Computational Statistics Exam

1 - Bivariate Smoothing

Martin Hoshi Vognsen

Implementation 1: Vectorized

Cubic spline: Order 4

Implementation 2: Demmler-Reinsch basis

Penalized normal equation for smoother:

$$\hat{\mathbf{f}} = \underbrace{\mathbf{\Phi}((\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{\Omega})^{-1}\mathbf{\Phi}^T}_{\mathbf{S}_\lambda}\mathbf{y} \tag{1}$$

$$\underset{n\times p}{\mathbf{\Phi}} = \underset{n\times n}{\mathbf{U}} \begin{bmatrix} \underset{n\times p}{\mathbf{D}} \\ \underset{(n-p)\times p}{\mathbf{0}} \end{bmatrix} \underset{p\times p}{\mathbf{V}^T} =: \mathbf{U\Sigma V}^T \quad (\text{here } p \le n) \tag{2}$$

$$\begin{aligned} S_\lambda &= \mathbf{\Phi}(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{\Omega})^{-1}\mathbf{\Phi}^T \\ &= \mathbf{U\Sigma V}^T\left((\mathbf{U\Sigma V})^T\mathbf{U\Sigma V}^T + \lambda\mathbf{\Omega}\right)^{-1}(\mathbf{U\Sigma V}^T)^T \\ &= \mathbf{U\Sigma V}^T\left(\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{U\Sigma V}^T + \lambda\mathbf{\Omega}\right)^{-1}\mathbf{V\Sigma}^T\mathbf{U}^T \\ &= \mathbf{U\Sigma V}^T\left(\mathbf{\Sigma}^2 + \lambda\mathbf{\Omega}\right)^{-1}\mathbf{V\Sigma U}^T \quad (\text{ortogonality}) \\ &= \mathbf{U}\left(\mathbf{I} + \lambda\mathbf{\Sigma}^+\mathbf{V}^T\mathbf{\Omega V\Sigma}^+\right)^{-1}\mathbf{U}^T \\ &= \mathbf{U}\left(\mathbf{I} + \lambda\widetilde{\mathbf{\Omega}}\right)^{-1}\mathbf{U}^T \\ &= \mathbf{U}\left(\mathbf{I} + \lambda\mathbf{W\Gamma W}^T\right)^{-1}\mathbf{U}^T \quad (\text{diagonalization}) \\ &= \mathbf{UW}\left(\mathbf{I} + \lambda\mathbf{\Gamma}\right)^{-1}\mathbf{W}^T\mathbf{U}^T \\ &= \widetilde{\mathbf{U}}\left(\mathbf{I} + \lambda\mathbf{\Gamma}\right)^{-1}\widetilde{\mathbf{U}}^T \quad (\widetilde{\mathbf{U}} := \mathbf{UW}) \end{aligned} \tag{3}$$

$$\widehat{\beta} = \widetilde{\mathbf{U}}^T\mathbf{y} \tag{4}$$

$$\widehat{\beta}_i(\lambda) = \frac{\widehat{\beta}_i}{1 + \lambda\ \mathrm{diag}(\Gamma)_i} \tag{5}$$

$$\widehat{\mathbf{f}} = \widetilde{\mathbf{U}}\widehat{\beta}(\lambda) \tag{6}$$

```
smoother_dr = function(y, lambda, U_tilde, Gamma) {
   beta_hat = crossprod(U_tilde, y) #(4)
   beta_hat_lam = beta_hat/(1+(lambda * Gamma))
     #(5)
   U_tilde %*% beta_hat_lam #\nsucceq(6)
DR = function(X, inner_knots, Omega){
   Phi <- splineDesign(c(rep(range(inner_knots),
     3), inner_knots), X)
   Phi_svd = svd(Phi)
   Omega_tilde = t(crossprod(Phi_svd$v, Omega %*%
     Phi_svd$v)) / Phi_svd$d
   Omega_tilde = t(Omega_tilde) / Phi_svd$d
   Omega_tilde_svd = svd(Omega_tilde)
   U_tilde = Phi_svd$u %*% Omega_tilde_svd$u
   W = Omega_tilde_svd$u
   Gamma = abs(diag(crossprod(W, Omega_tilde %*%
     W)))
   list(U_tilde = U_tilde, Gamma = Gamma,
     Omega_tilde = Omega_tilde, W = W)
}
```

Test data

Empirical data

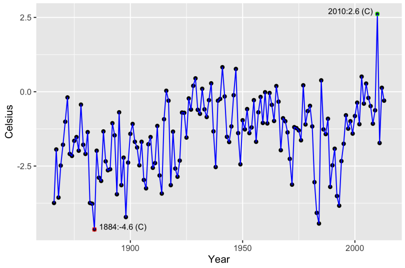

Figure 1: Nuuk temperatures 1866-2013: Annual means

Simulated data

$$Y = \sin(X) \cdot 10 - X + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 4)$$

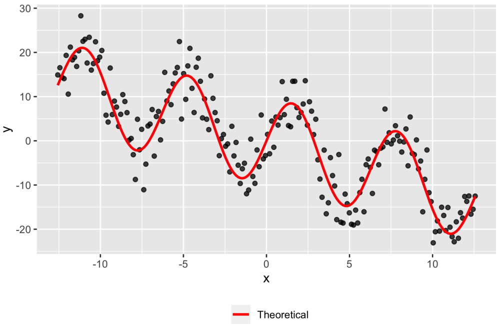

Figure 2: Simulated data

Optimal λ

Selecting λ with LOOCV
Empirical data
Note: The inclusion of one single data point (1866) changes `opt_lambda` from 130 to 74!

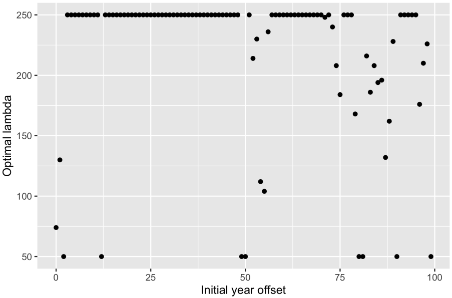

Figure 3: Optimal lambda w.r.t. initial year offset from 1866

Tests: Internal inspection

-

Tests: Correctness

Tests: Robustness

-

Profiling

```
> profvis(smoother_1(Y, 1, Phi, Omega),
interval = 0.01)
```

/r/CS1_functions.R	Memory	Time
`smoother_1 = function(y, lambda, Phi, Omega) {`		
`  Phi %*% solve(`		10
`    crossprod(Phi) + lambda * Omega,`		
`    t(Phi)`		
`    %*% y)`		
`}`		

Figure 4: Basic smoother function profile. Time in ms.

Code	File	Memory (MB)		Time (ms)
smoother_1		0	0	10
solve	CS1_functions.R	0	0	10
standardGeneric		0	0	10
+	CS1_functions.R	0	0	10
+		0	0	10
callGeneric		0	0	10
eval		0	0	10
eval		0	0	10
+		0	0	10
+		0	0	10
+		0	0	10
forceSymmetric		0	0	10
callGeneric		0	0	10
eval		0	0	10
eval		0	0	10
+		0	0	10
+		0	0	10
+		0	0	10
.Arith.Csparse		0	0	10
newTMat		0	0	10
new		0	0	10
initialize		0	0	10
initialize		0	0	10
callNextMethod		0	0	10
addNextMethod		0	0	10
addNextMethod		0	0	10
.findNextFromTable		0	0	10
new		0	0	10
initialize		0	0	10
initialize		0	0	10

Figure 5: Basic smoother function profile.

Benchmark

Zero-elements in Ω-matrix: 96%
Zero-elements in Φ-matrix: 98%
$\rightarrow$ sparse matrix

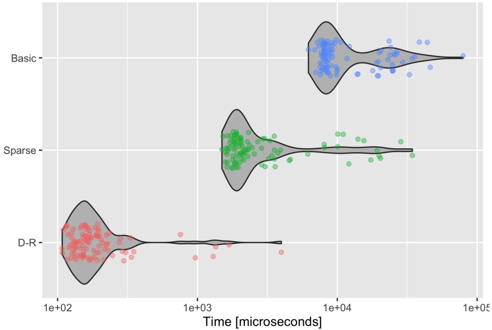

Figure 6: Top to bottom: Basic smoother, Basic smoother with sparse matrices (Φ and Ω), D-R smoother.

Basic smoother (vectorized):

```
smoother_1 = function(y, lambda, Phi, Omega) {
   Phi %*% solve(
      crossprod(Phi) + lambda * Omega,
      t(Phi) %*% y)
}
```

D-R smoother:

```
smoother_dr = function(y, lambda, U_tilde, Gamma) {
   beta_hat_lambda = crossprod(U_tilde, y)
   beta_hat_lambda = beta_hat_lambda/(1+(lambda *
     Gamma))
   U_tilde %*% beta_hat_lambda
}
```

Benchmark (cont.)

Alternative implementations

- RCPP
- OOP
- pipe

- - - END OF SLIDES - - -

Notes

Density estimation

- Circularity: Optimal bandwidth h_n depends via $||f''0||_2^2$ upon the unknown f_0 that we are trying to estimate. We therefore refer to h_n as an oracle bandwidth – it is the bandwidth that an oracle that knows f_0 would tell us to use. In practice, we will have to come up with an estimate of $||f''0||_2^2$ and plug that estimate into the formula for h_n. (CSwR 2.3)
    - AMISE (CSwR 2.3.3)
    - CV (CSwR 2.3.4)
- Silverman's rule-of-thumb (CSwR 2.3.3)
    - Tends to oversmooth
    - Sheather-Jones is suggested as a better default for `density` than Silverman's rule-of-thumb
- Time complexity (CSwR 1.1.5)
- Curse of dimensionality → sparse matrices (CSwR 1.1.5)
- At least 2 reasons why Monte Carlo integration is sometimes preferable (CSwR 1.2.2):
    - Straightforward to implement
        - often works quite well for multivariate and even high-dimensional integrals
        - grid-based numerical integration schemes scale poorly with the dimension.
    - Does not require that we have an analytic representation of the density.
- Choosing the right amount of regularization is just as important as choosing the method to use in the first place. (CSwR ch. 2.0)
- Method of sieves (CSwR 2.1.2)
- Basis expansion: h should grow rather slowly with n to avoid overfitting (CSwR 2.1.3)
- Kernel smoothing: Variance vs bias tradeoff (CSwR 2.2)

Bivariate smoothing

- Simpson's rule (CSwR 3.5.2)
- GCV vs LOOCV (CSwR 3.5.2+3.5.3)

Benchmarking

- CSwR 2.2.2: Deviations from straight lines on log-log plot. Writing run time as $Cn^a + R(n)$, residual term $R(n)$ often noticeable or even dominating and positive for small n. Only for large enough n power law term, Cn^a, will dominate.
- Consider time to compute sparse matrix S

Smoothing

- Why bivariate?
- Demmler-Reinsch splines: The closer `num_knots` is to `n`, the closer the end points are to zero (why?)
- Smoothing splines (Basis splines): See Elements Of Statistical Learning pp 141ff + 186ff.
  - `splineDesign(c(rep(range(inner_knots), 3), inner_knots), X)`
    rettes til
    `knots = sort(c(rep(range(inner_knots), 3), inner_knots)`
    `splineDesign(knots, inner_knots)`
    ?? Se ex med Nuuk data. Ikke sorteret. Hvorfor rep tre par til venstre og ingenting til højre: (min, max, min, max, min, max). Testet Nuuk data og simuleret data. Ingen synlig forskel med/uden sort.
- Demmler-Reinsch: Se Lay, s. 440: Når U og V har fuld rang anvendes "Reduced SVD". Men så er U og V ikke ortogonale.

*** ToDo ***

- select λ: I min kode anvendes GCV, men vi skal opgaven siger LOOCV
- `smoother_dr`: `Omega` indgår ikke
- Benchmark autoplot:
  - Add non-vectorized
  - Add `smooth.spline`...
- Benchmark: (see CSwR fig 2.6)
  - Test w.r.t. n...
  - Test w.r.t. number of splines...
  - Sparse matrix: See 3.3
    - Evt `image(Phi)`, image(Omega)
- Correctness:
  - Comparison with `smooth.spline`
    - OBS: smooth.spline anvender GCV i stedet for LOOCV!
    - OBS: `smooth.spline` heuristically selects less than n knots, med unless all.knots = TRUE (CSwR 3.5.3)
    - See `.nknots.smspl`
  - Compute differences in output (`range()`)
  - Test Ω (`pen_mat`) (CSwR 3.5.2). Ω implemented using Simpson's rule.
- Robustness: Extreme values/asymptotics, ...

Optimal λ

-