Recursive system identification

Week 9 – Advanced Topic 2

Norwegian University of Science and Technology

Outline

- System Identification
- Recursive system identification
- Algorithm explanation
- Code example
- Pros and Cons
- Key points

System Identification

- Build a mathematical model of a dynamic system
- Use input and output signals
- Process:
    - Measure the input x and output y signals
    - Select a model structure
    - Apply an estimation method to estimate the model parameters
    - Evaluate the model

Motivation: Recursive system identification

- Model with parameters $\hat{\theta}$ predicts outputs y from inputs x

- We have data from step 0 to step t-1: $\boldsymbol{x}_{0:t-1}$ and $\boldsymbol{y}_{0:t-1}$
  - Using $\boldsymbol{x}_{0:t-1}$ and $\boldsymbol{y}_{0:t-1}$ we can estimate our model parameters $\hat{\theta}_{t-1}$

- We then get one more datapoint (x_t, y_t)
  - How do calculate the next model estimate $\hat{\theta}_t$?

What do we do?

- Estimate $\hat{\theta}_t$ from all data $\boldsymbol{x}_{0:t}$ and $\boldsymbol{y}_{0:t}$
  - Need to save all data and computationally expensive

or

- Estimate $\hat{\theta}_t$ from $\hat{\theta}_{t-1}$ and only the newest datapoint (x_t, y_t)

NTNU

Recursive system identification

- We calculate the next $\hat{\theta}_t$ by doing a ‘simple’ modification of $\hat{\theta}_{t-1}$

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \Delta\hat{\theta}_t$$

- Where $\Delta\hat{\theta}_t$ is calculated from $\hat{\theta}_{t-1}$ and only the newest datapoint (x_t, y_t)

$$\Delta\hat{\theta}_t = f(\hat{\theta}_{t-1}, x_t, y_t)$$

Batch Least Squares Method

- Let's assume of a sensor that gives us:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} \quad C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_N \end{bmatrix}$$

- Problem formulation: $\min_{\mathbf{x}} \|\mathbf{y} - C\mathbf{x}\|_2^2$

- Solution: $\hat{\mathbf{x}} = (C^T C)^{-1} C^T \mathbf{y}$

Recursive Least Squares Method

Batch LS

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} \quad C = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_N \end{bmatrix}$$

$$\min_{\mathbf{x}} \|\mathbf{y} - C\mathbf{x}\|_2^2$$

$$\hat{\mathbf{x}} = (C^T C)^{-1} C^T \mathbf{y}$$

1. Gain matrix update

$$K_k = P_{k-1} C_k^T (R_k + C_k P_{k-1} C_k^T)^{-1}$$

2. Estimate update

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + K_k (\mathbf{y}_k - C_k \hat{\mathbf{x}}_{k-1})$$

3. Propagation of the estimation error covariance matric by using this equation

$$P_k = (I - K_k C_k) P_{k-1} (I - K_k C_k)^T + K_k R_k K_k^T$$

or this equation

$$P_k = (I - K_k C_k) P_{k-1}$$

Code example

The python notebook shows

1. **Generating synthetic data** with known model parameters (e.g. a quadratic function)
2. **Initialising** the RLS parameters
3. **Looping** through the RLS algorithm
4. **Comparing** estimated and true parameters

Code example – RLS algorithm

At each time step:

- **Predict output** based on previous time step
- **Calculate residual error**
- **Update**
  - Parameter estimates
  - Gain
  - Covariance matrix

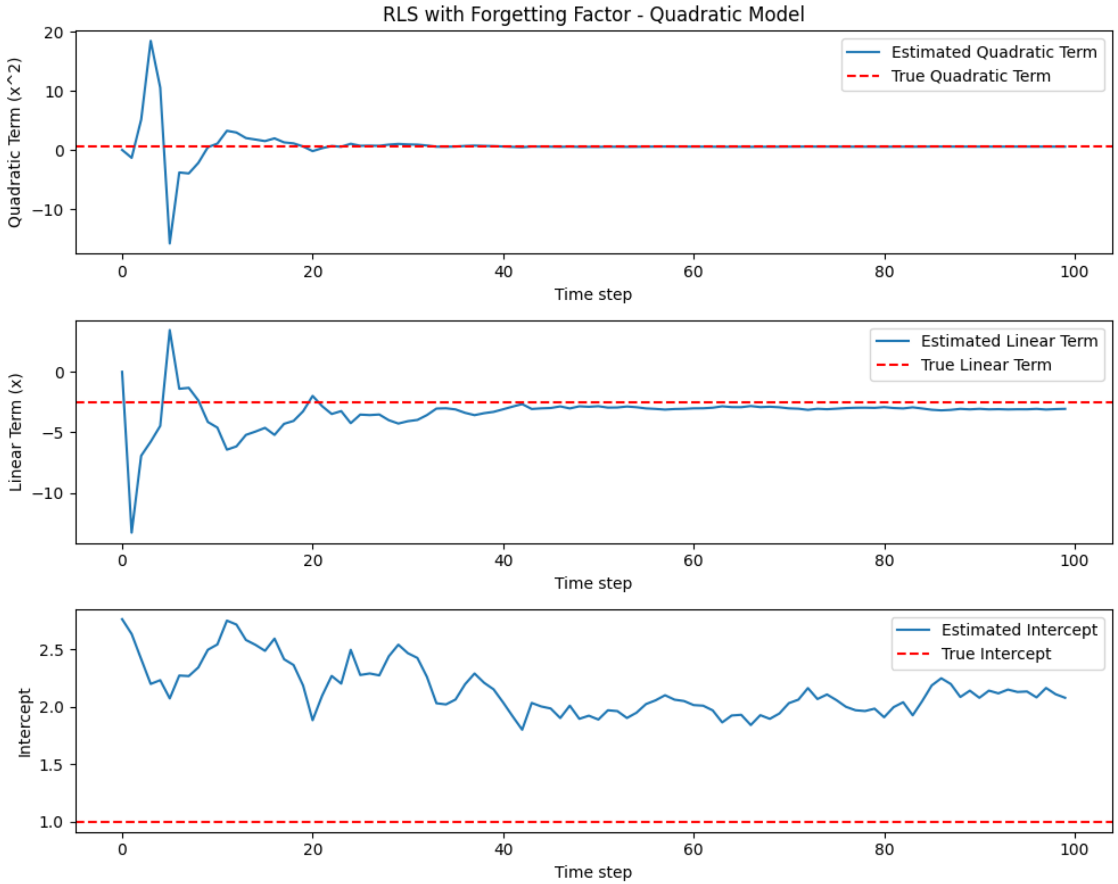
```python
# Step 3: Recursive Least Squares (RLS) algorithm with forgetting factor
for i in range(n):
    # Current data point
    x_i = X[i, :]  # Input vector for current data point (including intercept, x^2, x)
    y_i = y_true[i]  # Observed output for current data point

    # Predict output based on current parameter estimates
    y_hat = np.dot(theta_est, x_i)

    # Calculate prediction error (residual)
    error = y_i - y_hat

    # Compute Kalman gain (update step)
    P_x = np.dot(P, x_i)
    gain = P_x / (forgetting_factor + np.dot(x_i.T, P_x))

    # Update parameter estimates
    theta_est = theta_est + gain * error

    # Update the covariance matrix
    P = (P - np.outer(gain, P_x)) / forgetting_factor

    # Store the current estimates
    theta_estimates.append(theta_est.copy())
```

Code example – RLS parameter estimates over time for a QUADRATIC FUNCTION

Quadratic term

Linear term

Intercept – doesn't converge

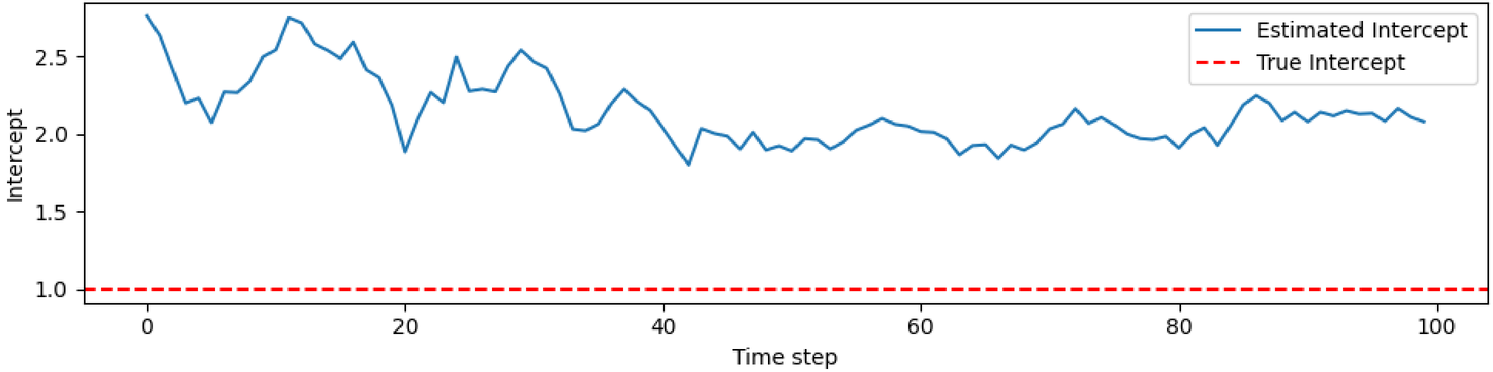


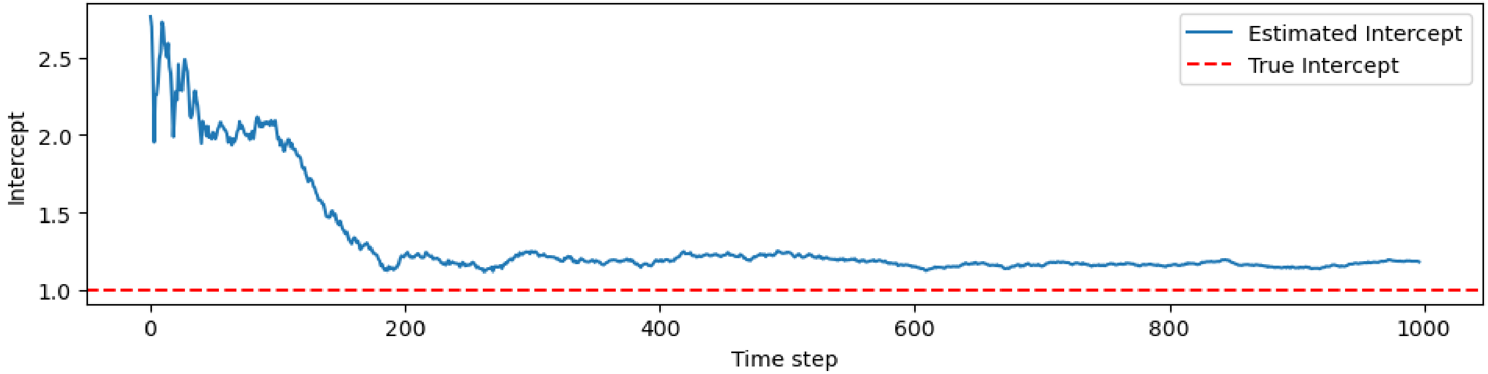
True model:

$$y = ax^2 + bx + c$$

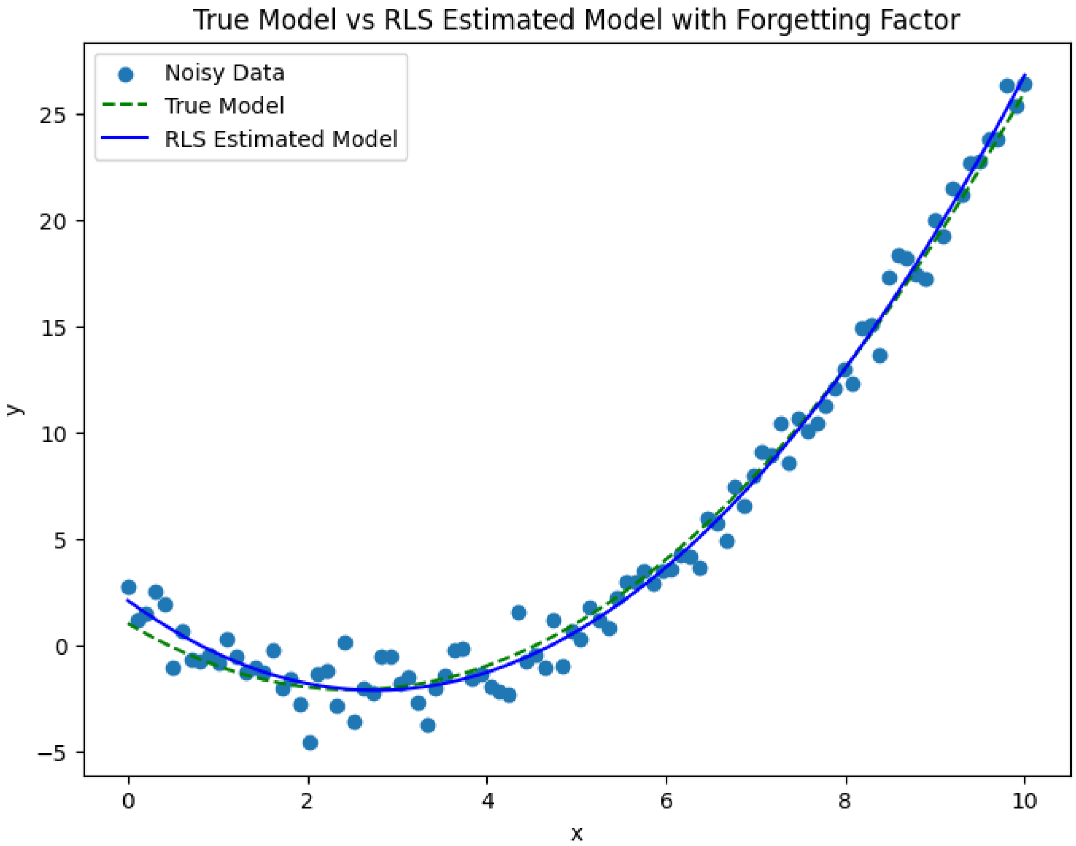
Code example – RLS parameter estimates over time

Increase time steps to 1000 – *better convergence*





Code example – True vs RLS fitted model



True model:

$y = ax^2 + bx + c$

NTNU

Coding in practice

In practice use e.g. **sysidentpy** which contains:

Model Structure Selection

Use State of the Art techniques to build your models.

Learn More >

Parameter Estimation

Use recursive methods, adaptive filters and many more.

Learn More >

Multiple NARMAX Classes

Create Polynomial, Fourier and Neural NARX models.

Learn More >

Use cases

Fault detection and diagnosis

Real time monitoring of dynamic system performance.

Adaptive control systems

Update of system parameters.

Filtering and signal processing

Noise/Echo cancelation

Time-varying signals

Real-time estimation in control systems

Parameter and state estimation

Biomedical applications

Health monitoring

Communication systems

Wireless and satelite communications

Pros Cons

	Pros	Cons
Recursive LS	➢ Simpler & efficient (memory) ➢ Real-Time Adaptation dynamically ➢ Computationally Efficient	➢ Less Accurate at Initial moment ➢ Struggles with noisy data or non-linear systems ➢ Need Proper Tuning ➢ Complex to be implemented
Batch LS	➢ Accurately estimate ➢ No Need for Sequential Updates:	➢ Not for realtime process ➢ Not adaptive to dynamic system

Conclusions

- System Identification is **crucial for** modeling and controlling **dynamic systems**.
- SystemID algorithem (such as Recursive LS) is **designed for dynamic / real-time applications** where data is continuously collected.

Keypoints:

- RLS more suitable for applications like real-time control, signal processing, and adaptive filtering
- RLS updates parameter estimates incrementally, making it more efficient for systems where model parameters must adapt to changing conditions.

Sources

- https://www.youtube.com/watch?v=uLbjeQrQJ3Q
- https://se.mathworks.com/help/ident/gs/about-system-identification.html
- https://aleksandarhaber.com/introduction-to-kalman-filter-derivation-of-the-recursive-least-squares-method-with-python-codes/

NTNU

Thank you!