CLASS NOTES

Models, Algorithms and Data: Introduction to computing

2019

Petros Koumoutsakos, Jens Honore Walther, Julija Zavadlav

(Last update: April 27, 2019)

IMPORTANT DISCLAIMERS

Much of the material (ideas, definitions, concepts, examples, etc) in these notes is taken for teaching purposes, from several references:

- Numerical Analysis by R. L. Burden and J. D. Faires
- The Nature of Mathematical Modeling by N. Gershenfeld
- A First Course in Numerical methods by U. M. Ascher and C. Greif

These notes are only informally distributed and intended ONLY as study aid for the final exam of ETHZ students that were registered for the course Models, Algorithms and Data (MAD): Introduction to computing 2019. The notes have been checked, however they may still contain errors so use with care.

LECTURE 4 Interpolation and extrapolation I: Lagrange interpolation

4.1 Introduction

As we learned in the first lecture function fitting may be understood as a *low order model* that describes the given data. The type of interpolation is equivalent to choosing a particular model and as such it reflects some prior knowledge about the data. The choice of the approximating function as well as the choice (when possible) of the sampling points, represents assumptions that we have made about the structure/architecture of the model.

Recall from the first part of of the lecture that we constructed a fitting function $f(x)$ as follows:

$$f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x) = y \tag{4.1.1}$$

The parameters here are:

- M: the number of terms (can be larger, equal or smaller than the number of data points)
- $\phi_k(x)$: the type of basis function

Here we call x_i are the abscissae, y_i the ordinate of the given data points. Furthermore we denote α_k as our coefficients.

In the first section we took the number of datapoints N much bigger then the number of basis functions M. We learned that in the case $M = N$ and a linearly independent basis function our system is solvable for the coefficients. This allows us to **interpolate** among the data (i.e. interpret the data for unseen values) and in order to **extrapolate** (i.e. predict based on the available data). In this sense extrapolation is equivalent to generalisation.

Example 4.1: Polynomial Function Interpolation Revisited

Consider three data points $(x_i, f(x_i))$ with $i = 1,2,3$ in a two dimensional Cartesian coordinate system given by $(-1,-1)$, $(0,0)$ and $(1,1)$. We want to find a polynomial of degree 2 to interpolate the three data points, that is $f(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$. We have a linear system of 3 equations with

3 unknowns to solve

$$\begin{bmatrix} \phi_1(x_1), & \phi_2(x_1) & \phi_3(x_1) \\ \phi_1(x_2), & \phi_2(x_2) & \phi_3(x_2) \\ \phi_1(x_3), & \phi_2(x_3) & \phi_3(x_3) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix}. \tag{4.1.2}$$

With the given data, we have the follow linear system to solve.

$$\begin{bmatrix} 1, & -1 & 1 \\ 1, & 0 & 0 \\ 1, & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \tag{4.1.3}$$

Although the 3×3 matrix is not sparse, we can still solve it quickly without pen or paper by observing its structure. Let us look at the second row first, we can tell directly $\alpha_1 = 0$. Subsequently, let us look at first and third rows together and remember $\alpha_1 = 0$, we can quickly determine $\alpha_3 = 0$ and $\alpha_2 = 1$. To summarize, the interpolating function we seek is $f(x) = x$.

Now consider the three data points again, plotting them it is obvious that they stay on the same straight line. This observation suggests that the interpolation procedure we just performed indeed exclude the contributions from constant and quadratic functions.

Imagine now that we have N data points and we use, for example, Gaussian elimination method to solve the resultant linear system. This would take $O(N^3)$ operations, which is quite expensive for the purpose of determining an interpolating function. It would be convenient if we can select the basis function so that the matrix of the linear system is diagonal and it takes almost no effort to get the coefficients α for the interpolating function. This motivates the construction of the Lagrangian interpolation method.

4.2 Lagrange Interpolation

One way to do achieve a diagonal matrix is to chose N polynomials of degree $N-1$: $\{l_k(x)\}_{k=1,\ldots,N}$, such that $l_i(x_j) = \delta_{ij}$[1]. This property is satisfied by

$$\boxed{l_k(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_N)}{(x_k-x_1)(x_k-x_2)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_N)}} \tag{4.2.1}$$

Using these functions we construct our interpolation function as

$$f(x) = \sum_{k=1}^{N} \alpha_k l_k(x)$$

[1]Recall that $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$.

In order to find the coefficients we insert one of our data-points

$$f(x_i) = \sum_{k=1}^{N} \alpha_k l_k(x_i) = \alpha_i = y_i.$$

Where we used the constructive property of the polynomials (i.e. that $l_k(x_k) = 1$ and $l_k(x_i) = 0$ for $k \neq i$). Thus we found our Lagragian interpolation function as

$$\boxed{f(x) = \sum_{k=1}^{N} y_k l_k(x)} \tag{4.2.2}$$

Important Notes:

- The Lagrange polynomials involve single polynomials which pass through all the data points.
- The approximation error of the Lagrange polynomial $f(x)$ of a function $y(x)$ that has been sampled at points (x_k, y_k) and for some $x_1 \leq \xi \leq x_N$ is given by:

  $$\left|y(x) - f(x)\right| = \left|\frac{y^{(n)}(\xi)}{n!}\Pi_{k=1}^{n}(x - x_k)\right|$$

  This term can be minimized if we chose the coordinates x_k of the sampling points to be the roots of the n-degree Chebychev polynomials $T_n(x)$. These points satisfy $T_n(x_k) = 0$ and they can be computed as: $x_k = -\cos(\frac{2k-1}{2n}\pi)$.

Exam checklist

Lagrange interpolation:

- Polynomials with degree $N-1$ for N data points
- Analytical expression from data points
- Sensitivity to noise
- Predictability issues
- Extensions to higher dimensions?

Exercises

Question 1: Lagrange vs Least Squares

We consider N data points in 2D ($\{x_i, y_i\}$ with $i = 1 \ldots N$), where we wish to find a function $f(x)$ such that $y_i \approx f(x_i)$. As an example, imagine that the data is given on a straight line (i.e. $y_i = a x_i + b$ for some real values a and b):

- How do you expect the Lagrange functions ($l_k(x)$ in Eq. (4.2.1)) to look like for $N = 3$? How about larger N?
  Solution: For $N = 3$, the Lagrange functions $l_k(x)$ are polynomials of order $N-1 = 2$ (quadratic). For larger N, the Lagrange functions $l_k(x)$ will always be polynomials of order $N-1$.

- How do you expect the Lagrange interpolator ($f(x)$ in Eq. (4.2.2)) to look like?
  Solution: The Lagrange interpolator is expected to be the equation of a line $f(x) = ax + b$ in every case (no matter how many N points we use). This means, that for an order-1 function (linear) the higher order (quadratic, cubic etc.) terms do not contribute to the interpolator. We provide a simple example in order for this to be clearer: Assume we have to interpolate through the following set of points that are drown from the linear function $f(x) = 2x+1$, $\{x_i, y_i\} = (0,1), (2,5), (4,9)$.
  Using $N = 2$ points:

$$
\begin{aligned}
l_1(x) &= \frac{x-2}{-2} = -\frac{1}{2}(x-2) \\
l_2(x) &= \frac{x-0}{2} = \frac{x}{2} \\
f(x) &= 1(-\frac{1}{2})(x-2) + 5\frac{x}{2} = 2x+1
\end{aligned}
$$

  Using $N = 3$ points:

$$
\begin{aligned}
l_1(x) &= \frac{(x-2)(x-4)}{(0-2)(0-4)} = \frac{(x-2)(x-4)}{8} = \frac{x^2-6x+8}{8} \\
l_2(x) &= \frac{(x-0)(x-4)}{(2-0)(2-4)} = -\frac{x^2-4x}{4} \\
l_3(x) &= \frac{(x-0)(x-2)}{(4-0)(4-2)} = \frac{x^2-2x}{8} \\
f(x) &= 1\frac{x^2-6x+8}{8} - 5\frac{2x^2-8x}{8} + 9\frac{x^2-2x}{8} = 2x+1
\end{aligned}
$$

  The resulting interpolator for $N = 3$ matches the result for $N = 2$. Generalizing, for a linear function, the final interpolator $f(x)$ using $N \geq 3$ points will be the same as computing it with $N = 2$ points.

- How do you expect those functions ($l_k(x)$ and $f(x)$) to change if we add a bit of noise to the data? Adding noise can be imagined as choosing $y_i = ax_i + b + \xi_i$, where ξ_i is sampled from a normal random distribution with mean 0 and standard deviation σ. How do you expect the value of σ to affect the Lagrange interpolator?
  Solution: By adding noise to the data: $y_i = ax_i + b + \xi_i$, $\xi_i \sim \mathcal{N}(0, \sigma^2)$ the Lagrange functions are not varied: $l_k(x)$ is a function of $\{x_i\}$ only. The Lagrange interpolator $f(x)$ is modified:

for data without noise: $y_i = ax_i + b$
for data with noise: $y_i^* = ax_i + b + \xi_i$

$$f(x) = \sum y_k l_k(x) = \sum (ax_k + b) l_k(x)$$
$$f^*(x) = \sum y_k^* l_k(x) = \sum (ax_k + b + \xi_k) l_k(x)$$

The difference in the interpolator is:

$$|f(x) - f^*(x)| = \sum_{k=1}^{N} |\xi_k l_k(x)| \leq \sum |\xi_k| \sum |l_k(x)|$$

We conclude that including even a bit of noise can cause significant oscillations, whereas oscillations become worse with increasing σ. If $\sigma \to 0$, then ξ_k is distributed close to 0, so error is considerably lower.

- Imagine we did a least squares fit to a linear function $y_i \approx a_L x_i + b_L$ and computed a_L, b_L as in Eq. (**??**) (note that there we used a slightly different notation). Would you expect to get $a_L = a$ and $b_L = b$, if the data had no noise? What if we added noise to the data?

  Solution: For least squares fit: $y_i = a_{LS} x_i + b_{LS}$. If data has no noise: $a_{LS} = a$, $b_{LS} = b$. If we add noise: $y_i^* = a_{LS}^* x_i + b_{LS}^* + \xi_i$. See exercise set 1 question 3:

$$|f(x) - f^*(x)| \leq |a - a_{LS}^*| x + |b - b_{LS}^*|$$
$$|a - a_{LS}^*| \leq \frac{C}{N} \sum_i^N \xi_i$$

Question 2: Computational complexity

Consider again the case of N data points in 2D: $\{x_i, y_i\}$ $(i = 1 \ldots N)$. Estimate the computational complexity in terms of number of basic floating point operations (FLOPs: add, subtract, multiply, divide) to evaluate $f(x)$ in Eq. (4.2.2) for a given value x. Estimate the computational complexity to evaluate the equivalent $f(x) = \sum_{k=1}^{N} \alpha_k \phi_k(x)$ (as in Eq. (4.1.1)) with basis functions $\phi_k(x) = x^{k-1}$ (as in Eq. (**??**)) with given coefficients α_k. Which one requires less FLOPs and hence is faster to evaluate? How many FLOPs do you save if you evaluate a simplified system such as $f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x)$ with $M < N$?

Solution: To evaluate $f(x) = \sum_{k=1}^{N} y_k l_k(x)$ where $l_k(x) = \prod_{i=1, i\neq k}^{N} \frac{(x-x_i)}{(x_k-x_i)}$ we have

$$f(x) = y_1 \frac{(x-x_2)(x-x_3)\ldots(x-x_N)}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_N)} + \ldots$$
$$+ y_N \frac{(x-x_1)(x-x_2)\ldots(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)\ldots(x_N-x_{N-1})}$$

For every l_k the $y_k \prod \frac{1}{(x_k-x_i)}$ is a constant and can be precomputed.

$$f(x) = c_1[(x-x_2)(x-x_3)\ldots(x-x_N)] + \cdots + c_N[(x-x_1)(x-x_2)\ldots(x-x_{N-1})]$$

We need N multiplications for every term and there are N terms (N basis functions). Thus, in total we need N^2 flops.

To evaluate $f(x) = \sum a_k \Phi_k(x)$ where $\Phi_k(x) = x^{k-1}$ we have

$$f(x) = a_N x^{N-1} + a_{N-1} x^{N-2} + \cdots + a_2 x^1 + a_1$$

If we compute x^{k-1} in each step from scratch it takes: $\frac{(N-1)(N-2)}{2}$ multiplications to compute x^{k-1} and $N-1$ multiplications to multiply with all x^{k-1} with a_k. Therefore, overall $\frac{N^2-N}{2}$ multiplications.
If we compute x^{k-1} using x^{k-2} (computed in previous step) we can save flops. We have to do $N-1$ multiplications for all x^{k-1} and $N-1$ multiplications to multiply with all x^{k-1} with a_k. Together, we need $2N-2$ multiplications.