RC23952 (W0605-020) May 3, 2006
Mathematics

A Neural Network Analogue of the Mammalian Local Cortical Circuit Implements Optimal Kalman Prediction and Control

Ralph Linsker
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
linsker@us.ibm.com

Abstract

The local cortical circuit (LCC) is a fundamental unit of mammalian neural processing found in neocortical areas subserving diverse sensory, motor, and other functions. A central challenge for neuroscience is to understand what core processing functions the LCC performs. Prediction, the estimation or inference of missing or noisy sensory data, and the goal-driven generation of control signals are important functions of biological brains. Kalman's classical solutions of the optimal estimation and control problems in simple linear systems, and their extensions, have been widely used in engineering for 45 years. It has been speculated that neural circuitry implementing solutions related to Kalman's might be important for enabling sensory processing and motor control, but no neural circuit and algorithm for the general Kalman solution has previously been described. Here we show how optimal Kalman estimation and control are learned and executed by a neural network having simple computational elements. The circuit architecture implied by this algorithm is similar in several ways to recurrent neural circuits in brain, and to LCC architecture in particular. These results suggest that the core functions of the LCC may include those of estimation (including prediction) and control, including more sophisticated nonlinear and context-dependent methods beyond Kalman's solutions, and that LCC studies may be guided by these connections between biological and engineering design. Conversely, further analysis of LCC circuitry and function may lead to advances in nonlinear estimation and control for engineering applications.

Mammalian neocortex exhibits significant uniformity at the 50- to 100-micron scale of the local cortical circuit (LCC, minicolumn, canonical microcircuit)[1, 2, 3, 4]. This uniformity motivates a search for a set of core LCC processing functions that may be common to sensory, motor, and other cortical areas, and that may enable the diverse functions of those areas[5, 6]. Inference, prediction, and estimation, and the goal-driven generation of control signals, are important functions of biological brains. The blending of 'bottom-up' sensory input and 'top-down' model-driven expectations has been discussed in the context of Bayesian inference and generative models, and various neural network (NN) algorithms are motivated by, approximate, or perform a portion of, the Bayesian inference process[7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. A general resemblance has been noted between (a) the feedforward and feedback connections within the LCC and between cortical areas, and (b) the use of bottom-up and top-down signals in the algorithms. Bayes-optimal behavior has been found in human psychophysics experiments[17].

Kalman filtering (KF)[18] is a classically important and exactly solvable special case of Bayesian inference. KF performs optimal prediction and estimation when the external system's dynamics and the measurement process are both described by linear equations. Kalman control (KC)[18] is closely related; it generates optimal control outputs to achieve goals (e.g., target acquisition) having a certain mathematical form. However, although KF and KC have been proposed to play important roles in biological neural processing[9, 19], and KF has been used with artificial NNs (e.g., to speed learning)[20], no NN algorithm that fully performs KF or KC (i.e., without making substantial simplifications or being limited to special subcases [21, 22]) has to my knowledge previously been described (see *Discussion*).

In this paper I derive a NN algorithm that performs both KF and KC, without such simplifications or limitations. In the limit of a large number of features (e.g., elements of a visual scene) being tracked, the algorithm exactly reproduces Kalman's solutions, including the exponentially rapid convergence of the learned KF and KC matrices. Furthermore, unlike Kalman's methods, in which the external system and measurement process parameters are assumed known in advance, the NN algorithm also learns the required parameters using only a stream of noisy measurement data.

Strikingly, the derivation implies significant constraints on the signal flow, multilayer architecture, and circuitry of a NN that can perform KF and KC. Many features of the resulting NN are found to resemble those of the LCC. These resemblances lend support to the conjecture that the LCC performs prediction and control (P&C) functions that are related to, although more powerful (e.g., in their handling of nonlinearities and context-dependence) than, Kalman's solutions.

The approach taken here is thus to (a) pose a well-defined computational task – Kalman P&C – that is a prototype of the more general P&C processes likely to be important in cortex; (b) select a simple typical set of allowed NN operations, rather than invoking more complex NN dynamics tailored to the task; (c) see whether a NN algorithm can be devised that does not compromise, change, or limit the computational task; (d) see what constraints that task imposes on the NN's circuitry and signal flows; and (e) compare the resulting NN circuitry with that of the biological system of interest.

Organization of the paper. The classical linear P&C problems and Kalman's solutions are summarized. A set of elementary NN operations is defined; the NN algorithms to be derived are required to use only these operations. We then transform Kalman's solutions into a set of matrix equations that eliminate all references to external system and measurement parameters that cannot be separately known by the NN. We derive NN algorithms that perform KF and KC without any approximations other than those resulting from statistical fluctuations that arise because the set of input features being tracked is finite. We then discuss the specific signal flows, layered organization of the computations, and circuit architecture that are implied by the algorithms and the allowed set of NN operations, and discuss these results in the context of the LCC's known structure and possible core functions.

Classical Kalman linear estimation and control

In classical linear estimation and control[18], an external system (the 'plant') is described by a state vector x_t (e.g., a point's trajectory) at each discrete time t, and the dynamical rule $x_{t+1} = Fx_t + Bu_t + m_t$, where m_t is plant noise (e.g., random buffeting of an object) having mean zero and covariance Q. Each measurement vector $y_t = Hx_t + n_t$, where n_t is measurement noise having mean zero and covariance R. Matrices F, B, H, Q, and R, and the optional vector u_t (an external driving term and/or a generated control term) are assumed known.

Classical Kalman estimation (filtering and prediction). Given measurements through time t, the goal of optimal filtering (or, respectively, one-step-ahead prediction) is to compute a posterior state estimate $\hat{x}_t$ (resp., a prior state estimate $\hat{x}_{t+1}^-$) that minimizes the generalized mean-square estimation error[*] $E[(\xi_t)'C\xi_t]$ (resp., $E[(\xi_{t+1}^-)'C\xi_{t+1}^-]$) where $\xi_t \equiv x_t - \hat{x}_t$, $\xi_{t+1}^- \equiv x_{t+1} - \hat{x}_{t+1}^-$, and C is a symmetric positive-definite matrix.

Kalman[18] showed that, under a variety of conditions, the optimal estimation solution for both filtering and prediction is given by the execution equations

$$\hat{x}_t = \hat{x}_t^- + K_t(y_t - H\hat{x}_t^-) \ ; \quad \hat{x}_{t+1}^- = F\hat{x}_t + Bu_t \ ; \tag{1}$$

and the learning equations

$$K_t = P_t^- H'(HP_t^- H' + R)^{-1} \ ; \quad P_{t+1}^- = F(I - K_t H)P_t^- F' + Q \ . \tag{2}$$

Equations 2 are initialized by assuming some distribution of values for ξ_0^- and setting $P_0^- \equiv E[\xi_0^-(\xi_0^-)']$. It then follows[18] that, for all t, $P_t^- = E[\xi_t^-(\xi_t^-)']$. Thus the KF matrix is learned iteratively, starting with an arbitrary matrix and converging exponentially rapidly to its final value as each new measurement is obtained. The classical KF learning algorithm involves multiplications of one matrix by another, and matrix inversion.

The model prediction $\hat{x}_t^-$ and the current measurement y_t are optimally blended using the KF. As expected intuitively, when the plant noise is much greater than the measurement noise, this blending gives greater weight to the current measurement; when the measurement noise is much greater, the model prediction receives greater weight.

Classical Kalman control. In the classical problem of optimal linear quadratic regulation, a controller is to generate a set of signals $\{u_t\}$ that minimizes the expected total cost J of approaching a desired target state (taken here to be $x = 0$ for simplicity) at time N. Here J reflects the cost of producing each control output (e.g., the energetic cost of moving a limb or firing a rocket thruster) plus a penalty that is a function of the difference between the actual state at each time step and the target state. Specifically, $J = E[\Sigma_{t=t_0}^{N}(u_t' g u_t + x_t' r x_t)]$, where g and r are specified symmetric positive-definite matrices.

The classical Kalman control (KC) algorithm[18], starting at a current time t_0, computes during the learning step a set of KC matrices $\{L_N, L_{N-1}, \ldots, L_{t_0}\}$,

[*]Notation: $E(\ldots)$ denotes expectation value, prime denotes transpose, and I is the identity matrix.

using an auxiliary symmetric positive-definite matrix S_τ where $S_N = r$ and

$$L_\tau = (B' S_\tau B + g)^{-1} B' S_\tau F \ ; \quad S_{\tau-1} = (F' - L'_\tau B') S_\tau F + r \ ; \tag{3}$$

for $\tau = N, N-1, \ldots, t_0$. Then, as time t advances from t_0 to $N-1$, the execution step $u_t = -L_t \hat{x}_t$ generates the optimal control signal u_t.

Thus the Kalman controller generates an optimal sequence of control outputs that cause the plant to approach a desired target state at a specified future time N. The KC matrices are learned iteratively, 'backward in time,' then executed forward in time.

NN operations

A linear NN[23, 24] comprises nodes (simplified 'neurons') i each having an activity v_i, and connections $i \to j$ each having a weight M_{ji}; v and M are real-valued. The output activity of node j is $z_j = \Sigma_i M_{ji} v_i$. We consider layered NNs. A set of source nodes in one layer, a set of target nodes in another, and the set of source-to-target connections, represent input vector v, output vector z, and (feedforward or feedback) weight matrix M, and satisfy $z = Mv$. For a set of nodes in a single layer, connected to itself via lateral connections, $z = Mv$ replaces v as the new activity vector after one pass through M. The activities of two sets of nodes may also be algebraically combined: $v^{\text{out}} = v^{(1)} \pm v^{(2)}$. We consider a simple bilinear Hebbian learning rule in which each weight is modified by an amount that depends only on its current value and the activities at either end of that connection at a given time; e.g.,

$$M \leftarrow (1 - s\gamma) M + s\gamma z v' \tag{4}$$

where $\gamma > 0$ is the learning rate, and $s = 1$ for Hebbian, or -1 for anti-Hebbian, learning. Additional constant terms and terms proportional to z or v may also be included. For lateral connections the zv' factor is replaced by vv' or zz'. (For other types of NNs, see *Discussion.*)

Note that a connection matrix can be multiplied by an activity vector, but not by another matrix. That is, given two sets of connections of weights $M^{(1)}$ and $M^{(2)}$, one cannot directly create a third set having the matrix-product weights $M = M^{(2)} M^{(1)}$, even though any given vector v may be successively multiplied by $M^{(1)}$, then by $M^{(2)}$. Similarly, a matrix M cannot be inverted to compute M^{-1}, although we can use M to compute $M^{-1} v$ for a given v

(see below). Thus Eqs. 2 cannot be implemented, as they stand, using the allowed NN operations.

We also consider an optional 'ganging' operation, in which there are N_{gang} sets of nodes, each set processing a different input vector (e.g., a feature in a visual scene) at the same time step, and a connection matrix M common to all sets. M may be learned using either (a) $M \leftarrow (1 - sN_{\text{gang}}\gamma)M + sN_{\text{gang}}\gamma\langle z(p)v'(p)\rangle$, where p denotes the pth set of nodes and $\langle \ldots \rangle$ is an average over p, or (b) Eq. 4, using for the last term $s\gamma z(p)v'(p)$ for each p in turn. An artificial NN can either (a) gang together the corresponding weights $M_{ji}(p)$ that process each feature p, so that all such weights are updated together[25], or (b) transport each feature's activity vectors in turn to a common connection matrix M for activity computation and M update.

We will need to compute an estimate of $E(zv')$ over statistical distributions of v and z. Repeated application of Eq. 4 computes a recency-weighted running average of the product zv' over the past $O(1/\gamma)$ activity pairs (v, z). If these are enough pairs to yield representative statistics, and if the v and z distributions have not changed significantly during the time $O(1/N_{\text{gang}}\gamma)$ required to generate the pairs, then M learns an estimate of $E(zv')$. When $E(v) = 0$, $E(vv')$ equals the covariance matrix $\text{Cov}(v)$.

We will also need to compute $M^{-1}v$, where M is an estimate of $E(vv')$. This can be done in at least two ways.

Method 1[26]: Each activity vector v is presented in turn as input to a set of nodes that have lateral connections with weight matrix D, which is updated using $D \leftarrow (1-\gamma)D - \gamma(cvv' - I)$, yielding $D = I - cM$. To obtain $M^{-1}v$, each of these activity vectors v is held at the input to the set of nodes (i.e., the input is not reset to zero), while the network iteratively processes the resulting activity through the set of lateral connections. The activity vector will then be, as the iterations proceed: v, $v + Dv$, $v + D(v + Dv) = (I + D + D^2)v$, etc. For $c > 0$ chosen[26] such that all eigenvalues of D have magnitude < 1, this activity converges to $(I + D + D^2 + \ldots)v = (I - D)^{-1}v = (1/c)M^{-1}v$, which is (apart from the constant factor $1/c$) the desired result. Thus the incremental updating of D and the computation of $M^{-1}v$ are jointly performed.

Method 2[27]: Each activity vector v is presented in turn to a set of nodes having lateral connections A. After one pass through the lateral connections, the activity vector is $z = Av$. A is updated using $A \leftarrow (1+\gamma)A - \gamma zz'$. Provided that asymmetric terms in A (arising from initial asymmetries and/or buildup of numerical errors) can be kept small, e.g., by damping the asymmetric terms, A will learn M^{-1}.

NN algorithms

Kalman estimation and system identification. For our neural algorithm (in contrast to Kalman's solution above), we assume that the NN is given *only* the stream of noisy measurements $\{y_t\}$; no plant, measurement, or noise covariance parameters are assumed known. To eliminate x in favor of y variables, we define $Y_t = Hx_t$, the ideal noiseless measurement of x_t. The goal of optimal filtering (respectively, prediction) stated earlier is then, given $\{y_0, \dots, y_t\}$, to compute a posterior (resp., prior) measurement estimate $\hat{y}_t$ (resp., $\hat{y}^-_{t+1}$) that minimizes $E[(Y_t - \hat{y}_t)'\tilde{C}(Y_t - \hat{y}_t)]$ (resp., $E[(Y_{t+1} - \hat{y}^-_{t+1})'\tilde{C}(Y_{t+1} - \hat{y}^-_{t+1})]$), where $\tilde{C} \equiv H'CH$.

We do not need to learn H or F, but only the combination[†] $\tilde{F} \equiv HFH^+$. Prior to KF learning, $\tilde{F}$ is learned by using the raw measurement stream to minimize the mean-square prediction error $E(\epsilon'_t\epsilon_t)$ with respect to $\tilde{F}$, where $\epsilon_t = -y_t + \tilde{F}y_{t-1} + \tilde{u}_{t-1}$ and $\tilde{u}_t \equiv HBu_t$. Stochastic gradient descent yields the update rule: $\tilde{F} \leftarrow \tilde{F} - \gamma_F \epsilon_t y'_{t-1}$, where γ_F is a learning rate. For this to be a local learning rule, ϵ_t and y_{t-1} must be available at the two ends of the $\tilde{F}$ connection matrix at the same time (see *Derivation of layered NN architecture*).

$R = E(nn')$ is learned from 'measurements' taken in offline mode (i.e., in the absence of external input, so that $y_t = n_t$) by Hebbian covariance learning, $R \leftarrow (1 - \gamma_R)R + \gamma_R yy'$.

To transform the classical KF equations into a form suitable for a neural algorithm, we define $Z_t \equiv HP^-_tH' + R$. Then $I - HK_t = RZ_t^{-1}$. The transformed matrix equation that corresponds exactly to Eqs. 2 is:

$$Z_{t+1} = \tilde{F}(I - RZ_t^{-1})R\tilde{F}' + HQH' + R \qquad (5)$$

(see *Supporting text*).

The execution equations are (cf. Eqs. 1)

$$\hat{y}_t = y_t - RZ_t^{-1}(y_t - \hat{y}^-_t)\ ; \quad \hat{y}^-_{t+1} = \tilde{F}\hat{y}_t + \tilde{u}_t\ . \qquad (6)$$

Therefore $\eta_t \equiv \hat{y}^-_t - y_t$ evolves as

$$\eta_{t+1} = -y_{t+1} + \tilde{F}(y_t + RZ_t^{-1}\eta_t) + \tilde{u}_t\ , \qquad (7)$$

[†]M^+ denotes the Moore-Penrose pseudoinverse of M. When M is a square matrix of full rank, $M^+ \equiv M^{-1}$.

and is initialized by assuming some distribution of values for η_0.

The relationship between η_t and Z_t is crucial for deriving the neural algorithm. If $Z_t = E(\eta_t \eta_t')$ (initialized by setting $Z_0 = E(\eta_0 \eta_0')$), then Eqs. 5 and 7 yield $Z_{t+1} = E(\eta_{t+1}\eta_{t+1}')$ (see *Supporting text*). Thus, in the idealized large-N_{gang} limit for which $\langle \eta_{t+1}\eta_{t+1}' \rangle \to E(\eta_{t+1}\eta_{t+1}')$, the learning rule $Z_{t+1} = \langle \eta_{t+1}\eta_{t+1}' \rangle$ and Eq. 7 exactly implements Eq. 5, and thereby yields a KF identical to the classical KF at each t. In practice, for finite N_{gang}, we use Method 1 or 2 to update Z or Z^{-1}, with optional ganging (see *NN operations*). If N_{gang} is finite but large enough so that we can choose $N_{\text{gang}}\gamma \approx 1$, then the KF learned at each time step approximates that obtained using Kalman's algorithm at the same time step. However, if $N_{\text{gang}}\gamma \ll 1$ (as, e.g., when $N_{\text{gang}} = 1$), it requires many time steps of the neural algorithm to learn the change in the KF matrices that occurs in one time step using the classical algorithm.

Once $\hat{y}$ is approximately equal to y (i.e., once Z has evolved sufficiently from its arbitrary initial value), $\tilde{F}$ learning may optionally continue using $\hat{y}_{t-1}$ and η_t in place of y_{t-1} and ϵ_t respectively; i.e., $\tilde{F} \leftarrow \tilde{F} - \gamma_F \eta_t \hat{y}_{t-1}'$.

Fig. 1 (solid path) shows the signal flow that integrates the above learning algorithms for Z (or Z^{-1}), $\tilde{F}$, and R, and the execution Eqs. 6 and 7. Assume for now that line Cu1 and/or Cu2 is cut; i.e., no control signals are computed. For the execution process, starting with the link (near upper left) labeled η_t, the computation sequence is: $\eta_t \to Z^{-1}\eta_t \to RZ^{-1}\eta_t \to \hat{y}_t = y_t + RZ^{-1}\eta_t \to \tilde{F}\hat{y}_t \to \hat{y}_{t+1}^{-} = \tilde{u}_t(\text{ext}) + \tilde{F}\hat{y}_t \to \eta_{t+1} = -y_{t+1} + \hat{y}_{t+1}$ (external driving term $\tilde{u}_t(\text{ext})$ is optional). Using Method 1, Z learning occurs at link LZ1; or, using Method 2, Z^{-1} learning occurs at link LZ2. $\tilde{F}$ learning uses $\hat{y}_{t-1}$ at link LF1 (held from the previous time step) and η_t at link LF2. For initial learning of $\tilde{F}$ (before Z is used), the circuit is cut at link CF, so that LF1 carries activity y_{t-1} and LF2 carries ϵ_t. In the offline mode for learning R, the circuit is cut at link CR, so that the following link LR carries activity $y_t = n_t$.

See *Supporting text* and Fig. 3 for simulations illustrating NN learning of $\tilde{F}$ and the KF.

Kalman control. We define: $\tilde{g} \equiv H'^{+}B'^{+}gB^{+}H^{+}$; $\tilde{r} \equiv H'^{+}rH^{+}$; $\tilde{L}_\tau \equiv -HBL_\tau H^{+}$; and $T_\tau \equiv H'^{+}S_\tau H^{+} + \tilde{g}$.

The transformed matrix equation that corresponds exactly to Eqs. 3 is

$$T_{\tau-1} = \tilde{F}'\tilde{g}(I - T_\tau^{-1}\tilde{g})\tilde{F} + \tilde{r} + \tilde{g} \ . \tag{8}$$

Similarly to the case of NN estimation, we will represent T as the covariance of the distribution of an activity vector w, so that the learning rule for T may be recast as an evolution equation for w. However, whereas the physically meaningful quantity η was the vector whose covariance equaled Z in the case of estimation, we now have to construct w from terms that are based on the goal of the control problem, i.e., the cost function to be minimized.

We introduce an activity vector w_τ, and construct a rule for computing $w_{\tau-1}$ in terms of w_τ, such that $E(ww')$ satisfies the same evolution equation as T (Eq. 8):

$$w_{\tau-1} = -\nu^g_{\tau-1} + \nu^r_\tau + \tilde{F}'(\nu^g_\tau + \tilde{g}T^{-1}_\tau w_\tau) \ . \tag{9}$$

(See *Supporting text* for proof.) Here ν^g_τ and ν^r_τ are random vectors, or internally generated 'noise,' drawn from distributions having mean zero and covariances $\tilde{g}$ and $\tilde{r}$. These noise generators are the means by which the neural system represents the cost matrices $\tilde{g}$ and $\tilde{r}$.

Learning: At the current time t_0, a set of KC matrices T_τ to be used at future times is learned by iteratively computing Eq. 9 for $\tau = N, N-1, \ldots, t_0+1$, starting with $w_N = \nu^r_{N+1} - \nu^g_N$ (corresponding to $S_N = r$). Neural control learning, using $T_{\tau-1} = \langle w_{\tau-1}w'_{\tau-1}\rangle$, exactly yields Kalman control in the idealized limit, $\langle w_{\tau-1}w'_{\tau-1}\rangle \to E(w_{\tau-1}w'_{\tau-1})$, of a large number of instances of w_τ (at each τ). In practice, $T \to (1-\gamma_T)T + \gamma_T ww'$ (or Method 2 for learning T^{-1}) approximates classical KC learning.

Execution: The computed T_t is used to compute the desired control signal

$$\tilde{u}_t = \tilde{L}_t\hat{y}_t = (-I + T^{-1}_t\tilde{g})\tilde{F}\hat{y}_t \ . \tag{10}$$

Depending upon the choice of implementation, one may either (a) store the sequence of matrices during learning and retrieve them in reverse order during execution (possibly using different parts of a larger NN for each matrix, not discussed here), or (b) retain only the last-computed matrix T_{t_0}, use it for execution at the current time t_0, then relearn the T matrices at the next time step t_0+1. Alternatively, one may approximate ideal KC by using the same computed matrix for several time steps.

Execution of control is shown in Fig. 1 (solid path) starting at link Cu1. The computation sequence is: $\tilde{F}\hat{y}_t \to \tilde{g}\tilde{F}\hat{y}_t \to T^{-1}\tilde{g}\tilde{F}\hat{y}_t \to \tilde{u}_t = -\tilde{F}\hat{y}_t + T^{-1}\tilde{g}\tilde{F}\hat{y}_t = (-I + T^{-1}\tilde{g})\tilde{F}\hat{y}$ at link Cu2. Output $\tilde{u}_t$ is the control signal, and is also provided as efferent-copy feedback to compute $\hat{y}^-_{t+1}$.

Fig. 1 (dashed path) shows the signal flows that implement learning of control; i.e., Eq. 9 and learning of T (or T^{-1}) and $\tilde{g}$. For w evolution, the computation sequence is (starting near lower right, at the link labeled

w_τ): $w_\tau \to T^{-1}w_\tau \to \tilde{g}T^{-1}w_\tau \to \nu^g_\tau + \tilde{g}T^{-1}w_\tau \to \tilde{F}'(\nu^g_\tau + \tilde{g}T^{-1}w_\tau) \to \nu^r_\tau + \tilde{F}'(\nu^g_\tau + \tilde{g}T^{-1}w_\tau) \to w_{\tau-1} = -\nu^g_{\tau-1} + \nu^r_\tau + \tilde{F}'(\nu^g_\tau + \tilde{g}T^{-1}w_\tau)$.

Either T learning occurs at link LT1, or T^{-1} learning occurs at link LT2. The $\tilde{F}'$ connections will join the same nodes as $\tilde{F}$, but in the reverse direction (discussed below); thus they are learned along with $\tilde{F}$ (prior to being used for control) using the same Hebb rule. $\tilde{g}$ is learned (analogously to R above) in a mode of circuit operation in which the dashed link Cg is cut, so that the next link Lg carries activity ν^g_τ, which is used for Hebbian learning of $\tilde{g} = E[\nu^g(\nu^g)']$.

For a summary of the operation of the composite KF and KC algorithm, see *Supporting text.*

Derivation of layered NN architecture (Fig. 2). The assignment of activity vectors to distinct NN layers, and details of the signal flow among layers, are strongly constrained by several requirements:

1. Each of the four matrices R, $\tilde{g}$, Z (or Z^{-1}), and T (or T^{-1}), is learned using a Hebb rule (Eq. 4 with $z \equiv v$, or Method 1 or 2) containing the factor vv' or zz'; i.e., the connection matrix joins an activity vector to itself. Thus each matrix describes a set of lateral connections, and is assigned to its own layer, denoted by R, g, Z, T in Fig. 2a, and shown separately for the solid (KC execution and KF) and dashed (KC learning) signal paths.

2. For Hebb learning, activity η_t must be present at the input to the Z (or Z^{-1}) connections; w_τ at the input to T (or T^{-1}); $y_t \equiv n_t$ at the input to R during R learning mode; and ν^g_τ at the input to $\tilde{g}$ during $\tilde{g}$ learning mode.

3. For Hebb learning of $\tilde{F}$, activities $\hat{y}_{t-1}$ and η_t must be present simultaneously at the two ends of the $\tilde{F}$ matrix. Thus $\hat{y}_{t-1}$ must be held as the activity of one set of nodes (see Fig. 2a solid-path layer R) until η_t has been computed at layer Z. $\tilde{F}$ is updated at the time indicated by links LF1 and LF2, but is used later in the signal flow, at the link labeled $\tilde{F}$.

4. The transpose matrix $\tilde{F}'$ is required for KC learning (Fig. 2a, dashed path). $\tilde{F}'$ is learned by the same algorithm, and at the same time, as $\tilde{F}$; thus it is assigned to connect the same two layers as $\tilde{F}$, but in the reverse direction. Fig. 2a assigns $\tilde{F}$ to run from layer R to g (solid

path), and $\tilde{F}'$ from g to R (dashed path); this requires η to be copied from layer Z to g as shown (just before LF2). [As a slight variant, $\tilde{F}$ could run instead from R to Z; then η_t would not need to be copied from Z to g, but the solid path would require a Z $\rightarrow$ g link just following the $\tilde{F}$ link from R to Z (Fig. 2a, solid path) for KC execution, and the dashed path would require a g $\rightarrow$ Z link following link Lg, to connect to $\tilde{F}'$.]

5. The measurement vector y_t is required twice as input: to layer Z, where it contributes to η_t, and to layer R, where it combines with $RZ^{-1}\eta_t$ to yield $\hat{y}_t$.

Thus Fig. 2a is not an arbitrary way of laying out the NN algorithms we have derived; rather, its four-layer organization and its signal flows are substantially determined (apart from small variations) by the algorithms and the requirements of a NN implementation.

The composite multilayer circuit. The static neural circuit for the integration of Kalman estimation and control – showing all connections, but omitting the explicit time flows – is shown in Fig. 2b. The signal flows of Figs. 1 and 2a can readily be traced through this circuit (see *Supporting text*). The circuit operation comprises several modes (requiring appropriate functional switching) including: (a) learning of the $\tilde{F}$ and $\tilde{F}'$ connection weight matrices (system identification); (b) normal 'online' KF and execution of KC; (c) iterative learning of the KC matrices T; and (d) initial or intermittent 'offline' learning of matrices R (for KF) and $\tilde{g}$ (for KC). Optional outputs $\hat{y}$, $\tilde{F}\hat{y}$, $\hat{y}^{-}$, and/or η can be provided (not shown) from layers R, g (or Z), Z, and Z (or g) respectively.

Discussion

We have shown how to perform the learning and execution of Kalman estimation and control, as well as system identification, using a neural network. The method is asymptotically exact in the limit that a large number of features are being tracked. The matrices Z and T that are iteratively computed to learn and execute KF or KC are each equal to the covariance of a distribution of computed activity vectors. These vectors evolve over time via a sequence of transformations (performed by the NN), and are used, via Hebbian learning, to update a matrix of connection weights that represents either the KF or KC matrix (or their inverses). The logical path of the derivation proceeds from the classical Kalman solutions, to a transformed set of equations that involves only quantities measured by the NN, to a set of signal flows and computations, to a layered NN architecture and circuitry that supports those computations. Both the signal flows and the NN architecture appear to be determined (apart from small variations); they do not appear to represent merely one among a large number of disparate possible architectures.

This work has implications for both engineering and neuroscience.

As a mathematical and engineering method, these NN algorithms may prove useful for implementing estimation, control, and system identification in special-purpose hardware comprising simple computational elements, especially with a large number of sets of such elements operating in parallel. Even when the plant or measurement parameters change with time, the present NN algorithms learn the new dynamics automatically and converge to the new optimal solution after a transient period of adjustment. The well-known extended Kalman filter (EKF)[20] and its variants yield approximate solutions for nonlinear plant and measurement processes, by repeatedly linearizing the dynamics about an operating point. Our NN algorithms likewise yield approximate solutions in these cases, since the learned plant and measurement parameters are automatically updated in response to the changing stream of noisy measurements.

A prior NN algorithm for KC[22], using temporal-difference learning, performs KC in the special case of stationary control, in which there is no specified time-to-target N, but not in the general KC case treated here. A KF-inspired NN algorithm[21] substantially alters Kalman's formulation; the resulting NN does not in general implement KF. Although the initial prediction error, starting with an arbitrary prediction, is shown to decrease rapidly with time[21], this provides no evidence that the KF has been even approximately learned; indeed, a similar reduction in error is found even when an arbitrary, non-optimal, and unchanging filter, differing greatly from the true KF, is used. See *Supporting text.*

Comparison of NN Kalman circuit with the LCC. This section is speculative, and several caveats apply when comparing our circuit with the LCC.

(1) We have solved the well-defined problem of learning and executing general Kalman estimation and control using a NN capable of certain elementary computations. If the LCC performs P&C, it surely performs a more powerful version of P&C – including the learning of higher-level features,

and other nonlinear and context-dependent analyses – although the LCC's algorithm may subsume KF and KC as simple special cases.

(2) Neuronal dynamics are much more complex than the NN operations allowed here. Despite this, it is commonly (and often fruitfully) assumed that reduced or simplified NN models can capture relevant dynamical features of biological neuronal networks. As an example, the node activity in a nonlinear (sigmoidal) version of the NN used here is often identified with an average neuronal firing rate; and connection weights, with synaptic efficacies.[‡] However, other types of NNs represent and process information in a variety of ways[23, 24], using, e.g., (a) place coding, in which the location of an active set of nodes (rather than the activity values themselves) encodes a vector; (b) coding via precise spike timing[28]; and (c) the related use of synchronous or phase-locked firing or oscillations for conveying information and/or for more efficient learning. Detailed neuronal dynamics also affect the relative timing of excitatory and inhibitory effects, the occurrence of bursting vs. tonic firing modes, etc., all of which are absent from our simple NN.

When a type of NN supports the basic operations used here – matrix-times-vector multiplication, addition of vectors, and bilinear Hebbian learning – then the derivation of the algorithm and architecture can proceed as described, unchanged at the level of abstraction of Figs. 1 and 2 (although the particular way in which a vector is multiplied by a matrix will depend on the NN type). When the NN supports a quite different set of operations, however, it is an open question whether and how neural KF and KC may be implemented, and what the resulting architecture will be.

(3) For our allowed set of NN operations, our exploration of the design constraints for performing general KF and KC suggests that the resulting signal flow and circuit are substantially determined (apart from small variations), but the non-existence of a significantly different design cannot be assured.

(4) Experimental knowledge of the detailed LCC connectivity is substantial, but not complete; e.g., an inhibitory cell may provide output to many layers, and the extent and importance of some of these connections are not clear.

[‡]Average firing rates must be nonnegative and synaptic efficacies cannot change sign. To modify our linear NN to satisfy these constraints, one could replace (a) each node by a rectifier plus two nodes having activities $(v, 0)$ if $v > 0$ and $(0, -v)$ if $v < 0$, and (b) each connection by a direct path plus a path having an inhibitory internode. These changes would not affect our results.

Bearing in mind these caveats, we first compare gross features of the NN and LCC, then proceed to more detailed comparisons.

First, both circuits are recurrent neural networks: there is an overall cycle of feedforward and feedback processing, which in the NN occurs once for each time step of the KF algorithm. This was to be expected, given the iterative nature of the classical Kalman algorithms.

Second, measurement input y is required at two NN layers. Its input to layer R can be considered the primary term, and its input to layer Z a secondary term, in the sense that the latter corrects the raw measurement by $RZ^{-1}(\hat{y}_t^- - y_t)$ to yield the optimal estimate. LCC input (from a 'lower' cortical area or thalamus) is likewise twofold, to layers 4 (dominant) and 6 (modulatory); in LCC, unlike the linear NN, these inputs can interact nonlinearly.

Third, the interlaminar signal flow for KF (learning and execution) and KC (execution only) (Fig. 2a and b, solid path) may be schematized, considering layers g and T as a unit, as Dgm. D1 (below, left):

$$\begin{tikzcd}
\{\ \mathrm{R} & \} \arrow[r] \arrow[d] & (\mathrm{g},\mathrm{T}) \arrow[r] \arrow[d] & \mathrm{Z} \arrow[r] & \mathrm{R} \\
y \arrow[u] & \hat{y} & \tilde{u} & y \arrow[u] & \textbf{[D1]}
\end{tikzcd}
\qquad
\begin{tikzcd}
4 \arrow[r] & 2/3 \arrow[r] \arrow[d] & 5 \arrow[r] \arrow[d] & 6 \arrow[r] & 4 \\
\mathrm{in} \arrow[u] & \mathrm{out}_S & \mathrm{out}_M & \mathrm{in} \arrow[u] & \textbf{[D2]}
\end{tikzcd}$$

(KC learning adds a g $\rightarrow$ R path.) By comparison, Gilbert's proposal[2] for the principal LCC signal flow among the layers 6, 5, 4, and 2/3 (a composite of layers 2 and 3) is Dgm. D2 (above, right). More recent work is consistent with, and expands upon, this basic flow[3, 4, 29]. Layer 4 is elaborated in visual cortex and is much less prominent in motor than in sensory cortex, while layer 5 is more prominent in motor cortex[1] and provides motor control output (e.g., from V1 to superior colliculus) denoted here by out_M. Layer 2/3 integrates contextual inputs from outside the classical receptive field, and provides output, denoted by out_S, to other cortical areas that process 'higher-level' perceptual features.

The resemblances between the NN and LCC signal flows, and their respective inputs, suggest at least a rough and tentative correspondence between (a) NN layer R, and LCC layers 4 and 2/3; (b) NN layer Z, and LCC layer 6; (c) NN layers g and T, and LCC layer 5; (d) NN inputs y to layers R and Z, and LCC sensory inputs to layers 4 and 6, respectively; (e) motor outputs $\tilde{u}$ and out_M, with an efferent copy to NN layer Z for prediction of the future plant state; (f) the optimal estimate $\hat{y}$ and out_S; and (g) the g $\rightarrow$ R path of

KC learning (Fig. 2b), and observed LCC connections from layer 5 $\rightarrow$ 2/3 (not shown in D1 and D2).

We treat the g and T layers together since their role is limited to control, and since LCC layer 5 appears to be most prominent in motor control areas of cortex. We suggest that the Kalman NN uses one layer (R) in place of two (LCC layers 4 and 2/3) because Kalman estimation does not involve the learning of higher-level features (e.g., orientation selectivity in V1), and we expect that more sophisticated (e.g., more strongly nonlinear and context-sensitive) NN prediction methods may require an additional layer as in the LCC.

$\tilde{F}$ (used for prediction) and $\tilde{F}'$ (used for learning of control) join the same pair of NN layers in opposite directions, and are learned together during system identification. This suggests that a biological network performing Kalman-like P&C may use a corresponding pair of functional mappings that are (approximately) the transpose of one another.

For an alternative mapping, we modify Dgm. D1 to obtain D3:

$$\begin{tikzcd}
\mathrm{R} \arrow[r] & \mathrm{g} \arrow[r] \arrow[d] & \mathrm{T} \arrow[r] \arrow[d] & \mathrm{Z} \arrow[r] & \mathrm{R} \\
y \arrow[u] & (\tilde{F}\hat{y}) & \tilde{u} & y \arrow[u] & \textbf{[D3]}
\end{tikzcd}$$

Additional direct g $\rightarrow$ Z and Z $\rightarrow$ g paths (Fig. 2b, solid) are omitted from D3 for notational simplicity. KC learning adds paths T $\rightarrow$ g $\rightarrow$ R (Fig. 2b, dashed).

On this view, one can posit a one-to-one correspondence between NN layers {R, g, T, Z} and LCC layers {4, 2/3, 5, 6}. Then an exact match would imply the existence of LCC paths 4 $\rightarrow$ 2/3 $\rightarrow$ 5 $\rightarrow$ 6 $\rightarrow$ 4, 5 $\rightarrow$ 2/3 (used for KC learning in the NN), and 2/3 $\rightarrow$ 6, and with lower-area input to 4 and 6, output from 5 to a lower area, and output from 2/3 to the same or a higher area. All of these paths fit within current understanding of LCC connectivity[2, 3, 4, 29]. An exact match would, however, also imply LCC paths 6 $\rightarrow$ 2/3 $\rightarrow$ 4. The 6 $\rightarrow$ 2/3 path has been described[30], but in the context of V1 complex-cell interconnections. Also, NN layer g is used only for KC (and as a layer through which $\tilde{F}\hat{y}$ passes (Fig. 2)), arguing against identifying g with 2/3, which is important in sensory processing. I thus expect the less-detailed resemblance between Dgms. D1 and D2 to be more robust than that between D3 and D2, as more complex P&C tasks and different sets of allowed NN operations are studied in future work.

Conclusion

We have shown how optimal Kalman estimation and control are learned and executed by a neural network having simple computational elements. The circuit architecture required by this algorithm is found to be similar in significant ways to recurrent neural circuits in brain, and to LCC architecture in particular. These results suggest that the LCC may perform the core functions of estimation (including prediction) and control, with greater power than KF and KC (to discover higher-level perceptual features and to solve more complex control problems), but perhaps subsuming Kalman's solutions as simple linear cases. The NN-LCC similarities may help to guide LCC studies and, conversely, further analysis of LCC circuitry and function may lead to advances in nonlinear estimation and control for engineering applications.

Acknowledgments

I thank Drs. Stan Goldin, Geoff Grinstein, and Roger Traub for valuable comments.

Supporting Text

Derivation of the neural Kalman estimation algorithm

This section deals with three closely related points:

1. The meaning of a time-dependent expectation value in the Kalman solutions, and the ways in which we use a NN to approximate it;
2. Proof that Eq. 5 and Eqs. 2 are equivalent; and
3. Proof that $Z_t = E(\eta_t \eta'_t)$ and the η evolution Eq. 7 imply $Z_{t+1} = E(\eta_{t+1}\eta'_{t+1})$ in the large N_{gang} limit.

(1) Kalman filter and control theory makes extensive use of expectation values of the form $E(z_t v'_t)$, where z_t and v_t are column vectors describing plant, measurement, or computed variables at time t, and z and v may denote the same or different vectors. An expectation value is, by definition, a property of the distributions from which z_t and v_t are drawn. The distributions are in general time-varying. Thus, at any specified time t, $E(z_t v'_t)$ equals the limit, as $N_{\text{ens}} \to \infty$, of the ensemble average $\langle z_t(p)v'_t(p)\rangle \equiv (1/N_{\text{ens}})\Sigma_{p=1}^{N_{\text{ens}}} z_t(p)v'_t(p)$, where each $z_t(p)$ and $v_t(p)$ is independently drawn from its respective distribution. Ideally, one may think of each pair as being obtained from measurements on each of a sufficiently large number of different external systems at the same time, where the process parameters (F, H, noise covariances) are the same for each of the systems.

In practice, (a) one may be dealing with a single system in which some number $N_{\text{gang}} \geq 1$ of features are being tracked; (b) the features may not all be independent of each other (e.g., if they are part of the same rigidly moving object); and (c) if N_{gang} is small, one may need to sample over a set of recent times $(t - \Delta t, \ldots, t)$ in order to compute an approximation of $E(z_t v'_t)$, which will only be valid provided the distributions change slowly over time Δt. The present work does assume that features can be treated as independent of one another, each feature obeying the plant dynamics; otherwise, coupled features can be combined into a single feature vector, or else nonlinear interactions or constraints beyond the scope of this paper may need to be considered.

The NN algorithms that we derive can be used in different ways, as described in the main text and illustrated below (see *Numerical Example* and Fig. 3). If the 'ganging' option (see *NN operations*) is available, and N_{gang}

is sufficiently large, then the average over features p at time t can be used: $E(z_t v_t') \approx \langle z_t(p) v_t'(p) \rangle$. If N_{gang} is small (or if ganging is not available, in which case $N_{\text{gang}} = 1$), then a recency-weighted running average, using N_{gang} events at each of a set of past times, is used (green curve of Fig. 3). If the distributions have changed slowly over the time interval of the averaging, then the ensemble average (over all p and recent t) will approximate $E(z_t v_t')$. If the distributions are changing too rapidly, then the ensemble average will eventually converge to the optimal KF, but at a slower rate than the classical KF solution (blue curve of Fig. 3).

(2) Eq. 2 for K_t, and the definition $Z_t \equiv HP_t^- H' + R$, yield $I - HK_t = I - HP_t^- H' Z_t^{-1} = I - (Z_t - R) Z_t^{-1} = RZ_t^{-1}$. Thus Eq. 2 yields

$$\begin{aligned} Z_{t+1} &= HF(I - K_t H) P_t^- F' H' + HQH' + R \\ &= \tilde{F}(I - HK_t) HP_t^- H' \tilde{F}' H' + HQH' + R \\ &= \tilde{F} R Z_t^{-1} (Z_t - R) \tilde{F}' H' + HQH' + R \\ &= \tilde{F}(I - RZ_t^{-1}) R \tilde{F}' H' + HQH' + R \ , \end{aligned} \tag{11}$$

which is Eq. 5.

(3) The plant and measurement equations for x_{t+1} and y_t yield

$$y_{t+1} = \tilde{F} H x_t + \tilde{u}_t + H m_t + n_{t+1} \ ; \tag{12}$$

thus

$$\begin{aligned} \eta_{t+1} &= -y_{t+1} + \tilde{F} y_t + \tilde{F} R Z_t^{-1} \eta_t + \tilde{u}_t \\ &= -H m_t - n_{t+1} + \tilde{F} n_t + \tilde{F} R Z_t^{-1} \eta_t \ . \end{aligned} \tag{13}$$

Since (a) the noise terms m_t, n_t, and n_{t+1} are mutually independent and have zero mean; (b) η_t depends on n_t (through y_t) but not on m_t or n_{t+1}; (c) R and Z are symmetric matrices; and (d) $E(m_t m_t') = Q$, $E(n_t n_t') = E(n_{t+1} n_{t+1}') = R$, $E(\eta_t n_t') = -E(n_t n_t') = -R$, and $E(\eta_t \eta_t') = Z_t$; we obtain

$$\begin{aligned} E(\eta_{t+1} \eta_{t+1}') &= HE(m_t m_t') H' + E(n_{t+1} n_{t+1}') + \tilde{F} E(n_t n_t') \tilde{F} + \tilde{F} R Z_t^{-1} E(\eta_t n_t') \tilde{F}' \\ &\quad + \tilde{F} E(n_t \eta_t') Z_t^{-1} R \tilde{F}' + \tilde{F} R Z_t^{-1} E(\eta_t \eta_t') Z_t^{-1} R \tilde{F}' \\ &= HQH' + R + \tilde{F} R \tilde{F}' - \tilde{F} R Z_t^{-1} R \tilde{F}' \\ &= \tilde{F}(I - RZ_t^{-1}) R \tilde{F}' + HQH' + R \ , \end{aligned} \tag{14}$$

which equals Z_{t+1} by Eq. 5.

Numerical example

Numerical simulations illustrate that the results of the NN estimation (KF) algorithm agree with the classical KF matrix solution, apart from random fluctuations that arise because the NN estimates the covariance of η^- using a finite ensemble, whereas the classical algorithm is given the exact plant state covariance Q and uses matrix operations (not available to the NN) to compute the covariance of the estimation error. These fluctuations vanish in the limit of large ensemble size, which can be realized either by ganging the simultaneous learning of sufficiently many input features, and/or by using sufficiently many NN time steps to correspond to a single classical time step.

In Figure 3a, one of the components of the 2×2 matrix $(I-HK_t) \equiv RZ_t^{-1}$ is plotted vs. time step t, for four cases. All curves start with the same arbitrary $(I-HK)$.

1. Black curve: The classical Kalman Eqs. 2 are run for 7 time steps. Note that results are identical when the transformed matrix Eq. 5 is used in place of Eqs. 2.

2. Blue curve: The neural-network KF algorithm, which is given by Eqs. 6 and 7, shown in block diagram form in Fig. 1, and as a signal flow diagram in a layered neural network in Fig. 2a. In this case one feature (a measurement vector y) is tracked for 700 time steps; i.e., $N_{\text{gang}} = 1$. The time scale is compressed 100-fold.

3. Green curve: The same neural algorithm, but tracking 100 simultaneously tracked features for 7 time steps (i.e., $N_{\text{gang}} = 100$). The black, blue, and green curves all use the correct fixed value of $\tilde{F}$.

4. Red curve: Same as for green curve, but here $\tilde{F}$ is initially arbitrary and is learned from the measurement stream. Time values are left-shifted by one unit to allow startup time for learning $\tilde{F}$.

The external system (plant) and measurement processes are defined by the following parameters for this example. (See *Classical Kalman linear estimation and control* for definitions): F and H are 2-d rotations by 15^o and 50^o respectively, and the plant and noise covariance matrices are $Q = 10^{-5}I$ and $R = 10^{-4}I$ respectively, where I is the 2×2 identity matrix.

The learning rates are preferably time-varying for more efficient learning. Here, the learning rates used for Z and (for the last curve above) $\tilde{F}$ are adaptively adjusted using the method of Murata et al. [Murata, N., Müller, K.-R., Ziehe, A., & Amari, S.-i. *Adv. Neural Info. Proc. Systems* **9**, 599-605 (1997)]. In their notation, the values of the rate control parameters, which we have made no attempt to optimize, are $\{\alpha, \beta, \gamma, \delta\} = \{0.5, 30, 0.05, 0.1\}$ for Z, and $\{0.1, 3, 0.05, 0.04\}$ for $\tilde{F}$.

Numerical agreement between our NN KC algorithm and classical Kalman control (again, apart from random fluctuations) is also obtained in simulations (not shown).

Fig. 3b illustrates that the NN algorithm learns $\tilde{F}$ from noisy measurements alone. The learned $\tilde{F}$ was computed as part of the simulation that generated the last (red) curve in Fig. 3a. By way of contrast, classical KF assumes that F and H are given, or are learned separately using a system identification method.

Derivation of the neural Kalman control algorithm

The proof is similar to that for the neural KF algorithm. We show first (1) that Eq. 8 and Eqs. 3 are equivalent; then (2) that $T_t = E(w_t w_t')$ and the w evolution Eq. 9 imply $T_{\tau-1} = E(w_{\tau-1} w_{\tau-1}')$ in the large-ensemble limit.

(1) Since $T_\tau \equiv H'^{+} S_\tau H^{+} + \tilde{g}$, $S_\tau = H'(T_\tau - \tilde{g})H$. Using Eqs. 3 and the definitions of $\tilde{g}$ and $\tilde{r}$,

$$\begin{aligned} F' - L_\tau' B' &= F' - F'H'(T_\tau - \tilde{g})HB[B'H'(T_\tau - \tilde{g})HB + g]^{-1}B' \\ &= F' - F'H'(T_\tau - \tilde{g})T_\tau^{-1}H'^{+} \\ &= F' - F'H'(I - \tilde{g}T_\tau^{-1})H'^{+} \\ &= H'\tilde{F}'\tilde{g}T_\tau^{-1}H'^{+} \ . \end{aligned} \tag{15}$$

Thus

$$\begin{aligned} S_{\tau-1} &= (F' - L_\tau' B')S_\tau F + r \\ &= H'\tilde{F}'\tilde{g}T_\tau^{-1}H'^{+}S_\tau H^{+}HF + r \\ &= H'\tilde{F}'\tilde{g}(I - T_\tau^{-1}\tilde{g})\tilde{F}H + r \ ; \end{aligned} \tag{16}$$

so

$$\begin{aligned} T_{\tau-1} &= H'^{+}S_{\tau-1}H^{+} + \tilde{g} \\ &= \tilde{F}'\tilde{g}(I - T_\tau^{-1}\tilde{g})\tilde{F} + \tilde{r} + \tilde{g} \ , \end{aligned} \tag{17}$$

which is Eq. 8.

(2) The internally generated noise terms $\nu^g_{\tau-1}$, ν^g_τ, and ν^r_τ are mutually independent and have zero mean. The vector w_τ depends on ν^g_τ (by Eq. 9) but not on $\nu^g_{\tau-1}$ or ν^r_τ, which are both generated only after w_τ has been computed, since the iterative calculation of w proceeds in order of decreasing τ. Since $\tilde{g}$, $\tilde{r}$, and T_τ are symmetric matrices, and $E[\nu^g_\tau(\nu^g_\tau)'] = E[\nu^g_{\tau-1}(\nu^g_{\tau-1})'] = \tilde{g}$, $E[\nu^r_\tau(\nu^r_\tau)'] = \tilde{r}$, $E(\nu^g_\tau w'_\tau) = -\tilde{g}$, and $E(w_\tau w'_\tau) = T_\tau$, we obtain

$$\begin{aligned} E(w_{\tau-1}w'_{\tau-1}) &= E[\nu^g_{\tau-1}(\nu^g_{\tau-1})'] + E[\nu^r_\tau(\nu^r_\tau)'] + \tilde{F}'E[\nu^g_\tau(\nu^g_\tau)']\tilde{F} \\ &\quad + \tilde{F}'\tilde{g}T_\tau^{-1}E(w_\tau w'_\tau)T_\tau^{-1}\tilde{g}\tilde{F} \\ &\quad + \tilde{F}'E(\nu^g_\tau w'_\tau)T_\tau^{-1}\tilde{g}\tilde{F} + \tilde{F}'\tilde{g}T_\tau^{-1}E[w_\tau(\nu^g_\tau)']\tilde{F} \\ &= \tilde{g} + \tilde{r} + \tilde{F}'\tilde{g}\tilde{F} - \tilde{F}'\tilde{g}T_\tau^{-1}\tilde{g}\tilde{F} \ , \end{aligned} \tag{18}$$

which equals $T_{\tau-1}$ by Eq. 8.

Summary of operation of composite algorithm for KF and KC

The circuit of Fig. 1 operates switchably in several distinct modes, each using a portion of the circuit with cuts (breaks in signal flow) as described. At start-up: Learn $\tilde{F}$ and $\tilde{F}'$; this circuit mode has a cut at CF, with signal flow along solid path. Learn R using offline sensor operation (no external input); this mode uses solid path with cut at CR. Then, for each (increasing) t:

1. Perform KF learning & execution (solid path). If not also performing KC, the KF mode has a cut at Cu1 or Cu2. If performing KC:
    (a) If KC matrix for this t has been stored, retrieve it. Otherwise: learn $\tilde{g}$ (mode: dashed path, cut at Cg) if not already done; iterate τ from goal-completion time N backward to t (mode: dashed path, t held constant while τ is decremented); and optionally store intermediate KC matrices.
    (b) KC execution: Use KC matrix to calculate $\tilde{u}$ (mode: solid path), and provide efferent copy to KF.
2. Optionally update $\tilde{F}$ and $\tilde{F}'$ during KF operation (mode: solid path, no cut at CF).
3. Optionally update R (offline mode: CR cut, solid path).

Signal flows in the NN circuit of Fig. 2b

The following notes are intended to aid in the tracing of the signal flows (of Fig. 2a) through the NN circuit wiring diagram of Fig. 2b.

KF flow (without KC execution) starts with y input to the left circle of layer Z (denoted Z-left); result η is multiplied by Z^{-1} by passage through the Z or Z^{-1} connections (see text, Methods 1 and 2) to Z-right; result is conveyed to R-right, then is multiplied by R by passing through the R connections to R-left, where y is added; result $\hat{y}$ is multiplied by the $\tilde{F}$ connections from R-left to g-left; result $\tilde{F}\hat{y}$ is conveyed to Z-left, where the cycle repeats for the incremented value of t. [An external control term $\tilde{u}(\text{ext})$ if present, is added at Z-left (not shown).]

KC execution adds the computation: $\tilde{F}\hat{y}$ at g-left is multiplied by $\tilde{g}$, passing to g-right; result is sent to T-right and multiplied by T^{-1}, passing to T-left; here $\tilde{F}\hat{y}$ is subtracted, via the direct link from g-left to T-left, yielding $\tilde{u}$, which is sent as output and also (as an efferent copy) to Z-left, where it is added to the $\tilde{F}\hat{y}$ computed during KF flow.

Finally, KC learning (using dashed lines and the lateral connection matrices) proceeds from T-left (with subtractive input ν^g yielding activity w), to T-right (being multiplied by T^{-1}), thence to g-right; to g-left with multiplication by g; result receives additive input ν^g, is multiplied by $\tilde{F}'$ enroute to R-left; and passes to T-left, repeating the cycle for the decremented value of τ.

Comments on a recent KF-inspired NN algorithm

In a recent paper, Szirtes et al.[21] (hereafter referred to as 'SPL') observe: 'Connectionist representation of [Kalman filter-like] mechanisms with Hebbian learning rules has not yet been derived.' They state: 'The first problem of the classical [Kalman] solution is that covariance matrices of [the plant and measurement] noises are generally assumed to be known. The second problem is that ... the algorithm requires the calculation of a matrix inversion, which is hard to interpret in neurobiological terms. [Here] we derive an approximation of the Kalman gain, which eliminates these problems.'

In this section I show that the sequence of alterations made by SPL to the KF equations is not justified, and that the SPL simulations, which show a reduction of prediction error with time, do not provide evidence that their algorithm is in fact computing an approximation of the optimal Kalman gain.

SPL denotes the 'Kalman gain' matrix by K^t, which corresponds to our FK_t (cf. our Eq. 1 and their Eq. 3). At the outset, their stated goal is actually not to compute the Kalman-optimal K^t. Instead, they consider only a family of matrices, which I will denote $\tilde{K}(\theta)$, parametrized by a vector θ, for which the elements of $\tilde{K}(\theta)$ are $\tilde{K}_{ij}(\theta) \equiv K_{ij}\theta_i$ for each column i, where K is a given and fixed arbitrary matrix. SPL's goal is to adaptively learn θ over time, using a NN algorithm, so that its final learned value θ^* has the property that $\tilde{K}(\theta^*)$ minimizes the prediction error over the family of all possible $\tilde{K}(\theta)$. Since K is arbitrary, the parametrized family of matrices may not include (or even contain a matrix that approximates) the actual optimal KF. (Note that if K^t is an $N \times N$ matrix, the set of all possible K^t is being replaced by a family parametrized by θ, which has only N, not N^2, components.)

Given this goal, the first step in the SPL derivation is to minimize the prediction error by performing stochastic gradient descent. This yields a pair of equations for θ_k^{t+1} and an auxiliary matrix W_{ik}^{t+1}, in terms of values at time t:

$$\theta_k^{t+1} = \theta_k^t + \alpha \Sigma_{lj} K_{kl} H_{lj} W_{jk} \epsilon_k^t \ ; \tag{19}$$

$$W_{ik}^{t+1} = \Sigma_j F_{ij} W_{jk}^t - \theta_i^t \Sigma_{lj} K_{il} H_{lj} W_{jk}^t + \delta_{ik} \epsilon_k^t \ . \tag{20}$$

To obtain their model 'O1' (SPL Eqs. 5-6), SPL introduces a random vector ξ, which 'can be regarded as sparse, internally generated noise,' in order 'to provide a conventional neuronal equation.' In SPL Eq. 6, the first two right-hand terms are accordingly equal to ξ_k times the corresponding terms of the above Eq. 20. This multiplication by ξ_k is, however, not justified. If, for example, ξ has zero mean (actually the ξ distribution is nowhere specified), each of those two (now incorrect) terms will average to zero, and will thus make no contribution, on average, to W_{ik}^{t+1}. Therefore model 'O1' is not a valid approximation.

Several additional alterations are then made to generate a succession of SPL models called 'O2' through 'O5.'

To obtain model 'O2' (SPL Eqs. 7-8), SPL writes 'to simplify the complexity of the iteration, we may suppose that the system is near optimal: $K \approx H^{-1}$.' Since the K referred to here is the arbitrary and fixed K that enters the above definition of $\tilde{K}(\theta)$, one is free to choose $K \approx H^{-1}$, or even $K = H^{-1}$, irrespective of whether 'the system is near optimal.' However, doing so in no way ensures that the learned θ will yield a $\tilde{K}$ that is approximately Kalman-optimal. Certainly the optimal KF is not in general approximately equal to H^{-1}; it may be quite far from this value, depending upon F and the relationship between the plant and measurement noise covariances. This (as well as some of the following points) is most easily confirmed by considering the simple 1-d case, in which all matrices are scalar quantities.

To obtain model 'O3' (SPL Eqs. 9-10), SPL states that because only diagonal elements W_{kk} enter SPL Eq. 8 for the evolution of θ, therefore 'we may neglect the off-diagonal elements of matrix W in Eq. 7' (the evolution equation for W). This conclusion is not warranted, since such neglect will introduce errors into the *diagonal* elements of W, and thereby into the evolution of θ.

To obtain model 'O4' (SPL Eqs. 11-12), SPL writes that a 'further simplification neglects the self-excitatory contribution, $F_{ii}W^t_{ii}\xi_i$.' However, this term is not in general small compared with the remaining terms, so such neglect is unjustified as an approximation.

Finally, to obtain model 'O5' (SPL Eqs. 13-14), SPL introduces the 'stabilized form' of model 'O4.' This means that instead of setting W^{t+1} equal to a specified function $f(W^t)$, as in model 'O4,' SPL re-defines W^{t+1} as $W^{t+1} = W^t + \gamma f(W^t)$, where γ is a (presumably small) learning rate. At this step, it would have been more appropriate to define instead $W^{t+1} = (1-\gamma)W^t + \gamma f(W^t)$, so that if $W^{t+1} = W^t$ in model 'O4,' it would do so in model 'O5' as well.

Thus each of models 'O1' through 'O5' is obtained by making an alteration or simplification that is not a justified approximation to the previous model, nor to the original stochastic gradient form given by our Eqs. 19 and 20 above.

SPL then presents simulations (SPL Fig. 1) that show a rapid decrease in prediction error $\| x - \hat{x} \|$ (their $\hat{x}$ is our $\hat{x}^-$), and in the related 'reconstruction error', $\| y - H\hat{x} \|$, with time. This is stated to be a 'comparison of direct [i.e., classical solution] and approximated Kalman filters.' However, it can easily be seen that even if one uses an *arbitrary and fixed* blending matrix (K^t in SPL's Eq. 3) to combine the prediction from the previous time step with the current measurement, the prediction error starting with an initial arbitrary guess – prior to making any measurements on the system – will at first rapidly (exponentially) decrease as more measurements are made, provided only that K^t is such that the posterior estimate of x^t tends to move toward, rather than away from, $H^{-1}y^t$ (i.e., provided that K^t actually blends the new measurement with the prior prediction, rather than repelling the predicted $H\hat{x}$ away from the new measurement).

SPL does not display the $\tilde{K}(\theta)$ that is learned using their algorithm, so one cannot tell whether or how that matrix converges to an approximation of the optimal Kalman filter. The asymptotic prediction errors obtained using (a) the optimal KF, (b) an arbitrary fixed KF, and (c) a KF that varies according to an arbitrary algorithm, will, subject to the proviso above, typically differ by a modest factor as long as the plant and measurement SNRs differ by only 8 dB as in SPL's simulation. Thus the differences in asymptotic error are not visible on the scale of SPL's Fig. 1, in which initial errors are huge (simply because the initial prediction is a guess made in the absence of prior measurements) compared with the asymptotic errors using any of these blending matrices.

It is also worth noting that the learning rate $\alpha = 0.01$ is held constant in SPL's simulation (according to the SPL Fig. 1 caption). However, a rate factor that is appropriate initially, when prediction errors are huge, must be increased substantially as prediction errors decrease, in order to ensure that learning continues. This raises the possibility that SPL learning may have stalled early in the simulation, although one cannot tell without seeing the evolution of θ or $\tilde{K}(\theta)$ with time. Even if α had been adaptively altered to avoid such stalling, however, the above discussion shows that the claim of approximately optimal KF learning using SPL's 'O1' through 'O5' algorithms is not warranted, and that the comparison of predicted errors vs. time in SPL's Fig. 1 does not indicate that KF-like learning has occurred.

References

[1] Mountcastle, V. B. *Perceptual neuroscience: the cerebral cortex* (Harvard Univ. Press, 1998).

[2] Gilbert, C. D. *Ann. Rev. Neurosci.* **6**, 217-247 (1983).

[3] Douglas, R. J. & Martin, K. A. C. *Ann. Rev. Neurosci.* **27**, 419-51 (2004).

[4] Callaway, E. M. *Ann. Rev. Neurosci.* **21**, 47-74 (1998).

[5] Grossberg, S. & Williamson, J. R. *Cerebral Cortex* **11**, 37-58 (2001).

[6] Poggio, T. & Bizzi, E. *Nature* **431**, 768-74 (2004).

[7] Lewicki, M. S. & Sejnowski, T. J. *Adv. Neural Info. Proc. Systems* **9**, 529-35 (1996).

[8] Hinton, G. E. & Ghahramani, Z. *Phil. Trans. Roy. Soc. B* **352**, 1177-90 (1997).

[9] Rao, R. P. N. *Vision Research* **39**, 1963-89 (1999).

[10] Lee, T. S. & Mumford, D. *J. Opt. Soc. Am. A* **20**, 1434-48 (2003).

[11] Rao, R. P. N. *Neural Computation* **16**, 1-38 (2004).

[12] Zemel, R. S., Huys, Q. J. M., Natarajan, R., & Dayan, P. *Adv. Neural Info. Proc. Systems* **17**, 1609-16 (2005).

[13] Yu, A. J. & Dayan, P. *Adv. Neural Info. Proc. Systems* **17**, 1577-84 (2005).

[14] Rao, R. P. N. *NeuroReport* **16**, 1843-48 (2005).

[15] Todorov, E. *Neural Computation* **17**, 1084-1108 (2005).

[16] George, D. & Hawkins, J. *Proc. 2005 IEEE Int. Joint Conf. Neural Networks* **3**, 1812-17 (2005).

[17] Körding, K. P. & Wolpert, D. M. *Nature* **427**, 244-47 (2004).

[18] Kalman, R. E. *J. Basic Engrg.* **82**, 35-45 (1960).

[19] Bousquet, O., Balakrishnan, K., & Honavar, V. *Proc. Pacific Symp. on Biocomputing*, 655-66 (1998).

[20] Haykin, S. *Kalman Filtering and Neural Networks* (Wiley-Interscience, 2001).

[21] Szirtes, G., Póczos, B., & Lőrincz, A. *Neurocomputing* **65-66**, 349-355 (2005).

[22] Szita, I. & Lőrincz, A. *Neural Computation* **16**, 491-499 (2004).

[23] Hertz, J., Krogh, A., & Palmer, R. G. *Introduction to the Theory of Neural Computation* (Addison-Wesley, 1991).

[24] Haykin, S. *Neural Networks: A Comprehensive Foundation* (2nd ed., Prentice Hall, 1999).

[25] Becker, S. & Hinton, G. E. *Nature* **355**, 161-163 (1992).

[26] Linsker, R. *Neural Computation* **4**, 691-702 (1992).

[27] Linsker, R. *Neural Networks* **18**, 261-265 (2005).

[28] Rieke, F., Warland, D., de Ruyter van Steveninck, R., & Bialek, W. *Spikes: Exploring the Neural Code* (MIT Press, 1999).

[29] Raizada, R. D. S. & Grossberg, S. *Visual Cognition* **8**, 431-466 (2001); see refs. cited in Table 1.

[30] Hirsch, J. A., Gallagher, C. A., Alonso, J.-M., & Martinez, L. M. *J. Neurosci.* **18**, 8086-94 (1998).

Figure captions

Figure 1. Block diagram of NN algorithm. Each ◯ block combines inputs; arrowhead inputs are added, filled-circle inputs are subtracted. Each $\otimes$ block multiplies its input by the matrix indicated alongside the block. Arrows through blocks denote a matrix to be learned. The solid path implements KF learning and execution, and KC execution; the dashed path, KC learning. Link symbols starting with C and L denote where signal flow is cut, and where signals are used for matrix learning, during specific modes of operation (see text).

Figure 2. Layered organization of signal flow and circuitry required by NN algorithm.

(a) Signal flow shown in four layers. Upper set (solid links) shows KF learning and execution, and KC execution, at time t; lower set (dashed links) shows KC learning at time index τ. All open circles within a single layer denote the same set of nodes in that layer, at each of many computing steps. Computation proceeds from left to right along the links (t and τ values are unchanged). At extreme right, t is incremented and τ decremented, and signal flow resumes at extreme left. Links labeled by a matrix denote that the activity vector at that link is multiplied by that matrix. Other link labels and arrow symbols are as in Fig. 1.

(b) NN circuitry required to enable this signal flow. Within each layer, the pair of open circles denotes the set of nodes, and the labeled line joining them denotes the weight matrix (or its inverse) of its lateral connections. Arrows and symbols as in (a).

Figure 3. (a) Example of classical and neural Kalman filter learning. The (2,2) component of the 2×2 matrix $(I - HK_t)$ is plotted vs. time step t. See *Supporting text* for details.

(b) Learning of (2,2) component of $\tilde{F}$ vs. t, starting with arbitrary matrix. Horizontal line denotes correct value.

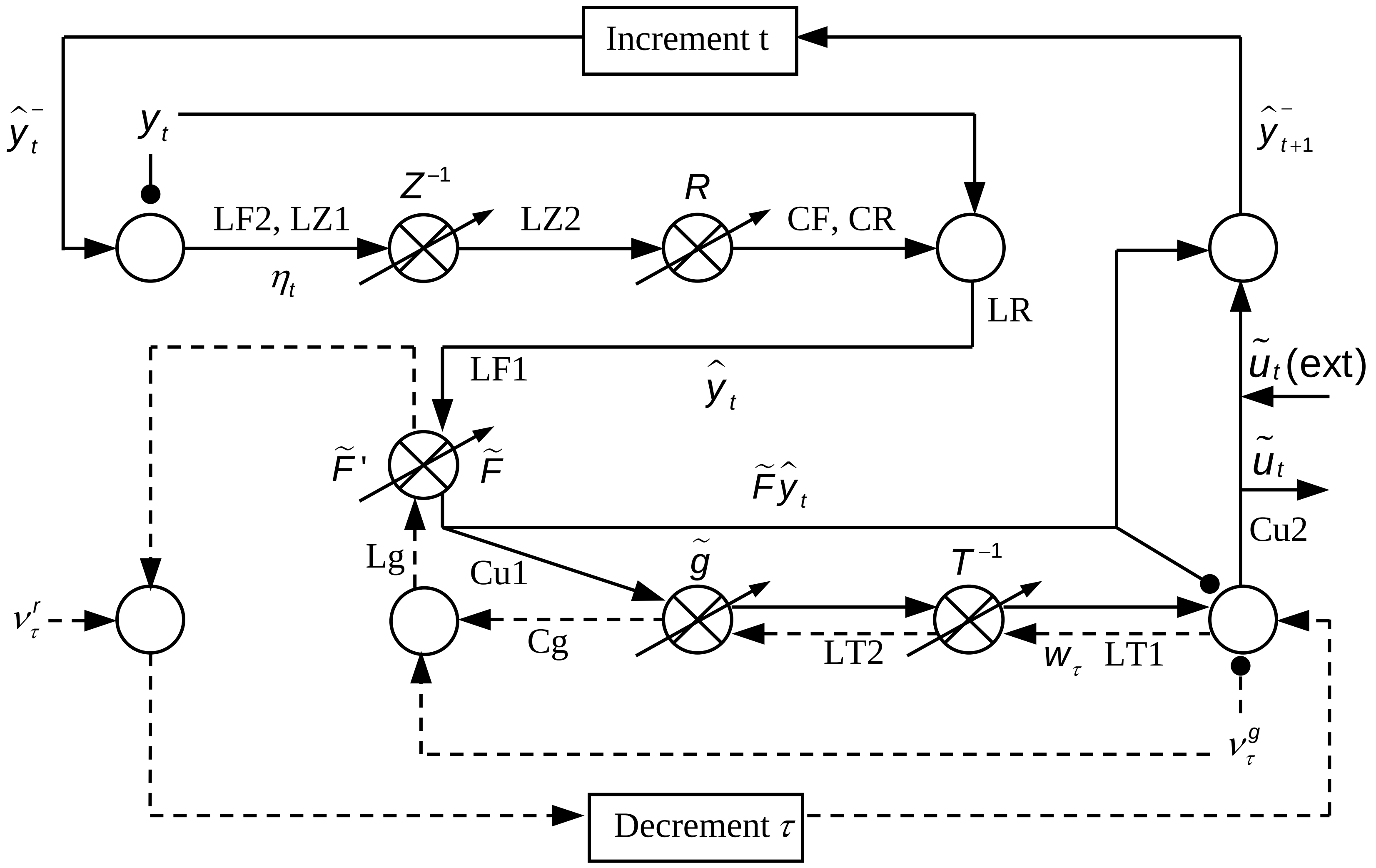

Increment t
$\hat{y}_t^-$
y_t
LF2, LZ1
η_t
Z^{-1}
LZ2
R
CF, CR
LR
$\hat{y}_{t+1}^-$
LF1
$\hat{y}_t$
$\tilde{u}_t$ (ext)
$\tilde{u}_t$
$\tilde{F}'$
$\tilde{F}$
$\tilde{F}\hat{y}_t$
Cu2
Lg
Cu1
$\tilde{g}$
T^{-1}
ν_τ^r
Cg
LT2
w_τ
LT1
ν_τ^g
Decrement τ

Fig. 1

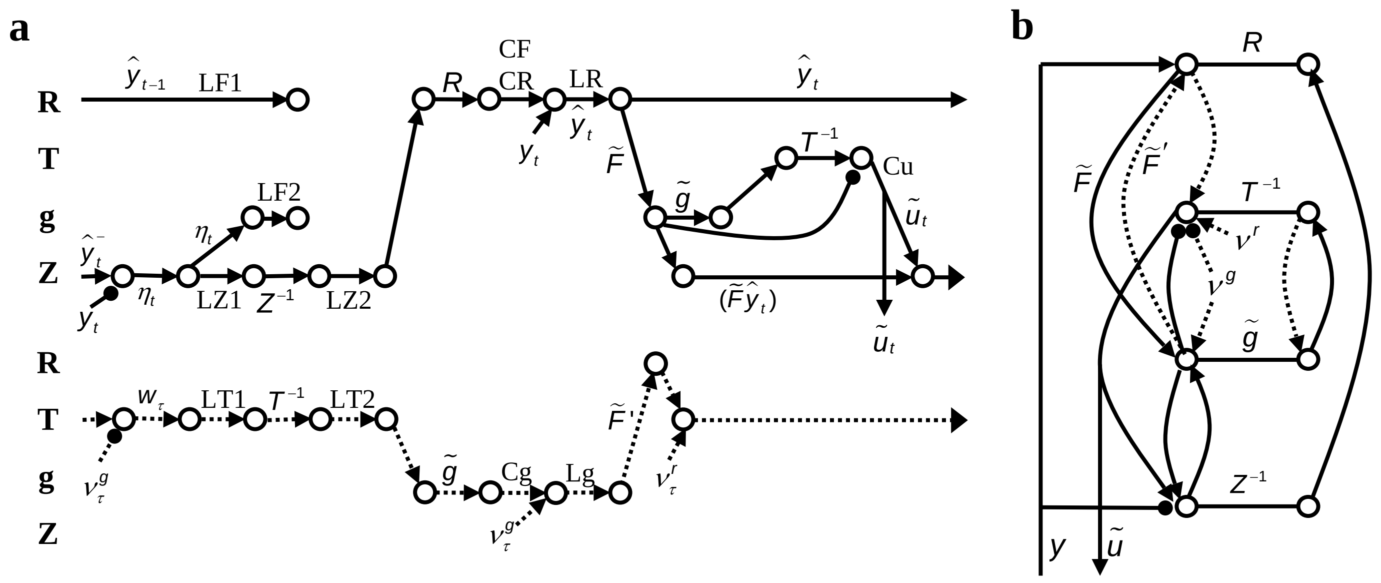
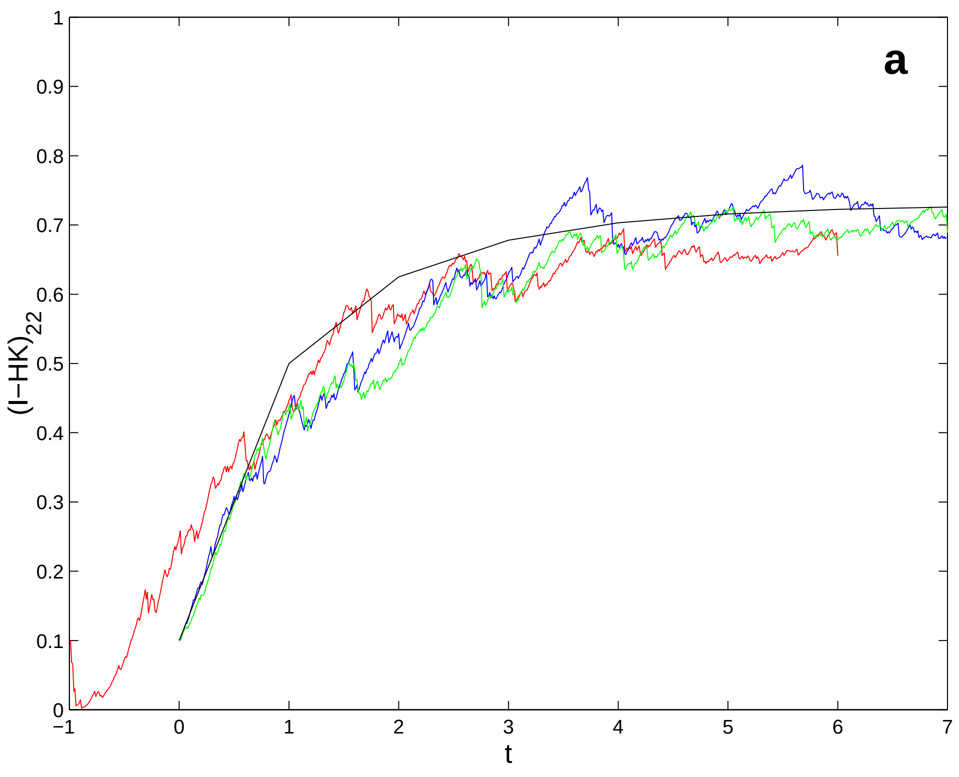
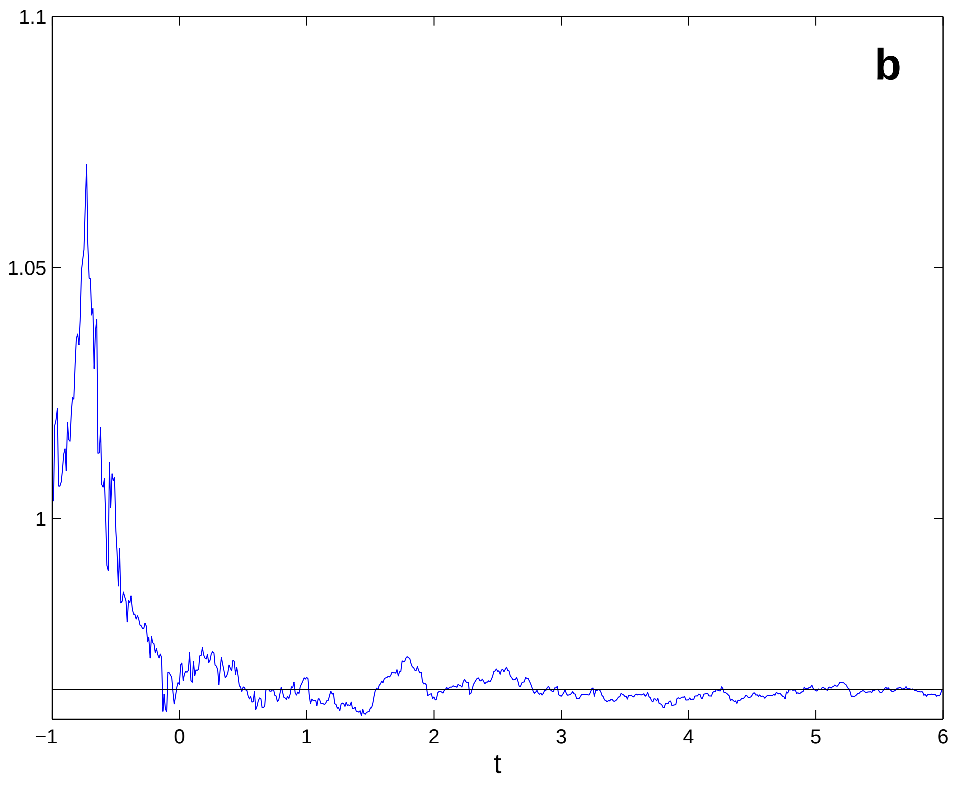

a
R
T
g
Z
R
T
g
Z
$\hat{y}_{t-1}$
LF1
R
CF
CR
LR
$\hat{y}_t$
y_t
$\hat{y}_t$
$\tilde{F}$
$\tilde{g}$
T^{-1}
Cu
$\tilde{u}_t$
LF2
η_t
$\hat{y}_t^-$
η_t
LZ1
Z^{-1}
LZ2
y_t
$(\tilde{F}\hat{y}_t)$
$\tilde{u}_t$
w_τ
LT1
T^{-1}
LT2
ν_τ^g
$\tilde{g}$
Cg
Lg
$\tilde{F}'$
ν_τ^r
ν_τ^g
b
R
$\tilde{F}$
$\tilde{F}'$
T^{-1}
ν^r
ν^g
$\tilde{g}$
Z^{-1}
y
$\tilde{u}$

Fig. 2



a
(I−HK)$_{22}$
t
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−1
0
1
2
3
4
5
6
7



b
t
1.1
1.05
1
−1
0
1
2
3
4
5
6

Fig. 3