> # Построение кубического сплайна

```
> restart;
  n := 10 :
  grid := Array(1 ..(n + 1), i -> (i - 1)/n) :
> gamma_1 := 0 :
  gamma_(n + 1) := 0 :
  h := Array(1 ..n, i->(grid[i + 1] - grid[i])) :
  y := Array(1 ..(n + 1), i ->f(grid[i])) :
> initMatrix := proc(i, j)
    if (i = 1 and j = 1) then return 1;
    elif (i = (n + 1) and j = (n + 1)) then return 1;
    elif (i = j) then return 2·(h[i - 1] + h[i]);
    elif (abs(i-j) = 1 and i ≠ 1 and i ≠ n + 1) then return h[min(i, j)];
    else return 0;
    end if;
  end proc:

> initVector := proc(i)
    if (i = 1) then return gamma_1
    elif (i = (n + 1)) then return gamma_(n + 1)
    else return 6((y[i + 1] - y[i])/h[i] - (y[i] - y[i - 1])/h[i])
    end if;
  end proc:
> with(LinearAlgebra) :
  A := Matrix(n + 1, n + 1, initMatrix) :
  b := Vector(n + 1, initVector) :
  gamma_sol := LinearSolve(A, b) :
  K1 := Array(1 ..n, i->(y[i]/h[i] - gamma_sol[i]·h[i]/6)) :
  K2 := Array(1 ..n, i->(y[i + 1]/h[i] - gamma_sol[i + 1]·h[i]/6)) :

> S := Array(1 ..n, i->(x-> gamma_sol[i]·(grid[i + 1] - x)^3/(6·h[i])
      + gamma_sol[i + 1]·(x - grid[i])^3/(6·h[i]) + K1[i]·(grid[i + 1] - x) +
  K2[i]·(x - grid[i]))) :
> cubicSplineInterp := proc(x)
  local i;
  for i from 1 to n do
```

```
    if (grid[i] ≤ x and x ≤ grid[i + 1]) then
      return S[i](x);
    end if;
  end do;
end proc:
```

>

> # Построение интерполяции B-сплайнами:

```
> n := 10 :
grid := Array(1..(n + 1), i → (i − 1)/n) :
> EPS := 10^(−12) :
> getX := proc(i)
  if (i > n + 1) then return grid[n + 1] + (i − n)·EPS
  elif (i < 1) then return grid[1] + (1− i) ·EPS
  else return grid[i]
  end if;
end proc:
> getC := proc(j)
  local x_0 := getX(j + 1);
  local x_1 := (getX(j + 1) + getX(j + 2))/2;
  local x_2 := getX(j + 2);
  return (−f(x_0) + 4·f(x_1) − f(x_2))/2;
 end proc:
> B := proc(j, d, x)
  if (d = 0) then return piecewise(getX(j) ≤ x and x < getX(j + 1), 1, 0)(x)
  else return (x − getX(j))/(getX(j + d)−getX(j)) ·B(j, d − 1 , x) + (getX(j + 1 + d) − x)/(getX(j + 1 + d) − getX(j + 1)) ·B(j
      + 1, d − 1, x)
  end if;
end proc:
> BSplineInterp := proc(x)
  local i;
  local k;
  for i from 1 to n do
    if (getX(i) ≤ x and x ≤ getX(i + 1)) then
      return add(B(k, 2, x)·getC(k) , k = (i − 2) ..i);
    end if;
  end do;
 end proc:
>
> Функции для подсчета ошибки :
calculateError := proc(actual, expected)
```

```
    local check_grid := Array(1..(10·n + 1), i → (i − 1)/(10·n));
    local max_error := 0;
    local i;
    for i from 1 to (10·n + 1) do
      max_error := max(max_error, abs(actual(check_grid(i)) − expected(check_grid(i))));
    end do;
    return max_error;
  end proc:
```

> # **Пример(0)**
> # Сравним получившуюся интерполяцию кубическими сплайнами с готовым решением Maple:

```
> f := x→sqrt(x);
  yVals := Array(1..(n + 1), i →f(grid[i])) :
```

$$f := x \mapsto \sqrt{x} \tag{1}$$

```
> with(Student[NumericalAnalysis]) :
  points := [seq([grid[i], yVals[i]], i = 1..(n + 1))] :
  mapleCubicInterp := MakeFunction(expand(Interpolant( CubicSpline(points, independentvar
      = x))), x) :
  print(Ошибка получившейся интерполяции );
  err_01 := calculateError(cubicSplineInterp, f ) :
  evalf(err_01, 2);
  print(Ошибка решения Maple) ;
  err_02 := calculateError(mapleCubicInterp, f ) :
  evalf(err_02, 2);
```

Ошибка получившейся интерполяции

0.069

Ошибка решения Maple

0.068 **(2)**

> # Получили примерно одинаковые значения, что свидетельствует о правильности написанного алгоритма.

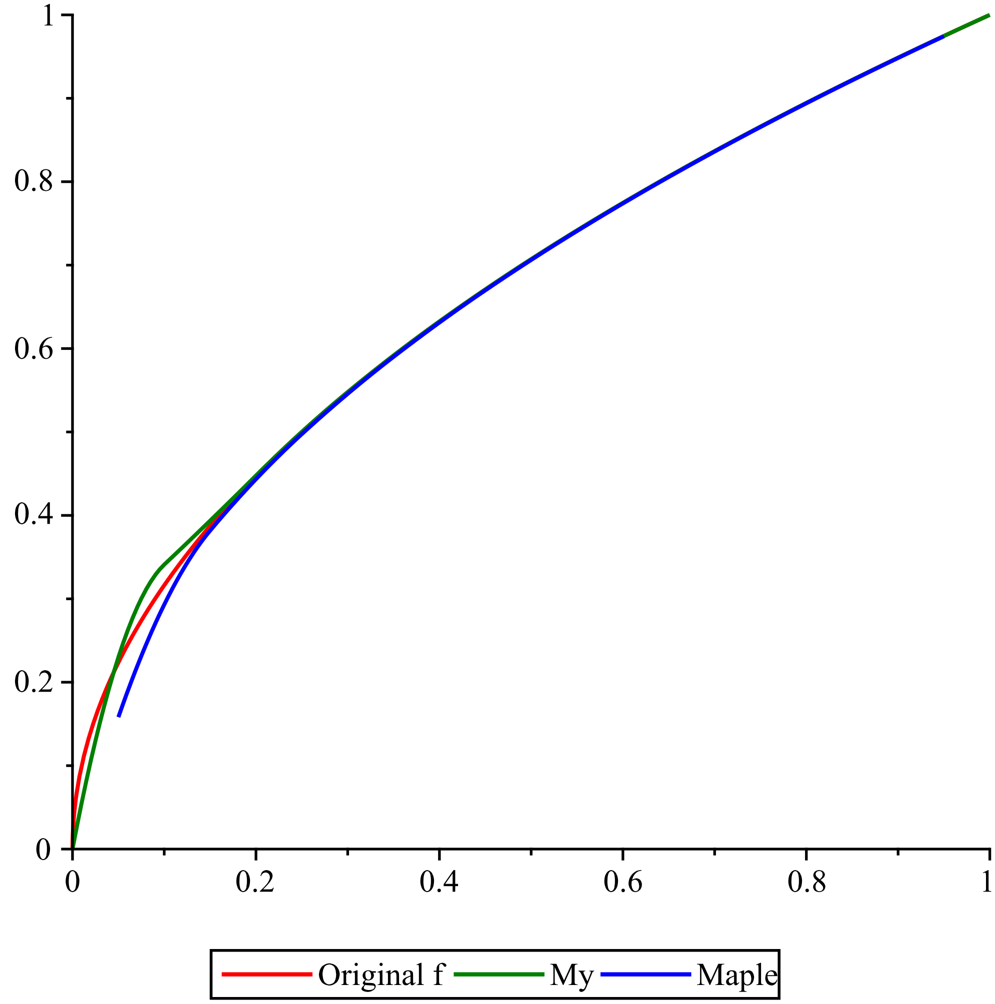
> # **Пример(1)**
> # Теперь сравним получившуюся интерполяцию B-сплайнами с готовым решением Maple:

```
> f := x→sqrt(x);
  yVals := Array(1..(n + 1), i →f(grid[i])) :
```

$$f := x \mapsto \sqrt{x} \tag{3}$$

```
> with(CurveFitting) :
  points := [seq([grid[i], yVals[i]], i = 1..(n + 1))] :
  mapleBSplineInterp := BSplineCurve(points, x, order = 3) :
```

```
plot( [ f, BSplineInterp, mapleBSplineInterp], 0 ..1, color = ["Red", "Green", "Blue"], legend
    = ["Original f", "My", "Maple"]);
```



> # Здесь мы видим, что как и у моего, так и у сплайна из пакета Maple появляютя неточности на левом конце отрезка. Также можно заметить, что мой сплайн построился полностью на отрезке, так как при посторении я доопределял его точками за границами отрезка, у которых абсциссы менялись с шагом EPS, а значения были равны значению функции на левом конце отрезка. В то же время сплайн из Maple "не достроен" рядом с концами отрезка, так как не было данных для доопределения.

> # ***Пример*** (**2**) :

В книге "С. П. Шарого "Курс вычислительных методов" в разделе про сплайны была упомянута функция Рунге, для которой при интерполяции полиномами наблюдается эффект осциляций. А также было отметечено, что "последовательность интерполяционных кубических сплайнов на равномерной сетке узлов всегда сходится к интерполируемой непрерывной функции". Значит, что ошибка при интерполяции

кубическими сплайнами для этой функции будет меньше. Давайте это проверим. Так как мы работаем на отрезке $[0, 1]$,
немного видоизменим функцию :

> $f := x \to \frac{1}{1 + 25\cdot(0.5 - x)^2}$;
> $yVals := Array(1..(n+1),\ i \to f(grid[i]))$:

$$f := x \mapsto \frac{1}{1 + 25\cdot(0.5 - x)^2} \quad \textbf{(4)}$$

> # Найдём максимальную ошибку при интерполяции кубическими сплайнами:

> $err\_1 := calculateError(cubicSplineInterp, f)$:
> $evalf(err\_1, 1)$;

$$0.003 \quad \textbf{(5)}$$

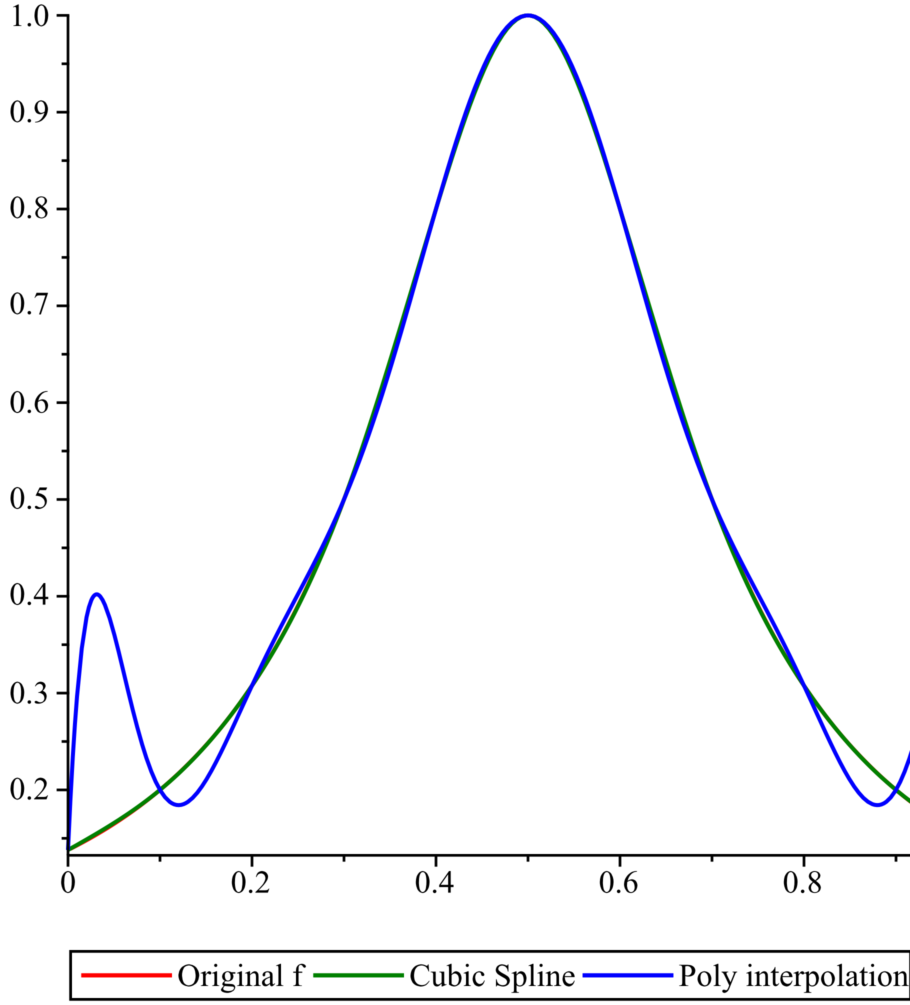
> # Для получения интерполяции полиномами будем использовать стандартные средства Maple. Найдем максимальную ошибку:

> $with(CurveFitting)$:
> $polyInterpolation := x \to PolynomialInterpolation(grid, yVals, x)$:
> $err\_2 := calculateError(polyInterpolation, f)$:
> $evalf(err\_2, 2)$;

$$0.25 \quad \textbf{(6)}$$

> # Мы видим, что ошибка отличается больше чем в 82 раза. Пронаблюдаем теперь это на графике:

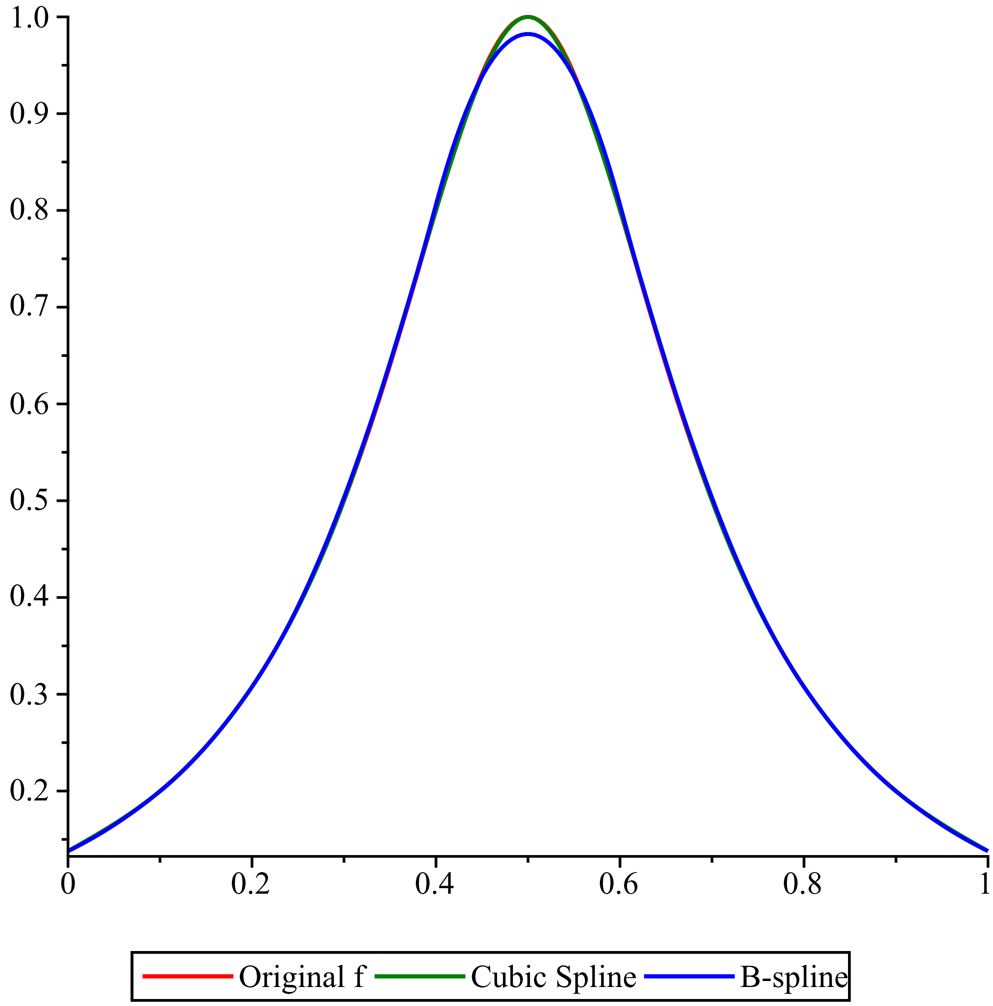
> $plot([f, cubicSplineInterp, polyInterpolation], 0..1, color = ["Red", "Green", "Blue"], legend = ["Original f", "Cubic Spline", "Poly interpolation"])$;



>

> # Таким образом, мы показали, что интерполяция кубическими сплайнами благодаря своим свойствам позволила избежать эффекта осциляций на концах отрезка. Теперь посмотрим, как с этой задачей справится B-spline:

> *plot*([*f*, *cubicSplineInterp* , *BSplineInterp*], 0 ..1, *color* = ["Red", "Green", "Blue"], *legend* = ["Original f", "Cubic Spline", "B-spline"]);



>

Оригинальную функцию на графике не видно, так как ее очень хорошо приближает кубический сплайн (ошибка 0.003).

>

> # У B−Spline не появляется эффектов осциляций на концах, он также лучше справляется с задачей, чем интерполяция полиномами, но при этом ошибка больше, чем у кубического сплайна, примерно в 6 раз:

> *err_3* ≔ *calculateError*(*BSplineInterp*, *f*) :
evalf(*err_3*, 2)

$$0.018 \tag{7}$$

> # Правда эту оценку нельзя считать полностью точной, так как наша погрешность составляет O(h^2), где h = 0.1 в нашем случае . Но на графике можно пронаблюдать, в каком месте B − Spline ошибается.

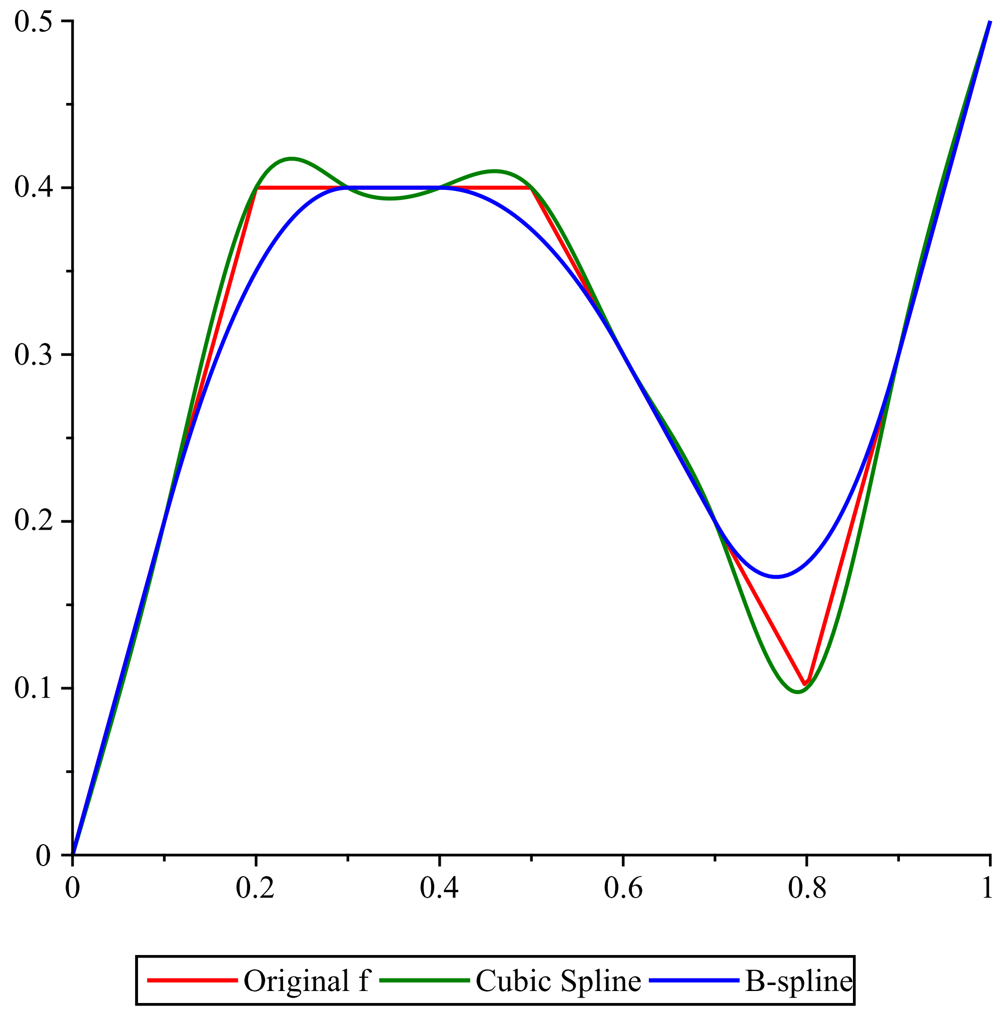
> # **Пример(3)** :

> # Посмотрим на пример из книги Tom Lyche and Knut Mørken, "Spline Methods".

> \# В разделе 5.2.2 "Numerical solution and examples" есть пример, который показывает, что кубические сплайны плохо приближают кусочно-линейные функции с острыми углами. Давайте его повторим, а также протестируем B-Сплайны.

> $f := x \to piecewise(x < 0.2, 2\,x, 0.2 \le x < 0.5, 0.4, 0.5 \le x < 0.8, 0.9 - x, 2\cdot(x - 0.75));$
plot([*f*, *cubicSplineInterp* , *BSplineInterp*], 0 ..1, *color* = ["Red", "Green", "Blue"], *legend* = ["Original f", "Cubic Spline", "B-spline"]);

$$f := x \mapsto \begin{cases} 2\cdot x & x < 0.2 \\ 0.4 & 0.2 \le x < 0.5 \\ 0.9 - x & 0.5 \le x < 0.8 \\ 2\cdot x - 1.50 & \textit{otherwise} \end{cases}$$



> \# BSpline хуже справился с задачей. Ошибка получилась в три раза больше:
print(*Ошибка кубического сплайна*);
err_01 := calculateError(*cubicSplineInterp*, f) :
evalf(*err_01*, 2);
print(*Ошибка BSpline*) ;

```
err_02 := calculateError(BSplineInterp, f ) :
evalf(err_02, 2);
```

Ошибка кубического сплайна

0.026

Ошибка BSpline

0.075 **(8)**

```
>
```