```
> restart
```

```
> Функция :
  f := x→sqrt(x)
```

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

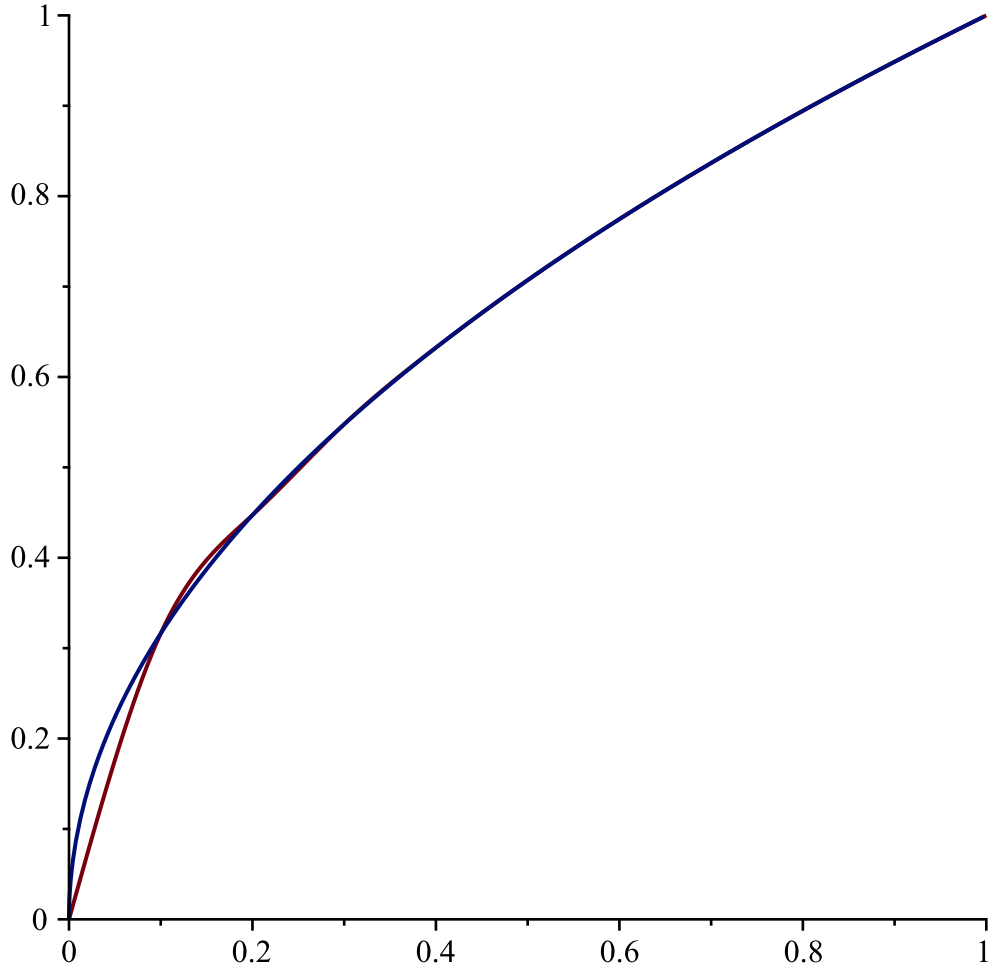
```
> Алгоритм интерполяции :
  n := 10;
  gamma_1 := 0;
  gamma_(n + 1) := 0;
```

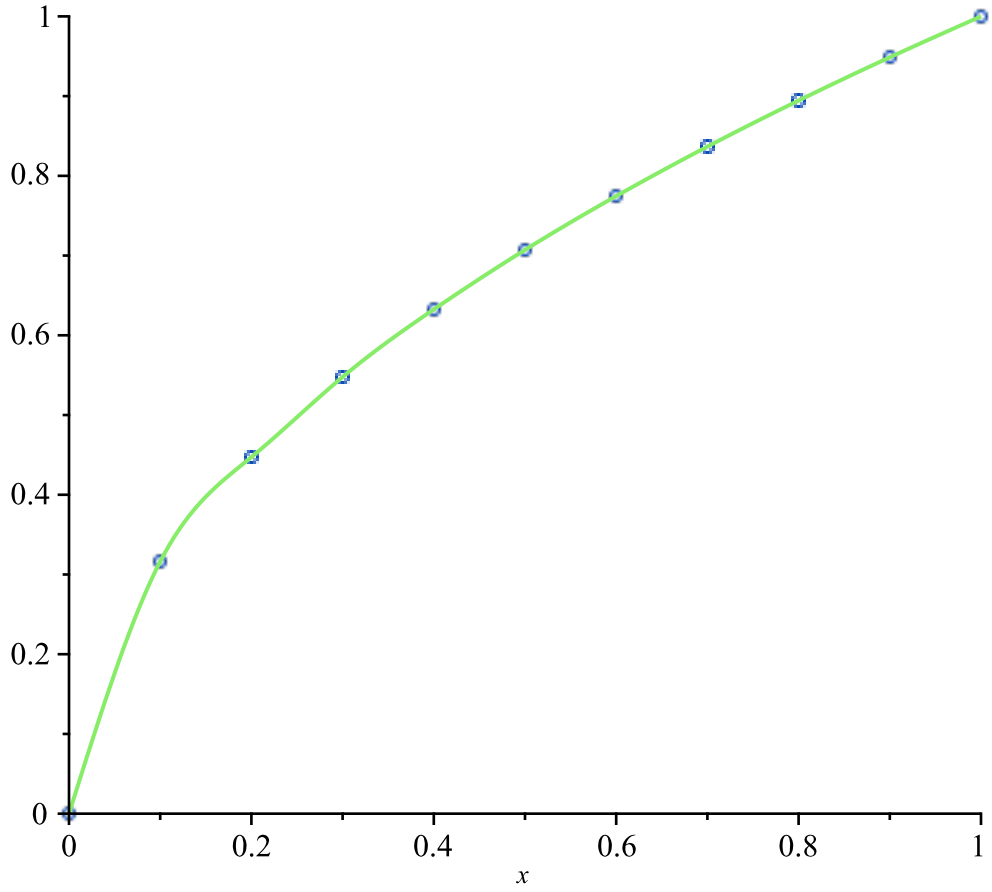
$$n := 10$$
$$gamma\_1 := 0$$
$$gamma\_(11) := 0 \tag{2}$$

```
> grid := Array(1..(n + 1), i → (i − 1)/n) :
  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]))  ) :
> approx := 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:
> Сравнение с алгоритмом из пакета Maple :
    with(Student[NumericalAnalysis]) :
     points := [seq([grid[i], y[i]], i = 1 ..(n + 1 ))] :
    mapleApprox := CubicSpline(points, independentvar = x) :
     plot({approx, f}, 0 ..1);
    Draw(mapleApprox)
```





Cubic spline interpolation with natural boundary conditions.

>

>

>

>