(1)

```
> f := x→sin(400 x^2);
```

$$f := x \mapsto \sin(400 \cdot x^2) \tag{2}$$

```
> # Cubic splines
  n := 10;
  segmentWidth := 1/n;
  gridPoints := [seq((i − 1)/n, i = 1 ..n + 1)];
  yValues := [seq(f(gridPoints[i]), i = 1 ..n + 1)] :;
```

$$n := 10$$

$$segmentWidth := \frac{1}{10}$$

$$gridPoints := \left[0, \frac{1}{10}, \frac{1}{5}, \frac{3}{10}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{7}{10}, \frac{4}{5}, \frac{9}{10}, 1\right] \tag{3}$$

(4)

```
> # Build tridiagonal matrix and right hand side (rhs) for the system
  triDiagonalMatrix := Matrix(n + 1, n + 1, (i, j) →
     if i = j and i > 1 and i < n + 1 then 2·(segmentWidth + segmentWidth)
    elif abs(i − j) = 1 then segmentWidth
    elif i = j then 1
    else 0
  end if) :;
  rhsVector := Vector(n + 1, (i) →

    if i = 1 or i = n + 1 then 0
     else 6·((yValues[i + 1] − yValues[i])/segmentWidth − (yValues[i] − yValues[i−1])/segmentWidth)
  end if) :;
```

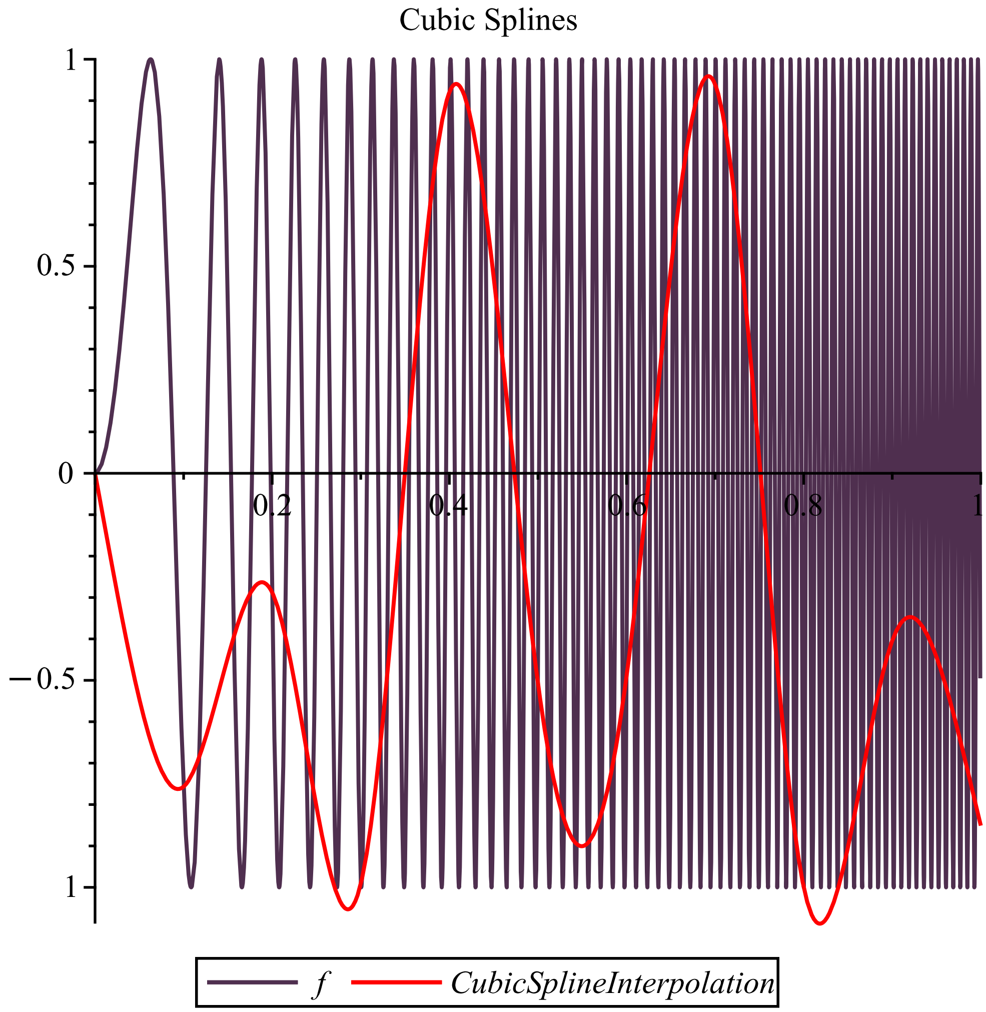
```
>  # Solve for gamma values
  with(LinearAlgebra) :
  gammaSolution := LinearSolve(triDiagonalMatrix, rhsVector) :;


  # Construct the cubic spline constants for each segment
  splineConstantsA := [seq(yValues[i + 1], i = 1 ..n)] :;
  splineConstantsB := [seq((yValues[i + 1] − yValues[i])/segmentWidth
    + segmentWidth·gammaSolution[i + 1]/3 + segmentWidth·gammaSolution[i]/6, i = 1 ..n)]
    :;
```

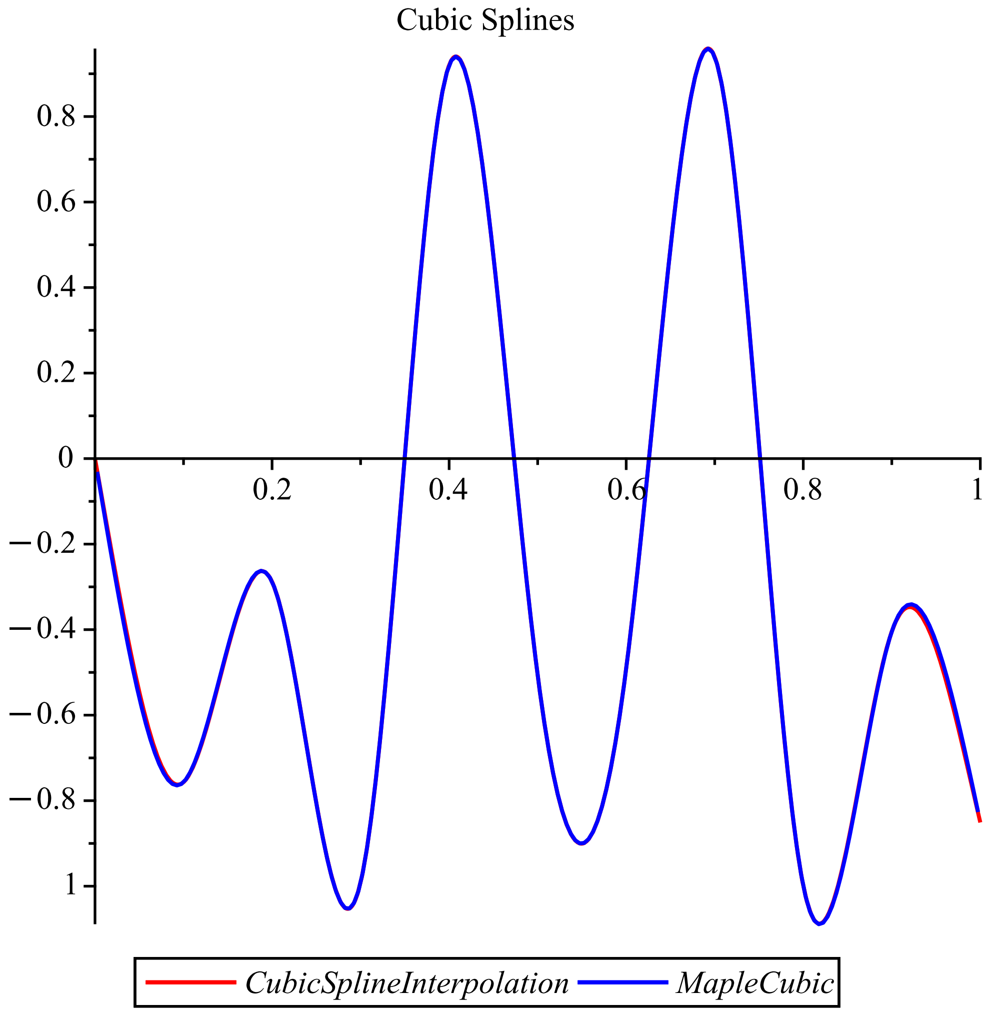
```
splineConstantsC := [seq((gammaSolution[i + 1] - gammaSolution[i]) / segmentWidth, i = 1 ..n)] : ;

# Define the piecewise cubic spline function
S := (i, x) -> splineConstantsA[i] + splineConstantsB[i]·(x - gridPoints[i + 1])
   + gammaSolution[i + 1]/2 ·(x - gridPoints[i + 1])^2 + splineConstantsC[i]/6 ·(x
   - gridPoints[i + 1])^3 :;
```

```
> # Define a function to create a cubic spline interpolation
  CubicSplineInterpolation := proc(x)
  local i;
  for  i from 1 to n do
    if (gridPoints[i] ≤ x and x ≤ gridPoints[i + 1]) then
      return S(i, x);
  end if;
  end do;
  end proc:
```

```
> with(CurveFitting) :;
  MapleCubic := t->Spline([seq(i, i = 0 ..1, 0.1)], [seq(f(i), i = 0 ..1, 0.1)], t, degree = 3) :;
```

```
> # Plot the original function and cubic spline interpolation
  plot([f, CubicSplineInterpolation], 0 ..1, title = "Cubic Splines", color = [violet, red], legend
     = [typeset(f), typeset(CubicSplineInterpolation) ]);
  plot([CubicSplineInterpolation, MapleCubic],  0 ..1, title = "Cubic Splines ", color = [red, blue],
      legend = [typeset(CubicSplineInterpolation), typeset(MapleCubic)]);
```


Cubic Splines
1
0.5
0
−0.5
1
0.2
0.4
0.6
0.8
1
f
CubicSplineInterpolation




```
> # We can already notice a problem
>
> # B-splines
  segmentRange := 0 ..1 :;
  stepSize := 1/10 :;
  controlPoints := 12 :;
  eps := 10^(-9) :;
  local i;
  xCoords := [ -2·eps, -eps, seq(i, i = segmentRange, stepSize), 1 + eps, 1 + 2·eps] :;
  yCoords := [ f(0), f(0), seq( f(i), i = segmentRange, stepSize), f(1), f(1) ] :;
```

$$i \qquad (5)$$

```
> # Define coefficients spline construction
  coeff0 := index → piecewise( index = 1, yCoords[1],
                1 < index and index < controlPoints,
```

$$-\frac{1}{2}\cdot yCoords[index+1] + 2\cdot f\left(\frac{1}{2}\cdot \mathrm{xCoords}[index+1] + \frac{1}{2}\cdot \mathrm{xCoords}[index+2]\right) - \frac{1}{2}\cdot yCoords[index+2],$$

$$index = controlPoints, yCoords[controlPoints+1]\Big) :;$$

```
> # Basis functions B−splines
BasisFunc[0] := (index, t) → piecewise(xCoords[index] ≤ t and t < xCoords[index + 1], 1, 0) :;
```

$$BasisFunc[1] := (index, t) \to \frac{(t - xCoords[index])\cdot \mathrm{BasisFunc}[0](index, t)}{(xCoords[index+1] - xCoords[index])} + \frac{(xCoords[index+2] - t)\cdot \mathrm{BasisFunc}[0](index+1, t)}{(xCoords[index+2] - xCoords[index+1])} :;$$

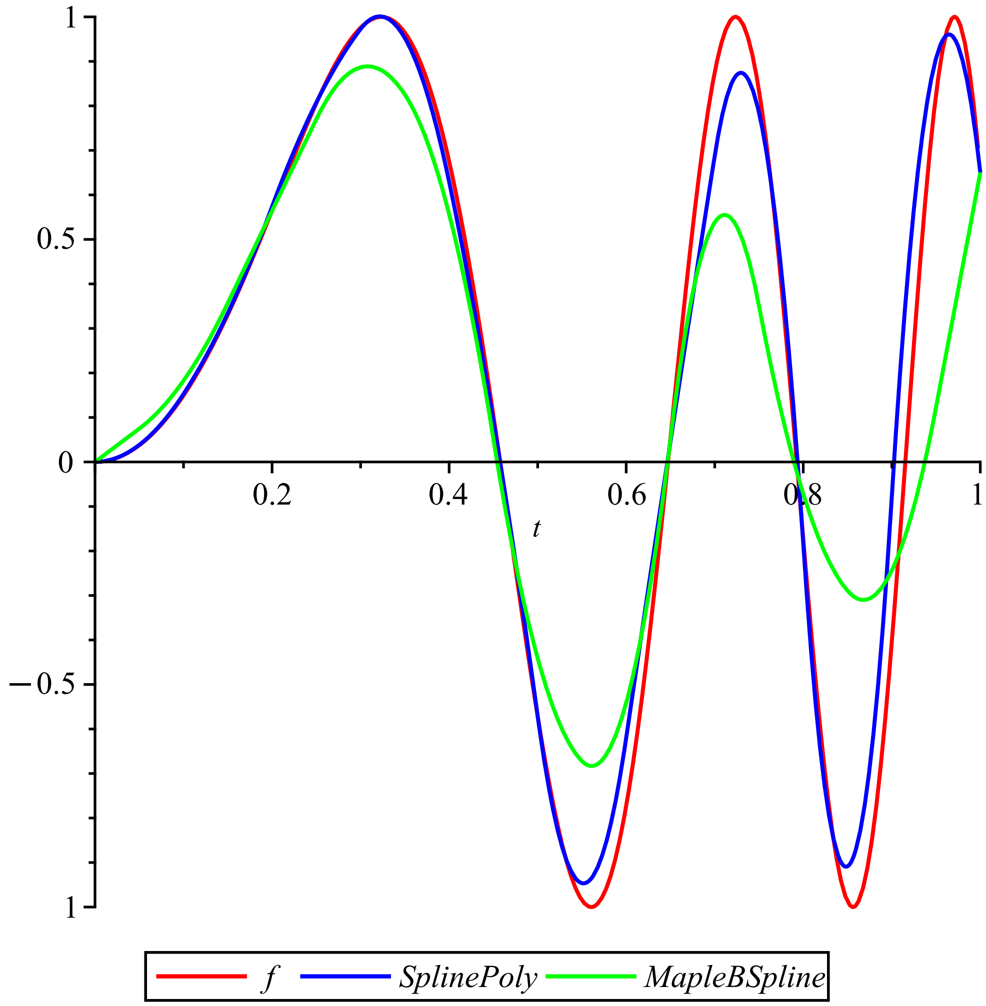
$$BasisFunc[2] := (index, t) \to \frac{(t - xCoords[index])\cdot \mathrm{BasisFunc}[1](index, t)}{(xCoords[index+2] - xCoords[index])} + \frac{(xCoords[index+3] - t)\cdot \mathrm{BasisFunc}[1](index+1, t)}{(xCoords[index+3] - xCoords[index+1])} :;$$

```
#Polynomial construction using the basis functions coefficients
SplinePoly := x→sum(coeff0(index)·BasisFunc[2](index, x), index = 1 .. controlPoints) ;
```

$$SplinePoly := x \mapsto \sum_{index=1}^{controlPoints} coeff0(index)\cdot BasisFunc_2(index, x) \tag{6}$$

```
> f := x→sin(15 x^2);

with(CurveFitting) :;
eps := 10^(−9) :;
MapleBSpline := x→BSplineCurve(
[−2·eps, −eps, seq(i, i = 0 ..1, 0.1), 1 + eps, 1 + 2·eps],
[f(0), f(0), seq(f(i), i = 0 ..1, 0.1), f(1), f(1)], x, order = 3) :;

plot([f(t), SplinePoly(t), MapleBSpline(t)], t = 0 ..1, color = [red, blue, green] , legend
    = [typeset(f), typeset(SplinePoly), typeset(MapleBSpline)]);

    # This difference in interpolation occurs due to the fact that the standard Maple library uses
    other coefficients c_i before B_{i, k}(x)
```

$$f := x \mapsto \sin(15\cdot x^2)$$



> # This difference in interpolation occurs due to the fact that the standard Maple library uses other coefficients c_i before B_{i, k}(x)

>

> # Error calculation

```
calculateMaxError := proc(originalFunc, interpolateFunc)
local errorFunc, errorList, point, interval, stepSize, points;
interval := 0 .. 1 :
stepSize := 1/100 :
points := [seq(i, i = interval, stepSize)] :
errorFunc := x → abs(originalFunc(x) − interpolateFunc(x));
errorList := [seq(errorFunc(point), point in points)];
return max(errorList) ;
end proc:
```

>

> # Test functions

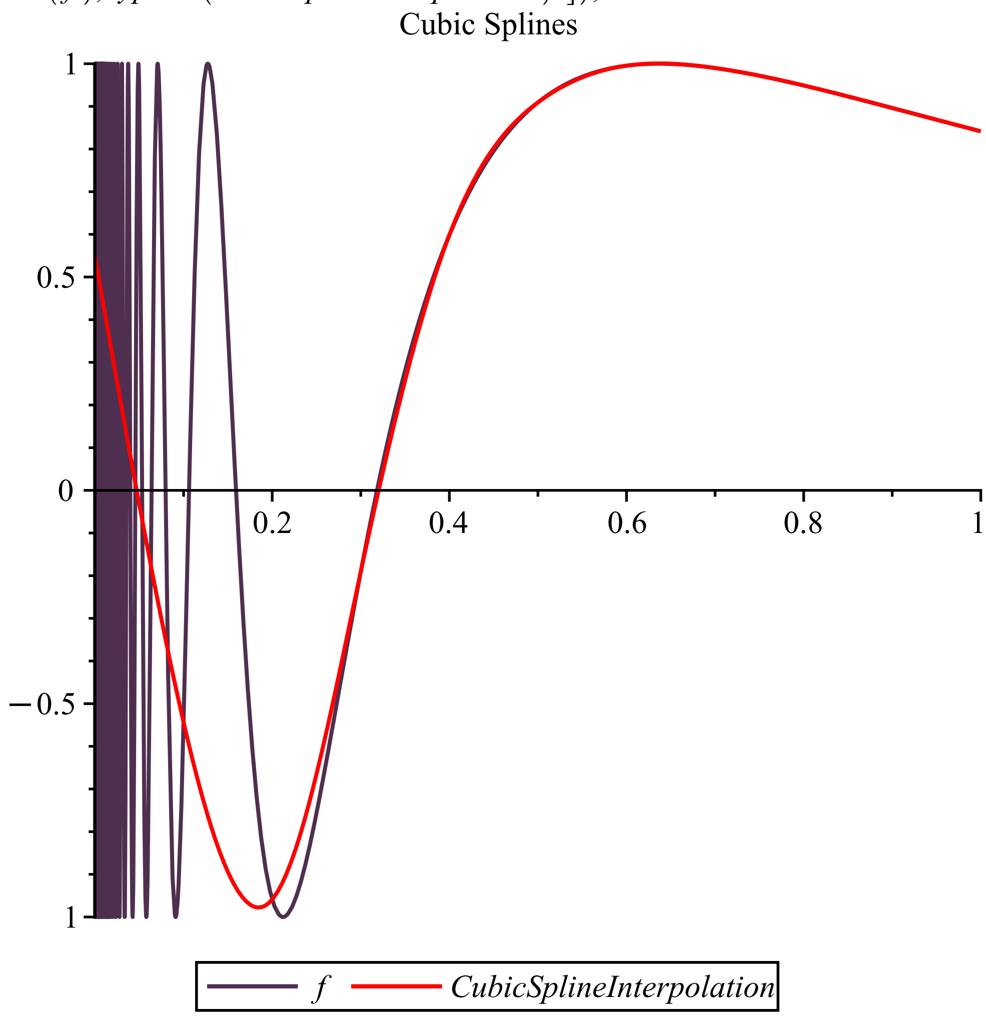
According to https://www.proven-reserves.com/CubicSplines.php (which is the part of "Numerical Analysis" by Richard L. Burden and J. Douglas Faires): "The cubic splines

interpolation algorithm does not work well for interpolation when the x values are large and have a large distance between them. Under these circumstances, cubic splines interpolation becomes very unstable making interpolations incorrect by many orders of magnitude". So let's test it, statement sounds like true, because we know only several points and have got only polynomial of the third degree, therefore, we cannot approximate the oscillating function well.

\> $f := x \rightarrow \sin\left(\frac{1}{x + 10^{-9}}\right);$

$$f := x \mapsto \sin\left(\frac{1}{x + \frac{1}{1000000000}}\right) \qquad \textbf{(7)}$$

\> *# Plot the original function and cubic spline interpolation*
plot([*f*, *CubicSplineInterpolation*], 0 ..1, *title* = "Cubic Splines", *color* = [*violet*, *red*], *legend* = [*typeset*(*f*), *typeset*(*CubicSplineInterpolation*)]);



\> *evalf*[10](*calculateMaxError*(*f*, *CubicSplineInterpolation*));

$$1.768099240 \qquad \textbf{(8)}$$

\> *# At the same time, according to the graph, it can be seen that in the interval where the function does not oscillate, cubic splines approximate the function well (well == error < 0.01).*

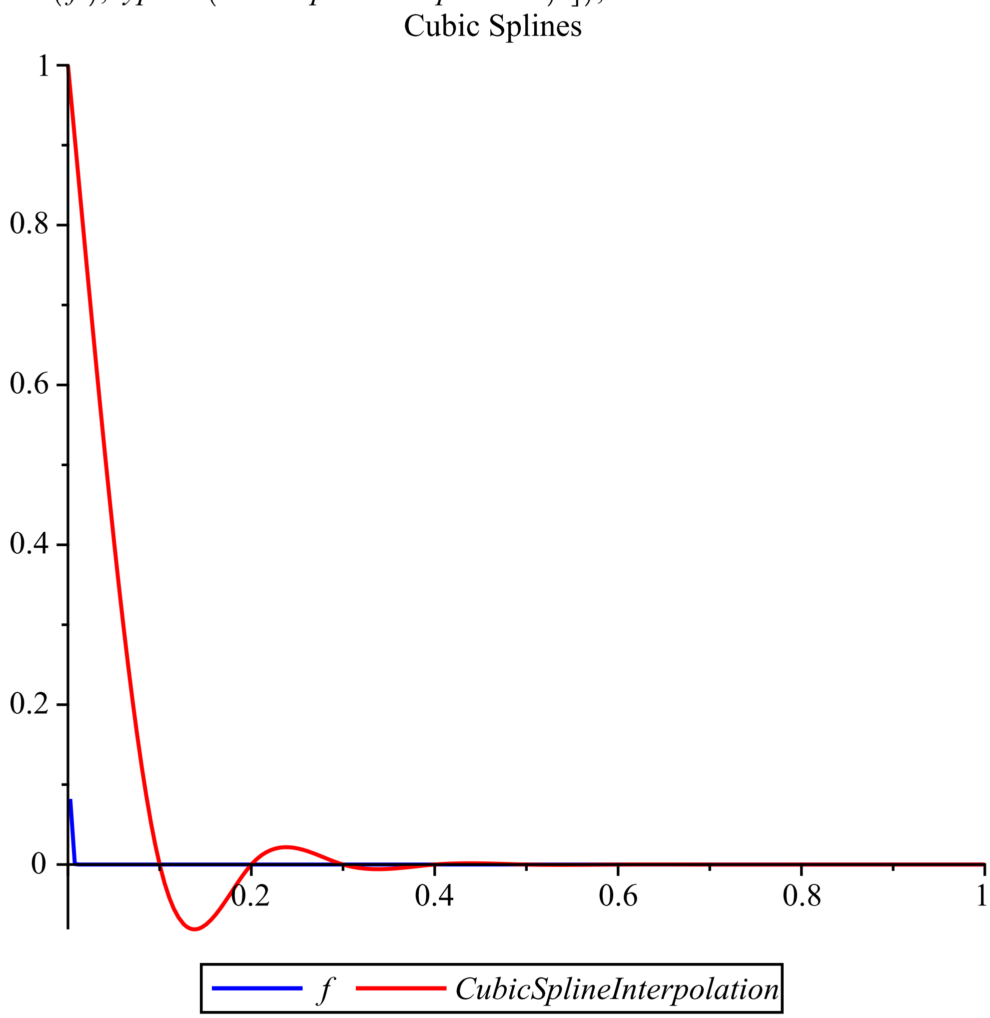
 # The same book claims: "The cubic splines algorithm produces surprising oscillation and instability when the independent variable scale is large and data measurements are sparse".

Let's test it too, this is more about the application of splines rather than specific functions, but it will also be interesting to consider.

> $f := x \rightarrow \exp(-1000\,x);$

$$f := x \mapsto \mathrm{e}^{-1000 \cdot x} \tag{9}$$

> $plot([f, CubicSplineInterpolation], 0..1, title = \text{"Cubic Splines"}, color = [blue, red], legend = [typeset(f), typeset(CubicSplineInterpolation)\ ]);$



> $evalf[10](calculateMaxError(f, CubicSplineInterpolation));$

$$0.8774053561 \tag{10}$$

> *# Both statements in question turned out to be true.*