Новосибирский государственный технический университет
Факультет прикладной математики и информатики
Кафедра теоретической и прикладной информатики

Лабораторная работа № 1

«Оценивание неизвестных параметров линейных дискретных систем»

по дисциплине

«Математические методы планирования эксперимента»

Группа: ПММ-61
Студент: Горбунов К. К.
Преподаватель: Чубич В. М.

Новосибирск, 2017 г.

Цель работы

Ознакомиться алгоритмом оценивания неизвестных параметров моделей стохастических линейных дискретных систем методом максимального правдоподобия (ММП).

Порядок выполнения лабораторной работы

1. Изучить соответствующий материал лекции на тему «Оценивание неизвестных параметров дискретных моделей».
2. Последовательно выполнить все задания к лабораторной работе.
3. Проверить правильность реализации алгоритмов и работоспособность программ.

Задание к лабораторной работе

1. Реализовать вычисление значения критерия максимального правдоподобия.
2. Проверить правильность исполнения программы на примерах.
3. Для стохастических линейных дискретных моделей при различных вариантах вхождения неизвестных параметров в уравнения состояния и измерения получить оценки параметров.

Теоретический материал

Описание модельной структуры

Модель стохастической динамической линейной дискретной системы в пространстве состояний в виде [1]:

$$\begin{cases} x(t_{k+1}) &= Fx(t_k) + Cu(t_k) + Gw(t_k), \\ y(t_{k+1}) &= Hx(t_{k+1}) + v(t_{k+1}), \qquad\qquad k = 0, \ldots, N-1 \end{cases} \tag{1}$$

Здесь:

$x(t_k)$ – n-вектор состояния;

F – матрица перехода состояния;

$u(t_k)$ – r-вектор управления (входного воздействия);

C – матрица управления;

$w(t_k)$ – p-вектор возмущений;

G – матрица влияния возмущений;

H – матрица наблюдения;

$v(t_{k+1})$ – m-вектор шума измерений;

$y(t_{k+1})$ – m-вектор наблюдений (измерений) отклика;

F, C, G, H — матрицы соответствующих размеров.

Априорные предположения:

- F устойчива;
- пары (F, C) и (F, G) управляемы;
- пара (F, H) — наблюдаема;

- $w(t_k)$ и $v(t_{k+1})$ — случайные векторы, образующие стационарные белые гауссовские последовательности, причем:

$$E[w(t_k)] = 0,\ E[w(t_k)w^T(t_l)] = Q\delta_{k,l}\ ;$$

$$E[v(t_{k+0}) = 0,\ E[v(t_{k+1})v^T(t_{l+1})] = R\delta_{k,l}$$

$$E[v(t_k)w^T(t_k)] = 0,$$

для любых $k, l = 0, 1, \ldots, N-1$ ($\delta_{k,l}$ — символ Кронекера);

- начальное состояние $x(0)$ имеет нормальное распределение с параметрами $\overline{x}(0)$ и $P(0)$ и не коррелирует с $w(t_k)$ и v_{k+1} при любых значениях k.

Будем считать, что подлежащие оцениванию параметры $\theta = (\theta_1, \theta_2, \ldots, \theta_s)$ могут входить в элементы матриц $F, C, G, H, Q, R, P(0)$ и в вектор $\overline{x}(0)$ в различных комбинациях.

Критерий идентификации

В качестве критерия идентификации используется логарифмическая функция правдоподобия. Она имеет вид [1]:

$$\chi(\theta) = -\ln L(\theta) = \frac{Nm}{2}\ ln2\pi + \frac{1}{2}\sum_{k=0}^{N-1}\left[\varepsilon^T(t_{k+1})B^{-1}(t_{k+1})\varepsilon(t_{k+1})\right] + \\ + \frac{1}{2}\sum_{k=0}^{N-1}\ln\det B^{-1}(t_{k+1}).$$

Алгоритм вычисления критерия следующий [1]:

1. для заданного θ найдем $F, C, G, Q, R, \overline{x}(0), P(0)$.
2. положим $k = 0$, $\chi(\theta) = \frac{Nm}{2}\ln 2\pi$, $\Delta_2 = 0$, $P(0|0) = P(0)$.
3. Используя выражения фильтра Калмана, вычислим:

$$P(t_{k+1}|t_k) = FP(t_k|t_k)F^T + GQG^T;$$

$$B(t_{k+1}) = HP(t_{k+1}|t_k)H^T + R;$$

$$K(t_{k+1}) = P(t_{k+1}|t_k)H^T B^{-1}(t_{k+1});$$

$$P(t_{k+1}|t_{k+1}) = [I - K(t_{k+1})H]\, P(t_{k+1}|t_k).$$

4. Вычислим $\Delta_2 = \Delta_2 + \frac{1}{2}\ln\det B(t_{k+1})$.

5. Увеличим k на единицу. Если $k \leq N-1$, перейдем на шаг 3. В противном случае — на шаг 6.

6. Положим $k = 0$.

7. Используя выражения фильтра Калмана, найдем:

$$\hat{x}(t_{k+1}|t_k) = F\hat{x}(t_k|t_k) + Cu(t_k);$$

$$\varepsilon(t_{k+1}) = y(t_{k+1}) - H\hat{x}(t_{k+1}|t_k); \hat{x}(t_{k+1}|t_{k+1}) = \hat{x}(t_{k+1}|t_k) + K(t_{k+1})\varepsilon(t_{k+1}).$$

8. Найдем значение приращения логарифмической функции правдоподобия Δ_1, соответствующее текущему значению времени:

$$\Delta_1 = \frac{1}{2}\left[\varepsilon(t_{k+1})\right]^T B^{-1}(t_{k+1})\left[\varepsilon(t_{k+1})\right].$$

9. Положим $\chi(\theta) = \chi(\theta) + \Delta_1$.

10. Увеличим k на единицу. Если $k \leq N-1$, перейдем на шаг 7. В противном случае — на шаг 11.

11. Положим $\chi(\theta) = \chi(\theta) + \Delta_2$ и закончим процесс.

Градиент критерия идентификации

Выражение для градиента критерия имеет вид [1]:

$$\frac{\partial \chi(\theta)}{\partial \theta_i} = \sum_{k=0}^{N-1} \left[\frac{\partial \varepsilon(t_{k+1})}{\partial \theta_i}\right]^T B^{-1}(t_{k+1}) \left[\varepsilon(t_{k+1})\right] -$$

$$-\frac{1}{2}\sum_{k=0}^{N-1} \left[\varepsilon(t_{k+1})\right]^T B^{-1}(t_{k+1}) \frac{\partial B(t_{k+1})}{\partial \theta_i} B^{-1}(t_{k+1}) \varepsilon(t_{k+1}) +$$

$$+\frac{1}{2}\sum_{k=0}^{N-1} \mathrm{Sp}\left[B^{-1}(t_{k+1}) \frac{\partial B(t_{k+1})}{\partial \theta_i}\right].$$

Алгоритм вычисления градиента критерия следующий [1]:

1. Для заданного θ найдем: $F, \frac{\partial F}{\partial \theta_i}, Q, \frac{\partial Q}{\partial \theta_i}, C, \frac{\partial C}{\partial \theta_i}, R, \frac{\partial R}{\partial \theta_i}, G, \frac{\partial G}{\partial \theta_i}, \overline{x}(0), \frac{\partial \overline{x}(0)}{\partial \theta_i}, H, \frac{\partial H}{\partial \theta_i}$, $P(0), \frac{\partial P(0)}{\partial \theta_i}$, $i = 1, 2, \ldots, s$.

2. Положим $k = 0$; $\frac{\partial \chi(\theta)}{\partial \theta_i} = 0$; $P(0|0) = P(0)$; $\frac{\partial P(0|0)}{\partial \theta_i} = \frac{\partial P(0)}{\partial \theta_i}$, $i = 1, 2, \ldots, s$; $\Delta_2' = 0$.

3. Используем выражения фильтра Калмана, вычислим:

$$P(t_{k+1}|t_k) = FP(t_k|t_k)F^T + GQG^T;$$

$$B(t_{k+1}) = HP(t_{k+1}|t_k)H^T + R;$$

$$K(t_{k+1}) = P(t_{k+1}|t_k)H^T B^{-1}(t_{k+1});$$

$$P(t_{k+1}|t_{k+1}) = \left[I - K(t_{k+1})H\right] P(t_{k+1}|t_k)$$

4. Найдем $\frac{\partial P(t_{k+1}|t_k)}{\partial \theta_i}$, $\frac{\partial B(t_{k+1})}{\partial \theta_i}$, $\frac{\partial K(t_{k+1})}{\partial \theta_i}$, $\frac{\partial P(t_{k+1}|t_{k+1})}{\partial \theta_i}$ для всех $i = 1, 2, \ldots, s$ по формулам, вытекающим из уравнений фильтра Калмана:

$$\frac{\partial P(t_{k+1}|t_k)}{\partial \theta_i} = \frac{\partial F}{\partial \theta_i} P(t_k|t_k)F^T + F\frac{\partial P(t_k|t_k)}{\partial \theta_i}F^T + FP(t_k|t_k)\frac{\partial F^T}{\partial \theta_i} +$$

$$+\frac{\partial G}{\partial \theta_i}QG^T + G\frac{\partial Q}{\partial \theta_i}G^T + GQ\frac{\partial G^T}{\partial \theta_i};$$

$$\frac{\partial B(t_{k+1})}{\partial \theta_i} = \frac{\partial H}{\partial \theta_i} P(t_{k+1}|t_k)H^T + H\frac{\partial P(t_{k+1}|t_k)}{\partial \theta_i}H^T + HP(t_{k+1}|t_k)\frac{\partial H^T}{\partial \theta_i} + \frac{\partial R}{\partial \theta_i};$$

$$\frac{\partial K(t_{k+1})}{\partial \theta_i} = \left[\frac{\partial P(t_{k+1}|t_k)}{\partial \theta_i} H^T + P(t_{k+1}|t_k)\frac{\partial H^T}{\partial \theta_i} - \right.$$
$$\left. - P(t_{k+1}|t_k) H^T B^{-1}(t_{k+1}) \frac{\partial B(t_{k+1})}{\partial \theta_i}\right] B^{-1}(t_{k+1});$$

$$\frac{\partial P(t_{k+1}|t_{k+1})}{\partial \theta_i} = [I - K(t_{k+1})H]\,\frac{\partial P(t_{k+1}|t_k)}{\partial \theta_i} -$$
$$- \left[\frac{\partial K(t_{k+1})}{\partial \theta_i} H + K(t_{k+1})\frac{\partial H}{\partial \theta_i}\right] P(t_{k+1}|t_k).$$

5. Вычислим $\Delta_2' = \Delta_2' + \frac{1}{2}\,\mathrm{Sp}\left[B^{-1}(t_{k+1})\frac{\partial B(t_{k+1})}{\partial \theta_i}\right]$.

6. Увеличим k на единицу. Если $k \le N-1$, перейдем на шаг 3. В противном случае — на шаг 7.

7. Положим $k = 0$.

8. Используя выражения фильтра Калмана, найдем:

$$\hat{x}(t_{k+1}|t_k) = F\hat{x}(t_k|t_k) + Cu(t_k);$$

$$\varepsilon(t_{k+1}) = y(t_{k+1}) - H\hat{x}(t_{k+1}|t_k);$$

$$\hat{x}(t_{k+1}|t_{k+1}) = \hat{x}(t_{k+1}|t_k) + K(t_{k+1})\varepsilon(t_{k+1}).$$

9. Найдем $\frac{\partial \hat{x}(t_{k+1}|t_k)}{\partial \theta_i}$, $\frac{\partial \varepsilon(t_{k+1})}{\partial \theta_i}$, $\frac{\partial \hat{x}(t_{k+1}|t_{k+1})}{\partial \theta_i}$ для всех $i = 1, 2, \ldots, s$ по формулам, вытакающим из уравнений фильтра Калмана:

$$\frac{\partial \hat{x}(t_{k+1}|t_k)}{\partial \theta_i} = \frac{\partial F}{\partial \theta_i}\hat{x}(t_k|t_k) + F\frac{\partial \hat{x}(t_k|t_k)}{\partial \theta_i} + \frac{\partial C}{\partial \theta_i}u(t_k);$$

$$\frac{\partial \varepsilon(t_{k+1})}{\partial \theta_i} = -\frac{\partial H}{\partial \theta_i}\hat{x}(t_{k+1}|t_k) - H\frac{\partial \hat{x}(t_{k+1}|t_k)}{\partial \theta_i};$$

$$\frac{\partial \hat{x}(t_{k+1}|t_{k+1})}{\partial \theta_i} = \frac{\partial \hat{x}(t_{k+1}|t_k)}{\partial \theta_i} + \frac{\partial K(t_{k+1})}{\partial \theta_i}\varepsilon(t_{k+1}) + K(t_{k+1})\frac{\partial \varepsilon(t_{k+1})}{\partial \theta_i}.$$

10. Найдем значение приращения градиента Δ_1', соответствующее текущему значению времени:

$$\Delta_1' = \left[\frac{\partial \varepsilon(t_{k+1})}{\partial \theta_i}\right]^T B^{-1}(t_{k+1}) \left[\varepsilon(t_{k+1})\right] - \\ -\frac{1}{2}\left[\varepsilon(t_{k+1})\right]^T B^{-1}(t_{k+1}) \frac{\partial B(t_{k+1})}{\partial \theta_i} B^{-1}(t_{k+1}) \varepsilon(t_{k+1}).$$

11. Положим $\frac{\partial \chi(\theta)}{\partial \theta_i} = \frac{\partial \chi(\theta)}{\partial \theta_i} + \Delta_1'$, $i = 1, 2, \ldots, s$.

12. Увеличим k на единицу. Если $k \leq N-1$, перейдем на шаг 8. В противном случае — на шаг 13.

13. Положим $\frac{\partial \chi(\theta)}{\partial \theta_i} = \frac{\partial \chi(\theta)}{\partial \theta_i} + \Delta_2'$, $i = 1, 2, \ldots, s$ и закончим процесс.

Примеры решений

Пример 1: параметры в матрице перехода F

$$\begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{pmatrix} \begin{pmatrix} x_1(t_k) \\ x_2(t_k) \end{pmatrix} + \begin{pmatrix} u_1(t_k) \\ u_2(t_k) \end{pmatrix} + \begin{pmatrix} w_1(t_k) \\ w_2(t_k) \end{pmatrix}$$

$$\begin{pmatrix} y_1(t_{k+1}) \\ y_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} + \begin{pmatrix} v_1(t_{k+1}) \\ v_2(t_{k+1}) \end{pmatrix}$$

$$\overline{x}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \; P(0) = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}, \; Q = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}, \; R = \begin{pmatrix} 0.15 & 0 \\ 0 & 0.15 \end{pmatrix}$$

Истинные значения параметров $\theta_{true} = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$

Управляющее воздействие — дискретная ступенчатая функция

$$u(t_k) = \begin{pmatrix} 10 \\ 10 \end{pmatrix}, \; k = 0, 1, 2, \ldots, 99$$

Область оценивания: $0.1 \leq \theta_i \leq 0.9$, $i = 1, \ldots, s$.

Оценивание параметров: результаты

Значение критерия при истинных значениях параметров

$$L(\theta_{true}) = 31.747.$$

Начальное приближение по параметрам

$$\theta_{init} = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}.$$

Значение критерия при начальном приближении

$$L(\theta_{init}) = 39260.481.$$

Полученные оценки параметров

$$\hat{\theta} = \begin{bmatrix} 0.498632 & 0.498493 \end{bmatrix}$$

Значение критерия

$$L(\hat{\theta}) = 31.167$$

Относительная погрешность в пространстве параметров:

$$\frac{||\theta_{true} - \hat{\theta}||}{||\theta_{true}||} = 0.287923\%$$

Относительная погрешность в пространстве откликов:

$$\frac{||y_{true} - \hat{y}||}{||y_{true}||} = 0.198772\%$$

График поверхности, отражающей зависимость критерия от параметров представлен далее.

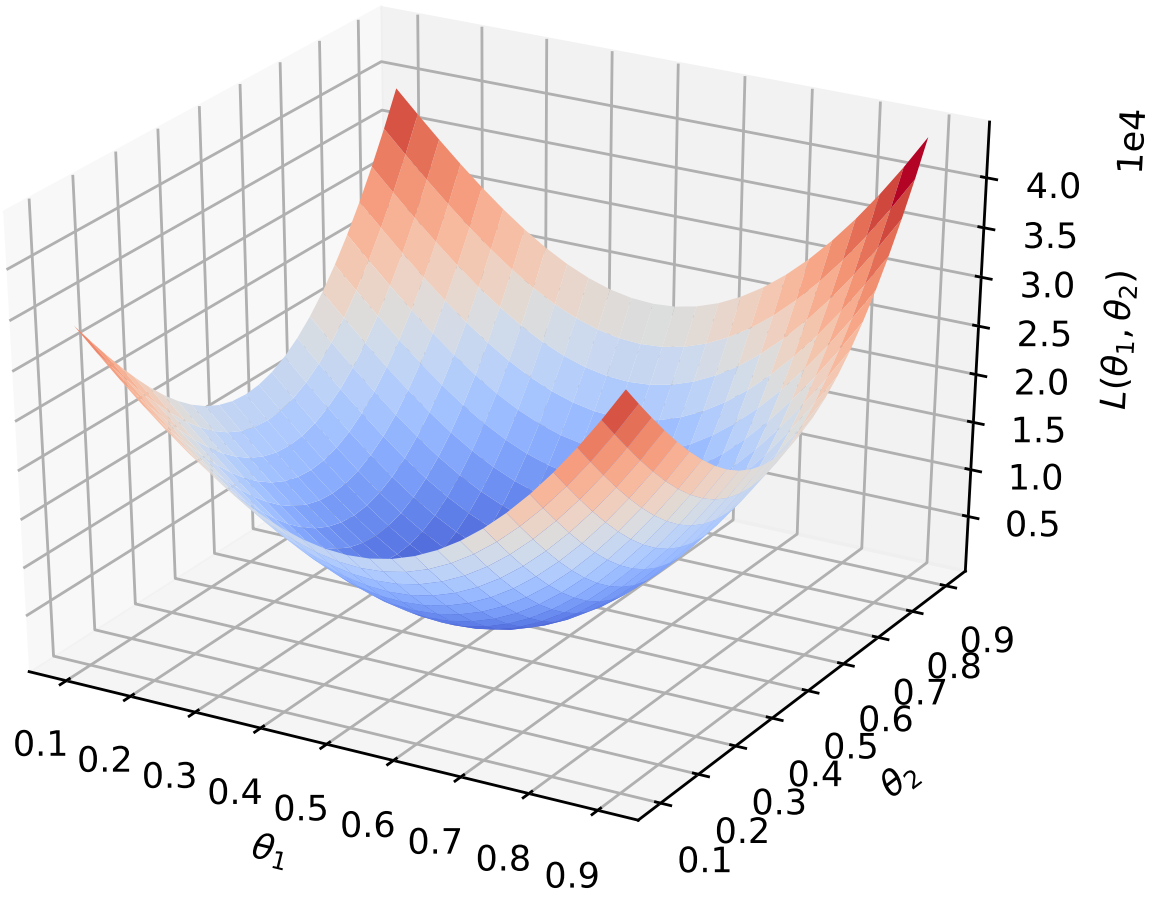

Рис. 1. Поверхность, отражающая зависимость критерия от параметров

Выводы: поверхность выпуклая, виден выраженный минимум.

Пример 2: параметры в матрице управления C

$$\begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_k) \\ x_2(t_k) \end{pmatrix} + \begin{pmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{pmatrix} \begin{pmatrix} u_1(t_k) \\ u_2(t_k) \end{pmatrix} + \begin{pmatrix} w_1(t_k) \\ w_2(t_k) \end{pmatrix}$$

$$\begin{pmatrix} y_1(t_{k+1}) \\ y_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} + \begin{pmatrix} v_1(t_{k+1}) \\ v_2(t_{k+1}) \end{pmatrix}$$

Истинные значения параметров $\theta_{true} = \begin{pmatrix} 1 & 1 \end{pmatrix}$

Остальные условия остаются такими же, как в примере 1.

Оценивание параметров: результаты

Значение критерия при истинных значениях параметров

$$L(\theta_{true}) = 81.748.$$

Начальное приближение по параметрам

$$\theta_{init} = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}.$$

Значение критерия при начальном приближении

$$L(\theta_{init}) = 28656.687.$$

Полученные оценки параметров

$$\hat{\theta} = \begin{bmatrix} 1.002779 & 0.998564 \end{bmatrix}$$

Значение критерия

$$L(\hat{\theta}) = 81.409$$

Относительная погрешность в пространстве параметров:

$$\frac{||\theta_{true} - \hat{\theta}||}{||\theta_{true}||} = 0.221208\%$$

Относительная погрешность в пространстве откликов:

$$\frac{||y_{true} - \hat{y}||}{||y_{true}||} = 0.006278\%$$

График поверхности, отражающей зависимость критерия от параметров представлен далее.

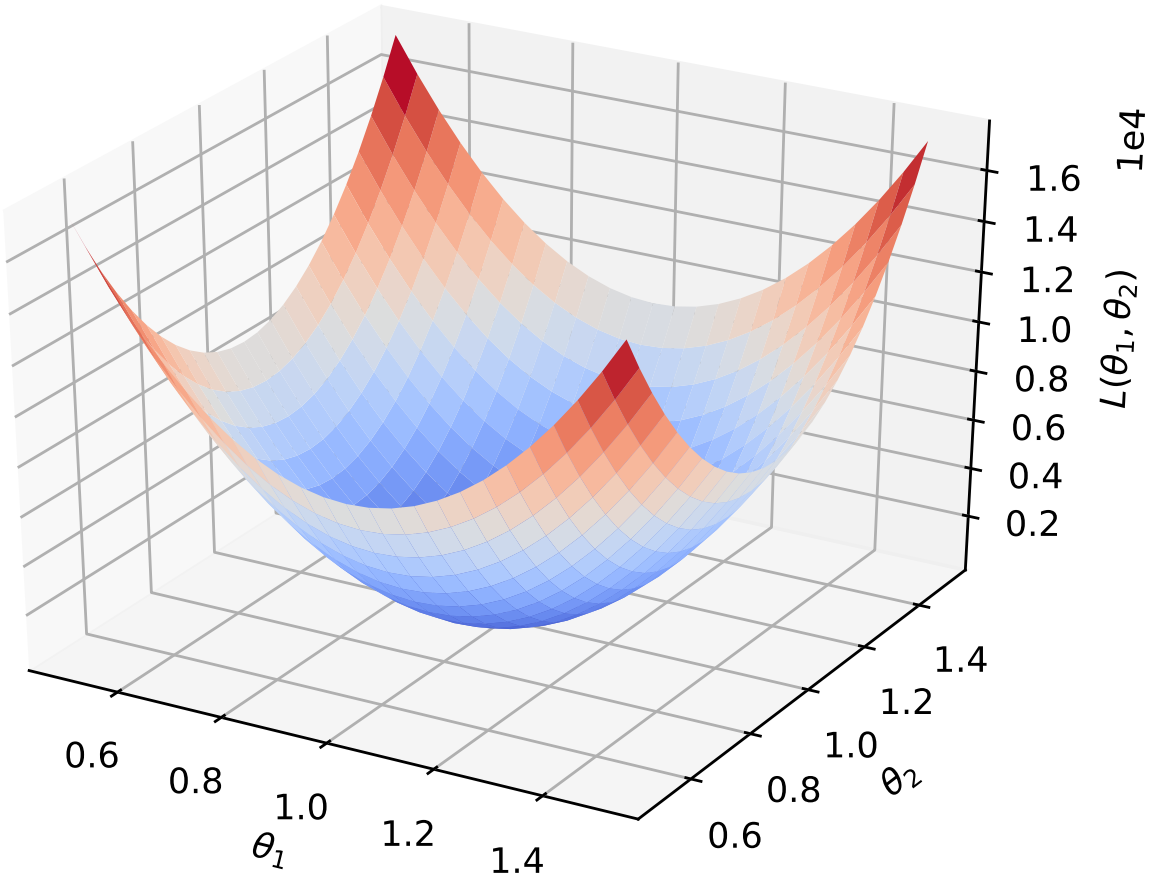

Рис. 2. Поверхность, отражающая зависимость критерия от параметров

Вывод: поверхность выпуклая, минимум выраженный, но менее, чем в примере 1. Это видно при сравнении масштабов осей Z графика в примере 1 и графика в данном примере.

Пример 3: параметры в матрице наблюдения H

$$\begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_k) \\ x_2(t_k) \end{pmatrix} + \begin{pmatrix} u_1(t_k) \\ u_2(t_k) \end{pmatrix} + \begin{pmatrix} w_1(t_k) \\ w_2(t_k) \end{pmatrix}$$

$$\begin{pmatrix} y_1(t_{k+1}) \\ y_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{pmatrix} \begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} + \begin{pmatrix} v_1(t_{k+1}) \\ v_2(t_{k+1}) \end{pmatrix}$$

Истинные значения параметров $\theta_{true} = \begin{pmatrix} 1 & 1 \end{pmatrix}$

Остальные условия остаются такими же, как в примере 1.

Оценивание параметров: результаты

Значение критерия при истинных значениях параметров

$$L(\theta_{true}) = 75.677.$$

Начальное приближение по параметрам

$$\theta_{init} = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}.$$

Значение критерия при начальном приближении

$$L(\theta_{init}) = 70109.294.$$

Полученные оценки параметров

$$\hat{\theta} = \begin{bmatrix} 0.998865 & 0.998058 \end{bmatrix}$$

Значение критерия

$$L(\hat{\theta}) = 75.502$$

Относительная погрешность в пространстве параметров:

$$\frac{||\theta_{true} - \hat{\theta}||}{||\theta_{true}||} = 0.159044\%$$

Относительная погрешность в пространстве откликов:

$$\frac{||y_{true} - \hat{y}||}{||y_{true}||} = 0.004525\%$$

График поверхности, отражающей зависимость критерия от параметров представлен далее.

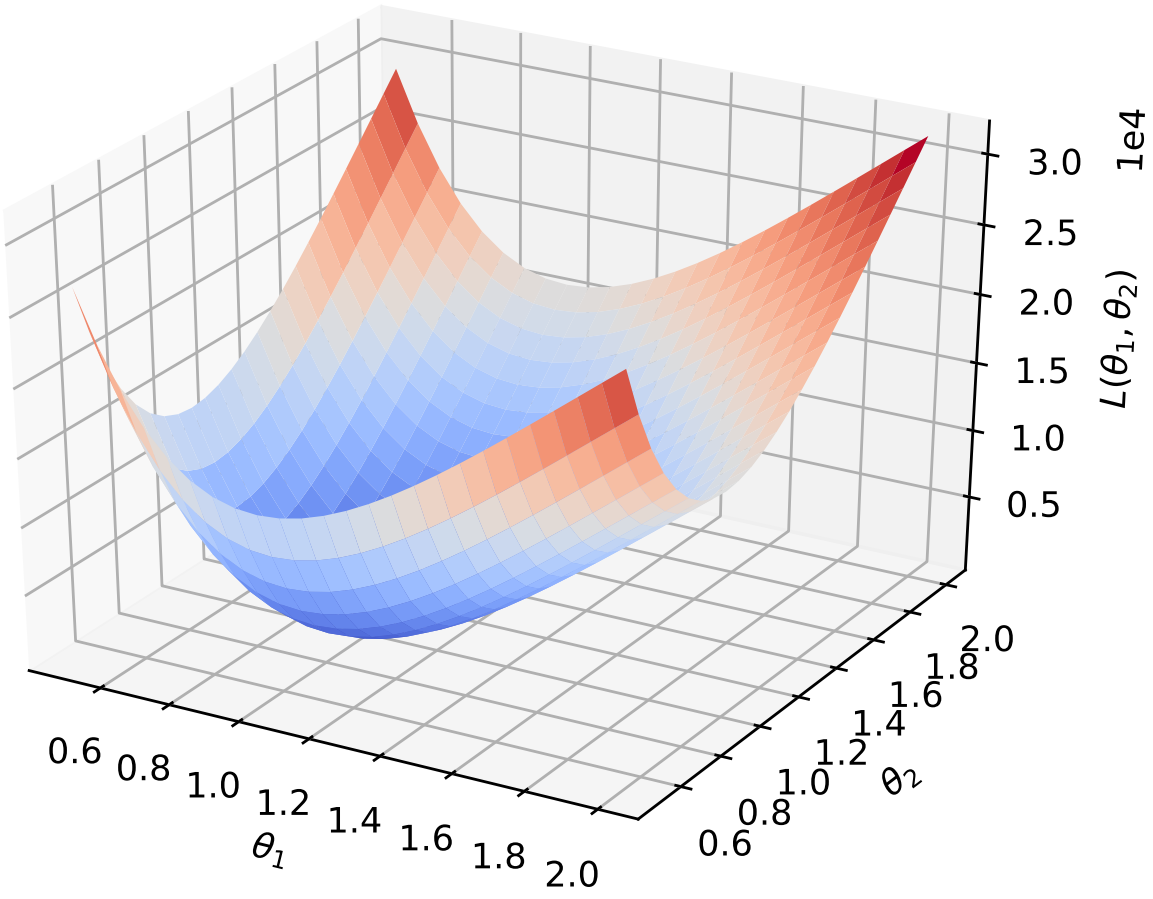

Рис. 3. Поверхность, отражающая зависимость критерия от параметров

Вывод: поверхность выпуклая, минимум выраженный.

Пример 4: параметры в матрице Q

$$\begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_k) \\ x_2(t_k) \end{pmatrix} + \begin{pmatrix} u_1(t_k) \\ u_2(t_k) \end{pmatrix} + \begin{pmatrix} w_1(t_k) \\ w_2(t_k) \end{pmatrix}$$

$$\begin{pmatrix} y_1(t_{k+1}) \\ y_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} + \begin{pmatrix} v_1(t_{k+1}) \\ v_2(t_{k+1}) \end{pmatrix}$$

$$Q = \begin{pmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{pmatrix},$$

Истинные значения параметров $\theta_{true} = \begin{pmatrix} 0.2 & 0.2 \end{pmatrix}$

Область оценивания: $0.01 \leq \theta_i \leq 0.2$, $i = 1, \ldots, s$.

Остальные условия остаются такими же, как в примере 1.

Оценивание параметров: результаты

Значение критерия при истинных значениях параметров

$$L(\theta_{true}) = 128.585.$$

Начальное приближение по параметрам

$$\theta_{init} = \begin{bmatrix} 0.2 & 0.1 \end{bmatrix}.$$

Значение критерия при начальном приближении

$$L(\theta_{init}) = 126.097.$$

Полученные оценки параметров

$$\hat{\theta} = \begin{bmatrix} 0.193001 & 0.076657 \end{bmatrix}$$

Значение критерия

$$L(\hat{\theta}) = 125.931$$

Относительная погрешность в пространстве параметров:

$$\frac{||\theta_{true} - \hat{\theta}||}{||\theta_{true}||} = 43.678633\%$$

Относительная погрешность в пространстве откликов:

$$\frac{||y_{true} - \hat{y}||}{||y_{true}||} = 0.007733\%$$

График поверхности, отражающей зависимость критерия от параметров представлен далее.

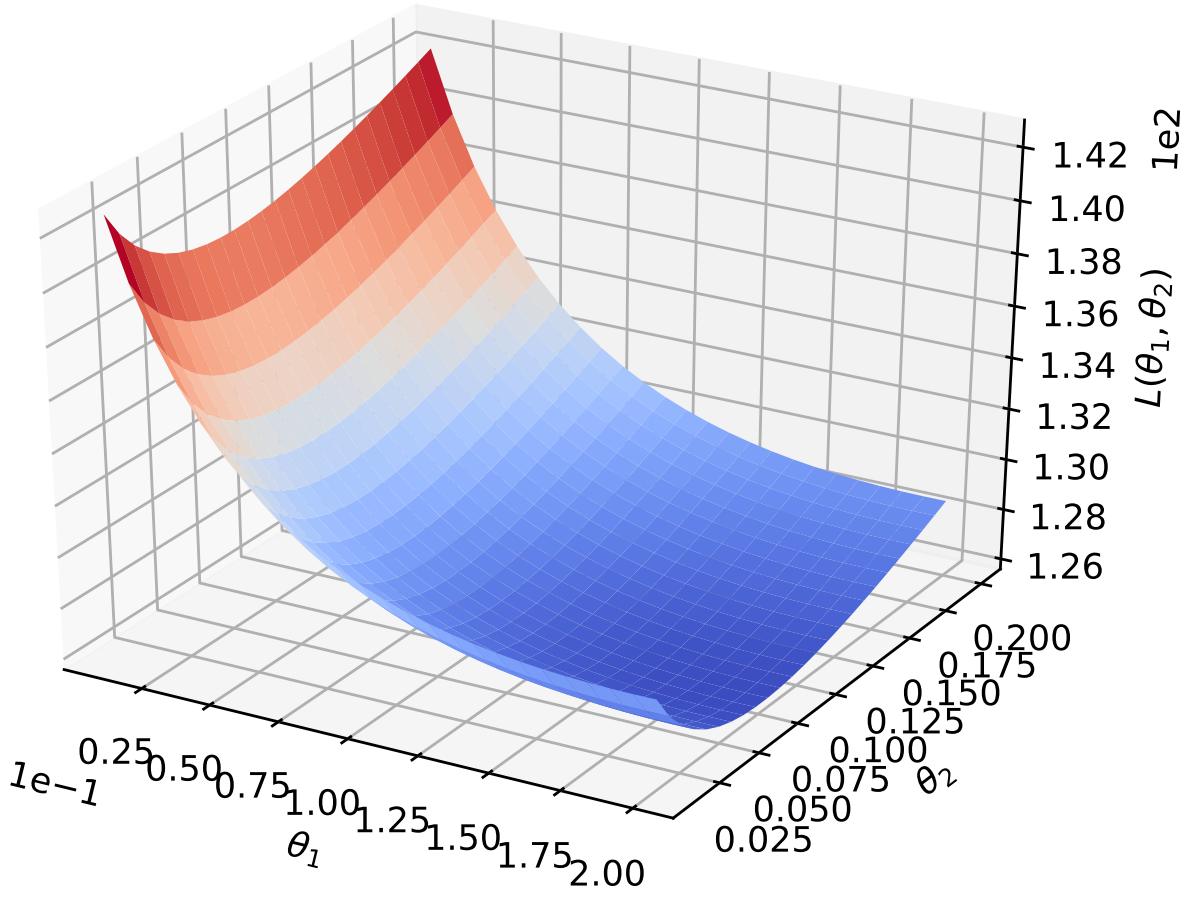

Рис. 4. Поверхность, отражающая зависимость критерия от параметров

Поверхность выпуклая, но выраженного минимума в окрестности истинных значений параметров нет.

Пример 5: параметры в матрице R

$$\begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_k) \\ x_2(t_k) \end{pmatrix} + \begin{pmatrix} u_1(t_k) \\ u_2(t_k) \end{pmatrix} + \begin{pmatrix} w_1(t_k) \\ w_2(t_k) \end{pmatrix}$$

$$\begin{pmatrix} y_1(t_{k+1}) \\ y_2(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x_1(t_{k+1}) \\ x_2(t_{k+1}) \end{pmatrix} + \begin{pmatrix} v_1(t_{k+1}) \\ v_2(t_{k+1}) \end{pmatrix}$$

$$R = \begin{pmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{pmatrix},$$

Оценивание параметров: результаты

Истинные значения параметров $\theta_{true} = \begin{pmatrix} 0.2 & 0.2 \end{pmatrix}$

Область оценивания: $0.1 \leq \theta_i \leq 0.3$, $i = 1, \ldots, s$.

Остальные условия остаются такими же, как в примере 1.

Оценивание параметров: результаты

Значение критерия при истинных значениях параметров

$$L(\theta_{true}) = 107.518.$$

Начальное приближение по параметрам

$$\theta_{init} = \begin{bmatrix} 0.2 & 0.1 \end{bmatrix}.$$

Значение критерия при начальном приближении

$$L(\theta_{init}) = 118.751.$$

Полученные оценки параметров

$$\hat{\theta} = \begin{bmatrix} 0.146664 & 0.169546 \end{bmatrix}$$

Значение критерия

$$L(\hat{\theta}) = 105.458$$

Относительная погрешность в пространстве параметров:

$$\frac{||\theta_{true} - \hat{\theta}||}{||\theta_{true}||} = 21.714524\%$$

Относительная погрешность в пространстве откликов:

$$\frac{||y_{true} - \hat{y}||}{||y_{true}||} = 0.001791\%$$

График поверхности, отражающей зависимость критерия от параметров представлен далее.

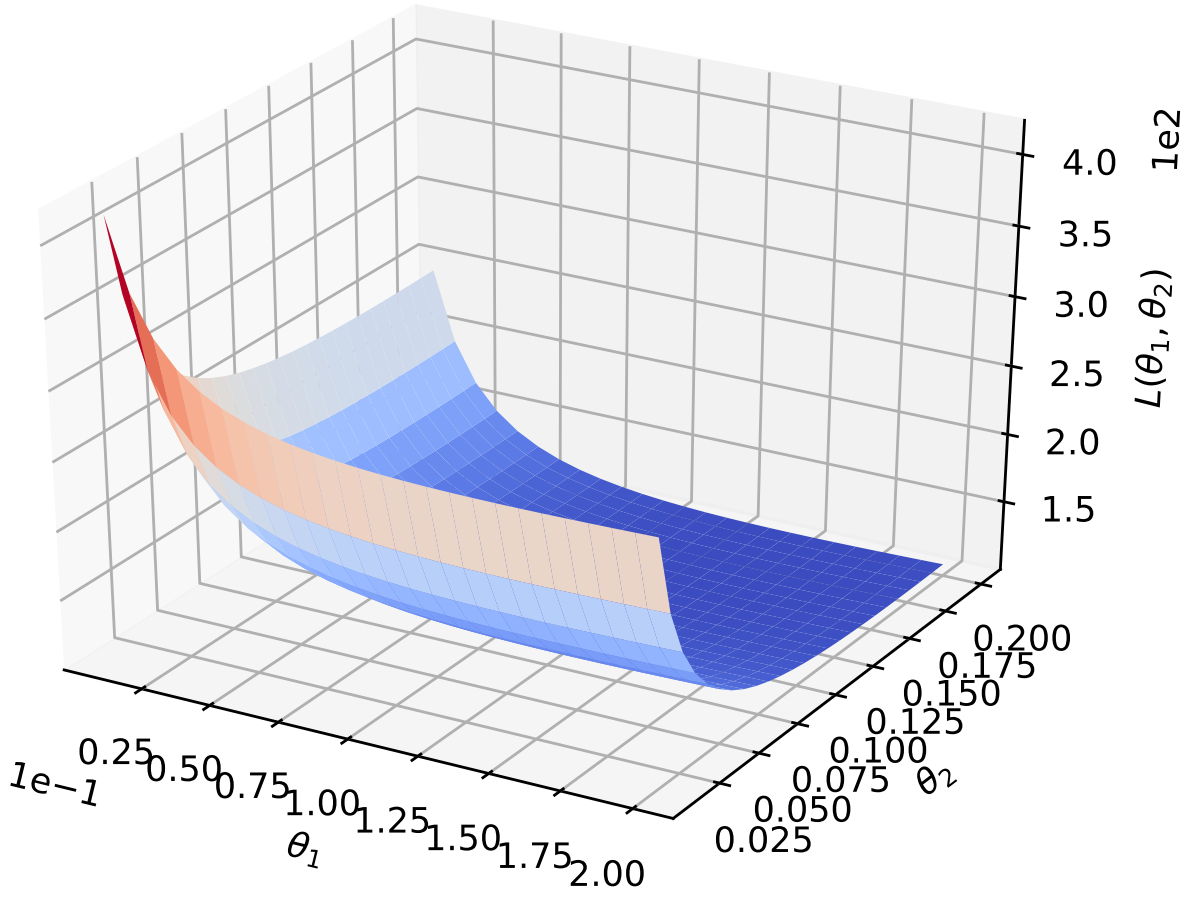

Рис. 5. Поверхность, отражающая зависимость критерия от параметров

Заключение

Матрицы перехода, управления, наблюдений хорошо поддаются оцениванию. Ковариационные матрицы шумов объекта, наблюдений плохо поддаются оцениванию.

Список литературы

[1] Активная параметрическая идентификация стохастических линейных систем: монография / В.И. Денисов, В.М. Чубич, О.С. Черникова, Д.И. Бо-

былева. — Новосибирск : Изд-во НГТУ, 2009. — 192 с. (Серия «Монографии НГТУ»).

Исходные тексты программ

```
import math
import tensorflow as tf
import control
import numpy as np
from tensorflow.contrib.distributions import MultivariateNormalFullCovariance
import scipy


class Model(object):

    # TODO: introduce some more default argument values, check types, cast if
    # neccessary
    def __init__(self, F, C, G, H, x0_mean, x0_cov, w_cov, v_cov, th):
        """
        Arguments are all callables (functions) of 'th' returning python lists
        except for 'th' itself (of course)
        """

        # TODO: evaluate and cast everything to numpy matrices first
        # TODO: cast floats, ints to numpy matrices
        # TODO: allow both constant matrices and callables

        # store arguments, after that check them
        self.__F = F
        self.__C = C
        self.__G = G
        self.__H = H
        self.__x0_mean = x0_mean
        self.__x0_cov = x0_cov
        self.__w_cov = w_cov
        self.__v_cov = v_cov
        self.__th = th

        # evaluate all functions
        th = np.array(th)
        F = np.array(F(th))
        C = np.array(C(th))
        H = np.array(H(th))
        G = np.array(G(th))
        w_cov = np.array(w_cov(th))    # Q
        v_cov = np.array(v_cov(th))    # R
        x0_m = np.array(x0_mean(th))
        x0_cov = np.array(x0_cov(th))  # P_0

        # get dimensions and store them as well
        self.__n = n = F.shape[0]
```

```
        self.__m = m = H.shape[0]
        self.__p = p = G.shape[1]
        self.__r = r = C.shape[1]

        # generate means
        w_mean = np.zeros([p, 1], np.float64)
        v_mean = np.zeros([m, 1], np.float64)

        # and store them
        self.__w_mean = w_mean
        self.__v_mean = v_mean

        # check conformability
        u = np.ones([r, 1])
        # generate random vectors
        # squeeze, because mean must be one dimensional
        x = np.random.multivariate_normal(x0_m.squeeze(), x0_cov)
        w = np.random.multivariate_normal(w_mean.squeeze(), w_cov)
        v = np.random.multivariate_normal(v_mean.squeeze(), v_cov)

        # shape them as column-vectors
        x = x.reshape([n, 1])
        w = w.reshape([p, 1])
        v = v.reshape([m, 1])

        # if model is not conformable, exception would be raised (thrown) here
        F * x + C * u + G * w
        H * x + v

        # check controllability, stability, observability
        self.__validate()

        # if the execution reached here, all is fine so
        # define corresponding computational tensorflow graphs
        self.__define_observations_simulation()
        self.__define_likelihood_computation()

    def __define_observations_simulation(self):
        # TODO: reduce code not to create extra operations

        self.__sim_graph = tf.Graph()
        sim_graph = self.__sim_graph

        r = self.__r
        m = self.__m
        n = self.__n
        p = self.__p

        x0_mean = self.__x0_mean
        x0_cov = self.__x0_cov

        with sim_graph.as_default():
```

```
th = tf.placeholder(tf.float64, shape=[None], name='th')

# TODO: this should be continuous function of time
# but try to let pass array also
u = tf.placeholder(tf.float64, shape=[r, None], name='u')

t = tf.placeholder(tf.float64, shape=[None], name='t')

# TODO: refactor

# FIXME: gradient of py_func is None
# TODO: embed function itself in the graph, must rebuild the graph
# if the structure of the model change
# use tf.convert_to_tensor
F = tf.convert_to_tensor(self.__F(th), tf.float64)
F.set_shape([n, n])

C = tf.convert_to_tensor(self.__C(th), tf.float64)
C.set_shape([n, r])

G = tf.convert_to_tensor(self.__G(th), tf.float64)
G.set_shape([n, p])

H = tf.convert_to_tensor(self.__H(th), tf.float64)
H.set_shape([m, n])

x0_mean = tf.convert_to_tensor(x0_mean(th), tf.float64)
x0_mean = tf.squeeze(x0_mean)

x0_cov = tf.convert_to_tensor(x0_cov(th), tf.float64)
x0_cov.set_shape([n, n])

x0_dist = MultivariateNormalFullCovariance(x0_mean, x0_cov,
                                           name='x0_dist')

Q = tf.convert_to_tensor(self.__w_cov(th), tf.float64)
Q.set_shape([p, p])

w_mean = self.__w_mean.squeeze()
w_dist = MultivariateNormalFullCovariance(w_mean, Q, name='w_dist')

R = tf.convert_to_tensor(self.__v_cov(th), tf.float64)
R.set_shape([m, m])
v_mean = self.__v_mean.squeeze()
v_dist = MultivariateNormalFullCovariance(v_mean, R, name='v_dist')

def sim_obs(x):
    v = v_dist.sample()
    v = tf.reshape(v, [m, 1])
    y = H @ x + v  # the syntax is valid for Python >= 3.5
    return y

def sim_loop_cond(x, y, t, k):
```

```
        N = tf.stack([tf.shape(t)[0]])
        N = tf.reshape(N, ())
        return tf.less(k, N-1)

    def sim_loop_body(x, y, t, k):

        # TODO: this should be function of time
        u_t_k = tf.slice(u, [0, k], [r, 1])

        def state_propagate(x):
            w = w_dist.sample()
            w = tf.reshape(w, [p, 1])
            Fx = tf.matmul(F, x, name='Fx')
            Cu = tf.matmul(C, u_t_k, name='Cu')
            Gw = tf.matmul(G, w, name='Gw')
            x = Fx + Cu + Gw
            return x

        tk = tf.slice(t, [k], [2], 'tk')

        x_k = x[:, -1]
        x_k = tf.reshape(x_k, [n, 1])

        x_k = state_propagate(x_k)

        y_k = sim_obs(x_k)

        # TODO: stack instead of concat
        x = tf.concat([x, x_k], 1)
        y = tf.concat([y, y_k], 1)

        k = k + 1

        return x, y, t, k

    x = x0_dist.sample(name='x0_sample')
    x = tf.reshape(x, [n, 1], name='x')

    # this zeroth measurement should be thrown away
    y = sim_obs(x)
    k = tf.constant(0, name='k')

    shape_invariants = [tf.TensorShape([n, None]),
                        tf.TensorShape([m, None]),
                        t.get_shape(),
                        k.get_shape()]

    sim_loop = tf.while_loop(sim_loop_cond, sim_loop_body,
                             [x, y, t, k], shape_invariants,
                             name='sim_loop')

    self.__sim_loop_op = sim_loop
```

```
# defines graph
def __define_likelihood_computation(self):

    self.__lik_graph = tf.Graph()
    lik_graph = self.__lik_graph

    r = self.__r
    m = self.__m
    n = self.__n
    p = self.__p

    x0_mean = self.__x0_mean
    x0_cov = self.__x0_cov

    with lik_graph.as_default():
        # FIXME: Don't Repeat Yourself (in simulation and here)
        th = tf.placeholder(tf.float64, shape=[None], name='th')
        u = tf.placeholder(tf.float64, shape=[r, None], name='u')
        t = tf.placeholder(tf.float64, shape=[None], name='t')
        y = tf.placeholder(tf.float64, shape=[m, None], name='y')

        N = tf.stack([tf.shape(t)[0]])
        N = tf.reshape(N, ())

        F = tf.convert_to_tensor(self.__F(th), tf.float64)
        F.set_shape([n, n])

        C = tf.convert_to_tensor(self.__C(th), tf.float64)
        C.set_shape([n, r])

        G = tf.convert_to_tensor(self.__G(th), tf.float64)
        G.set_shape([n, p])

        H = tf.convert_to_tensor(self.__H(th), tf.float64)
        H.set_shape([m, n])

        x0_mean = tf.convert_to_tensor(x0_mean(th), tf.float64)
        x0_mean.set_shape([n, 1])

        P_0 = tf.convert_to_tensor(x0_cov(th), tf.float64)
        P_0.set_shape([n, n])

        Q = tf.convert_to_tensor(self.__w_cov(th), tf.float64)
        Q.set_shape([p, p])

        R = tf.convert_to_tensor(self.__v_cov(th), tf.float64)
        R.set_shape([m, m])

        I = tf.eye(n, n, dtype=tf.float64)

        def lik_loop_cond(k, P, S, t, u, x, y):
            return tf.less(k, N-1)
```

```
def lik_loop_body(k, P, S, t, u, x, y):

    # TODO: this should be function of time
    u_t_k = tf.slice(u, [0, k], [r, 1])

    # k+1, cause zeroth measurement should not be taken into account
    y_k = tf.slice(y, [0, k+1], [m, 1])

    t_k = tf.slice(t, [k], [2], 't_k')

    # TODO: extract Kalman filter to a separate class
    def state_predict(x):
        Fx = tf.matmul(F, x, name='Fx')
        Cu = tf.matmul(C, u_t_k, name='Cu')
        x = Fx + Cu
        return x

    def covariance_predict(P):
        GQtG = tf.matmul(G @ Q, G, transpose_b=True)
        PtF = tf.matmul(P, F, transpose_b=True)
        P = tf.matmul(F, P) + PtF + GQtG
        return P

    x = state_predict(x)

    P = covariance_predict(P)

    E = y_k - tf.matmul(H, x)

    B = tf.matmul(H @ P, H, transpose_b=True) + R
    invB = tf.matrix_inverse(B)

    K = tf.matmul(P, H, transpose_b=True) @ invB

    S_k = tf.matmul(E, invB @ E, transpose_a=True)
    S_k = 0.5 * (S_k + tf.log(tf.matrix_determinant(B)))

    S = S + S_k

    # state update
    x = x + tf.matmul(K, E)

    # covariance update
    P = (I - K @ H) @ P

    k = k + 1

    return k, P, S, t, u, x, y

k = tf.constant(0, name='k')
P = P_0
S = tf.constant(0.0, dtype=tf.float64, shape=[1, 1], name='S')
x = x0_mean
```

```
            # TODO: make a named tuple of named list
            lik_loop = tf.while_loop(lik_loop_cond, lik_loop_body,
                                     [k, P, S, t, u, x, y], name='lik_loop')

            dS = tf.gradients(lik_loop[2], th)

            self.__lik_loop_op = lik_loop
            self.__dS = dS

    def __isObservable(self, th=None):
        if th is None:
            th = self.__th
        F = np.array(self.__F(th))
        C = np.array(self.__C(th))
        n = self.__n
        obsv_matrix = control.obsv(F, C)
        rank = np.linalg.matrix_rank(obsv_matrix)
        return rank == n

    def __isControllable(self, th=None):
        if th is None:
            th = self.__th
        F = np.array(self.__F(th))
        C = np.array(self.__C(th))
        n = self.__n
        ctrb_matrix = control.ctrb(F, C)
        rank = np.linalg.matrix_rank(ctrb_matrix)
        return rank == n

    # FIXME: fix to discrete
    def __isStable(self, th=None):
        if th is None:
            th = self.__th
        F = np.array(self.__F(th))
        eigv = np.linalg.eigvals(F)
        real_parts = np.real(eigv)
        return np.all(real_parts < 0)

    def __validate(self, th=None):
        # FIXME: do not raise exceptions
        # TODO: prove, print matrices and their criteria
        if not self.__isControllable(th):
            # raise Exception('''Model is not controllable. Set different
            #                 structure or parameters values''')
            pass

        if not self.__isStable(th):
            # raise Exception('''Model is not stable. Set different structure or
            #                 parameters values''')
            pass

        if not self.__isObservable(th):
```

```
            # raise Exception('''Model is not observable. Set different
            #                 structure or parameters values''')
            pass

    def sim(self, u, th=None):
        if th is None:
            th = self.__th

        k = u.shape[1]
        t = np.linspace(0, k-1, k)

        self.__validate(th)
        g = self.__sim_graph

        if t.shape[0] != u.shape[1]:
            raise Exception('''t.shape[0] != u.shape[1]''')

        # run simulation graph
        with tf.Session(graph=g) as sess:
            t_ph = g.get_tensor_by_name('t:0')
            th_ph = g.get_tensor_by_name('th:0')
            u_ph = g.get_tensor_by_name('u:0')
            rez = sess.run(self.__sim_loop_op, {th_ph: th, t_ph: t, u_ph: u})

        return rez

    def lik(self, u, y, th=None):

        # hack continuous to discrete system
        k = u.shape[1]
        t = np.linspace(0, k-1, k)

        if th is None:
            th = self.__th

        # to numpy 1D array
        th = np.array(th).squeeze()

        # self.__validate(th)
        g = self.__lik_graph

        # TODO: check for y also
        if t.shape[0] != u.shape[1]:
            raise Exception('''t.shape[0] != u.shape[1]''')

        # run lik graph
        with tf.Session(graph=g) as sess:
            t_ph = g.get_tensor_by_name('t:0')
            th_ph = g.get_tensor_by_name('th:0')
            u_ph = g.get_tensor_by_name('u:0')
            y_ph = g.get_tensor_by_name('y:0')
            rez = sess.run(self.__lik_loop_op, {th_ph: th, t_ph: t, u_ph: u,
                                                y_ph: y})
```

```
        # FIXME: fix to discrete
        N = len(t)
        m = y.shape[0]
        S = rez[2]
        S = S + N*m * 0.5 + np.log(2*math.pi)

        return S

    def __L(self, th, u, y):
        return self.lik(u, y, th)

    def __dL(self, th, u, y):
        return self.dL(u, y, th)

    def dL(self, u, y, th=None):
        if th is None:
            th = self.__th

        # hack continuous to discrete system
        k = u.shape[1]
        t = np.linspace(0, k-1, k)

        # to 1D numpy array
        th = np.array(th).squeeze()

        # self.__validate(th)
        g = self.__lik_graph

        if t.shape[0] != u.shape[1]:
            raise Exception('''t.shape[0] != u.shape[1]''')

        # run lik graph
        with tf.Session(graph=g) as sess:
            t_ph = g.get_tensor_by_name('t:0')
            th_ph = g.get_tensor_by_name('th:0')
            u_ph = g.get_tensor_by_name('u:0')
            y_ph = g.get_tensor_by_name('y:0')
            rez = sess.run(self.__dS, {th_ph: th, t_ph: t, u_ph: u, y_ph: y})

        return rez[0]

    def mle_fit(self, th, u, y):
        # TODO: call slsqp
        th0 = th
        th = scipy.optimize.minimize(self.__L, th0, args=(u, y),
                                     jac=self.__dL, options={'disp': True})
        return th
```