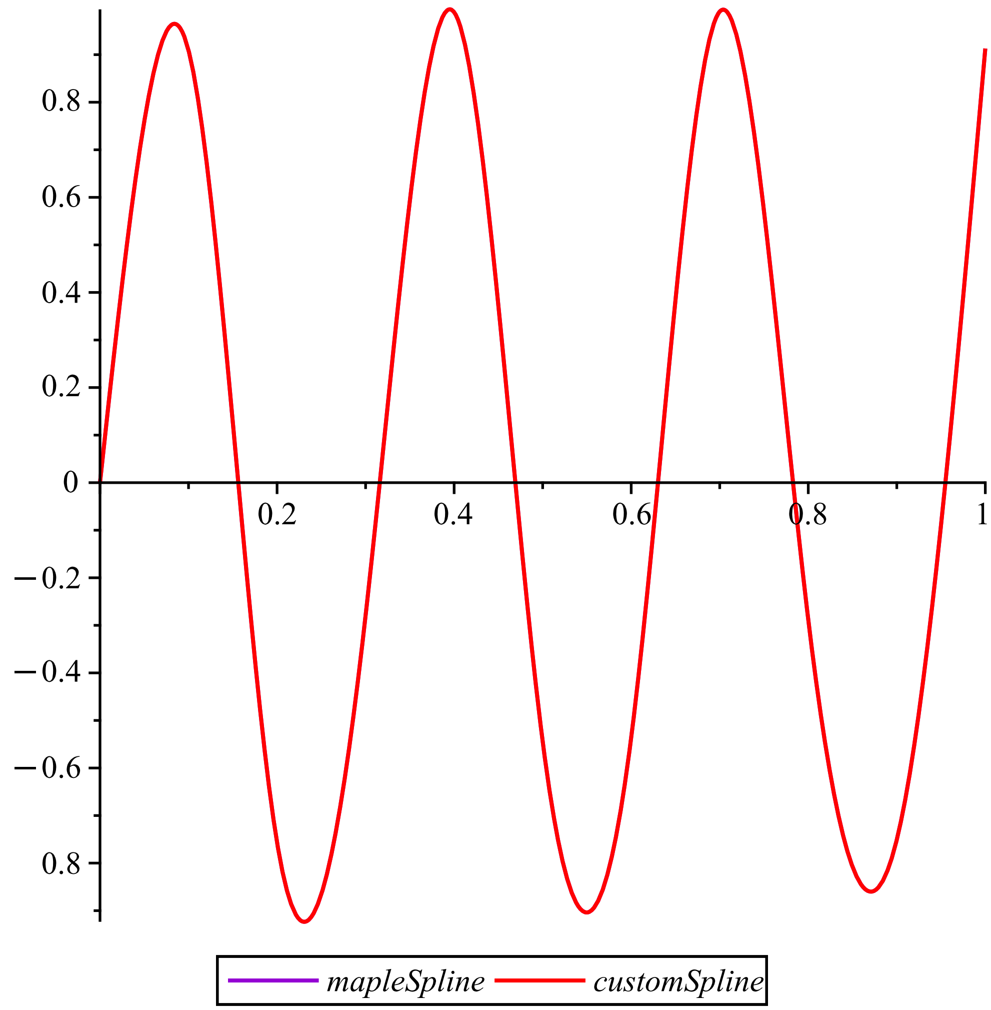
```
>
> with(CurveFitting) :
  with(LinearAlgebra) :
> CubicSpline := proc(f, step := 0.1)
    local i, x1, x2, splineFunc, points, A, b, n, coeffs, spl,  p1, p2, p3, p4, p5, p6, p7, p8, p9, p10;;

   points := [seq([i, f(i)], i = 0 ..1, step)];
    n := nops(points) − 1 ;

    A := Matrix(4 n, 4 n, 0);
    b := Vector(4 n, 0);



    for i from 1 to n do
       x1 := points[i, 1];
       x2 := points[i + 1, 1];
      A[2 i − 1, (4 i − 3) ..4 i] := ⟨x1^3, x1^2, x1, 1⟩;
      b[2 i − 1] := points[i, 2];
      A[2 i , (4 i − 3) ..4 i] := ⟨x2^3, x2^2, x2, 1⟩;
      b[2 i] := points[i + 1, 2];
      if i < n then
        A[2 n + i, 4 i − 3 ..4 i] := ⟨3 x2^2, 2 x2, 1, 0⟩;
        A[2 n + i, 4 i + 1 ..4 i + 4] := ⟨−3 x2^2, −2 x2, −1, 0⟩;
        A[3 n − 1 + i, 4 i − 3 ..4 i] := ⟨6 x2, 2 , 0, 0⟩;
        A[3 n − 1 + i, 4 i + 1 ..4 i + 4] := ⟨−6 x2, −2 , 0, 0⟩;
      end if;

   end do;
   A[ 4 n − 1, 1 ..3] := ⟨6 points[1][1], 2, 0⟩;
   b[4 n − 1] := 0;
   A[4 n, 4 n − 3 ..4 n − 2] := ⟨6 points[−1][1], 2⟩;
   b[4 n] := 0;
   coeffs := LinearSolve(A, b);
    # brute method, very bad; reduce??????
   p1 := x^3 * coeffs[1] + x^2 * coeffs[2] + x * coeffs[3] + coeffs[4];
   p2 := x^3 * coeffs[5] + x^2 * coeffs[6] + x * coeffs[7] + coeffs[8];
   p3 := x^3 * coeffs[9] + x^2 * coeffs[10] + x * coeffs[11] + coeffs[12];
   p4 := x^3 * coeffs[13] + x^2 * coeffs[14] + x * coeffs[15] + coeffs[16];
   p5 := x^3 * coeffs[17] + x^2 * coeffs[18] + x * coeffs[19] + coeffs[20];
   p6 := x^3 * coeffs[21] + x^2 * coeffs[22] + x * coeffs[23] + coeffs[24];
   p7 := x^3 * coeffs[25] + x^2 * coeffs[26] + x * coeffs[27] + coeffs[28];
   p8 := x^3 * coeffs[29] + x^2 * coeffs[30] + x * coeffs[31] + coeffs[32];
   p9 := x^3 * coeffs[33] + x^2 * coeffs[34] + x * coeffs[35] + coeffs[36];
  p10 := x^3 * coeffs[37] + x^2 * coeffs[38] + x * coeffs[39] + coeffs[40];
  spl := piecewise(x < 0.1, p1,  x < 0.2, p2, x < 0.3, p3, x < 0.4, p4, x < 0.5, p5,  x < 0.6, p6,
       x < 0.7, p7, x < 0.8, p8, x < 0.9, p9,  x ≤ 1, p10, 0);
```

```
    return MakeFunction(spl, x);
  end proc:
```

```
> customSpline := CubicSpline(x→ sin(20 x)) :
  mapleSpline := x→Spline([seq([i, sin(20 i)], i = 0 ..1, 0.1)], x, degree = 3) :
```

Warning, (in mapleSpline) `i` is implicitly declared local

```
> plot([mapleSpline , customSpline] , 0 ..1, color = ["DarkViolet", "Red"], legend = [mapleSpline ,
      customSpline] )
```



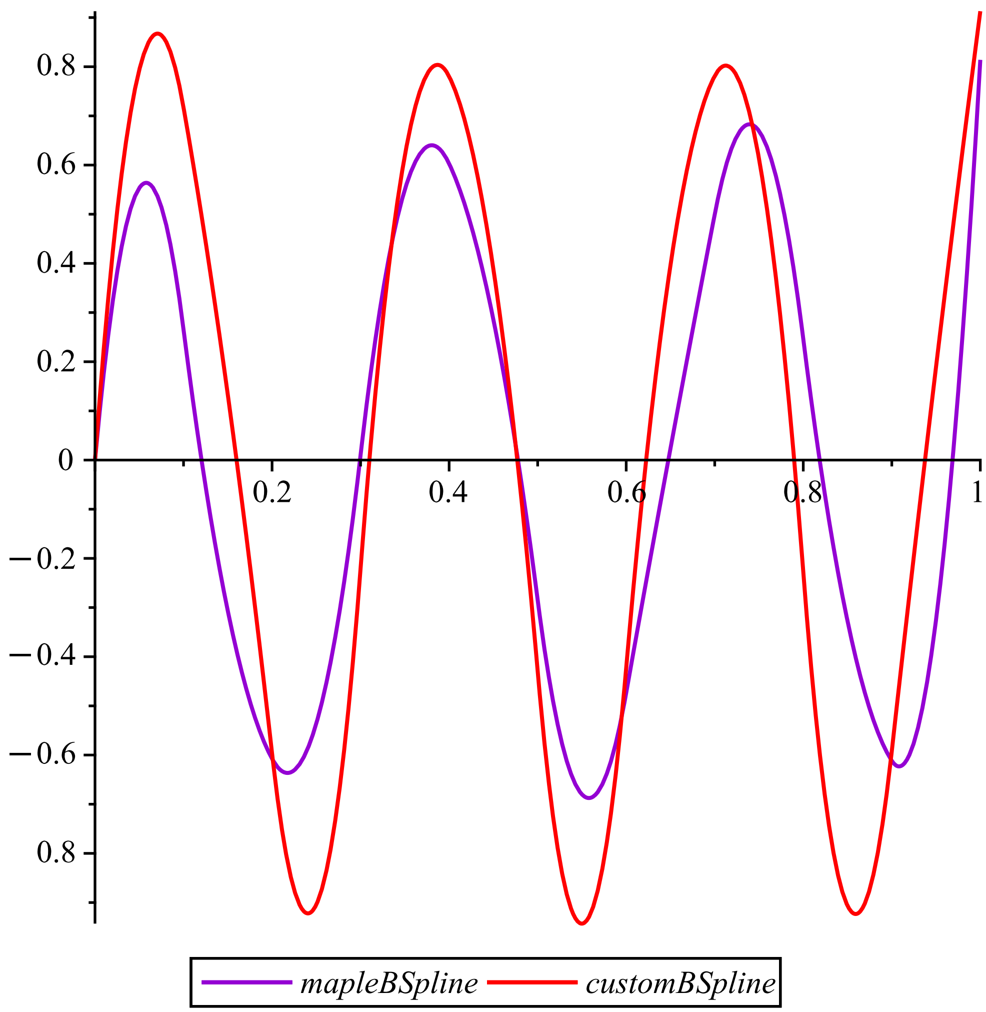

```
> EvalError := proc(f, spl)
    local i, maxDeviation;
    maxDeviation := max(seq(abs(f(i) − spl(i)), i = 0 ..1, 0.01));
   return maxDeviation;
  end proc:
```

```
> EvalError(mapleSpline, customSpline)
```

$$5.00306907014192 \times 10^{-10} \qquad (1)$$

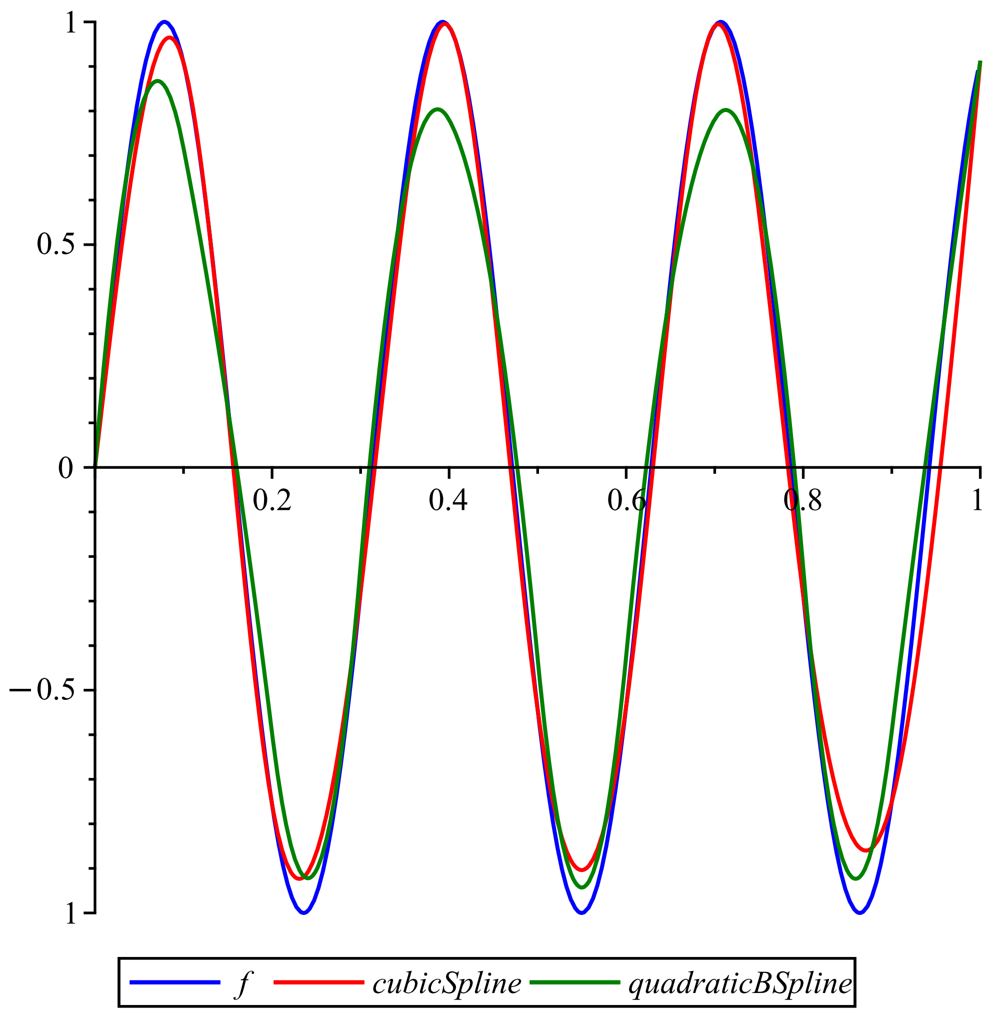
```
> QuadraticBSpline := proc(f, step := 0.1)
    local B, i, x0, x1, x2, knots, lmbd, n, customBSpline, k, eps;
   k := 3;
    eps := 0.0000001;
    # add border knots (?)
```

```
    knots := [seq(i, i=0 - k*eps..0 - eps, eps),  seq(i,   i= 0 ..1, step), seq(i, i=1 + eps..1
      + k· eps, eps)   ];
    n := nops(knots);
    lmbd := Vector(n + 10, 0);

 B := proc(i, k, t, T)
    if k=1 then
      if  T[i] ≤ t and t < T[i + 1]  then
         return 1;
       else
         return 0;
     end if;
    else
      return  (t - T[i])/(T[i + k - 1] - T[i]) B(i, k - 1, t, T) + (T[i + k] - t)/(T[i + k] - T[i + 1]) B(i + 1, k - 1, t,
      T);
    end if;
  end proc:


  for i from 2 to n - k  do
    if i=2 then
       lmbd[i] := f(knots[i]);
    elif i=n - k  then
       lmbd[i] := f(knots[i + 1]) ;
   else
       x0 := knots[i + 1];
       x1 := (knots[i + 1] + knots[i + 2])/2;
       x2 := knots[i + 2];
       lmbd[i] := (-f(x0) + 4 f(x1) - f(x2))/2;

   end if;
  end do;

  customBSpline :=  x→add(lmbd[i] B(i, k, x, knots), i=1 ..n - k);
  return customBSpline;
  end proc:

> customBSpline := QuadraticBSpline(x→sin(20 x)) :
  mapleBSpline := BSplineCurve([seq([i, sin(20 i)],  i= 0 ..1, 0.09)] , x, order=3, knots=[ 0,
       0, seq(i,   i= 0 ..1, 0.1), 1, 1.000001  ]) :
  mapleBSpline := MakeFunction(mapleBSpline[2], x) :
> plot([mapleBSpline, customBSpline] , 0 ..1, color = ["DarkViolet", "Red"], legend
       = [mapleBSpline, customBSpline]  )
```




```
> # sin(20x)
f := x→sin(20 x);
plot( [ f, CubicSpline( f ), QuadraticBSpline( f ) ] , 0 ..1, color = ["Blue", " Red", "Green"], legend
      = [ f, cubicSpline, quadraticBSpline]   ) ;
```

$$f := x \mapsto \sin(20 \cdot x)$$



> *EvalError*(*f*, *CubicSpline*(*f*));
EvalError(*f*, *QuadraticBSpline*(*f*));

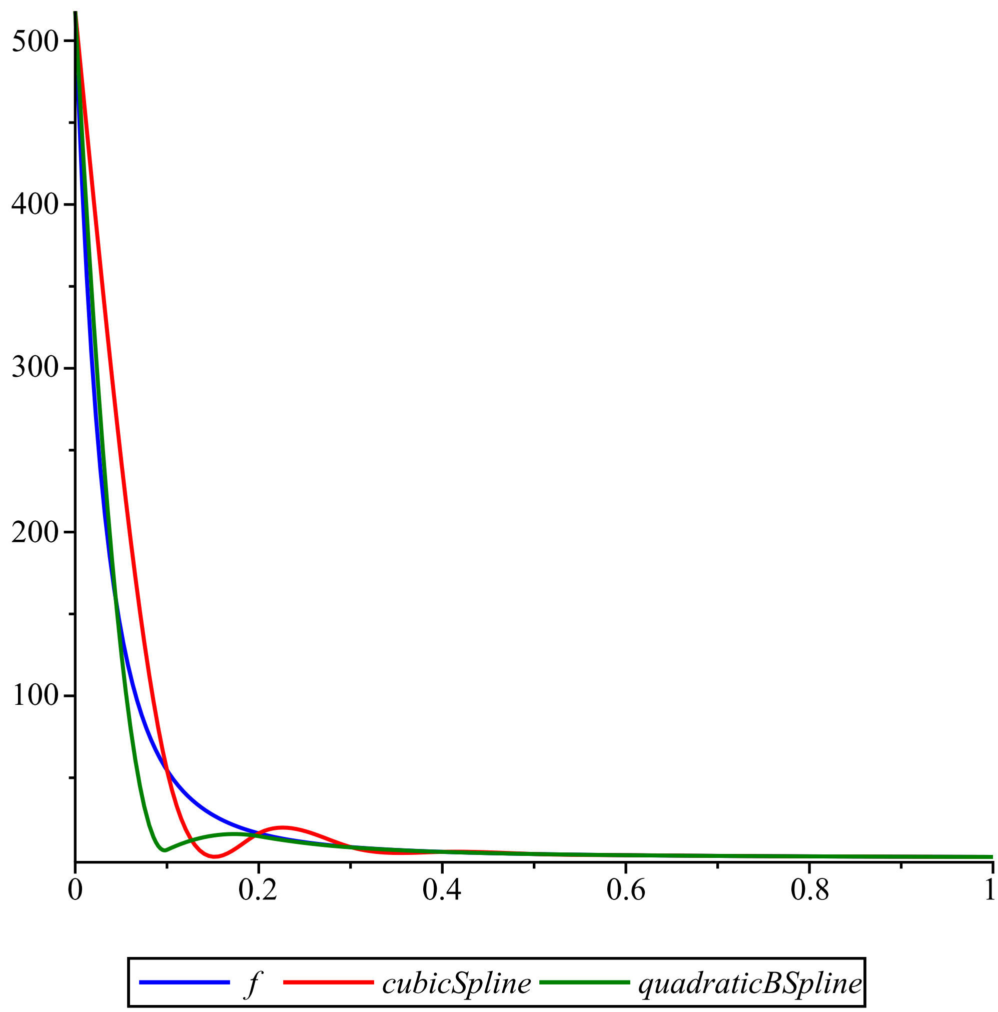
$$0.248065974415797$$

$$0.2093370978 \qquad (2)$$

> *# По выбранной нами метрике мы получаем, что квадратичный B-Spline лучше кубического сплайна (чего нельзя сказать однозначно по графику).*

> $f := x \rightarrow \exp\left(\frac{1}{(x+0.4)^2}\right);$
plot([*f*, *CubicSpline*(*f*), *QuadraticBSpline*(*f*)] , 0 ..1, *color* = ["Blue", " Red", "Green"], *legend* = [*f*, *cubicSpline*, *quadraticBSpline*]) ;

$$f := x \mapsto e^{\frac{1}{(x+0.4)^2}}$$



> *EvalError*(*f*, *CubicSpline*(*f*));
> *EvalError*(*f*, *QuadraticBSpline*(*f*));

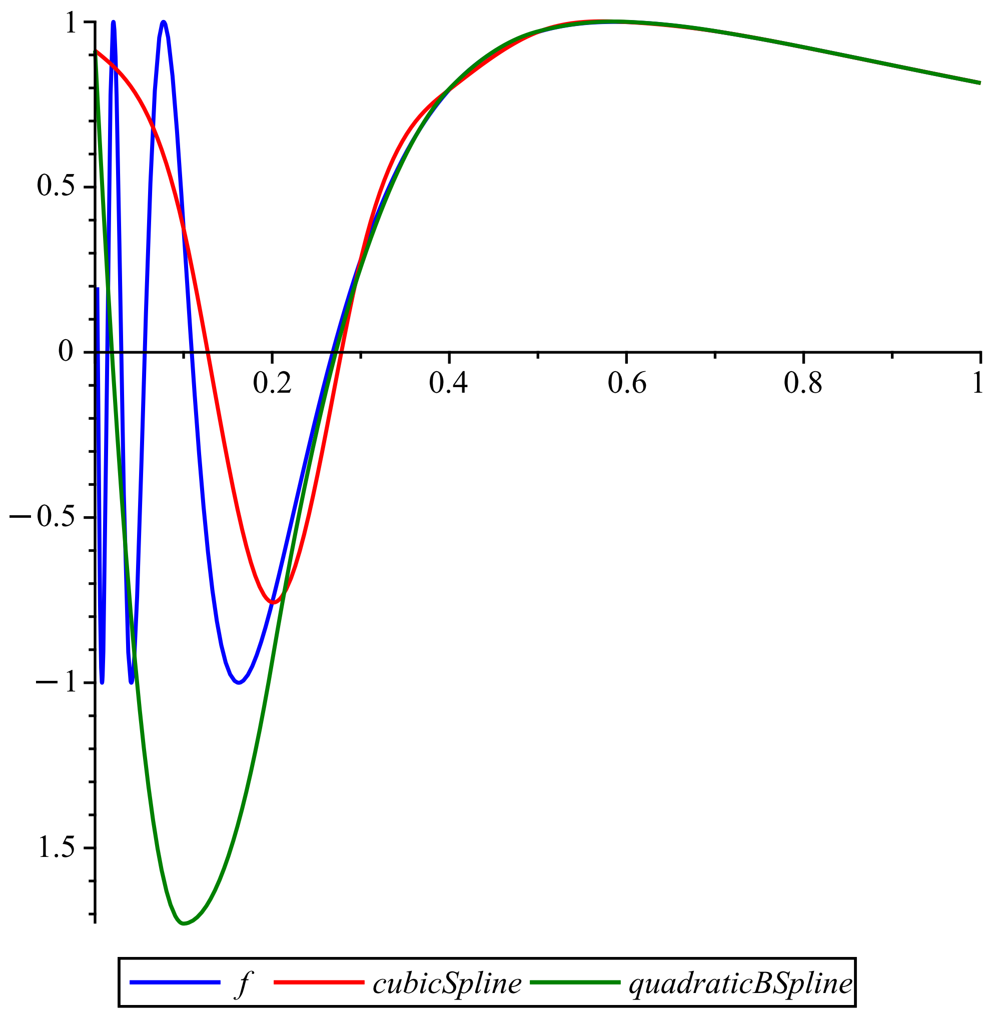
$$125.212669768365$$

$$55.67235005 \tag{3}$$

> # *На данном примере оба интерполянта немонотоны, однако интерполируемая функция монотона.*

> $f := x \rightarrow \sin\left(\dfrac{1}{x + 0.05}\right);$
> $plot([f, CubicSpline(f), QuadraticBSpline(f)], 0 ..1, color = [\text{"Blue"}, \text{" Red"}, \text{"Green"}], legend = [f, cubicSpline, quadraticBSpline])$;

$$f := x \mapsto \sin\left(\frac{1}{x + 0.05}\right)$$



> *EvalError*(*f*, *CubicSpline*(*f*));
EvalError(*f*, *QuadraticBSpline*(*f*));

1.79923370272734

2.611145430 **(4)**

> # *Последний пример наглядно показывает, как ведут себя сплайны при попытке аппроксимации сильноколеблющейся функции. Однако, в здесь меньшую ошибку имеет кубический сплайн.*