

## Interpolation

Files for the present tutorial can be found using following links:

- [Header file inter.h](#)
- [Source file inter.cpp](#)

### 1. Interpolation in engineering practice

Today, we will deal with the issue of interpolation. Suppose that from a tank that can be placed at different heights, water flows out of the pipe until it is empty. We made the experiment by placing the tank at several different heights (eg 2, 3 and 4 m) and for each of these heights we measure the time after which the tank was empty. Note, the pipe outlet remains on the same level. In the design process (optimization?), we usually want to be able to predict the emptying time for any tank placement height. This height is between the set extreme values, i.e. at an altitude of 3.27 m. Therefore, we must be able to use our measurement data to carry out a curve that will reliably approximate this relationship. We are talking about interpolation, i.e. interpolating a set of discrete data (interpolation nodes) into a continuous relationship defined for each argument lying between the points (interpolation nodes) on which we base interpolation.

#### Meritum

Interpolation is the issue of leading a curve through all points from the data set so as to obtain the course of the assumed dependence between the measurement points. The number of degrees of freedom of the fitted curve must be equal to the number of points in the data set (the number of equations must be equal to the number of unknowns).

### 2. Polynomial interpolation

Let us consider a Lagrange interpolation polynomial of order  $n$  defined as:

$$w_n(x) = \sum_{i=0}^n y_i \prod_{j=0 \wedge j \neq i}^n \frac{x - x_j}{x_i - x_j}$$

where  $(x_i, y_i)$  denote the  $i$ -th coordinate of single measurement point (interpolation node).

### Exercises

1. Write a program generating  $n$  points (interpolation nodes) using following definitions:

$$(x_i, y_i) \rightarrow (x_i, \exp(-x_i^2))$$

where

$$\begin{aligned} x_i &= a + i \cdot h, & y_i &= \exp(-x_i^2) \\ h &= \frac{b - a}{n - 1} \\ i &= 0, \dots, n - 1 \end{aligned}$$

and  $a = -2$  i  $b = 2$ .

**Repetition:** usage of `malloc` function for dynamic memory allocation of a 1D array:

```
double *x;
x = (double *) malloc(n * sizeof(double));
if (x == NULL) {
    fprintf(stderr, "malloc: can not allocate x.\n");
    exit(1);
}
/* free memory x */
free(x);
```

2. Compute value of the Lagrange interpolation polynomial in points with coordinates:

$$t_i = a + i \cdot \frac{h}{4}, \quad i = 0, \dots, 4n - 4.$$

Using function `lagrange(double *x, double *y, int n, double xx)` compute the value of the interpolation polynomial in the arbitrary point `xx` located in between the interpolation nodes. An example of basic usage of `lagrange()` function is given below:

```
int main() {
    double x[3] = {0, 1, 2};
    double y[3] = {0, 1, 4};
    double f, t = 1.5
    /* Compute value of the Lagrange polynomial of order 2 at t = 1.5 */
    f = lagrange(x, y, 3, t);
}
```

3. Print obtained results (for each  $t_i$ ) on the screen.
4. Display result on the diagram using function `scale(double x0, double y0, double x1, double y1)` defined in `winbgi2` library. Experiment with different `x0,y0,x1,y1` parameters. An example below shows how to use it to display `sin(x)` function:

```
int main() {
    double pi = 4. * atan(1.);
    graphics(600, 400);
    scale(0, -1.2, 7, 1.2);    // xmin = 0, ymin = -1.2, xmax = 7, ymax =
    double x = 0;

    while(x < 2 * pi) {
        point(x, sin(x));
        x += 0.01;
    }
    wait();
}
```

5. Write results to the file.
6. Compare results for different number of interpolation nodes  $n$ .
7. For a selected  $n$  calculate and display maximal error of the interpolation.
8. Repeat your results for different set of interpolation nodes (using different function to create values `y[i]` in interpolation nodes). Look for the results obtained using function `|x|`. Can you explain why they look like that (see definition of the interpolation error)?

## Note

Parameters of the function `lagrange(double *x, double *y, int n, double xx)` denote:

- `x`, `y` - pointers to  $n$ -element arrays holding coordinates of the interpolation nodes,
- `n` - the size of the vector (number of its elements),
- `xx` - current value of the interpolating polynomial argument for which we compute the value of the Lagrange polynomial.

## 3. Runge effect

Let us check how to improve quality of the interpolation by introduction of a more clever, non-uniform interpolation nodes distribution. We will use function `|x|` in this example to generate the set of interpolation nodes  $(x_i, |x_i|)$ .

### Chebyshev distribution of interpolation nodes

Up to now, interpolation nodes were distributed uniformly in the interpolation domain. Let us now to choose the distribution in such a way that they are denser close to the ends of the interpolation domain. This can be done as follows (see Lecture notes for theoretical explanation).

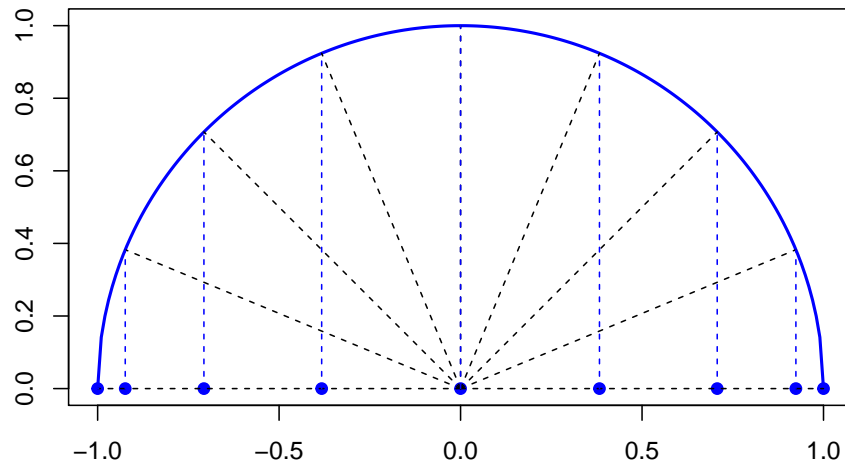
We want to distribute interpolation nodes in the range  $x = [-1, 1]$ .

Now, let us plot half-circle with center in the middle of this range  $x = 0$  and radius  $R = 1$ . The uniform division of an resulting arc (measuring in local arc coordinate) on  $n - 1$  fragments allows to generate abscissas of the interpolation nodes given by the formula:

$$x_i = -\cos\left(\frac{i \cdot \pi}{n-1}\right), \quad i = 0, \dots, n-1$$

In the more general case of the range  $x = [a, b]$  the following formula is valid:

$$x_i = -\frac{b-a}{2} \cos\left(\frac{i \cdot \pi}{n-1}\right) + \frac{a+b}{2}, \quad i = 0, \dots, n-1$$



### Exercise

Modify your program in such a way, that interpolation nodes are generated according to the above defined formula for  $x_i$ . Check, how interpolation errors change when functions:  $|x|$  or  $1/(1 + 10x^2)$  are used to generate the values  $y_i$  in interpolation nodes distributed (non-)uniformly. Can you see a difference?