Programmation - LagrangeInterpolation - 100 points

Kévin DUVERGER

Table des matières

1 Résolution LagrangeInterpolation :

Pour les détails de comment l'interpolation de lagrange fonctionne, je fais un peu de publicité pour mon site, vous pouvez aller sur le lien suivant : `https://kevin-duverger.fr/Maths/Curiosites/InterpolationLagrange`.
Pour vous résumer le principe de façon très rapide, l'idée est que l'on nous donne k points $p_i = (x_i, y_i)$ et on veut trouver un polynôme de degré $k-1$ qui passe par tous ces points. L'idée est de calculer une base de polynômes f_i pour lesquels $f_i(x_i) = 1$ et $f_i(x_j) = 0$ pour $j \neq i$, de cette façon la réponse à l'exercice est $f = y_1 \times f_1 + y_2 \times f_2 + ... + y_k \times f_k$.
Chacun des polynômes de la base recherchée s'exprime de la façon suivante :

$$f_i(x) = \frac{x_1 - x}{x_1 - x_i} \times \frac{x_2 - x}{x_2 - x_i} \times ... \times \frac{x_{i-1} - x}{x_{i-1} - x_i} \times \frac{x_{i+1} - x}{x_{i+1} - x_i} \times ... \times \frac{x_{k-1} - x}{x_{k-1} - x_i} \times \frac{x_k - x}{x_k - x_i}$$

L'idée au numérateur est de faire en sorte que la fonction s'annule en $x_1, x_2, ..., x_k$ mais pas en x_i (remarquez que le terme $x_i - x$ n'est pas présent). Le dénominateur est pris de telle sorte que si l'on rentre $x = x_i$, toutes ces fractions vaudront 1 et vous savez que $1 \times 1 \times ... \times 1 = 1$ ce qui nous donne la seconde demande !

On peut donc encore une fois tester la connexion vers `146.59.227.136` avec le port `23407` :

```
Your goal is to find a polynomial that goes through the following points :
        P1 = (6,-75)
        P2 = (19,-43)
        P3 = (4,32)
        P4 = (-12,27)
        P5 = (18,38)
        P6 = (-20,54)
        P7 = (0,99)
        P8 = (-19,28)
Now, you can give your polynomial in the following form : num1,den1|num2,den2|... (all num / den must be integers)
The polynomial will then be parsed as (num1/den1) * x^7 + (num2/den2) * x^6 + ...
Now please enter your polynomial (you have 5 seconds) :
```

On nous donne donc toujours 8 points et il faut retourner le bon polynôme.

La première étape du programme sera de parser ce que le serveur nous envoie pour récupérer tous les x_i et les y_i. Une fois ces derniers trouvés, nous pouvons calculer la base de polynômes grâce au x_i comme indiqués (il faudra pouvoir multiplier les petits polynômes de degré 1 pour avoir le polynôme final de degré $k-1$). Pour finir, il suffira de multiplier les coefficients de chacun de ces polynômes par les y_i pour obtenir la fonction f finale à envoyer au serveur. Bien entendu, il faudra retourner les valeurs exactes de chacun des coefficients qui prendront la forme de fractions (comme le challenge le demande dans le message précédent).
Voici donc le code associé :

```python
import math
import os, sys, socket

NUMERATOR = 0
DENOMINATOR = 1

def addFractions(fraction1, fraction2) :
    result = [ fraction1[NUMERATOR] * fraction2[DENOMINATOR] + fraction2[NUMERATOR] * fraction1[DENOMINATOR], fraction1[DENOMINATOR] * fraction2[DENOMINATOR] ]
    commonFactor = math.gcd(result[NUMERATOR], result[DENOMINATOR])
    result[NUMERATOR] //= commonFactor
    result[DENOMINATOR] //= commonFactor
    return result

def subFractions(fraction1, fraction2) :
    fraction2[NUMERATOR] *= -1
    toReturn = addFractions(fraction1, fraction2)
    fraction2[NUMERATOR] *= -1
    return toReturn

def multFractions(fraction1, fraction2) :
    result = [ fraction1[NUMERATOR] * fraction2[NUMERATOR], fraction1[DENOMINATOR] * fraction2[DENOMINATOR] ]
    commonFactor = math.gcd(result[NUMERATOR], result[DENOMINATOR])
    result[NUMERATOR] //= commonFactor
    result[DENOMINATOR] //= commonFactor
    return result

def generatePolynomial(xList, xIndex) :
    # First, generating the elements of the product
    littlePolynomials = []
    for i in range(0, len(xList)) :
        if i != xIndex :
```

```
            denominator = xList[xIndex] - xList[i]
            xFraction   = [ 1, denominator ]
            _1Fraction = [ xList[i], -denominator ] if xList[i] > 0 and denominator < 0 else [ -xList[i], denominator ]
            littlePolynomials.append([ xFraction, _1Fraction ]) # fraction in front of x, then in front of constant term
    # Then, multiplying all of those polynomials
    finalPolynomial = [ littlePolynomials[0][0], littlePolynomials[0][1] ]
    for i in range(1, len(littlePolynomials)) :
        polynomial1 = [ [ 0, 1 ] ] + [ multFractions(finalPolynomial[j], littlePolynomials[i][1]) for j in range(0, len(finalPolynomial)) ]
        polynomial2 = [ multFractions(finalPolynomial[j], littlePolynomials[i][0]) for j in range(0, len(finalPolynomial)) ] + [ [ 0, 1 ] ]
        finalPolynomial = [ addFractions(polynomial1[j], polynomial2[j]) for j in range(0, len(polynomial1)) ]
    # Returning the result
    return finalPolynomial

def findLagrangeSolution(points) :
    # First, creating the xList
    xList = [ points[i][0] for i in range(0, len(points)) ]
    # Then, finding all the base polynomials
    basePolynomials = []
    for i in range(0, len(points)) :
        basePolynomials.append(generatePolynomial(xList, i))
    # Finally, finding the final polynomial
    finalPolynomial = [ [ 0, 1 ] ] * len(points)
    for i in range(0, len(points)) :
        finalPolynomial = [ addFractions(finalPolynomial[j], [ basePolynomials[i][j][NUMERATOR] * points[i][1], basePolynomials[i][j][DENOMINATOR] ]) for j in range(0, len(points)) ]
    return finalPolynomial


server = "146.59.227.136"
port = 23407

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# First, getting the data
data = ma_socket.recv(8192).decode("utf-8")
print(data)
data = data.split("\n")
points = []
for i in range(2, 10) :
    openParenPos = data[i].find("(")
    commaPos     = data[i].find(",")
    closParenPos = data[i].find(")")
    points.append([ int(data[i][openParenPos + 1:commaPos]), int(data[i][commaPos + 1:closParenPos]) ])

# Now, solving the problem
result = findLagrangeSolution(points)
toSend = "|".join([ ",".join([ str(result[i][j]) for j in range(0, len(result[i])) ]) for i in range(0, len(result)) ])
ma_socket.sendall((toSend + "\n").encode("utf-8"))
print(ma_socket.recv(8192).decode("utf-8"))

ma_socket.close()
```