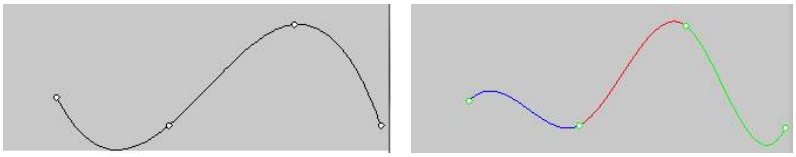
TD 2 - A basic editor for curve interpolation

(Luca Castelli Aleardi)

The goal of this TD session is to design a basic editor for 2D objects: you are asked to implement some of the methods for curve interpolation described today.



Examples of curve interpolation schemes (Lagrange and cubic spline interpolations)

Votre opinion sur le cours:

Votre opinion sur le TD:

Submit

0. Getting started (a Processing frame for testing interpolation schemes)

Files to download today

Here are some files to download (and extract to the /src folder of your Eclipse project):

- geometry.zip: some classes for manipulating geometric objects (such as points, vectors, affine and projective transformations in 2D).
- core.jar: the Processing library.
- Jama.jar: java library for matrix computations.
- TC.jar: java library for input/output.

Java documentation

Here is the javaDoc concerning the packages and classes used today:

- Jama documentation;
- Processing documentation;
- TD documentation.

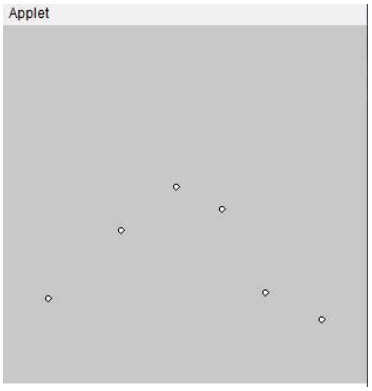
Testing classes draw

Class Draw provides a basic 2D editor for drawing (points and segments), and handling mouse events (for adding/moving/deleting points, zoom in/out, rotations, ...).

- complete class Transformation_2 implementing the constructors
  - Transformation_2(Vector_2 v)
  - Transformation_2(double s1, double s2)
  - Transformation_2(double theta)
  - Transformation_2(double theta, Point_2 center)

which correspond to: translations, scaling and rotations.

Execute class Draw to test your code.



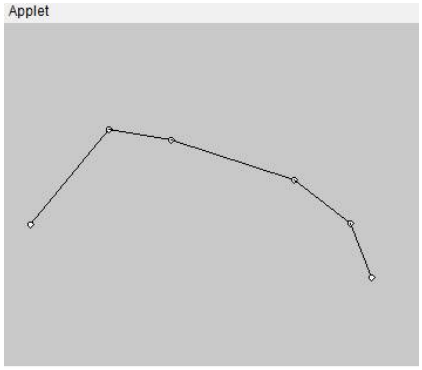
Example of execution of class Draw:
you should obtain a frame to display points and segments.
Click with **left button** to add a new point,
Click with **right button**, to delete a point,
Press **'d'** to remove the last point.
Press **'s'** to change the interpolation method (by default, at the beginning there is no interpolation)

Additional features (you have to implement them)
Press **'t'** to perform a translation.
Press **'i'** or **'o'** to perform a scaling (zoom in/out).
Press **'r'** to perform a rotation.

1. Linear interpolation

Implementing linear interpolation

- complete class LinearInterpolation implementing method interpolate() which computes and plots the linear interpolation.



Example: linear interpolation of 6 points:
$(x_0, y_0) \dots (x_5, y_5)$.

2. Function interpolation

2.1 Lagrange interpolation: solve linear system of equations

Given a set of n points to interpolate, you have to solve a system of n equations (with n variables), represented in matrix form (as illustrated during the Lecture), as follows:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ \dots & & & & \\ 1 & x_{N-1} & x_{N-1}^2 & \dots & x_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{N-1} \end{pmatrix} = \begin{pmatrix} \sum_{k=0}^{N-1} a_k x_0^k \\ \sum_{k=0}^{N-1} a_k x_1^k \\ \dots \\ \sum_{k=0}^{N-1} a_k x_{N-1}^k \end{pmatrix}$$

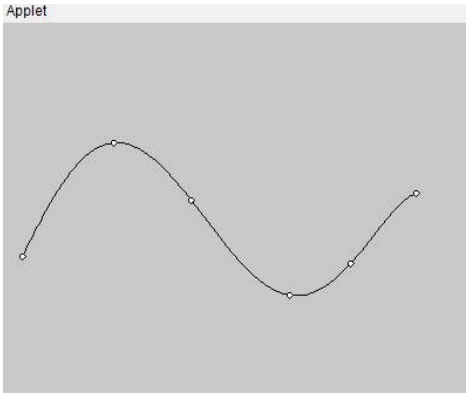
- complete class LagrangeInterpolation implementing method interpolate() which computes Lagrange interpolation.

Suggestions:
you can solve this task defining:

- a method double[] computeCoefficients(Point_2[] P) which returns an array containing the coefficients of the polynomial expression, given the input points,
- a method void plotPolynomial(double[] coeff, double min, double max, int n) which allows to plot a given polynomial expression in a given range.

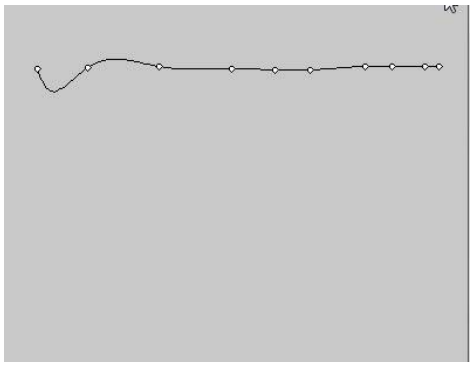
Recall that Jama library computes **solutions of linear systems**:

- use x=A.solve(b) to obtain the vector solution of system Ax=b.

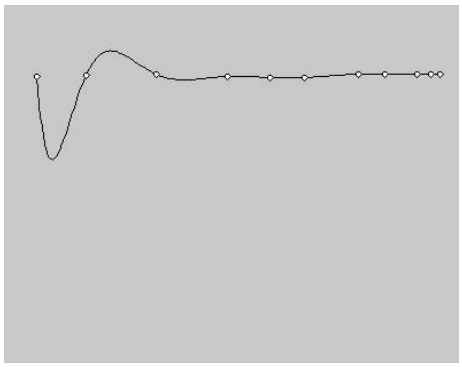


Example: Lagrange interpolation of 6 points:
(x_0, y_0) ... (x_5, y_5).
The goal is to compute the coefficients of the polynomial
expression for **f(x)**, by solving a linear system of equations.

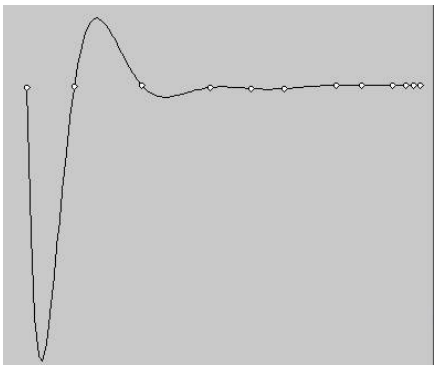
The sequence of images below illustrates some drawbacks of Lagrange interpolation



Interpolating 10 points



Addition of a point (the rightmost one)



After adding a further point

2.2 Cubic spline interpolation: matrix form

Given a set of n points to interpolate, you have to solve a system of 4(n-1) equations (with 4(n-1) variables), represented in matrix form as below.

- complete class CubicSplineInterpolation implementing the cubic spline interpolation (you can first consider the case of 4 points).

Suggestion: to specify colors use method frame.stroke(int r, int g, int b) . For example:
frame.stroke(255, 0, 0) allows to choose color red for drawing lines.

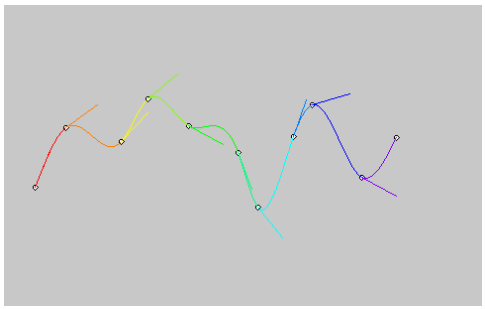
Example: cubic spline interpolation of 4 points:
(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3).
The goal is to compute cubic polynomial expressions for
f_0, f_1, f_2

$$
\begin{pmatrix}
0 & 1 & 2x_0 & 3x_0^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & x_0 & x_0^2 & x_0^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & x_1 & x_1^2 & x_1^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2x_1 & 3x_1^2 & 0 & -1 & -2x_1 & -3x_1^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 6x_1 & 0 & 0 & -2 & -6x_1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & x_1 & x_1^2 & x_1^3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & x_2 & x_2^2 & x_2^3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2x_2 & 3x_2^2 & 0 & -1 & -2x_2 & -3x_2^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 6x_2 & 0 & 0 & -2 & -6x_2 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_2 & x_2^2 & x_2^3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_3 & x_3^2 & x_3^3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2x_3 & 3x_3^2
\end{pmatrix}
\begin{pmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{pmatrix}
=
\begin{pmatrix} s_0 \\ y_0 \\ y_1 \\ 0 \\ 0 \\ y_1 \\ y_2 \\ 0 \\ 0 \\ y_2 \\ y_3 \\ s_3 \end{pmatrix}
\begin{matrix}
f_0'(x_0) = s_0 \\
f_0(x_0) = y_0 \\
f_0(x_1) = y_1 \\
f_0'(x_1) = f_1'(x_1) \\
f_0''(x_1) = f_1''(x_1) \\
f_1(x_1) = y_1 \\
f_1(x_2) = y_2 \\
f_1'(x_2) = f_2'(x_2) \\
f_1''(x_2) = f_2''(x_2) \\
f_2(x_2) = y_2 \\
f_2(x_3) = y_3 \\
f_2'(x_3) = s_3
\end{matrix}
$$

Example: cubic spline interpolation of 4 points: **(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)**.

2.3 Drawing tangents (bonus)

- modify the class CubicSplineInterpolation in order to draw the tangent vectors (at interpolated points).



3. Curve interpolation

3.1 Hermite cubic spline interpolation

Now, let us suppose we are given:

- a set of n points to interpolate (added by the user in the frame),
- a set of n vectors $\{\mathbf{t_i}\}$, where $\mathbf{t_i}$ is tangent to the curve at point $\mathbf{p_i=(x_i, y_i)}$.

Remark: for the sake of simplicity, the slopes (the n vectors $\{\mathbf{t_i}\}$) are provided as predefined parameters in the code (their are constant).

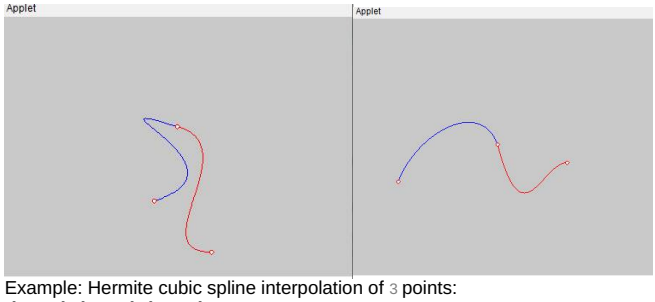
Goals and constraints: we want to find a cubic parametric curve f(t)=(x(t), y(t)), such that:

-) **f** interpolates points $\mathbf{p_i=(x_i, y_i)}$,

-) **f** respects C^1 constraints (given by tangents)

- complete class HermiteSplineInterpolation implementing the Hermite cubic spline interpolation.

Suggestion: as before, solve the problem in its matrix form, recalling that you should use curve parametrization.



Example: Hermite cubic spline interpolation of 3 points: **(x_0, y_0), (x_1, y_1), (x_2, y_2)**.
The goal is to compute parametric expressions for **f_0, f_1.**

$\mathrm{f}(t) = (x(t), y(t))$ $\quad \mathrm{x}(t) = a_x + b_x t + c_x t^2 + d_x t^3$ $\quad \frac{dt}{dx} = \frac{1}{p_1^x - p_0^x}$

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix}}_{W_x} \begin{pmatrix} a_x \\ b_x \\ c_x \\ d_x \end{pmatrix}}_{A_x} = \begin{pmatrix} \dots \\ \dots \\ \dots \\ \dots \end{pmatrix} \begin{matrix} \mathrm{x}(0) = p_0^x \\ \mathrm{x}(1) = p_1^x \\ (x \text{ coordinate of}) \text{ slope at } \mathbf{p}_0 \\ (x \text{ coordinate of}) \text{ slope at } \mathbf{p}_1 \end{matrix}$$

3.2 Drawing tangents and update slopes (bonus: difficult)

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{pmatrix}}_{W_y} \begin{pmatrix} a_y \\ b_y \\ c_y \\ d_y \end{pmatrix}}_{A_y} = \begin{pmatrix} \dots \\ \dots \\ \dots \\ \dots \end{pmatrix} \begin{matrix} \mathrm{y}(0) = p_0^y \\ \mathrm{y}(1) = p_1^y \\ (y \text{ coordinate of}) \text{ slope at } \mathbf{p}_0 \\ (y \text{ coordinate of}) \text{ slope at } \mathbf{p}_1 \end{matrix}$$

- modify the class HermiteSplineInterpolation in order to draw the tangent vectors (at interpolated points).
- Modify the class Draw: make use of the mouse drag events to update the slope of tangent vectors in order to adapt in an interactive way the shape of the curve