Linear Affine SDDEs: Moment Equations, Method of Steps, and a Julia Implementation

January 22, 2026

Abstract

We derive and implement deterministic differential–delay equations (DDEs) for the first and second moments of a linear, affine stochastic delay differential equation (SDDE) with affine multiplicative noise. The result is a closed system for the mean, the second (raw) moment, and a ladder of cross-moments at integer multiples of the delay. We then show how to integrate these DDEs in `Julia` using the `DifferentialEquations.jl` ecosystem, covering practical details such as state layout, absolute-time history access, time-varying coefficients, and validation in a scalar test case with known behavior.

1 Model

Consider the n-dimensional SDDE

$$\mathrm{d}x(t) = \big(A(t)x(t) + B(t)x(t-\tau) + c(t)\big)\,\mathrm{d}t \;+\; \big(\alpha(t)x(t) + \beta(t)x(t-\tau) + \gamma(t)\big)\,\mathrm{d}W_t, \qquad (1)$$

where $A(t), B(t), \alpha(t), \beta(t) \in \mathbb{R}^{n\times n}$, $c(t), \gamma(t) \in \mathbb{R}^n$, $\tau > 0$, and W_t is a scalar Wiener process.[1] We assume an initial segment (history) $x(s) = \phi(s)$ for $s \in [-\tau, 0]$ (deterministic unless stated otherwise).

Define the mean $\mu(t) = \mathbb{E}[x(t)]$, the raw second moment $M(t) = \mathbb{E}[x(t)x(t)^\top]$, and for $k \geq 1$ the cross-moments

$$S_k(t) \;=\; \mathbb{E}\Big[x(t)\,x\big(t-k\tau\big)^\top\Big]. \qquad (2)$$

2 Moment equations

Taking expectations of (**??**) gives the closed mean DDE

$$\dot{\mu}(t) = A(t)\,\mu(t) + B(t)\,\mu(t-\tau) + c(t). \qquad (3)$$

The Itô's product rule simply stats that taking derivatives has to include the quadratic variation term.

$$\mathrm{d}x(t)x(t)^\top = \mathrm{d}x(t)x(t)^\top + x(t)\mathrm{d}x(t)^\top + \mathrm{d}x(t)\mathrm{d}x(t)^\top\,. \qquad (4)$$

So, for example if you have a system like this: $\mathrm{d}x(t) = fx(t)\mathrm{d}t + gx(t)\mathrm{d}W_t$, then the quadratic variation term is $gx(t)gx(t)^\top\mathrm{d}t$.

$$\mathrm{d}x(t)x(t)^\top = f(t)x(t)^\top\mathrm{d}t + x(t)^\top f(t)\mathrm{d}t + g(t)g(t)^\top\mathrm{d}W_t^2. \qquad (5)$$

And we take the expectation of both sides and devide formally by $\mathrm{d}t$ to get the moment equation for the second moment.

[1] For m-dimensional W_t with covariance Q, replace $LL^\top$ by $LQL^\top$ throughout.

For the second moment, apply Itô's product rule to $x(t)x(t)^\top$ and note that the quadratic variation contributes $L(t)L(t)^\top$ with $L(t) = \alpha(t)x(t) + \beta(t)x(t-\tau) + \gamma(t)$:

$$\begin{aligned}\dot{M}(t) = A(t)M(t) + M(t)A(t)^\top + B(t)M(t-\tau) + S_1(t)B(t)^\top + c(t)\,\mu(t)^\top + \mu(t)\,c(t)^\top \\ + \mathbb{E}\Big[L(t)L(t)^\top\Big],\end{aligned} \tag{6}$$

and a standard expansion yields

$$\begin{aligned}\mathbb{E}\Big[LL^\top\Big] &= \alpha(t)M(t)\alpha(t)^\top + \beta(t)M(t-\tau)\beta(t)^\top + \alpha(t)S_1(t)\beta(t)^\top + \beta(t)S_1(t)^\top\alpha(t)^\top \\ &+ \alpha(t)\mu(t)\,\gamma(t)^\top + \gamma(t)\mu(t)^\top\alpha(t)^\top + \beta(t)\mu(t-\tau)\,\gamma(t)^\top + \gamma(t)\mu(t-\tau)^\top\beta(t)^\top + \gamma(t)\gamma(t)^\top,\end{aligned} \tag{7}$$

For cross-moments there is no Itô correction because $x(t-k\tau)$ has no differential at time t. This essentially means that when we apply the Itoˆs product rule to $x(t)x(t-\tau)^\top$, we get the following:

$$\mathrm{d}x(t)x(t-\tau)^\top = f(t)x(t-\tau)^\top\mathrm{d}t + x(t)^\top f(t-\tau)\mathrm{d}t + g(t)g(t-\tau)^\top\mathrm{d}W_t\mathrm{d}W_{t-\tau}. \tag{8}$$

The expectation of the product of the two non-overlapping Wiener increments is zero, i.e. $\mathbb{E}[\mathrm{d}W_t\mathrm{d}W_{t-k\tau}] = 0$ because they are independent.

For $S_1(t)$ one gets

$$\dot{S}_1(t) = A(t)S_1(t) + S_1(t)A(t-\tau)^\top + B(t)M(t-\tau) + S_2(t)B(t-\tau)^\top + c(t)\,\mu(t-\tau)^\top + \mu(t)\,c(t-\tau)^\top. \tag{9}$$

$$\dot{S}_2(t) = A(t)S_2(t) + S_2(t)A(t-2\tau)^\top + B(t)S_1(t-\tau) + S_3(t)B(t-2\tau)^\top + c(t)\,\mu(t-2\tau)^\top + \mu(t)\,c(t-2\tau)^\top. \tag{10}$$

In general, for $k \geq 1$,

$$\boxed{\dot{S}_k(t) = A(t)S_k(t) + S_k(t)A(t-k\tau)^\top + B(t)\,S_{k-1}(t-\tau) + S_{k+1}(t)B(t-k\tau)^\top + c(t)\,\mu(t-k\tau)^\top + \mu(t)\,c(t-k} \tag{11}$$

with the convention $S_0(t) = M(t)$ and $S_{-1}(t) \equiv M(t-\tau)$ when $k = 1$.

Covariance form. Let $P(t) = M(t) - \mu(t)\mu(t)^\top$ and $R_k(t) = S_k(t) - \mu(t)\mu(t-k\tau)^\top$. One can rewrite (**??**) for (P, R_k), but for implementation it is simpler to evolve raw moments and form covariances a posteriori.

Steady state (delay Lyapunov). If the system is mean-square stable, cross-covariances $S(\theta) = \lim_{t\to\infty}\mathbb{E}[x(t)x(t-\theta)^\top]$ satisfy $\mathrm{d}S/\mathrm{d}\theta = AS$ for $\theta \in (0,\tau)$ with $S(0) = P$, yielding $S(\tau) = e^{A\tau}P$ and the algebraic *delay Lyapunov* equation

$$AP + PA^\top + BS(\tau)^\top + S(\tau)B^\top + \Gamma = 0, \qquad \Gamma = \mathbb{E}[LL^\top]_{\mathrm{ss}}. \tag{12}$$

3 Method of steps and closure on a finite horizon

On $[0,\tau]$, all delayed quantities (e.g. $M(t-\tau)$, $\mu(t-\tau)$, $S_k(t-\tau)$) depend only on the history and are therefore known functions. Hence (**??**) form a closed linear ODE in time. On each subsequent interval $[m\tau, (m+1)\tau]$, the delayed terms are known from the previously computed solution, so again we solve a closed ODE. For a finite horizon $[0,T]$, choosing $K = \lfloor T/\tau \rfloor$ ensures $t - (K+1)\tau < 0$ for all $t \in [0,T]$, so the highest cross-moment that appears, $S_{K+1}(t)$, multiplies the *history* only and is known:

$$S_{K+1}(t) \;=\; \mu(t)\,\phi\big(t - (K+1)\tau\big)^\top \quad \text{(deterministic history)}. \tag{13}$$

4 Implementation in Julia

We integrate the moment DDEs using `DifferentialEquations.jl`. The state is

$$y(t) = \left[\ \mu(t);\ \operatorname{vec}(M(t));\ \operatorname{vec}(S_1(t));\ \ldots;\ \operatorname{vec}(S_K(t))\ \right].$$

We define a layout to pack/unpack μ, M, and $\{S_k\}$. The RHS evaluates the coefficient matrices/vectors (possibly time-varying) at the current t, uses the solver-supplied *absolute-time* history accessor $h(p, s)$ to fetch past states at $s = t - \tau$ and $s = t - k\tau$, and forms the derivatives from (**??**).

Absolute-time history. In the `DDEProblem` API, the history accessor is *absolute time*: calling `h(p, s)` returns the state at time s (for $s \leq 0$ the user-provided history; for $s > 0$ the previously integrated/interpolated solution). This is why we use `h(p, t - lag)` rather than offsets.

Core code (excerpt)

```
# coefficients can be constants or functions of time
mat_at(M, t) = M isa Function ? M(t) : M
vec_at(v, t) = v isa Function ? v(t) : v

function E_LL(t, t, t, , , M, C, N)
    t*M*t' + t*N*t' + t*C*t' + t*C'*t' +
    t**t' + t*'*t' + t**t' + t*'*t' + t*t'
end

function mom_rhs!(dy, y, h, p, t)
    A = mat_at(p.A,t); B = mat_at(p.B,t)
     = mat_at(p.,t);  = mat_at(p.,t)
    c = vec_at(p.c,t);  = vec_at(p.,t)
    , M, S = unpack_state(y, p.layout)

    # past states at absolute times
    , M, S = unpack_state(h(p, t - p.), p.layout)
    edge, _, _ = unpack_state(h(p, t - (p.K+1)*p.), p.layout)
    S_Kp1 =  * edge'

    C = (p.K>=1) ? S[1] : zeros(p.layout.n, p.layout.n)

    d = A* + B* + c
    dM = A*M + M*A' + B*M + C*B' + c*' + *c' +
         E_LL(, , , , , M, C, M)

    for k in 1:p.K
        Sk = S[k]
        Skm1_delay = (k==1) ? M : S[k-1]
        Skp1 = (k<p.K) ? S[k+1] : S_Kp1
        km, _, _ = unpack_state(h(p, t - k*p.), p.layout)
        dSk = A*Sk + Sk*A' + B*Skm1_delay + Skp1*B' + c*km' + *c'
        dy[p.layout.idx_S[k]] .= vec(dSk)
    end

    dy[p.layout.idx_] .= d
    dy[p.layout.idx_M] .= vec(dM)
end
```

The full implementation (packing, history builder, driver, etc.) matches the code you now use. Coefficients may be either constants or functions $t \mapsto$ matrix/vector, and the history is supplied by a user function $\phi(t)$ returning the deterministic initial state when $t \le 0$.

5 Validation on a scalar test

Take $n = 1$ with $A = -6$, $B = 0$, $\alpha = 0$, $\beta = 2$, $\gamma = 1$, $\tau = 1$, and deterministic history $\phi \equiv 3$. On $[0, 1]$,

$$\begin{aligned} \dot{M} &= -12M + \underbrace{\big(4M(t-\tau) + 4\mu(t-\tau) + 1\big)}_{\mathbb{E}[(\beta x_{-\tau}+\gamma)^2]} = -12M + 49, \\ \mu(t) &= 3e^{-6t}, \qquad M(0) = 9. \end{aligned}$$

Therefore the variance on the first step is

$$\mathrm{Var}(t) = M(t) - \mu(t)^2 = \frac{49}{12}\big(1 - e^{-12t}\big), \qquad 0 \le t \le 1,$$

which rises from 0 to ≈ 4.083 by $t = 1$, and then begins to bend as the delayed inputs $M(t-1)$ and $\mu(t-1)$ decay. The numerical integration matches this shape when the absolute-time history access is used consistently in the RHS.

6 Extensions

Multi-dimensional noise. If W is m-dimensional with covariance Q (possibly time-varying), the only change is

$$\mathbb{E}[LL^\top] \;\longrightarrow\; \mathbb{E}[LQL^\top]$$

in (**??**), i.e. insert Q between the left/right factors in each bilinear term.

Random history. If the history is random with known first/second moments and independent of future noise, build the history state $y(t)$ for $t \le 0$ from those moments instead of $\phi\phi^\top$. In that case $S_{K+1}(t)$ should be computed via the history accessor to use the correct cross-moment with the random history.

Covariances directly. One can evolve P and R_k directly; evolving raw moments tends to be simpler and numerically robust, with covariances formed as $P = M - \mu\mu^\top$ and $R_k = S_k - \mu\,\mu(\cdot)^\top$ for diagnostics.

7 Usage outline

```
# Define A,B,, as constants or functions t->Matrix; c, as vectors or t->Vector
n = 2; A(t) = -0.8I(n); B(t) = 0.1I(n); (t) = 0.3I(n); (t) = 0.1I(n)
c(t) = zeros(n); (t) = [0.2, 0.0]
 = 0.7; T = 5.0
(t) = [1.0, 0.0] # deterministic history for t  0

sol, L = solve_moments(A,B,c,,,; =, T=T, =, saveat=0:0.05:T)
t, Mt, St = get_moments_at(sol, L, 3.0) # read moments at t=3.0
```

8 Conclusion

The linear–affine SDDE (**??**) admits a closed deterministic DDE system for the mean, second moment, and a finite ladder of cross-moments on any finite horizon. Implemented with the method of steps and absolute-time history access, the approach is efficient and stable in practice, and readily accommodates time-varying coefficients and multi-dimensional noise.