Data Analysis and Machine Learning Lectures: Cubic Splines and Gradient Methods

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

Jan 27, 2018

Cubic Splines

Cubic spline interpolation is among one of the most used methods for interpolating between data points where the arguments are organized as ascending series. In the library program we supply such a function, based on the so-called cubic spline method to be described below.

A spline function consists of polynomial pieces defined on subintervals. The different subintervals are connected via various continuity relations.

Assume we have at our disposal $n + 1$ points $x_0, x_1, \ldots x_n$ arranged so that $x_0 < x_1 < x_2 < \ldots x_{n-1} < x_n$ (such points are called knots). A spline function s of degree k with $n + 1$ knots is defined as follows

- On every subinterval $[x_{i-1}, x_i)$ s is a polynomial of degree $\leq k$.

- s has $k - 1$ continuous derivatives in the whole interval $[x_0, x_n]$.

Splines

As an example, consider a spline function of degree $k = 1$ defined as follows

$$s(x) = \begin{bmatrix} s_0(x) = a_0 x + b_0 & x \in [x_0, x_1) \\ s_1(x) = a_1 x + b_1 & x \in [x_1, x_2) \\ \ldots & \ldots \\ s_{n-1}(x) = a_{n-1} x + b_{n-1} & x \in [x_{n-1}, x_n] \end{bmatrix}.$$

In this case the polynomial consists of series of straight lines connected to each other at every endpoint. The number of continuous derivatives is then $k - 1 = 0$, as expected when we deal with straight lines. Such a polynomial is quite easy to construct given $n + 1$ points $x_0, x_1, \ldots x_n$ and their corresponding function values.

Splines

The most commonly used spline function is the one with $k = 3$, the so-called cubic spline function. Assume that we have in adddition to the $n + 1$ knots a series of functions values $y_0 = f(x_0), y_1 = f(x_1), \ldots y_n = f(x_n)$. By definition, the polynomials s_{i-1} and s_i are thence supposed to interpolate the same point i, that is

$$s_{i-1}(x_i) = y_i = s_i(x_i),$$

with $1 \leq i \leq n - 1$. In total we have n polynomials of the type

$$s_i(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + a_{i2}x^3,$$

yielding $4n$ coefficients to determine.

Splines

Every subinterval provides in addition the $2n$ conditions

$$y_i = s(x_i),$$

and

$$s(x_{i+1}) = y_{i+1},$$

to be fulfilled. If we also assume that s' and s'' are continuous, then

$$s'_{i-1}(x_i) = s'_i(x_i),$$

yields $n - 1$ conditions. Similarly,

$$s''_{i-1}(x_i) = s''_i(x_i),$$

results in additional $n - 1$ conditions. In total we have $4n$ coefficients and $4n - 2$ equations to determine them, leaving us with 2 degrees of freedom to be determined.

Splines

Using the last equation we define two values for the second derivative, namely

$$s''_i(x_i) = f_i,$$

and

$$s''_i(x_{i+1}) = f_{i+1},$$

and setting up a straight line between f_i and f_{i+1} we have

$$s''_i(x) = \frac{f_i}{x_{i+1} - x_i}(x_{i+1} - x) + \frac{f_{i+1}}{x_{i+1} - x_i}(x - x_i),$$

and integrating twice one obtains

$$s_i(x) = \frac{f_i}{6(x_{i+1} - x_i)}(x_{i+1} - x)^3 + \frac{f_{i+1}}{6(x_{i+1} - x_i)}(x - x_i)^3 + c(x - x_i) + d(x_{i+1} - x).$$

Splines

Using the conditions $s_i(x_i) = y_i$ and $s_i(x_{i+1}) = y_{i+1}$ we can in turn determine the constants c and d resulting in

$$s_i(x) = \frac{f_i}{6(x_{i+1} - x_i)}(x_{i+1} - x)^3 + \frac{f_{i+1}}{6(x_{i+1} - x_i)}(x - x_i)^3$$
$$+ \left(\frac{y_{i+1}}{x_{i+1} - x_i} - \frac{f_{i+1}(x_{i+1} - x_i)}{6}\right)(x - x_i) + \left(\frac{y_i}{x_{i+1} - x_i} - \frac{f_i(x_{i+1} - x_i)}{6}\right)(x_{i+1} - x).$$
$$(1)$$

Splines

How to determine the values of the second derivatives f_i and f_{i+1}? We use the continuity assumption of the first derivatives

$$s'_{i-1}(x_i) = s'_i(x_i),$$

and set $x = x_i$. Defining $h_i = x_{i+1} - x_i$ we obtain finally the following expression

$$h_{i-1}f_{i-1} + 2(h_i + h_{i-1})f_i + h_i f_{i+1} = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}),$$

and introducing the shorthands $u_i = 2(h_i + h_{i-1})$, $v_i = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1})$, we can reformulate the problem as a set of linear equations to be solved through e.g., Gaussian elemination

Splines

Gaussian elimination

$$\begin{bmatrix} u_1 & h_1 & 0 & \dots & & & & \\ h_1 & u_2 & h_2 & 0 & \dots & & & \\ 0 & h_2 & u_3 & h_3 & 0 & & \dots & \\ \dots & & \dots & \dots & \dots & \dots & \dots & \\ & \dots & & & 0 & h_{n-3} & u_{n-2} & h_{n-2} \\ & & & & & 0 & h_{n-2} & u_{n-1} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_{n-2} \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}.$$

Note that this is a set of tridiagonal equations and can be solved through only $O(n)$ operations.

Splines

The functions supplied in the program library are *spline* and *splint*. In order to use cubic spline interpolation you need first to call

```
spline(double x[], double y[], int n, double yp1,  double yp2, double y2[])
```

This function takes as input $x[0, .., n-1]$ and $y[0, .., n-1]$ containing a tabulation $y_i = f(x_i)$ with $x_0 < x_1 < .. < x_{n-1}$ together with the first derivatives of $f(x)$ at x_0 and x_{n-1}, respectively. Then the function returns $y2[0, .., n-1]$ which contains the second derivatives of $f(x_i)$ at each point x_i. n is the number of points. This function provides the cubic spline interpolation for all subintervals and is called only once.

Splines

Thereafter, if you wish to make various interpolations, you need to call the function

```
splint(double x[], double y[], double y2a[], int n, double x, double *y)
```

which takes as input the tabulated values $x[0, .., n-1]$ and $y[0, .., n-1]$ and the output y2a[0,..,n - 1] from *spline*. It returns the value y corresponding to the point x.

Conjugate gradient (CG) method

The success of the CG method for finding solutions of non-linear problems is based on the theory of conjugate gradients for linear systems of equations. It belongs to the class of iterative methods for solving problems from linear algebra of the type

$$\hat{A}\hat{x} = \hat{b}.$$

In the iterative process we end up with a problem like

$$\hat{r} = \hat{b} - \hat{A}\hat{x},$$

where $\hat{r}$ is the so-called residual or error in the iterative process.

When we have found the exact solution, $\hat{r} = 0$.

Conjugate gradient method

The residual is zero when we reach the minimum of the quadratic equation

$$P(\hat{x}) = \frac{1}{2}\hat{x}^T \hat{A}\hat{x} - \hat{x}^T \hat{b},$$

with the constraint that the matrix $\hat{A}$ is positive definite and symmetric. If we search for a minimum of the quantum mechanical variance, then the matrix $\hat{A}$, which is called the Hessian, is given by the second-derivative of the function we want to minimize. This quantity is always positive definite. In our case this corresponds normally to the second derivative of the energy.

Conjugate gradient method, Newton's method first

We seek the minimum of the energy or the variance as function of various variational parameters. In our case we have thus a function f whose minimum we are seeking. In Newton's method we set $\nabla f = 0$ and we can thus compute the next iteration point

$$\hat{x} - \hat{x}_i = \hat{A}^{-1}\nabla f(\hat{x}_i).$$

Subtracting this equation from that of $\hat{x}_{i+1}$ we have

$$\hat{x}_{i+1} - \hat{x}_i = \hat{A}^{-1}(\nabla f(\hat{x}_{i+1}) - \nabla f(\hat{x}_i)).$$

Simple example and demonstration

The function f can be either the energy or the variance. If we choose the energy then we have

$$\hat{\alpha}_{i+1} - \hat{\alpha}_i = \hat{A}^{-1}(\nabla E(\hat{\alpha}_{i+1}) - \nabla E(\hat{\alpha}_i)).$$

In the simple harmonic oscillator model, the gradient and the Hessian $\hat{A}$ are

$$\frac{d\langle E_L[\alpha]\rangle}{d\alpha} = \alpha - \frac{1}{4\alpha^3}$$

and a second derivative which is always positive (meaning that we find a minimum)

$$\hat{A} = \frac{d^2\langle E_L[\alpha]\rangle}{d\alpha^2} = 1 + \frac{3}{4\alpha^4}$$

Simple example and demonstration

We get then

$$\alpha_{i+1} = \frac{4}{3}\alpha_i - \frac{\alpha_i^4}{3\alpha_{i+1}^3},$$

which can be rewritten as

$$\alpha_{i+1}^4 - \frac{4}{3}\alpha_i\alpha_{i+1}^4 + \frac{1}{3}\alpha_i^4.$$

Conjugate gradient method

In the CG method we define so-called conjugate directions and two vectors $\hat{s}$ and $\hat{t}$ are said to be conjugate if

$$\hat{s}^T\hat{A}\hat{t} = 0.$$

The philosophy of the CG method is to perform searches in various conjugate directions of our vectors $\hat{x}_i$ obeying the above criterion, namely

$$\hat{x}_i^T\hat{A}\hat{x}_j = 0.$$

Two vectors are conjugate if they are orthogonal with respect to this inner product. Being conjugate is a symmetric relation: if $\hat{s}$ is conjugate to $\hat{t}$, then $\hat{t}$ is conjugate to $\hat{s}$.

Conjugate gradient method

An example is given by the eigenvectors of the matrix

$$\hat{v}_i^T \hat{A} \hat{v}_j = \lambda \hat{v}_i^T \hat{v}_j,$$

which is zero unless $i = j$.

Conjugate gradient method

Assume now that we have a symmetric positive-definite matrix $\hat{A}$ of size $n \times n$. At each iteration $i + 1$ we obtain the conjugate direction of a vector

$$\hat{x}_{i+1} = \hat{x}_i + \alpha_i \hat{p}_i.$$

We assume that $\hat{p}_i$ is a sequence of n mutually conjugate directions. Then the $\hat{p}_i$ form a basis of R^n and we can expand the solution $\hat{A}\hat{x} = \hat{b}$ in this basis, namely

$$\hat{x} = \sum_{i=1}^{n} \alpha_i \hat{p}_i.$$

Conjugate gradient method

The coefficients are given by

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{n} \alpha_i \mathbf{A}\mathbf{p}_i = \mathbf{b}.$$

Multiplying with $\hat{p}_k^T$ from the left gives

$$\hat{p}_k^T \hat{A} \hat{x} = \sum_{i=1}^{n} \alpha_i \hat{p}_k^T \hat{A} \hat{p}_i = \hat{p}_k^T \hat{b},$$

and we can define the coefficients α_k as

$$\alpha_k = \frac{\hat{p}_k^T \hat{b}}{\hat{p}_k^T \hat{A} \hat{p}_k}$$

Conjugate gradient method and iterations

If we choose the conjugate vectors $\hat{p}_k$ carefully, then we may not need all of them to obtain a good approximation to the solution $\hat{x}$. We want to regard the conjugate gradient method as an iterative method. This will us to solve systems where n is so large that the direct method would take too much time.

We denote the initial guess for $\hat{x}$ as $\hat{x}_0$. We can assume without loss of generality that

$$\hat{x}_0 = 0,$$

or consider the system

$$\hat{A}\hat{z} = \hat{b} - \hat{A}\hat{x}_0,$$

instead.

Conjugate gradient method

One can show that the solution $\hat{x}$ is also the unique minimizer of the quadratic form

$$f(\hat{x}) = \frac{1}{2}\hat{x}^T\hat{A}\hat{x} - \hat{x}^T\hat{x}, \quad \hat{x} \in \mathbf{R}^n.$$

This suggests taking the first basis vector $\hat{p}_1$ to be the gradient of f at $\hat{x} = \hat{x}_0$, which equals

$$\hat{A}\hat{x}_0 - \hat{b},$$

and $\hat{x}_0 = 0$ it is equal $-\hat{b}$. The other vectors in the basis will be conjugate to the gradient, hence the name conjugate gradient method.

Conjugate gradient method

Let $\hat{r}_k$ be the residual at the k-th step:

$$\hat{r}_k = \hat{b} - \hat{A}\hat{x}_k.$$

Note that $\hat{r}_k$ is the negative gradient of f at $\hat{x} = \hat{x}_k$, so the gradient descent method would be to move in the direction $\hat{r}_k$. Here, we insist that the directions $\hat{p}_k$ are conjugate to each other, so we take the direction closest to the gradient $\hat{r}_k$ under the conjugacy constraint. This gives the following expression

$$\hat{p}_{k+1} = \hat{r}_k - \frac{\hat{p}_k^T\hat{A}\hat{r}_k}{\hat{p}_k^T\hat{A}\hat{p}_k}\hat{p}_k.$$

Conjugate gradient method

We can also compute the residual iteratively as

$$\hat{r}_{k+1} = \hat{b} - \hat{A}\hat{x}_{k+1},$$

which equals

$$\hat{b} - \hat{A}(\hat{x}_k + \alpha_k\hat{p}_k),$$

or

$$(\hat{b} - \hat{A}\hat{x}_k) - \alpha_k\hat{A}\hat{p}_k,$$

which gives

$$\hat{r}_{k+1} = \hat{r}_k - \hat{A}\hat{p}_k,$$