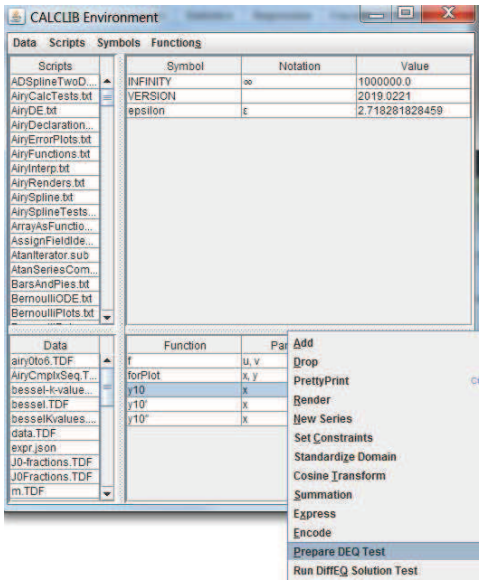
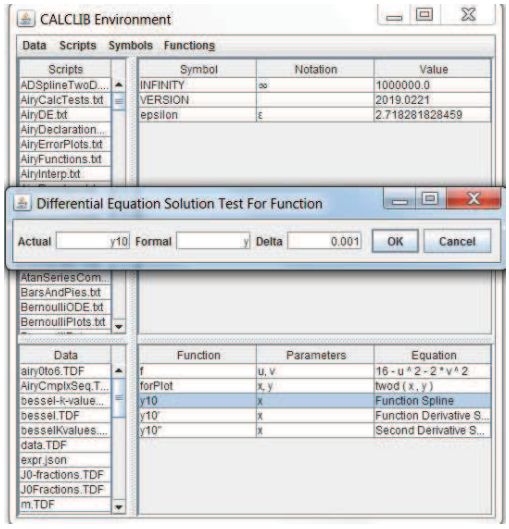
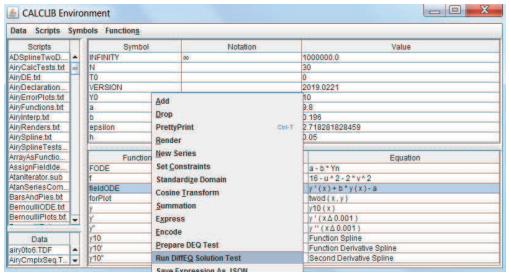
This repeats for as many knots as are required</td>
</tr>
<tr>
<td> 
</td>
<td>Now the Solution Test “Spline” button will generate the spline for the solution approximation</td>
</tr>
</table>

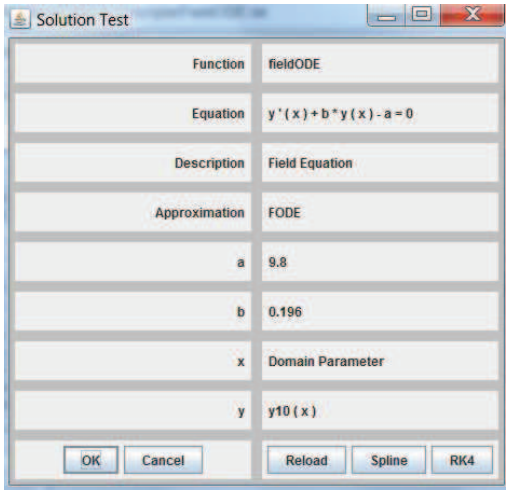
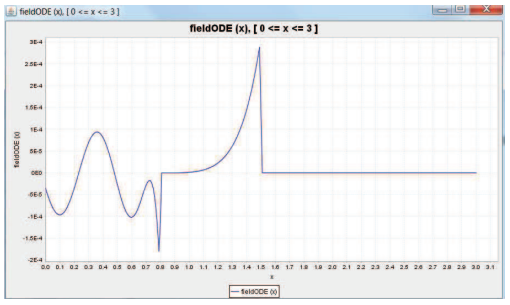
	Now the solution function “y” can be found in the functions list along with the derivatives y’ and y’’
	The function can be saved as a JSON expression for later import as a common singleton function Hence the solution is available as a function and the accuracy across the domain is know from the error plot

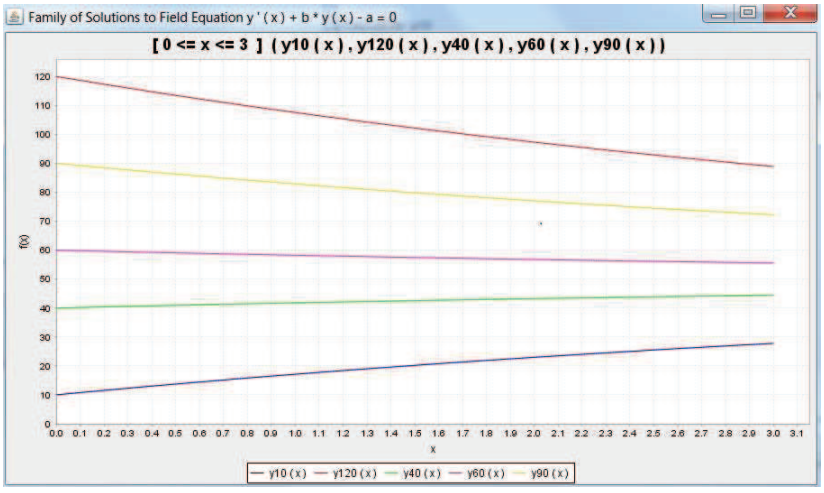
Test the Solution Quality

	The Solution Test screen now shows the solution "y" has a "Function Spline" available for test
	The OK button show a form for entering domain parameters
	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
	Enter domain description
	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

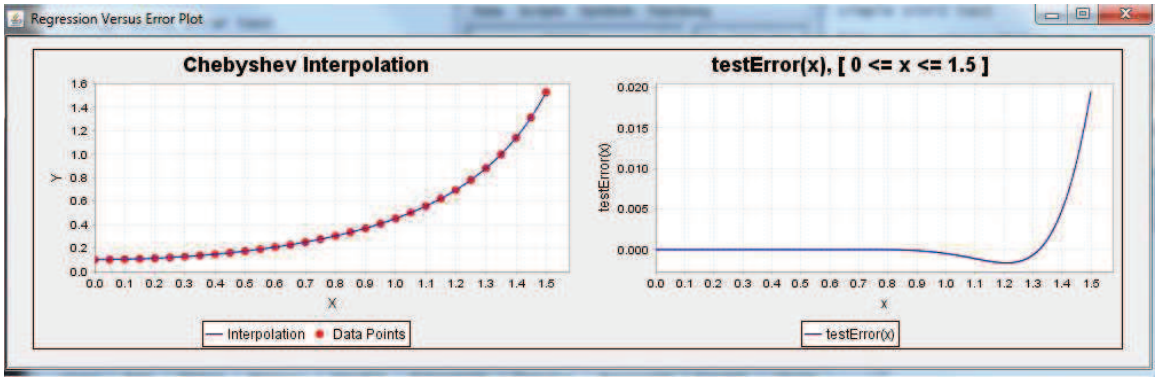
A simple example of use of **RungeKutta.txt**

// Riccati

!! u(x) = x / 2
Y0 = 0.1 ; T0 = 0 ; h = 0.05 ; N = 30
!! f(Tn,Yn) = Yn^2 + Tn * u(Tn) * Yn + u(Tn)

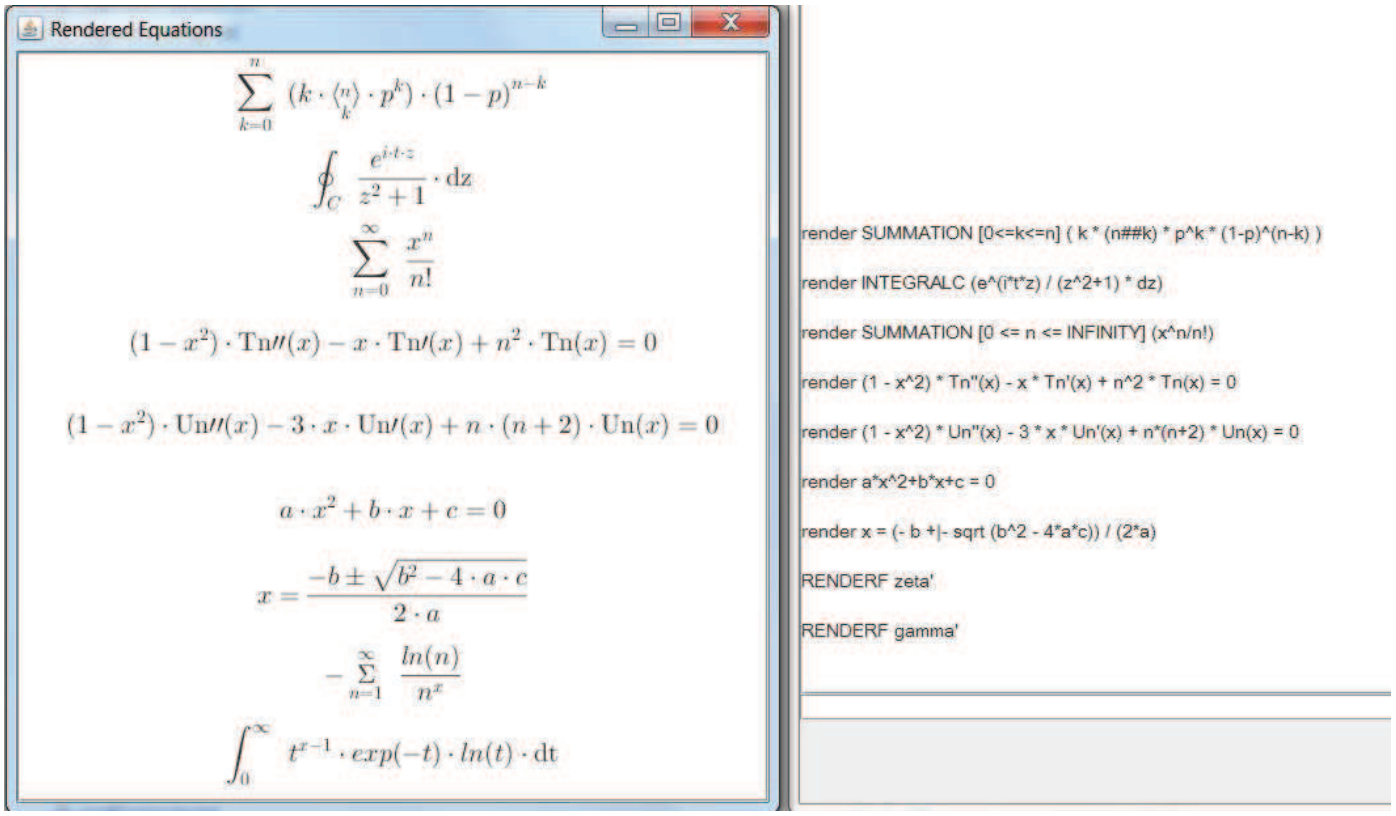
Y = (0.1, 0.10113171857626335, 0.10354424073431215, 0.10726542307837783,
0.11232678123700574, 0.11876514779425473, 0.12662446126207821,
0.1359577239358084, 0.14682917975976403, 0.15931677951018194,
0.1735150208399999, 0.18953827665792933, 0.20752475923532693,
0.22764131262151488, 0.250089287107206, 0.2751118334633071,
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.



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`