

Recursive and adaptive estimation

Classical results and how to modify them

Erik Lindström

Centre for Mathematical Sciences
Lund University

LU/LTH & DTU

Overview

Introduction

Naive recursive estimators

Recursive LS

Recursive Pseudo-Linear Regression

Recursive Prediction Error Method

Recursive Maximum Likelihood

Stochastic Approximation

Filtering

Why **R**ecursive and/or **A**daptive estimation

A recursive estimator is defined as

$$\theta_{n+1} = \theta_n + f(\theta_n, Y_{n+1} | Y_{1:n})$$

Useful for

R Large data sets - computational considerations

R Streaming data

A Knowledge that the system is changing, but little knowledge of how

A As a final tweaking of the model

► **Note** that the stability of the system may be negatively impacted when using adaptive methods, cf. the Lyapunov theory on Random Coefficient AR models during the first lecture.

Why **R**ecursive and/or **A**daptive estimation

A recursive estimator is defined as

$$\theta_{n+1} = \theta_n + f(\theta_n, Y_{n+1} | Y_{1:n})$$

Useful for

R Large data sets - computational considerations

R Streaming data

A Knowledge that the system is changing, but little knowledge of how

A As a final tweaking of the model

► **Note** that the stability of the system may be negatively impacted when using adaptive methods, cf. the Lyapunov theory on Random Coefficient AR models during the first lecture.

Why **R**ecursive and/or **A**daptive estimation

A recursive estimator is defined as

$$\theta_{n+1} = \theta_n + f(\theta_n, Y_{n+1} | Y_{1:n})$$

Useful for

- R** Large data sets - computational considerations
- R** Streaming data
- A** Knowledge that the system is changing, but little knowledge of how
- A** As a final tweaking of the model
- ▶ **Note** that the stability of the system may be negatively impacted when using adaptive methods, cf. the Lyapunov theory on Random Coefficient AR models during the first lecture.

Why **R**ecursive and/or **A**daptive estimation

A recursive estimator is defined as

$$\theta_{n+1} = \theta_n + f(\theta_n, Y_{n+1} | Y_{1:n})$$

Useful for

- R** Large data sets - computational considerations
 - R** Streaming data
 - A** Knowledge that the system is changing, but little knowledge of how
 - A** As a final tweaking of the model
- **Note** that the stability of the system may be negatively impacted when using adaptive methods, cf. the Lyapunov theory on Random Coefficient AR models during the first lecture.

Why **R**ecursive and/or **A**daptive estimation

A recursive estimator is defined as

$$\theta_{n+1} = \theta_n + f(\theta_n, Y_{n+1} | Y_{1:n})$$

Useful for

- R** Large data sets - computational considerations
- R** Streaming data
- A** Knowledge that the system is changing, but little knowledge of how
- A** As a final tweaking of the model
- ▶ **Note** that the stability of the system may be negatively impacted when using adaptive methods, cf. the Lyapunov theory on Random Coefficient AR models during the first lecture.

Different types

- ▶ Forgetting type estimators
- ▶ Converging estimators

Ex: $Z_i \in \mathcal{N}(\mu, 1)$. Estimate the mean (μ) as

$$\hat{\mu}_N = \frac{1}{N} \sum Z_i$$

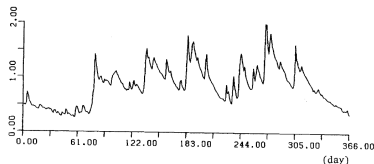
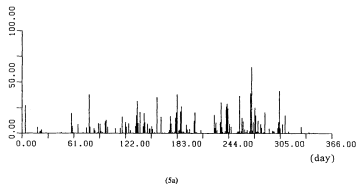
or as

$$\hat{\mu}_N = Z_N?$$

Different properties and applications!

Kanna river revisited

Rain (left) and flow (right)



An autoregression model would need time varying parameters

Naive approaches - Windowed estimation

- ▶ Use sliding window $\{y_{t-u:t}\}$ to estimate parameters

$$\hat{\theta}_t = \operatorname{argmax} \sum_{n=t-u}^t \log p(y_n | y_{t-u}, \dots, y_{n-1})$$

- ▶ Followed by

$$\hat{\theta}_{t+1} = \operatorname{argmax} \sum_{n=t-u+1}^{t+1} \log p(y_n | y_{t-u+1}, \dots, y_{n-1})$$

Properties?

Recursive LS

- ▶ Linear models can be written as

$$Y = Z\theta + e$$

- ▶ Estimate is given by

$$\hat{\theta} = (Z^T Z)^{-1}(Z^T Y)$$

This can be written in recursive form!

Recursive LS for AR(p) processes

- Optimize

$$\hat{\theta}_t = \operatorname{argmin} \sum_{s=p}^t (Y_s - X_s^T \theta)^2$$

- where

$$X_t^T = [-Y_{t-1}, \dots, -Y_{t-p}]$$

and

$$\theta^T = [\theta_1, \dots, \theta_p]$$

- This can be written as

$$\begin{aligned} \hat{\theta}_t &= (Z^T Z)^{-1} (Z^T Y) = R_t^{-1} h_t \\ R_t &= Z^T Z = \sum X_s X_s^T \\ h_t &= Z^T Y = \sum X_s Y_s \end{aligned} \tag{1}$$

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t \pm X_t X_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned}$$

(2)

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t \pm X_t X_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned}$$

(2)

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t \pm X_t X_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned}$$

(2)

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t \pm \mathbf{X}_t \mathbf{X}_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned}$$

(2)

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t \pm X_t X_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned}$$

(2)

Recursive LS

- ▶ We can write $R_t = R_{t-1} + X_t X_t^T$
- ▶ and $h_t = h_{t-1} + X_t Y_t$

Considering a recursive estimator, where only recent information influence the estimate, we write

$$\begin{aligned}
 \hat{\theta}_t &= R_t^{-1} h_t \\
 &= R_t^{-1} (h_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t) \\
 &= R_t^{-1} (R_{t-1} \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1}) \\
 &= R_t^{-1} \left((R_{t-1} + X_t X_t^T) \hat{\theta}_{t-1} + X_t Y_t - X_t X_t^T \hat{\theta}_{t-1} \right) \\
 &= R_t^{-1} \left(R_t \hat{\theta}_{t-1} + X_t (Y_t - X_t^T \hat{\theta}_{t-1}) \right) \\
 &= \hat{\theta}_{t-1} + R_t^{-1} X_t (Y_t - X_t^T \hat{\theta}_{t-1})
 \end{aligned} \tag{2}$$

Recursive LS

- We have that $R_t = R_{t-1} + X_t X_t^T$
- but are interested in R_t^{-1}

The matrix inversion lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}$$

gives $A = R_{t-1}$, $B = X_t$, $C = I$ and $D = X_t^T$

$$R_t^{-1} = R_{t-1}^{-1} - R_{t-1}^{-1}X_t(X_t^T R_{t-1}^{-1}X_t + I)^{-1}X_t^T R_{t-1}^{-1} \quad (3)$$

The RLS algorithm is then given by two simple matrix expressions!

Recursive LS

- ▶ We have that $R_t = R_{t-1} + X_t X_t^T$
- ▶ but are interested in R_t^{-1}

The matrix inversion lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}$$

gives $A = R_{t-1}$, $B = X_t$, $C = I$ and $D = X_t^T$

$$R_t^{-1} = R_{t-1}^{-1} - R_{t-1}^{-1}X_t(X_t^T R_{t-1}^{-1}X_t + I)^{-1}X_t^T R_{t-1}^{-1} \quad (3)$$

The RLS algorithm is then given by two simple matrix expressions!

Recursive LS

- ▶ We have that $R_t = R_{t-1} + X_t X_t^T$
- ▶ but are interested in R_t^{-1}

The matrix inversion lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}$$

gives $A = R_{t-1}$, $B = X_t$, $C = I$ and $D = X_t^T$

$$R_t^{-1} = R_{t-1}^{-1} - R_{t-1}^{-1}X_t(X_t^T R_{t-1}^{-1}X_t + I)^{-1}X_t^T R_{t-1}^{-1} \quad (3)$$

The RLS algorithm is then given by two simple matrix expressions!

Recursive LS

- ▶ We have that $R_t = R_{t-1} + X_t X_t^T$
- ▶ but are interested in R_t^{-1}

The matrix inversion lemma

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}$$

gives $A = R_{t-1}$, $B = X_t$, $C = I$ and $D = X_t^T$

$$R_t^{-1} = R_{t-1}^{-1} - R_{t-1}^{-1}X_t(X_t^T R_{t-1}^{-1}X_t + I)^{-1}X_t^T R_{t-1}^{-1} \quad (3)$$

The RLS algorithm is then given by two simple matrix expressions!

Adaptive Recursive LS

Known parameter variations should be included in the model, *slow unknown variations* can be dealt with using adaptive estimation.

Optimize

$$\hat{\theta}_t = \operatorname{argmin} \sum_{s=p}^t \beta(t, s) (Y_s - X_s^T \theta)^2$$

where

$$\begin{aligned} \beta(t, s) &= \lambda(t) \beta(t-1, s) \\ \beta(t, t) &= 1 \end{aligned} \tag{4}$$

Hence is $\beta(t, s) = \prod_{j=s+1}^t \lambda(j)$.

Again, recursive equations can be found!

Adaptive Recursive LS

Known parameter variations should be included in the model, *slow unknown variations* can be dealt with using adaptive estimation.

Optimize

$$\hat{\theta}_t = \operatorname{argmin} \sum_{s=p}^t \beta(t, s) (Y_s - X_s^T \theta)^2$$

where

$$\begin{aligned} \beta(t, s) &= \lambda(t) \beta(t-1, s) \\ \beta(t, t) &= 1 \end{aligned} \tag{4}$$

Hence is $\beta(t, s) = \prod_{j=s+1}^t \lambda(j)$.

Again, recursive equations can be found!

Adaptive Recursive LS

The solution is given by

$$\hat{\theta}_t = \tilde{R}_t^{-1} \tilde{h}_t$$

where

- ▶ $\tilde{R}_t = \lambda(t) \tilde{R}_{t-1} + X_t X_t^T$
- ▶ $\tilde{h}_t = \lambda(t) \tilde{h}_{t-1} + X_t Y_t$

And the rest is identical to the standard RLS.

- ▶ Interpretation of λ in terms of effective variance.

Recursive Pseudo-Linear Regression

- ▶ Extend

$$Y = X\theta + e$$

- ▶ To

$$Y = X(\theta)\theta + e$$

Includes e.g. ARMA and non-linear models!

(Adaptive) RPLR - Think ARMA models!

Let

$$\hat{\theta}_t = \operatorname{argmin} S_t(\theta)$$

where

$$S_t(\theta) = \sum_s^t \beta(t, s) (Y_s - X_s^T(\theta)\theta)^2$$

- ▶ $S_t(\theta) = \lambda(t)S_{t-1}(\theta) + (Y_t - X_t^T(\theta)\theta)^2$
- ▶ Taylor expand around $\hat{\theta}_{t-1}$

(Adaptive) RPLR

- Taylor expansion

$$S_t(\theta) \approx S_t(\hat{\theta}_{t-1}) + \nabla S_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}) + \frac{1}{2}(\theta - \hat{\theta}_{t-1})^T H_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}), \quad (5)$$

where H_t is the Hessian.

- $\nabla S_t(\hat{\theta}_{t-1}) \approx -2X_t(\hat{\theta}_{t-1}) \left(Y_t - X_t^T(\hat{\theta}_{t-1})\hat{\theta}_{t-1} \right)$
- $R_t = \frac{1}{2}H_t = \lambda(t)R_{t-1} + X_t(\hat{\theta}_{t-1})X_t^T(\hat{\theta}_{t-1})$
- This gives the estimators as

$$\hat{\theta}_t = \hat{\theta}_{t-1} + R_t^{-1}X_t(\hat{\theta}_{t-1}) \left(Y_t - X_t^T(\hat{\theta}_{t-1})\hat{\theta}_{t-1} \right)$$

- Implications in terms of robustness?

(Adaptive) RPLR

- Taylor expansion

$$S_t(\theta) \approx S_t(\hat{\theta}_{t-1}) + \nabla S_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}) + \frac{1}{2}(\theta - \hat{\theta}_{t-1})^T H_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}), \quad (5)$$

where H_t is the Hessian.

- $\nabla S_t(\hat{\theta}_{t-1}) \approx -2X_t(\hat{\theta}_{t-1}) \left(Y_t - X_t^T(\hat{\theta}_{t-1})\hat{\theta}_{t-1} \right)$
- $R_t = \frac{1}{2}H_t = \lambda(t)R_{t-1} + X_t(\hat{\theta}_{t-1})X_t^T(\hat{\theta}_{t-1})$
- This gives the estimators as

$$\hat{\theta}_t = \hat{\theta}_{t-1} + R_t^{-1}X_t(\hat{\theta}_{t-1}) \left(Y_t - X_t^T(\hat{\theta}_{t-1})\hat{\theta}_{t-1} \right)$$

- Implications in terms of robustness?

(Adaptive) RPEM

Let

$$\hat{\theta}_t = \operatorname{argmin} S_t(\theta)$$

where

$$S_t(\theta) = \sum_s^t \beta(t, s) (Y_s - \hat{Y}_{s|s-1}(\theta))^2$$

- ▶ Approximate by a $2nd$ order polynomial
- ▶ Optimize using Newton-Raphson

(Adaptive) RPEM

- Taylor expansion

$$\begin{aligned} S_t(\theta) &\approx S_t(\hat{\theta}_{t-1}) + \nabla S_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}) \\ &\quad + \frac{1}{2}(\theta - \hat{\theta}_{t-1})^T H_t(\hat{\theta}_{t-1})(\theta - \hat{\theta}_{t-1}), \end{aligned} \quad (6)$$

where H_t is the Hessian.

- Solution is given by

$$\hat{\theta}_t = \hat{\theta}_{t-1} - H_t(\hat{\theta}_{t-1})^{-1} \nabla S_t(\hat{\theta}_{t-1})$$

(Adaptive) RPEM

Note that

- ▶ $S_t(\theta) = \lambda(t)S_{t-1}(\theta) + (Y_t - \hat{Y}_{t|t-1}(\theta))^2$
- ▶ $\nabla S_t(\theta) = \lambda(t)\nabla S_{t-1}(\theta) + 2(Y_t - \hat{Y}_{t|t-1}(\theta))\nabla \hat{Y}_{t|t-1}(\theta)$
- ▶ $\nabla S_t(\hat{\theta}_{t-1}) \approx 2(Y_t - \hat{Y}_{t|t-1}(\theta))\nabla \hat{Y}_{t|t-1}(\hat{\theta}_{t-1})$
- ▶ The Hessian is given by

$$H_t(\theta) = 2 \sum \beta(t, s) \nabla \hat{Y}_{s|s-1}(\theta) \nabla \hat{Y}_{s|s-1}^T(\theta) - 2 \sum \beta(t, s) \nabla \nabla \hat{Y}_{s|s-1}(\theta) (Y_s - \hat{Y}_{s|s-1}(\theta)) \quad (7)$$

- ▶ $H_t(\hat{\theta}_{t-1}) \approx \lambda(t)H_{t-1} + 2\nabla \hat{Y}_{t|t-1}(\hat{\theta}_{t-1})\nabla \hat{Y}_{t|t-1}^T(\hat{\theta}_{t-1})$

(Adaptive) RPEM

This gives

- ▶ $R_t = \frac{1}{2}H_t$
- ▶ $\hat{\theta}_t = \hat{\theta}_{t-1} + R_t^{-1} \nabla \hat{Y}_{t|t-1}(\hat{\theta}_{t-1}) \left(Y_t - \hat{Y}_{t|t-1}(\hat{\theta}_{t-1}) \right)$
- ▶ $R_t = \lambda(t)R_{t-1} + \nabla \hat{Y}_{t|t-1}(\hat{\theta}_{t-1}) \nabla \hat{Y}_{t|t-1}^T(\hat{\theta}_{t-1})$

Use matrix inversion lemma to obtain an efficient recursion.

Recursive ML

It is possible to construct optimal recursive estimators for non-Gaussian models via likelihood theory

$$\blacktriangleright \hat{\theta}_t = \operatorname{argmax} \sum_{n=1}^t \log p(y_n | y_{1:n-1}, \theta) = \operatorname{argmax} \ell_t(\theta)$$

Taylor expand and maximize

$$\nabla \ell_t(\hat{\theta}_t) \approx \nabla \ell_t(\hat{\theta}_{t-1}) + \nabla \nabla \ell_t(\hat{\theta}_{t-1})(\hat{\theta}_t - \hat{\theta}_{t-1}) \quad (8)$$

$$= \nabla \ell_{t-1}(\hat{\theta}_{t-1}) + \nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1}) \quad (9)$$

$$+ \nabla \nabla \ell_t(\hat{\theta}_{t-1})(\hat{\theta}_t - \hat{\theta}_{t-1}) = 0. \quad (10)$$

Simplification gives

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \frac{1}{t} \hat{I}^F(\hat{\theta}_{t-1})^{-1} \nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$$

Sketch of proof

$$\ell_t(\theta) \approx \ell_t(\hat{\theta}_t) + \nabla \ell_t(\cdot)^T (\theta - \hat{\theta}_{t-1}) + \frac{1}{2} (\theta - \hat{\theta}_{t-1})^T \nabla \nabla \ell_t(\cdot) (\theta - \hat{\theta}_{t-1})$$

with $\nabla \ell_t(\cdot) = \nabla \ell_t(\cdot)$

- Compute gradient wrt θ leading to

$$\nabla \ell_t(\theta) \approx 0 + \nabla \ell_t(\cdot) + \nabla \nabla \ell_t(\cdot) (\hat{\theta}_t - \hat{\theta}_{t-1}) \triangleq 0$$

- Recalling properties of the Fisher Information and large sample approximation leads to

$$\nabla \ell_t(\hat{\theta}_{t-1}) - t \hat{I}^F(\hat{\theta}_{t-1})(\hat{\theta}_t - \hat{\theta}_{t-1}) = 0$$

- Using that $\nabla \ell_t(\hat{\theta}_{t-1}) \approx \nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$ yields the result

Approximative ML, computational simplification

What if $\nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$ is expensive to compute?

- ▶ A stochastic process is *mixing* if its values at widely-separated times are *asymptotically independent*
- ▶ Examples of this is Markov chains, but the concept is more general.
- ▶ Approximate $\nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$ by the **fixed-lag** (L) estimate $\nabla \log p(y_n | y_{n-L:n-1}, \hat{\theta}_{t-1})$ as the error vanishes as $L \rightarrow \infty$.
- ▶ This has been used for fixed-lag EM-algorithms, or when approximating an MA processes with a $AR(p)$ model, p being large.

Approximative ML, computational simplification

What if $\nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$ is expensive to compute?

- ▶ A stochastic process is *mixing* if its values at widely-separated times are *asymptotically independent*
- ▶ Examples of this is Markov chains, but the concept is more general.
- ▶ Approximate $\nabla \log p(y_n | y_{1:n-1}, \hat{\theta}_{t-1})$ by the **fixed-lag** (L) estimate $\nabla \log p(y_n | y_{n-L:n-1}, \hat{\theta}_{t-1})$ as the error vanishes as $L \rightarrow \infty$.
- ▶ This has been used for fixed-lag EM-algorithms, or when approximating an MA processes with a $AR(p)$ model, p being large.

Robbins-Monro stochastic approximation

- ▶ This is a special case of the Robbins-Monro stochastic approximation algorithm
- ▶ Problem: $x^* = \operatorname{argmin} G(x)$
- ▶ Introduce $x_{n+1} = x_n + \frac{a}{(1+n+A)^\alpha} g(x_n)$ with $\alpha \in (1/2, 1]$
- ▶ where x is a parameter, a some positive def. matrix, and $g(x)$ is a noisy gradient of G ,

$$g(x) = \nabla G(x) + \epsilon.$$

- ▶ It then holds that

$$x_n \xrightarrow{a.s.} x^* \tag{11}$$

$$N^{\alpha/2}(x_n - x^*) \xrightarrow{d} N(0, \Sigma) \tag{12}$$

Interpretations. Weak convergence, implications for Σ .

Robbins-Monro stochastic approximation

- ▶ This is a special case of the Robbins-Monro stochastic approximation algorithm
- ▶ Problem: $x^* = \operatorname{argmin} G(x)$
- ▶ Introduce $x_{n+1} = x_n + \frac{a}{(1+n+A)^\alpha} g(x_n)$ with $\alpha \in (1/2, 1]$
- ▶ where x is a parameter, a some positive def. matrix, and $g(x)$ is a noisy gradient of G ,

$$g(x) = \nabla G(x) + \epsilon.$$

- ▶ It then holds that

$$x_n \xrightarrow{a.s.} x^* \tag{11}$$

$$N^{\alpha/2}(x_n - x^*) \xrightarrow{d} N(0, \Sigma) \tag{12}$$

Interpretations. Weak convergence, implications for Σ .

SP/FD stochastic approximation

- ▶ The gradient can be approximated by finite difference at the cost of slower convergence, $\mathcal{O}(n^{-1/2}) \rightarrow \mathcal{O}(n^{-1/3})$.
- ▶ but clever methods (SPSA) are still fairly fast
- ▶ Idea: Many steps are taken, and the gradient is being averaged over the iterations.
- ▶ SPSA only evaluates a single central finite difference (randomly selected) per iteration and averages again over the iterations.

Result: Computational gain is asymp. equal to the dimension of x (which can be huge for deep nets).

Two scale Stochastic Approximation

Also known as *Averaged Stochastic Gradient* method.

- ▶ Very simple idea, the SA iterates converge slowly towards x^* .
- ▶ Optimal rate of convergence is difficult to obtain in practice (depends of the Hessian of the unknown function).
- ▶ Solution: Average over the sequence.

Two scale Stochastic Approximation

- Algorithm (note that $a_n = an^{-\alpha}$ for $\alpha \in (1/2, 1)$):

$$z_{n+1} = z_n + a_n g(z_n) \quad (13)$$

$$x_{n+1} = x_n + \frac{1}{n}(z_{n+1} - x_n) \quad (14)$$

- Surprising result: Optimal rate of convergence,
 $N^{1/2}(x_n - x^*) \xrightarrow{d} N(0, \tilde{\Sigma})$!
- Matlab code: `StochApproxMain.m`

Stochastic approximation in Machine Learning

Training often done using gradient or stochastic gradient methods.

- ▶ Gradient methods are expensive, but asymptotically faster
- ▶ Stochastic gradient provides rapid initial convergence.

Use block update as a compromise. Choose *random subset* $I_n \subset \{1, \dots, N\}$ of size B and use

$$x_{n+1} = x_n + \frac{a}{(1 + n + A)^\alpha} \frac{1}{B} \sum_{k \in I_n} g_k(x_k)$$

with $\alpha \in (1/2, 1]$ and g_k being the k th term in the log likelihood function. This is still a noisy gradient, but the blocks increase the computational demand by B and reduce the variance by $1/B$.

Note: Random reshuffling/permutation is more efficient than random sampling.

Stochastic approximation in Machine Learning

Training often done using gradient or stochastic gradient methods.

- ▶ Gradient methods are expensive, but asymptotically faster
- ▶ Stochastic gradient provides rapid initial convergence.

Use block update as a compromise. Choose *random subset* $I_n \subset \{1, \dots, N\}$ of size B and use

$$x_{n+1} = x_n + \frac{a}{(1 + n + A)^\alpha} \frac{1}{B} \sum_{k \in I_n} g_k(x_k)$$

with $\alpha \in (1/2, 1]$ and g_k being the k th term in the log likelihood function. This is still a noisy gradient, but the blocks increase the computational demand by B and reduce the variance by $1/B$.

Note: Random reshuffling/permutation is more efficient than random sampling.

Stochastic approximation in Machine Learning

Training often done using gradient or stochastic gradient methods.

- ▶ Gradient methods are expensive, but asymptotically faster
- ▶ Stochastic gradient provides rapid initial convergence.

Use block update as a compromise. Choose *random subset* $I_n \subset \{1, \dots, N\}$ of size B and use

$$x_{n+1} = x_n + \frac{a}{(1 + n + A)^\alpha} \frac{1}{B} \sum_{k \in I_n} g_k(x_k)$$

with $\alpha \in (1/2, 1]$ and g_k being the k th term in the log likelihood function. This is still a noisy gradient, but the blocks increase the computational demand by B and reduce the variance by $1/B$.

Note: Random reshuffling/permutation is more efficient than random sampling.

Stochastic approximation in Machine Learning

Training often done using gradient or stochastic gradient methods.

- ▶ Gradient methods are expensive, but asymptotically faster
- ▶ Stochastic gradient provides rapid initial convergence.

Use block update as a compromise. Choose *random subset* $I_n \subset \{1, \dots, N\}$ of size B and use

$$x_{n+1} = x_n + \frac{a}{(1 + n + A)^\alpha} \frac{1}{B} \sum_{k \in I_n} g_k(x_k)$$

with $\alpha \in (1/2, 1]$ and g_k being the k th term in the log likelihood function. This is still a noisy gradient, but the blocks increase the computational demand by B and reduce the variance by $1/B$.

Note: Random reshuffling/permutation is more efficient than random sampling.

SAGA and SVRG

SAGA and SVRG provides variance reduction, leading to faster convergence! SVRG theory is easier.

- ▶ Idea based on control variates in Monte Carlo.
- ▶ Substantial variance reduction when two stoch variables X and Y are correlated. Often $Y = g(X) - E[g(X)]$
- ▶ Define $Z = X + Y$ where $E[Y] = 0$ and $\text{corr}(X, Y) = \rho$
- ▶ Then $\text{var}[Z] = (1 - \rho^2)\text{var}[X]!$
- ▶ Again, use random blocks or indices I_n , and define $X = g_{I_n}(x_n)$ and $Y = (g_{I_n}(\tilde{x}_n) - \tilde{\mu})$ with $\tilde{\mu} = g(\tilde{x}_n)$ being the full gradient.
- ▶ This causes $\rho^2 \rightarrow 1$ as n increases (as step size of sequence decreases), leading to substantial variance reduction!

SVRG algorithm

Initialize $\tilde{x}_0$

- ▶ For $k = 1, \dots$ (run epochs)
 - ▶ Set $\tilde{x} = \tilde{x}_{k-1}$
 - ▶ Compute $\tilde{\mu} = \frac{1}{N} \sum_{i=1}^N g_i(\tilde{x})$ (the average gradient)
 - ▶ Set $x_0 = \tilde{x}$. For $l = 1 : m$
 - ▶ Sample i_l from $\{1, \dots, N\}$
 - ▶ Set $x_l = x_{l-1} - \eta (g_{i_l}(x_{l-1}) - g_{i_l}(\tilde{x}) + \tilde{\mu})$
- Set $\tilde{x}_k = x_m$

Very fast convergence, as fast as gradient methods on convex problem, and almost as fast on strongly convex problems, while still inheriting all nice properties of the stochastic gradient method.

Filtering

- ▶ Recursive estimation using non-linear filters
- ▶ Augment

$$x_{n+1} = f(x_n) + e_{n+1} \quad (15)$$

$$y_{n+1} = h(x_{n+1}) + w_{n+1} \quad (16)$$

- ▶ to

$$\begin{pmatrix} x_{n+1} \\ \theta_{n+1} \end{pmatrix} = \begin{pmatrix} f(x_n) \\ \theta_n \end{pmatrix} + \begin{pmatrix} e_{n+1}^x \\ e_{n+1}^\theta \end{pmatrix} \quad (17)$$

$$y_{n+1} = h(x_{n+1}, \theta_{n+1}) + w_{n+1} \quad (18)$$

Estimation is then "trivial", using EKF, UKF (better) or SMC methods

- ▶ Matlab code: `FilteringStateEstimation.m`

Consistent estimates in the filtering setup

- ▶ Estimate is often biased.
- ▶ Idea. Let $\text{Var}[e^\theta] \rightarrow 0$
- ▶ Formalized in the 'iterated filtering framework'
- ▶ Can show consistency $\theta_{n+1} \xrightarrow{\text{a.s.}} \theta_0$
- ▶ Matlab code: `MainFilter.m`