$restart$
$with(ArrayTools)$

$$[AddAlongDimension, Alias, AllNonZero, AnyNonZeros, Append, BlockCopy, CircularShift, ComplexAsFloat, Compress, Concatenate, Copy, DataTranspose, Diagonal, Dimensions, ElementDivide, ElementMultiply, ElementPower, Extend, Fill, FlipDimension, GeneralInnerProduct, GeneralOuterProduct, HasNonZero, HasZero, Insert, IsEqual, IsMonotonic, IsSubsequence, IsZero, Lookup, LowerTriangle, MultiplyAlongDimension, NumElems, Partition, Permute, PermuteInverse, RandomArray, ReduceAlongDimension, RegularArray, Remove, RemoveSingletonDimensions, Replicate, Reshape, Reverse, ScanAlongDimension, SearchArray, Size, SortBy, SuggestedDatatype, SuggestedOrder, SuggestedSubtype, Uncompress, UpperTriangle] \tag{1}$$

$segment := 0 .. 1$

$$segment := 0 ..1 \tag{2}$$

$step := 0.1$

$$step := 0.1 \tag{3}$$

Procedure that takes xs and ys and build an interpolant with them

$interpolate := \mathbf{proc}(xs, ys, func)$

$\mathbf{local}\, B\_0 := (i, x) \rightarrow piecewise(xs[i] \le x < xs[i+1], 1, 0);$

$\mathbf{local}\, B\_1 := (i, x) \rightarrow \frac{x - xs[i]}{xs[i+1] - xs[i]} \cdot B\_0(i, x) + \frac{xs[i+2] - x}{xs[i+2] - xs[i+1]} \cdot B\_0(i+1, x);$

$\mathbf{local}\, B\_2 := (i, x) \rightarrow \frac{x - xs[i]}{xs[i+2] - xs[i]} \cdot B\_1(i, x) + \frac{xs[i+3] - x}{xs[i+3] - xs[i+1]} \cdot B\_1(i+1, x);$

$\mathbf{local}\, x\_0 := j \rightarrow xs[j+1];$

$\mathbf{local}\, x\_1 := j \rightarrow \frac{xs[j+1] + xs[j+2]}{2};$

$\mathbf{local}\, x\_2 := j \rightarrow xs[j+2];$

$\mathbf{local}\ \lambda := j \rightarrow piecewise\Big(j = 1, func(xs[1]), 1 < j < nops(xs) - 2, \frac{1}{2}(-func(x\_0(j)) + 4 \cdot func(x\_1(j)) - func(x\_2(j))), j = nops(xs) - 2, func(xs[nops(xs) - 1])\Big);$

$\mathbf{local}\, interpolant := x \rightarrow \lambda(1) \cdot B\_2(1, x) + \lambda(2) \cdot B\_2(2, x) + \lambda(3) \cdot B\_2(3, x) + \lambda(4) \cdot B\_2(4, x) + \lambda(5) \cdot B\_2(5, x) + \lambda(6) \cdot B\_2(6, x) + \lambda(7) \cdot B\_2(7, x) + \lambda(8) \cdot B\_2(8, x);$

$\mathbf{return}\ interpolant;$

$\mathbf{end\ proc}$

$$interpolate := \mathbf{proc}(xs, ys, func) \tag{4}$$

$\mathbf{local}\ B\_0, B\_1, B\_2, x\_0, x\_1, x\_2, \lambda, interpolant;$

$B\_0 := (i, x) \rightarrow piecewise(xs[i] <= x \ \mathbf{and}\ x < xs[i+1], 1, 0);$

$B\_1 := (i, x) \rightarrow (x - xs[i]) * B\_0(i, x) / (xs[i+1] - xs[i]) + (xs[i+2] - x) * B\_0(i+1, x) / (xs[i+2] - xs[i+1]);$

$B\_2 := (i, x) \rightarrow (x - xs[i]) * B\_1(i, x) / (xs[i+2] - xs[i]) + (xs[i+3] - x) * B\_1(i+1, x)$

```
/ (xs[i + 3] − xs[i + 1]);
x_0 := j→xs[j + 1];
x_1 := j→1/2*xs[j + 1] + 1/2*xs[j + 2];
x_2 := j→xs[j + 2];
λ := j→piecewise(j = 1, func(xs[1]), 1 < j and j < nops(xs) − 2, − 1/2*func(x_0(j)) + 2
*func(x_1(j)) − 1/2*func(x_2(j)), j = nops(xs) − 2, func(xs[nops(xs) − 1]));
interpolant := x→λ(1)*B_2(1, x) + λ(2)*B_2(2, x) + λ(3)*B_2(3, x) + λ(4)*B_2(4, x)
+ λ(5)*B_2(5, x) + λ(6)*B_2(6, x) + λ(7)*B_2(7, x) + λ(8)*B_2(8, x);
return interpolant
end proc
```

 # Procedure that interpolates the given function on 0 .. 0.1, 0.1 .. 0.2, ... , 0.9 .. 1 segments with the step of 0.01
 # And returns an array of average absolute deviations on each segment

```
examine_my_spline := proc(func)
  local i, j, k, smaller_xs_grid, smaller_ys_grid, positive_difference;
  local zero_one_grid := seq(j, j = 0 .. 1, 0.1);
  local average_absolute_deviations := Array([ ]);
   for i from 2 to 11 do
     smaller_xs_grid := [seq(k, k = zero_one_grid[i − 1] ..zero_one_grid[i], 0.01)];
     smaller_ys_grid := map(func, smaller_xs_grid);
     positive_difference := x → abs(func(x) − interpolate(xs, ys, func)(x));
     average_absolute_deviations := Append(average_absolute_deviations,
     max(map(positive_difference, smaller_xs_grid)));
  end do;
  return average_absolute_deviations;
end proc
```

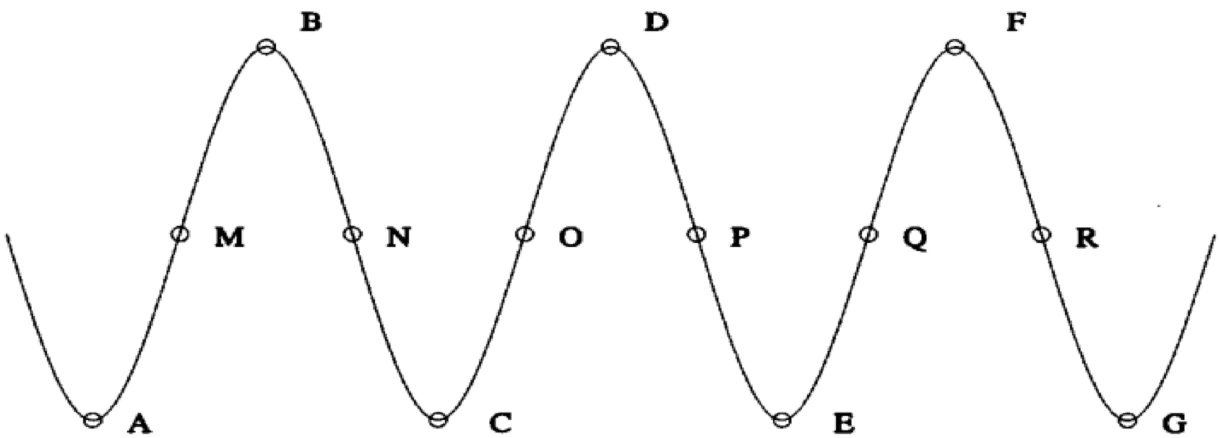
```
examine_my_spline := proc(func)                                                    (5)
  local i, j, k, smaller_xs_grid, smaller_ys_grid, positive_difference, zero_one_grid,
  average_absolute_deviations;
  zero_one_grid := seq(j, j = 0 ..1, 0.1);
  average_absolute_deviations := Array([ ]);
  for i from 2 to 11 do
    smaller_xs_grid := [seq(k, k = zero_one_grid[i − 1]..zero_one_grid[i], 0.01)];
    smaller_ys_grid := map(func, smaller_xs_grid);
    positive_difference := x→abs(func(x) − interpolate(xs, ys, func)(x));
    average_absolute_deviations := ArrayTools:-Append(average_absolute_deviations,
    max(map(positive_difference, smaller_xs_grid)))
  end do;
  return average_absolute_deviations
```

```
end proc
examine_maple_spline := proc(func)
   local i, j, k, smaller_xs_grid, smaller_ys_grid, positive_difference, maple_spline;
   local zero_one_grid := seq(j, j = 0 .. 1, 0.1);
   local average_absolute_deviations := Array([ ]);
   local pairs := Array([ ]);
    for i from 2 to 11 do
      smaller_xs_grid := [seq(k, k = zero_one_grid[i − 1] ..zero_one_grid[i], 0.01)];
      smaller_ys_grid := map(func, smaller_xs_grid);
      pairs :=  [seq([xs[i], ys[i]], i = 1 .. 11)];
      maple_spline := x → spline(pairs, x, 'quadratic');
      positive_difference := x → abs(func(x) − maple_spline(x));
      average_absolute_deviations := Append(average_absolute_deviations,
      max(map(positive_difference, smaller_xs_grid)));
   end do;
   return average_absolute_deviations;
end proc
```

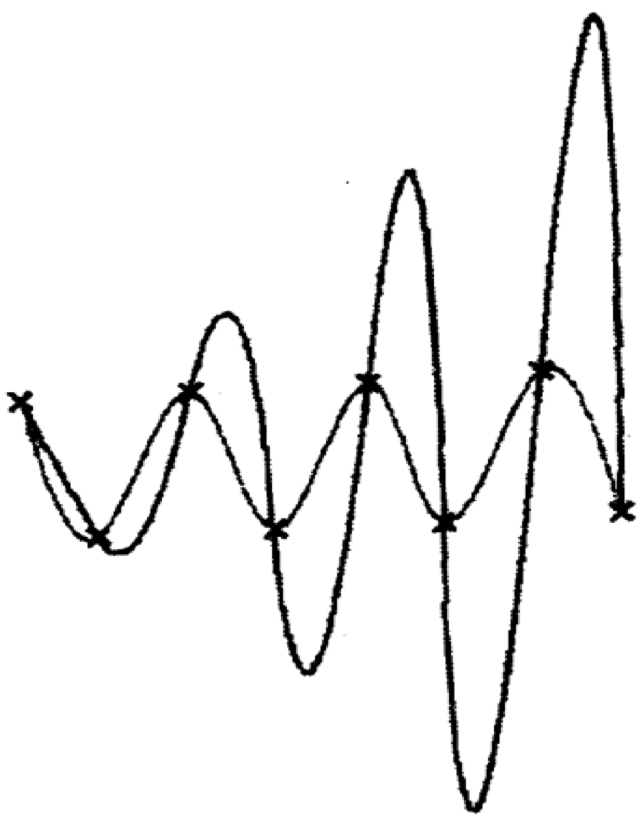
examine_maple_spline := proc(func) **(6)**

```
    local i, j, k, smaller_xs_grid, smaller_ys_grid, positive_difference, maple_spline, zero_one_grid,
    average_absolute_deviations, pairs;
    zero_one_grid := seq(j, j = 0 ..1, 0.1);
    average_absolute_deviations := Array([ ]);
    pairs := Array([ ]);
    for i from 2 to 11 do
        smaller_xs_grid := [seq(k, k = zero_one_grid[i − 1] ..zero_one_grid[i], 0.01)];
        smaller_ys_grid := map(func, smaller_xs_grid);
        pairs := [seq([xs[i], ys[i]], i = 1 ..11)];
        maple_spline := x→spline(pairs, x, 'quadratic');
        positive_difference := x→abs(func(x) − maple_spline(x));
        average_absolute_deviations := ArrayTools:-Append(average_absolute_deviations,
        max(map(positive_difference, smaller_xs_grid)))
    end do;
    return average_absolute_deviations
end proc
```

 # Test functions
 # 1. "Zigzagged data points and a desired interpolation curve"
 # Let's take a look at the article "Quadratic B-Spline Curve Interpolation " by Fuhua Cheng, Xuefu Wang and B. A Barsky
 # In the article they work with the set of points for which they want to get a curve akin to sine or cosine wave



 # However, this is what they get with quadratic B-splines



 # Let's try to conduct a similar expeiment and interpolate a function with a similar curve as the desired one from the article

$$cos\_44 := x \rightarrow \frac{\cos(44 \cdot x)}{2}$$

$$cos\_44 := x \mapsto \frac{\cos(44 \cdot x)}{2} \tag{7}$$

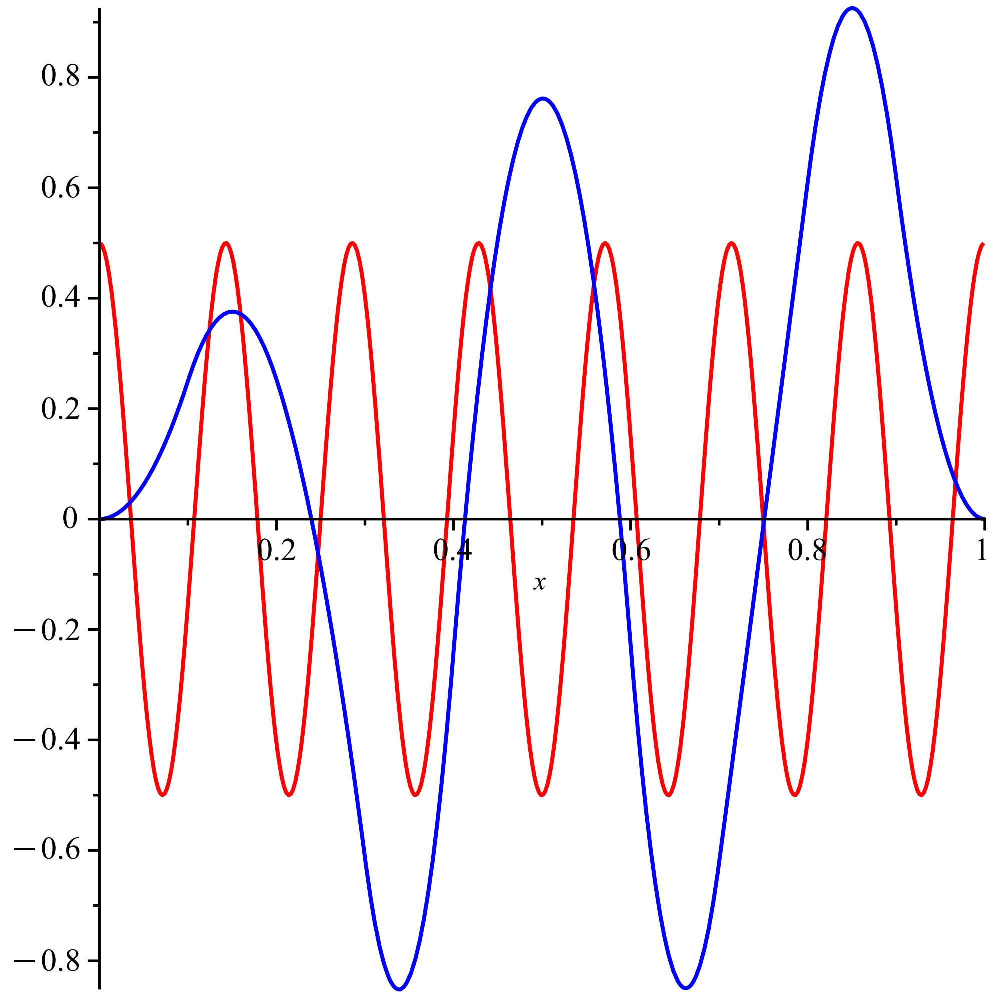
$xs := [seq(i, i = segment, step)]$

$$xs := [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0] \quad \textbf{(8)}$$

$ys := map(cos\_44, xs)$

$$ys := \left[\frac{1}{2}, -0.1536664350, -0.4055465070, 0.4029419788, 0.1578718774, -0.4999804132, 0.1494489532, 0.4081192618, -0.4003058812, -0.1620649511, 0.4999216543\right] \quad \textbf{(9)}$$

$plot([cos\_44(x), interpolate(xs, ys, cos\_44)(x)], x = segment, color = [red, blue])$



 # The similar problems with this function are seen when using quadratic B-spline interpolation. Let's also examine average absolute deviation on smaller grids:

$examine\_my\_spline(cos\_44)$

$$[0.6246271026, 0.6584104880, 1.016757920, 1.016757920, 1.261618476, 1.261618476, 1.029821944, 1.029821944, 1.010106161, 0.8635950867] \quad \textbf{(10)}$$

 # Average absolute deviation is very high (>0.6) on all the intervals

 # Let's compare our results for this function with built-in quadratic splines in Maple

 # The average absolute deviation is much lower but it takes a lot of effort **to** *construct regular quadratic splines, because a big system of equations must be solved*

In contrast, B-splines are much more easily constructed but are generally worse in interpolation

2. Runge's function
This example is described in more details in "cubic_splines.mw", however it's a good test function in case of B-splines as well, as B-Spline
Interpolation also metigates Runge's phenomenon quite well.

$runge\_function := x \rightarrow \frac{1}{1 + 25 \cdot x^2}$

$$runge\_function := x \mapsto \frac{1}{1 + 25 \cdot x^2} \quad \textbf{(11)}$$

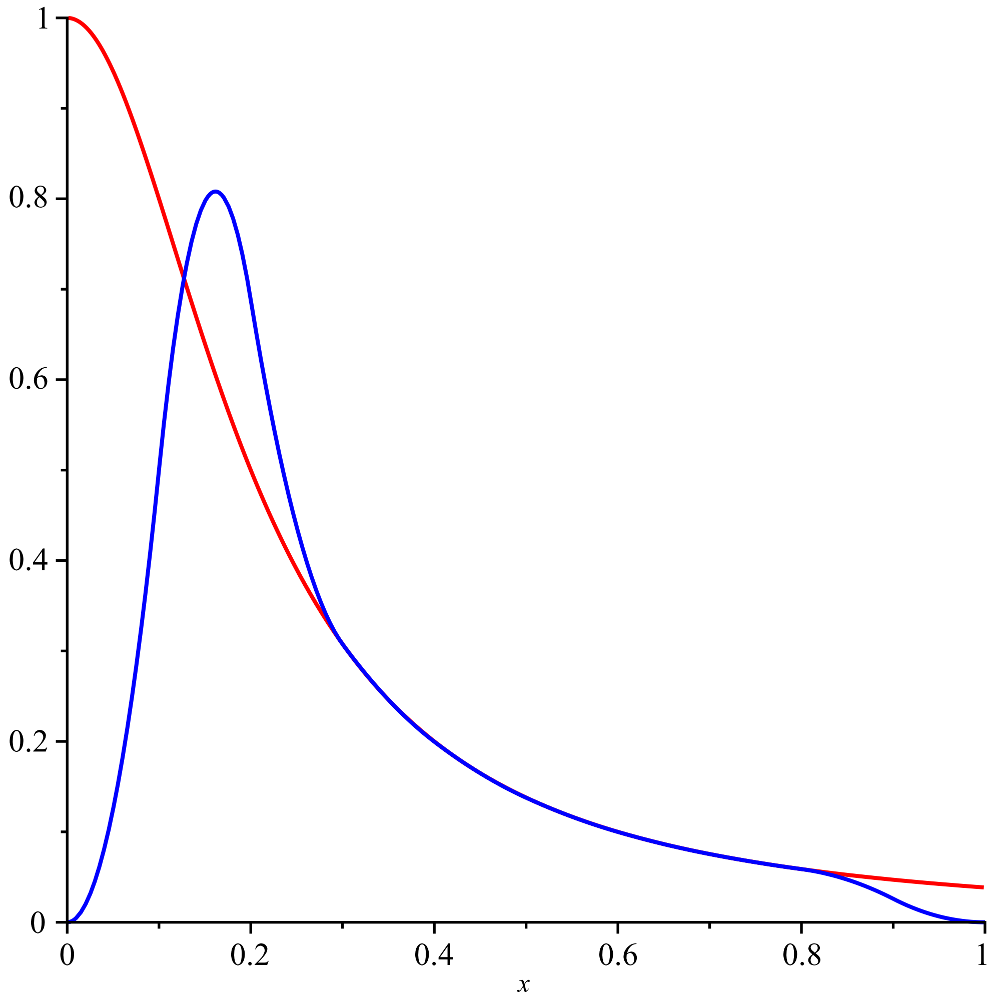
$xs := [seq(i, i = segment, step)]$

$$xs := [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0] \quad \textbf{(12)}$$

$ys := map(runge\_function, xs)$

$ys := [1, 0.8000000000, 0.5000000000, 0.3076923077, 0.2000000000, 0.1379310345, 0.1000000000, 0.07547169811, 0.05882352941, 0.04705882353, 0.03846153846]$ **(13)**

$plot([runge\_function(x), interpolate(xs, ys, runge\_function)(x)], x = segment, color = [red, blue])$



$examine\_my\_spline(runge\_function)$

$[1., 0.3000000000, 0.1883208255, 0.0005042464, 0.0003035357, 0.0001597769,$ **(14)**

0.00007587543, 0.00003690152, 0.02107039536, 0.03895195770]

 # Although, the approximation is not as good as with regular cubic and quadratic splines:

examine_maple_spline(*runge_function*)

[0.0304151198515676, 0.0106780842616218, 0.00294319500211893, 0.000678546604454267, 0.000228443821297941, 0.0000435487708747573, 0.0000226459870713014, $6.65638681154879 \times 10^{-6}$, 0.0000300527994015060, 0.000100864428234537] **(15)**