VSPLINE: a subroutine library for non-parametric fixed-interval smoothing with vector splines

Jeffrey A. Fessler*
Information Systems Laboratory
Department of Electrical Engineering
Stanford University · Stanford, CA 94305
E-mail: fessler@isl.stanford.edu
(415) 723-1904

July 10, 2003

1 Introduction

This document describes a software library called VSPLINE that is available via electronic-mail from NETLIB [1]. The VSPLINE library contains a set of subroutines that computes a non-parametric estimate of a smooth vector-valued function from noisy measurements, both linear and nonlinear. The algorithms are derived and described in detail in [2, 3]; this documentation briefly outlines the main ideas and explains the source code interface.

2 Spline smoothing of vector measurements

Consider the *vector* measurement model:

$$\boldsymbol{y}_n = f(\boldsymbol{g}(t_n)) + \boldsymbol{\varepsilon}_n, \;\; n = 1, \ldots, N, \tag{1}$$

$$\boldsymbol{g}(t_n) \in \Re^M, \;\; \boldsymbol{\varepsilon}_n, \boldsymbol{y}_n \in \Re^L, \;\; \boldsymbol{\varepsilon}_n \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}_n), \;\; E\{\boldsymbol{\varepsilon}_n \boldsymbol{\varepsilon}_m'\} = 0, \; n \neq m,$$

with known symmetric and positive-definite error covariances $\boldsymbol{\Sigma}_n$. The goal of smoothing is to estimate the smooth vector-valued function $\boldsymbol{g}$, given the measurements $\{\boldsymbol{y}_n\}_{n=1}^N$.

The nonparametric approach to this problem prescribes a compromise between the conflicting goals of fit to the data and smoothness of the estimated functions. An estimate is obtained by using the following

*This work was supported in part by National Institute of Health contract NO1-HV-38045 and grant R01-HL-39045, National Science Foundation contract ECS-8213959, and GE Medical Systems Group contract 22-84.

criterion, which is the natural generalization of a similar criterion used for scalar measurements:

$$\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{g}} \sum_{n=1}^{N} (\boldsymbol{y}_n - f(\boldsymbol{g}(t_n)))' \boldsymbol{\Sigma}_n^{-1} (\boldsymbol{y}_n - f(\boldsymbol{g}(t_n))) + R_{\boldsymbol{\alpha}}(\boldsymbol{g}), \tag{2}$$

where

$$R_{\boldsymbol{\alpha}}(\boldsymbol{g}) = \sum_{m=1}^{M} \alpha_m \int (\ddot{g}_m(t))^2 \, dt.$$

The smoothing parameter $\boldsymbol{\alpha}$ controls the tradeoff between residual error and smoothness. Although $\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}(t)$ is a continuous function, it is uniquely determined by its values at the knots $t_1, \ldots, t_N$, denoted $\hat{\boldsymbol{y}}_n = \hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}(t_n)$.

The `vspline` library contains special purpose subroutines for efficiently minimizing (2). Two cases are implemented: "linear" smoothing for $f(\boldsymbol{g}) = \boldsymbol{g}$, and "nonlinear" smoothing where the nonlinear function f and its derivatives must be implemented by the user.

Subroutine `vspline`, described below, computes $\{\hat{\boldsymbol{y}}_n\}_{n=1}^N$ given $\boldsymbol{\alpha}$, $\{\boldsymbol{y}_n\}_{n=1}^N$, $\{t_n\}_{n=1}^N$, and $\{\boldsymbol{\Sigma}_n^{-1}\}_{n=1}^N$. The computational requirements [2] are $O(M^3 \cdot N)$.

3 Choosing the smoothing parameters - Cross Validation

The smoothing parameter $\boldsymbol{\alpha}$ can be chosen automatically [2] by minimizing the cross validation (CV) score:

$$\boldsymbol{\alpha}_{\mathrm{CV}} \stackrel{\triangle}{=} \arg\min_{\boldsymbol{\alpha}} \mathrm{CV}(\boldsymbol{\alpha}),$$

$$\mathrm{CV}(\boldsymbol{\alpha}) \stackrel{\triangle}{=} \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{y}_n - f(\hat{\boldsymbol{g}}_{\boldsymbol{\alpha},-n}(t_n)))' \boldsymbol{\Sigma}_n^{-1} (\boldsymbol{y}_n - f(\hat{\boldsymbol{g}}_{\boldsymbol{\alpha},-n}(t_n))). \tag{3}$$

$\hat{\boldsymbol{g}}_{\boldsymbol{\alpha},-n}$ is the solution to the smoothing problem (2) with N-1 data points, posed while excluding the pair $(t_n, \boldsymbol{y}_n)$. Each data pair is dropped in turn, the smoothed curve $\hat{\boldsymbol{g}}_{\boldsymbol{\alpha},-n}$ is estimated, and the predicted value $\hat{\boldsymbol{g}}_{\boldsymbol{\alpha},-n}(t_n)$ is compared with the unused measurement $\boldsymbol{y}_n$. If the CV score is small, then we have chosen the smoothing parameter that makes the estimated curve a good self predictor.

Although equation (3) illustrates the idea behind cross validation, it is computationally inefficient. We have shown that, in the linear case, one can rewrite (3) as:

$$\mathrm{CV}(\boldsymbol{\alpha}) = \frac{1}{N} \sum_{n=1}^{N} \| \boldsymbol{\Sigma}_n^{-\frac{1}{2}} (\boldsymbol{I}_M - \boldsymbol{A}_{(nn)}(\boldsymbol{\alpha}))^{-1} (\boldsymbol{y}_n - \hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}(t_n)) \|^2, \tag{4}$$

where $\boldsymbol{A}_{(nn)}(\boldsymbol{\alpha})$ is the n'th $M \times M$ block diagonal submatrix of the influence matrix. In the nonlinear case, (4) is a useful approximation, as shown in [3]. The properties of this approximation are not completely understood.

By using the Hutchinson and de Hoog algorithm [5], (4) is computed in only O($M^3 N$) operations.

Subroutine `bestcv`, described below, computes $\boldsymbol{\alpha}_{\mathrm{CV}}$ and $\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}_{\mathrm{CV}}}$. The VSPLINE library also contains routines for computing Generalized Cross Validation and Unbiased Risk scores [2].

4 Software Interface - Linear Problems

4.1 Conventions

We have adopted the Numerical Recipes in C [4] convention of using unit offset arrays where convenient (almost everywhere). If an array is declared `double vect[100];` then `(&vect[0]-1)` or `(vect-1)` should be passed to the subroutines below.

4.2 Subroutine `vspline`

The syntax for `vspline` is:

```
int vspline(ys, alpha, tn, yn, inv_sigma, iid, M, N)
float *alpha;
double *ys, *tn, *yn, *inv_sigma;
int iid, M, N;
```

`M` is the dimension (length) the measurement vectors $\boldsymbol{y}_n$. `N` is the number of samples. `alpha` is the M smoothing parameters. `tn` is the N sample points. If the samples are uniform, then set `tn` to `(double *) NULL`. The measurements are stored in `yn` in the order $\{y_{1,1}, \ldots, y_{1,M}, \ldots, y_{N,1}, \ldots, y_{N,M}\}$. If `iid` is 1, the covariances are assumed to be identical, and `inv_sigma` is the M^2 elements of $\boldsymbol{\Sigma}^{-1}$, stored by columns or rows (it is symmetric). However, if `iid` is 0, the covariances can vary with n, and `inv_sigma` is a $N \cdot M^2$ array of the concatenation of the elements of $\boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_N^{-1}$ in the natural order.

On return, `ys` is the $N \cdot M$ array of $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_N$, stored in the same order as `yn`. `vspline` returns 1 if all goes well, 0 if there is an error. The possible errors are: running out of memory, `tn` not strictly increasing, or singular covariance matrices.

4.3 Subroutine `bestcv`

The syntax for `bestcv` is identical to that of `vspline`. The difference is that `alpha` is used as an initial estimate for $\boldsymbol{\alpha}_{\mathrm{CV}}$. On return, `alpha` is set to $\boldsymbol{\alpha}_{\mathrm{CV}}$, and `ys` is the smoothed estimate at that value of $\boldsymbol{\alpha}$.

4.4 Subroutine `cubic_interp`

Each component function of $\hat{\boldsymbol{g}}_{\boldsymbol{\alpha}}(t)$ is a scalar cubic spline. To compute values of $\hat{g}_{\boldsymbol{\alpha},m}(t)$ for values of t other than the knots, call `cubic_interp`.

```
int cubic_interp(yk, tn, ys, tk, N, K)
double *yk, *tn, *ys, *tk;
int N, K;
```

`tn` is the N sample points, and `ys` is the N smoothed estimates for one of the component functions, e.g. $\{\hat{y}_{n,m}\}_{n=1}^{N}$. `tk` is K new points at which $\hat{g}_{\boldsymbol{\alpha},m}(t)$ is to be evaluated. On return, `yk` is the K interpolated samples. `cubic_interp` returns 1 if successful, 0 otherwise. `cubic_interp` is not very efficient, and may not be exact for values of t outside of $[t_1, t_N]$.

5 Installation

VSPLINE is distributed as a Bourne Shell "bundle". Create a directory called `vspline`, move the bundled file to that directory, and type `sh` *filename* at a UNIX prompt. The file `README` contains important installation instructions.

When installed, the C source code resides in several subdirectories. `matrix` contains very general vector and matrix utility subroutines. Simply by changing the definitions in `matrix/matrix.h`, one could

use a different library. `smooth` contains utilities for banded matrices, and the smoothing subroutines themselves. Finally, `Test` contains programs we used to test the library. The test programs were applied to the data files in `Data`; to verify your installation, follow the instructions in `Data/README`. One `Makefile` in the parent directory controls compilation of all three subdirectories. This Makefile creates five libraries: `libnonlin.a, libsmooth.a, liboptima.a, libmatrix.a, libiodep.a`, which should be linked *in the order given* when compiling. The Makefile in the Test directory provides a good example.

`vspline` resides in `smooth/example1.c`, `bestcv` in `smooth/example2.c`. A subroutine called `vect_smooth` in `smooth/example1.c` gives another example of how the VSPLINE library can used. `cubic_interp` resides in `smooth/interp.c`. The LaTeXsource for this document is in `smooth/Doc`.

References

[1] J. J. Dongarra and E. Grosse, "Distibution of mathematical software via electronic mail," *Comm. ACM*, vol. 30, pp. 403–407, Oct. 1987.

[2] J. Fessler, "Nonparametric fixed-interval smoothing with vector splines," *IEEE Transactions on Signal Processing*. Apr. 1991.

[3] J. Fessler, "Nonlinear smoothing with vector splines," *IEEE Transactions on Signal Processing*. Apr. 1991.

[4] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. Cambridge Univ. Press, 1988.

[5] M. Hutchinson and F. deHoog, "Smoothing noisy data with spline functions," *Numerische Mathematik*, vol. 47, pp. 99–106, 1985.