cs 514, lecture 08-10may: spline approximation

When the data

$$((\tau_i, y_i) : i = 1{:}N) \tag{1}$$

are not exact, e.g., y_i may only be an approximation to $g(\tau_i)$, or there are more data than needed to recover $g \in X$ to the desired accuracy, one does not try to match the data exactly, but simply finds an 'approximate interpolant' from some subspace F that has $\dim F < N$ but can be expected to recover the 'essential part' of the information about g in the data.

There are at least two standard ways to proceed, *least-squares approximation* and *smoothing*.

Look at **discrete least-squares approximation from** $F \subset X = \mathbb{R}^I$ for some interval I containing the sequence τ. With V a basis for F, look again at the linear system

$$\sum_j v_j(\tau_i) a_j = y_j, \quad j = 1{:}N,$$

for the coordinate vector $\mathbf{a}$ for the 'interpolant' $V\mathbf{a}$ from F to the data. If the coefficient matrix

$$A := A_\tau := \Lambda'_\tau V$$

of this linear system is invertible, then there is a unique interpolant, and we get its coefficients just by solving the linear system. In `Matlab`, this amounts to the command

```
a = A\y;
```

which is very user-friendly.

Remarkably, the very same command also suffices for computing the discrete least-squares approximant from F to these data. This is, by definition, the unique element $P_F\mathbf{y} = V\mathbf{a} \in F$ that minimizes

$$\|\mathbf{y} - f\|_2^2 := \sum_i (y_i - f(\tau_i))^2$$

over all $f \in F$.

To be sure, the asserted *uniqueness* requires the assumption that the Gramian matrix $A = \Lambda'_\tau V$ is 1-1; in particular, this requires that it have at least as many rows as it has columns and be *of full rank*, i.e., its column sequence be linearly independent. But, with that assumption, the matrix $A'A$ is also 1-1 and, since it is square, therefore it is invertible, hence the so-called **normal equations**

$$A'A\mathbf{a} = A'\mathbf{y}$$

have a unique solution. These equations are so called because they identify $\mathbf{c} := A\mathbf{a}$ as the unique element of $\operatorname{ran} A$ for which $\mathbf{y} - A\mathbf{a}$ is perpendicular to $\operatorname{ran} A$, hence for which $\|\mathbf{y} - \mathbf{c}\|_2$ is minimized. In formula,

$$A\mathbf{a} = A(A'A)^{-1}A'\mathbf{y}$$

is the element in $\operatorname{ran} A$ closest to $\mathbf{y}$. Do appreciate the by now familiar structure here: $A\mathbf{a}$ appears as the element of the linear space $\operatorname{ran} A$ recovered from the data $A'\mathbf{y} = (\langle \mathbf{y}, A(:,i)\rangle : i = 1{:}n)$ about $\mathbf{y}$ as written in terms of the basis A for $\operatorname{ran} A$. From this,

$$P_F\mathbf{y} = V(A'A)^{-1}A'\mathbf{y}$$

provides the discrete least-squares approximant to the given data from F. However, for stability reasons, `Matlab` does not actually solve the normal equations when it constructs the vector `A\y`.

Here is an example that further exercises your growing understanding of the use of bases. Suppose that we would like to compute the discrete least-squares approximant to the data $\mathbf{y} \sim \Lambda'_\tau g$ by C^2-cubic splines with break-sequence $\boldsymbol{\xi} = (\xi_1 < \cdots < \xi_{\ell+1})$ and $I = [\xi_1 \,..\, \xi_{\ell+1}]$. Then our F is $S_{4,\mathbf{t}}$, with

$$\mathbf{t} = (t_1 \le \cdots \le t_{n+4}) = (\xi_1, \xi_1, \xi_1, \boldsymbol{\xi}, \xi_{\ell+1}, \xi_{\ell+1}, \xi_{\ell+1})$$

(why?). In particular,

$$n = \ell + 3.$$

Hence, the command `spap2(augknt(`$\boldsymbol{\xi}$`,4),4,`τ`,y)` would provide the desired least-squares approximant, using the B-spline basis. But `spap2` is not part of plain `Matlab`, so, what to do?

Recall that plain `Matlab` does provide a cubic spline interpolant. Precisely, with `xi` $= \boldsymbol{\xi}$ and `z` an arbitrary *row* vector with n entries, the `Matlab` command

```
cs = spline(xi,z);
```

provides (the ppform of) a C^2-cubic spline $f = P_{\boldsymbol{\xi}}\mathbf{z}$ with break-sequence $\boldsymbol{\xi}$, i.e., $f \in F$, that matches the data

$$\Lambda' f := (Df(x_1), f(x_1), f(x_2), \ldots, f(x_{n-1}), f(x_n), Df(x_n)) = \mathbf{z}.$$

($P_{\boldsymbol{\xi}}\mathbf{z}$ is known as a **complete cubic spline interpolant** to the data $\mathbf{z} = \Lambda' f$.) In particular,

$$\Lambda' P_{\mathbf{x}} = \mathrm{id}.$$

This implies that $\Lambda'|_F$ maps F *onto* $\mathrm{I\!R}^n$, hence, since $\dim F = n$, $\Lambda'|_F$ must also be 1-1, hence invertible. But since $P_{\boldsymbol{\xi}}$ is a right inverse for $\Lambda'|_F$, it must be the inverse for it and, in particular, $P_{\boldsymbol{\xi}}$ must be *linear*, 1-1, and onto F, hence a basis for F. It is this basis for F we now will work with.

Set $[v_1, \ldots, v_n] = V := P_{\boldsymbol{\xi}}$. Then, for $j = 1{:}n$, `vj= spline(x,`$\mathbf{i}_j$`)` is the jth term in this basis, with

$$\mathbf{i}_j = [\underbrace{0, \ldots, 0}_{j-1 \text{ terms}}, 1, 0, 0, \ldots] \in \mathrm{I\!R}^n$$

the jth unit vector of length n written as a row-vector, i.e., the jth row of the identity matrix of order n. Since `spline(xi,z,xx)` is set up to return the values at `xx` of the function `spline(xi,z)`, we therefore have

$(v_j(\tau_1), \ldots, v_j(\tau_N)) =$ `spline(xi,`$\mathbf{i}_j$`,tau)`

provided `tau` is the row-vector containing τ. More than that, `spline` is set up to work on *vector-valued* data. Specifically, if `z` is a $d \times n$-matrix, then `spline(xi,z)` is the d-vector valued function whose jth component is the complete cubic spline interpolant to the data `z(j,:)`. In particular, the command `spline(xi,eye(n), tau)` returns the $n \times N$-matrix whose jth row is the vector $(v_j(\tau_1), \ldots, v_j(\tau_N))$. In other words, `spline(xi,eye(n), tau)` is the *transpose* A' of the matrix $A = \Lambda'_\tau V$ we seek.

Since `(A\y)'` is the same as `y'/A'`, it follows that `y'/spline(xi,eye(n), tau)` is the row-vector containing the coefficients $\mathbf{a} = (a_i)$ we seek, in order to put together the spline $V\mathbf{a} = P_{\boldsymbol{\xi}}\mathbf{a}$, hence that spline is obtained by the *single* command

```
l2 = spline(xi,y/spline(xi,eye(n),tau));
```

The idea behind the above construction can be used to construct a least-squares approximant from the range of any linear recovery map P for which one has a computer implementation.

Note again that, strictly speaking, `f = spline(xi,z)` is not the spline recovered from the data `z`, but only a representation for it from which, presumably, we can obtain other information about it; e.g., `ppval(f,tau)` derives from that representation the values of that spline at the entries of `tau`. Hence recovery is, strictly speaking, just a *change* in representation.

