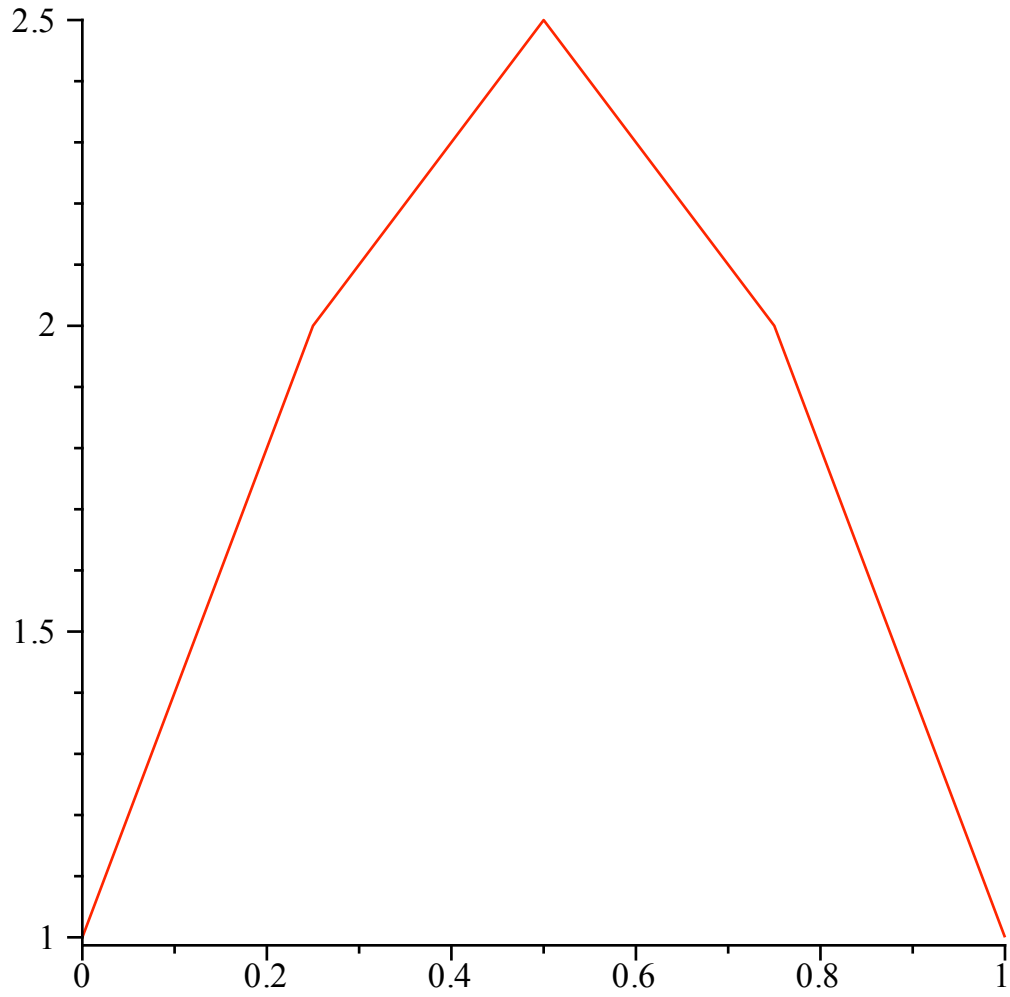
2. Interpolation

2.2 Interpolation linéaire

2.2.2 Algorithme

```
> restart :
> interpolin := proc( fX, x )
  local k, N, alpha :
  N := nops( fX ) - 1 :
  if ( x = 1 ) then
  return fX [ N + 1 ] :
  fi:
  k := floor ( N·x ) :
  alpha := x·N - k;
  fX [ k + 1 ] + ( fX [ k + 2 ] - fX [ k + 1 ]) ·alpha;
  end proc:
> evalf (interpolin( [ 1, 2, 2.5, 2, 1 ], 0.3 ) );
                                  2.10                                  (1.1.1.1)
> plot( [ [ 0, 1 ], [ 0.25, 2 ], [ 0.5, 2.5 ], [ 0.75, 2 ], [ 1, 1 ] ]);
```



>

2.2.3 Exercices

Exerxice 4

> $T := 23;$

$$T := 23 \qquad \textbf{(1.1.2.1.1)}$$

> $interpolin\left([1.308, 1.141, 1.005, 0.896, 0.804, 0.727, 0.661], \frac{(T-10)}{40-10} \right);$

$$0.9396000000 \qquad \textbf{(1.1.2.1.2)}$$

>

Exercice 5

> $interpolin2 := \mathbf{proc}(fX, x, a, b)$

$$interpolin\left(fX,\ \frac{(x-a)}{b-a} \right);$$

end proc:

```
>
```

```
> #Deuxième methode
```

$$interpolin2bis := \mathbf{proc}(fX, x, a, b)$$

local N, h, k :

$$N := nops(fX) - 1 :$$

if $(x = b)$ **then**

return $fX[N+1]$:

fi:

$$h := \frac{(b-a)}{N} :$$

$$k := \mathrm{floor}\left(\frac{(x-a)}{h} \right) :$$

$$fX[k+1] + \frac{(fX[k+2] - fX[k+1])}{h} \cdot (x - (a + k \cdot h)) :$$

end proc:

```
>
```

```
>
```

Exercice 7

```
> interpolin2([1.308, 1.141, 1.005, 0.896, 0.804, 0.727, 0.661], T, 10, 40);
```

0.9396000000 **(1.1.2.3.1)**

```
> interpolin2bis([1.308, 1.141, 1.005, 0.896, 0.804, 0.727, 0.661], T, 10, 40);
```

0.9396000000 **(1.1.2.3.2)**

```
>
```

Exercice 8

```
> (512.35 - 362.78)/(20 - 15)*(16 - 15) + 362.78;
```

392.6940000 **(1.1.2.4.1)**

```
> interpolin2([0, 227.04, 362.78, 512.35, 602.97, 901.67], 16, 0, 30);
```

462.4933333 **(1.1.2.4.2)**

```
> interpolin2([362.78, 512.35], 16, 15, 20);
```

392.6940000 **(1.1.2.4.3)**

```
>
```

Exercice 9

$$localisation := \mathbf{proc}(X, x)$$

local i;

```
    if (x = X[nops(X)]) then
    return nops(X) − 1 :
    fi:
    i := 1;
    while (x ≥ X[i + 1]) do
    i := i + 1 :
    od:
    i := i − 1 :
    end proc:
> localisation([0, 0.2, 1, 1.5, 10, 100], 2);
```

3 **(1.1.2.5.1)**

```
> localisationbis := proc(X, x)
    local i, N, k;
    N := nops(X) − 1;
    if (x = X[N + 1]) then
    return N :
    fi:
    for k from 1 to N do
    if ((x ≥ X[k]) and (x < X[k + 1])) then
    i := k − 1;
    fi:
    od:
    i;
    end proc:
> localisationbis([0, 0.2, 1, 1.5, 10, 100], 2);
```

3 **(1.1.2.5.2)**

```
>
>
```

Exercice 10

```
> interpolin3 := proc(X, fX, x)
    local k, N :
    k := localisation(X, x);
    N := nops(X) − 1 :
    if (k = N) then
    return fX[N + 1] :
    fi:
    interpolin2([fX[k + 1], fX[k + 2]], x, X[k + 1], X[k + 2]);
    end proc:
> interpolin3([0, 10, 15, 20, 22.5, 30], [0, 227.04, 362.78, 512.35, 602.97,
      901.67], 16);
```

392.6940000 **(1.1.2.6.1)**

```
>
```

2.3 Interpolation cubique

2.3.3 Algorithme

```
> interpolcub := proc( fX, x )
  local N, k, alpha, s :
  N := nops( fX ) - 3 :
  if ( x = 1 ) then
  return fX [ N + 2 ] :
  fi:
  k := floor ( N·x ) :
  alpha := N·x - k :
  s := - fX [k + 1 ]·alpha·(1 - alpha )·(2 - alpha ) / 6 :
  s := s + fX [k + 2 ]·(1 + alpha )·(1 - alpha )·(2 - alpha ) / 2 :
  s := s + fX [k + 3 ]·(1 + alpha )·alpha·(2 - alpha ) / 2 :
  s := s - fX [k + 4 ]·(1 + alpha )·alpha·(1 - alpha ) / 6 :
  end proc:
>
```