Chapter 6: Models

Before You Start

What you are assumed to know. You write software for a living, you read a function signature and know what its arguments and return value commit it to, and you remember basic probability from a first course: what a probability is, what "given" means, and that a distribution is a spread of values rather than one value. You have read the previous five chapters. From Chapter 3 you have three words, recalled here in one clause each: a **model** is a description of how something behaves, and it does not run; a **simulation** is one execution of a model over time from a starting state; a **simulator** is the software that performs simulations. From Chapter 4 you have the seven building blocks, in particular the **models** block, which that chapter gave a slot behind an interface – given a state and a horizon, return a predicted state – and the **state** block, the twin's current belief about the physical twin. From Chapter 5 you have the tuple, the sampling interval T_s and the event time, and the offshore turbine whose generator-side bearing temperature ran through that chapter as its worked stream. From Chapter 2 you have the plant rig's four equations and its dose rule, which reappear once, in the one section of this chapter where the rig is allowed.

What you are not assumed to know. Calculus, linear algebra, numerical methods, any modelling tool and any control theory. Every result this chapter leans on from those subjects is recalled in one clause and pointed at the section of the mathematics appendix that builds it from a loop, a record or a table: *Quantities, Rates and Equations* for what a rate is and what an equation with a rate in it says; *From an Equation to a Loop* for Euler's step and what a solver does; *Many Quantities at Once* for vectors, matrices, the state-space form and modes; *Probability and Distributions* for Bayes' rule and the Gaussian; *State Machines with Probabilities* for the Markov chain and the hidden state seen through a noisy sensor; and *What a Solver and a Simulator Do* for the event queue and the real-time factor. If a paragraph below feels like it is skipping a step, the appendix section named beside it is where the step is.

What this chapter covers. Chapter 4 named the model as one of seven blocks and gave it a slot: given a state and a horizon, return a predicted state. This chapter is about what goes in that slot. It first says what every model has – a state, an input, a parameter set, a step function and an output function – and gives the four criteria a model inside a twin is judged on. It then introduces one running example, an offshore wind turbine, and models it five ways, once per **model type**: a lumped first-principles model of the rotor and the gearbox oil; a state machine of its operating modes and a discrete-event model of the vessel that repairs it; a finite-element model of its tower and blades; a reduced and a surrogate version of that model; and a probabilistic model of a wearing bearing, updated by Bayes' rule. Each model type closes on the same two sub-headings, what it costs and how it is kept current, and one section then generalises the second of those into the three rates at which a twin's model changes. A choosing table, the rig, five faded examples, a summary, exercises and further reading close the chapter.

What this chapter deliberately does not cover. How a solver chooses its step, which belongs to the simulators half of this part and to the appendix, and co-simulation, which belongs with the interface standards it is built on to the standards chapter. State estimation – how the twin corrects the model's state from readings – which is the state and estimator chapter's; this chapter hands it the discrete state-space form and stops. Neural networks,

hybrid physics-plus-network models, generative models and foundation models, which are parked for the AI chapter and named here only as parked. Whether a model deserves to be believed, which is the credibility chapter's. Structural change to the model set, which belongs to configuration and the manager chapter.

Learning objectives. By the end of this chapter you will be able to:

1. **Name** the state, inputs, outputs and parameters of a described system and **write** its step function.
2. **Classify** a modelling need onto the five model types and **defend** the choice against the four criteria.
3. **Turn** a rate rule into a loop and **say** what the step size costs.
4. **Explain** in words what a mesh, a mode and a reduced basis are and **judge** when each is worth paying for.
5. **Write** a degradation model as a random process and **update** one of its parameters from a reading with Bayes' rule.
6. **State** the three rates at which a twin's model changes and **name** which chapter owns each.

Every section that follows serves at least one of those six, and the exercises at the end say which.

What Every Model Has

Every model in this chapter, whatever its physics and whatever its size, is the same five things to a software engineer. Figure 1 draws them, and this section names them, because once the five are named the five model types that follow are five ways of writing one of them.

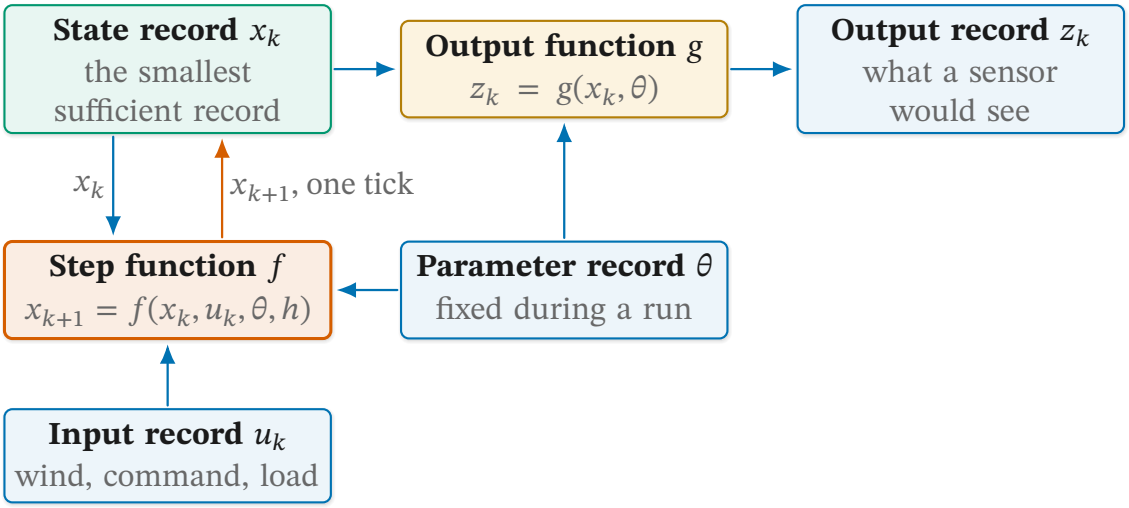


Figure 1: The anatomy of a model. Four records and two functions: the step function advances the state record by one interval from the input and the parameters, and the output function reads from the state what a sensor would report.

Four records and two functions

The state record x is the smallest set of numbers that, together with everything done to the system from now on, fixes everything the system will do from now on. That definition is worth reading twice, because it is a test, not a description. A turbine's rotor speed is state: if you know it now and you know the wind and the generator command from now on, you know the speed from now on. Last week's mean wind speed is not state: knowing it adds nothing once the present speed is known. The word is the same as Chapter 4's **state** block, and the relation is exact: the model's state is what the twin's state block holds an estimate of.

The input record u is what is done to the system from outside during a run – the wind, a command from the grid operator, a load on a bridge, a dose of water. The model reads inputs; it never predicts them. A quantity the model predicts is state; a quantity the model is told is input. The same physical thing can be either, depending on where the model's boundary is drawn: the wind is an input to a turbine model and a state of a weather model.

The parameter record θ holds the numbers that stay fixed during one run and may change between runs – an inertia, a resistance, a heat-transfer coefficient, a wear rate. The distinction from state is one of time scale and of who changes them: a solver changes the state every step; an engineer, or later in this chapter a Bayesian update, changes a parameter between runs. Chapter 2's drying rate d and dose response g were parameters in exactly this sense, and the whole of the Keeping a Model Current section is about how parameters move.

The step function f takes the state, the input and an interval h and returns the state one interval later:

$$x_{k+1} = f(x_k, u_k, \theta, h) \tag{1}$$

Equation 1 is the whole of what a simulator calls, in a loop, to produce a simulation. Chapter 3 said a model does not run and a simulation is one execution of it; the step function is the model's half of that bargain, and the loop is the simulator's.

The output function g reads from the state what a sensor would report:

$$z_k = g(x_k, \theta) \tag{2}$$

The output z in Equation 2 is a separate record from the state because a sensor rarely reads the state directly. An accelerometer on a tower reports an acceleration, not the tower's displacement; a thermocouple reports the temperature at one point of a gearbox, not the temperature of the oil as a whole. The output function is where the model says what its state would look like through the sensors Chapter 5 described, and it is the half the state and estimator chapter will lean on hardest: a residual is z predicted minus z measured.

Table 1 collects the five with the turbine's rotor as the instance, so that each has a concrete face before the sections that follow give it several.

Table 1: The five parts of a model, with the turbine's rotor as the instance. The right-hand column is the test that separates state from parameter: what changes it, and how often.

Part	Written as	For the turbine's rotor	Who changes it
state record	x	rotor speed	the step, every interval
input record	u	wind speed, generator command	the world, every interval
parameter record	θ	rotor inertia, blade geometry	an engineer, or an update, between runs
step function	$x_{k+1} = f(x_k, u_k, \theta, h)$	new speed from torque imbalance and inertia	never during a run
output function	$z_k = g(x_k, \theta)$	the speed the SCADA counter reports	never during a run

Three notions of time, three signatures for the step

The interval h in the step function hides a choice. There are three ways a model can think about time, and they give the step three different signatures.

Continuous time. The model says how fast each state is changing right now – a **rate rule**, this chapter's plain word for what the mathematics appendix builds under *Quantities, Rates and Equations* and calls an ordinary differential equation (ODE). A rate rule has no step in it; a solver supplies one, and the appendix's *From an Equation to a Loop* shows Euler's step doing so: new value equals old value plus rate times step. The step function of a continuous-time model is therefore "call the solver for h seconds", and h is the solver's choice as much as the modeller's.

Sampled time. The model is written directly as a rule from this sample to the next, at a fixed interval – the difference equation of the appendix. Chapter 2's dose rule is one: it reads the moisture every T_c hours and never asks what happened in between. Here h is fixed by the sensor or the controller and the step function is exactly Equation 1 with h constant.

Event-driven time. Nothing happens until something happens. The model's step is "advance the clock to the time of the next event and apply it", and h is whatever gap the events leave, different every step. The appendix's *What a Solver and a Simulator Do* runs this step from a priority queue, and the Discrete-State and Discrete-Event Models section builds a model that lives in it.

The three are not rivals. A twin of a turbine holds a continuous-time model of the rotor, a sampled-time model of the ten-minute SCADA statistics and an event-driven model of the vessel that comes when a gearbox fails, and the simulators half of this part is largely about making the three tick together.

Deterministic or stochastic

A **deterministic** step returns one next state. A **stochastic** step returns a distribution over next states – or, in code, one draw from it, with the understanding that a different draw

was possible. The difference is not in the physics but in what the model admits it does not know. A wearing bearing wears at a rate nobody can pin down to three decimal places, and a model that returns one wear level next month is overstating what it knows; a model that returns a spread is not. The Probabilistic Models and Bayesian Updating section is where the step starts returning distributions, and the fourth criterion below is why.

White, grey and black box

The literature sorts models by how much of the physics is written in them. A **white-box** model is written from physical laws – first principles such as Newton's second law, closed with constitutive relations for friction, damping and boundaries; a **black-box** model is fitted from measured data with no law inside; and a **grey-box** model sits between, with the physics guiding a structure whose parameters, or whose correction terms, are fitted from data [1]. The structural-health-monitoring literature reads the digital twin itself as the middle position: the model-based approach is the white box, the vibration-based approach the black box, and the twin stands between them [2]. The first three model types in this chapter are white-box models that become grey the moment a parameter is fitted to this asset, which every one of them is. The black end of the spectrum – a network trained to map inputs to outputs – is parked for the AI chapter, and the Reduced-Order and Surrogate Models section says what a fitted function may and may not stand in for.

The four criteria

A model can be a good model and a bad model for a twin. What makes the difference is not accuracy alone but four properties, and every model type in this chapter is judged against the same four. They are the book's own framing, distilled from what the published twins in the sections below had to get right.

1. **Evaluable from this asset's state with today's readings.** The model's inputs must be things the twin actually has: the SCADA stream, the accelerometers, the bearing temperature Chapter 5 carried. A model that needs a wind field measured at fifty points along the blade, when the twin has one anemometer, is a good model of a turbine and a useless model for this twin.
2. **Updatable.** When the asset changes – wears, is repaired, is re-commissioned – the model must be able to follow. A model whose parameters are baked into a compiled binary nobody can rebuild fails this criterion however accurate it was on delivery.
3. **Inside the decision's time budget.** Chapter 1 priced each value pattern by the latency of its decision. A model that takes four hours to run cannot serve a decision due in ten minutes, and the appendix's *What a Solver and a Simulator Do* gives the one number, the real-time factor, that says whether it can.
4. **Honest about uncertainty.** The model must say how sure it is, so that a decision taken on its output can weigh the risk. A remaining life of "fourteen months" is less useful than "between nine and twenty months, most likely fourteen", and often less true.

The fourth is the one software engineers find strangest, because most functions we write return values, not distributions. It is also the one the last of the five model types exists to satisfy.

What the six value patterns ask of a model

Chapter 1's six value patterns – monitor, diagnose, predict, decide, control, certify – each stress a different subset of the four criteria, and Table 2 says which. Read it down a column to see which criterion is most often decisive: the time budget appears in every row from predict onward, which is why the two cheap model types carry most of the chapter's decisions.

Table 2: The six value patterns of Chapter 1 against the four criteria each stresses most. A model chosen for monitoring may fail the moment the twin is asked to predict.

Value pattern	What the model must do	Criteria stressed
Monitor	read the state as the sensors would: the output function	evaluable
Diagnose	predict what the readings should have been, to form a residual	evaluable, updatable
Predict	run forward from the estimated state, with a spread	time budget, honest about uncertainty
Decide	run forward many times, once per option	time budget, honest about uncertainty
Control	run forward inside the actuator's loop	time budget, evaluable, updatable
Certify	stand in for the asset with a stated error	honest about uncertainty, updatable

One reading of Table 2 to carry forward: the same twin usually climbs Chapter 1's ladder over its life, from monitoring to prediction to decision, and the model that served the first pattern on that ladder is rarely the one that serves the last. The Choosing a Model Type section returns to this, once the five types are on the table.

The Running Example: An Offshore Wind Turbine

One asset carries the five sections that follow, so that each model type is met as an answer to a question about the same machine rather than as a technique in the abstract. The turbine is the one Chapter 5 used for its worked stream, made concrete enough to model.

The machine. Three blades on a hub turn a low-speed shaft at around one revolution every four seconds in a good wind. A gearbox steps that up by a factor near a hundred to a generator, which the turbine's controller loads with a torque that sets how much power leaves. The nacelle holding all of this sits on a steel tower some eighty metres tall, and the tower stands on a foundation: a monopile driven into the seabed for a fixed turbine, or a floating spar or semi-submersible held by mooring lines for one in deep water. The two published twins this chapter leans on are both of floating turbines – a physics-based twin validated on a full-scale floating prototype that estimates the tower's loads and its fatigue life from a handful of standard measurements [3], and a diagnostic twin of an operational floating turbine that watches its gearbox temperatures for anomalies [4] – and the offshore-structures

literature treats the tower and foundation as a structure whose measured dynamics should be compared with a predictive model [5]. Nothing in the sections below depends on whether the foundation floats; where it matters, the text says so.

What the operator sees. The turbine is in one of a handful of modes at any moment: idle below the cut-in wind speed, starting up, producing, curtailed when the grid operator asks for less than the wind would give, stopped on a fault, or under maintenance. Those six words are the whole of the turbine to the control room, and a section below models them as exactly that.

What happens when something breaks. Nobody drives to an offshore turbine. A crew-transfer vessel, or for a heavy component a jack-up vessel with a crane, is dispatched from port, and it sails only when the wave height and wind are below the vessel's limits, a **weather window** that in a North Sea winter may not open for days. A fault therefore costs the lost production from the moment of the fault to the end of the repair, and most of that time is spent waiting for a vessel and a window. That waiting is the thing a discrete-event model is for.

What the twin can read. Table 3 lists the sensors, at the rates Chapter 5 gave them. The supervisory control and data acquisition (SCADA) system reports wind speed, active and reactive power, rotor and generator speed, blade pitch and the temperatures of the shaft bearing, brakes, generator and gearbox oil [4]; on a commercial turbine those arrive as ten-minute statistics [6]. A condition-monitoring system samples drivetrain vibration at hertz to tens of hertz. Tower accelerometers, and on a floater inclination and position sensors, sample at kilohertz, and the published structural twin needs only those plus the SCADA power, pitch and rotor speed to estimate the loads along the tower [3].

Table 3: What the turbine's twin can read, and which model each stream feeds. The rates are Chapter 5's; the right-hand column is this chapter's map from data to model.

Sensor	Reads	Rate	Feeds the model of
SCADA counters	power, rotor and generator speed, pitch	ten-minute statistics; hertz on a research turbine	rotor and drivetrain; operating mode
SCADA thermocouples	shaft bearing, brake, generator and gearbox oil temperatures	ten-minute statistics	gearbox oil; bearing wear
condition monitoring	drivetrain vibration	hertz to tens of hertz	bearing wear
tower accelerometers	tower acceleration at two or three heights	kilohertz	tower and blade structure; loads
floater sensors	inclination, position	hertz	structure, on a floating foundation
control room log	mode changes, alarms, vessel dispatches	on each event	operating mode; vessel dispatch

The decisions. Each model type below serves one decision, and Table 4 states them in advance so that each section can open on its question. The decisions are chosen so that no single model type could serve them all, which is the chapter's argument in one table: a twin of a real asset carries several models, and choosing among the types is choosing per decision, not per asset.

Table 4: The turbine's five decisions and the model type that serves each. The Choosing a Model Type section returns to this table with the reasons.

Decision	Question the operator asks	Model type that serves it
power and thermal expectation	is the rotor speeding up as it should, and will the gearbox oil overheat in the next ten minutes?	lumped first-principles
availability and repair planning	which mode is the turbine in, and when will a vessel get to it?	discrete-state and discrete-event
structural integrity	has the tower's stiffness changed since commissioning?	high-fidelity field
load monitoring every second	what load is the tower carrying right now, and how much fatigue did this gust cost?	reduced-order and surrogate
bearing replacement	how long before the main bearing must be replaced, with what confidence?	probabilistic, Bayesian updating

Lumped-Parameter First-Principles Models

Is the rotor speeding up as it should, and will the gearbox oil overheat in the next ten minutes? Both halves of that question are "how much" questions about a few quantities, and the model type that answers them is the oldest one: write down what the physics says about how each quantity changes, and step it forward.

Conservation is where rate rules come from

Every rate rule in this section, and most in engineering, comes from one sentence: **what flows in, minus what flows out, is what accumulates.** Energy into a gearbox minus energy out is energy stored in its oil, which is a temperature rise. Torque turning a rotor minus torque resisting it is stored angular momentum, which is a speed rise. Water into a pot minus water out is stored moisture, which was Chapter 2. The sentence is a rate rule as soon as "accumulates" is read as "the rate of change of the stored quantity is", and the mathematics appendix's *Quantities, Rates and Equations* is where that reading is built.

Lumped means the quantity is treated as one number for a whole component – one temperature for the gearbox oil, one speed for the rotor – rather than a value at every point. When a component's properties are close enough to uniform over its geometry, its parameters can be aggregated over the whole object and the object reduced to a point, and such lumped-parameter models are written as ordinary differential equations, or as differential-algebraic

equations when constraints join them [7]. The High-Fidelity Field Models section is what happens when that assumption is dropped.

Three roles, four domains

Once the rate rules are written for a few systems a pattern shows. Every element plays one of three roles: it **stores** a quantity, it **dissipates** it, or it **supplies the effort** that drives it. A capacitor stores charge, a resistor dissipates it and a battery supplies the voltage; a mass stores momentum, friction dissipates it and a force drives it; a tank of oil stores heat, a cooler dissipates it and the gearbox's losses supply it. Table 5 lays the analogy out across the four domains a turbine's twin will meet. This is the observation the bond-graph formalism is built on – it unifies the physical domains at the lumped level by recognising the analogy between energy storage and dissipation across electrical, mechanical, thermal and hydraulic systems, and connects elements by the power that flows between them [7] – told here without its notation, because the observation is what a software engineer needs and the notation is not.

Table 5: Storage, dissipation and effort in four domains. Each row is one idea in four costumes; the last row is the rate rule each domain's conservation sentence becomes.

Role	Mechanical (rotation)	Electrical	Thermal	Hydraulic
stores	inertia holds angular momentum	capacitor holds charge	thermal mass holds heat	tank holds volume
dissipates	friction, drag	resistor	cooler, conduction to ambient	pipe friction, leak
supplies effort	applied torque	voltage source	heater, losses as heat	pump
the stored quantity's rate	torque imbalance over inertia	current over capacitance	net heat flow over thermal mass	net flow over tank area

The payoff of Table 5 is that a software engineer who has understood one column has understood the others. The turbine's rotor is the first column and its gearbox oil is the third, and the two rate rules below are the last row of each.

Equations, not assignments

A programmer's instinct is to write the rotor model as assignments: compute the torque, then the acceleration, then the new speed. A modeller in an equation-based language writes relations instead – torque equals inertia times acceleration – without saying which side is computed from which. This is the **acausal** style: a declarative style based on equations rather than assignments, in which which variables are inputs and which are outputs is left unspecified and is fixed only when causality is assigned before the model runs [7]. The point is reuse. A gearbox model written acausally can be driven from its input shaft or from

its output shaft without being rewritten, because the equations do not know which end is upstream; an assignment-based model would need a second version.

The price is a translator. Before an acausal model can run, something has to sort its equations into an order in which each unknown is computed from things already known – which is exactly the assignment program the programmer would have written by hand, generated instead. Modelica is the language most twins in the literature use for this, and a Modelica model reaches a twin as a compiled unit: TwinOps, for instance, exports Modelica models as functional mock-up units, generates the interface and glue code for each, and co-simulates them with an architecture model while the same pipeline builds the real system [8]. Sorting happens at that compile step, and when it fails – because the equations do not determine the unknowns, or determine them twice – the failure is a compile error and not a wrong number at run time, which is the second reason engineers accept the translator.

Constraints are equations with no rate in them, holding at every instant beside the rate rules. The gear ratio is one: the generator's speed is the rotor's speed times the ratio, always, not "eventually". A published architecture for cyber-physical twins decomposes the system level by level from the whole to the components, writes each component's physics as ordinary differential equations and the connections between components as algebraic equations, and the model is the coupling of the two [9]. The appendix's *Stiff Equations for Digital Twins* is where the solver's side of that coupling is discussed; here it is enough that a constraint is a relation the state must satisfy, and a translator uses it to eliminate a variable rather than to step one.

The turbine: six lines and a step function

Two states, two rate rules, one constraint, and a step. The symbols are in Table 6.

Table 6: The turbine's lumped model: two states, four inputs, four parameters and the step. The values are illustrative for a turbine of about two megawatts and are used in the worked step.

Symbol	Meaning	Unit	Role	Value used below
ω_{r}	rotor speed	radians per second	state	1.70
T_{oil}	gearbox oil temperature	degrees Celsius	state	48.0
Q_{a}	aerodynamic torque on the rotor	newton metres	input, from wind and pitch	1.50×10^{6}
Q_{g}	generator torque, referred to the rotor shaft	newton metres	input, from the controller	1.44×10^{6}

Symbol	Meaning	Unit	Role	Value used below
P_{loss}	gearbox loss power, as heat into the oil	watts	input, from the power flowing through	4.0×10^4
T_{amb}	nacelle air temperature	degrees Celsius	input	12.0
J	rotor and drivetrain inertia, referred to the rotor shaft	kilogram square metres	parameter	6.0×10^6
C_{th}	thermal capacity of the oil and gearbox casing	joules per kelvin	parameter	2.0×10^6
k_{th}	heat-loss coefficient from oil to nacelle air	watts per kelvin	parameter	1.0×10^3
n_{gear}	gear ratio	–	parameter	97
h	step size	seconds	the loop's	1

The rotor. Torque in minus torque out is stored angular momentum; angular momentum is inertia times speed; so the speed's rate is the torque imbalance over the inertia:

$$J \frac{d\omega_{\text{r}}}{dt} = Q_{\text{a}} - Q_{\text{g}} \tag{3}$$

The oil. Heat in minus heat out is stored heat; stored heat is thermal capacity times temperature; heat out is the loss to the nacelle air, proportional to the temperature difference:

$$C_{\text{th}} \frac{dT_{\text{oil}}}{dt} = P_{\text{loss}} - k_{\text{th}} \left(T_{\text{oil}} - T_{\text{amb}}\right) \tag{4}$$

Equation 3 says that a rotor left alone keeps its speed, and Equation 4 that oil left alone settles where loss equals gain; the step below moves both by one second.

The constraint. The generator turns at the gear ratio times the rotor speed, $\omega_{\text{g}} = n_{\text{gear}}\, \omega_{\text{r}}$, and the generator torque the controller sets on the fast shaft appears at the rotor as Q_{g}, the fast-shaft torque times the ratio. Two algebraic lines, no rates.

The output function. The SCADA counter reports the rotor speed in revolutions per minute, $z_1 = 60\, \omega_{\text{r}}/(2\pi)$, and the oil thermocouple reports $z_2 = T_{\text{oil}}$. Two more lines, and the model is six lines long.

The step. Apply Euler's step of the appendix – new value equals old value plus rate times step – to each rate rule, with the values of Table 6:

```
def step(x, u, theta, h):
    omega_r, T_oil = x
    Q_a, Q_g, P_loss, T_amb = u
    J, C_th, k_th = theta
    d_omega = (Q_a - Q_g) / J                          # the rotor's
 ↪  rate
    d_T = (P_loss - k_th * (T_oil - T_amb)) / C_th     # the oil's rate
    return (omega_r + h * d_omega, T_oil + h * d_T)
```

Run it once by hand. The torque imbalance is $Q_\mathrm{a} - Q_\mathrm{g} = 6.0 \times 10^4$ newton metres, so the rotor's rate is $6.0\times10^4/6.0\times10^6 = 0.010$ radians per second per second, and after one second $\omega_\mathrm{r} = 1.71$. The oil's excess over the air is $T_\mathrm{oil} - T_\mathrm{amb} = 36$ degrees, the loss is $k_\mathrm{th} \times 36 = 3.6\times10^4$ watts against 4.0×10^4 watts coming in, so the oil warms at $4.0\times10^3/2.0\times10^6 = 0.002$ degrees per second and after one second $T_\mathrm{oil} = 48.002$. The oil is heading for the temperature at which loss equals gain, $T_\mathrm{amb} + P_\mathrm{loss}/k_\mathrm{th} = 52$ degrees, and at this rate it will take most of the next ten minutes to get close: that is the "will it overheat" half of the question answered, with numbers, from six lines.

Two things to notice, because they recur. First, the two states move on wildly different time scales – the rotor changes by a per cent in a second, the oil by a hundredth of a degree – and the step size was set by the fast one. The appendix's *Stiff Equations for Digital Twins* explains why that is forced and what a better solver does about it; the cost here is that the oil rule is stepped six hundred times per SCADA reading to advance a fraction of a degree. Second, the model predicts T_oil, and the SCADA system measures it: the difference between the two, once a reading arrives, is a residual, and a residual that grows is a bearing warming or a cooler failing. That is the diagnostic twin's whole method, and it is why the published diagnostic twin of a floating turbine chose the gearbox temperatures as the quantities to model: they change slowly, so they need only the low SCADA rates, and an abnormal temperature is a component misbehaving [4].

Where else this model type appears

The rotor-and-oil model is small. The same model type scales to a whole system by composition. The cyber-physical architecture cited above builds a lunar probe's power system as a library of lumped component models – environment, solar array, battery pack, controller, load, distribution – each a set of rate rules, joined by algebraic constraints into a system model that can be simulated at several levels of granularity [9]. In buildings, Spawn couples an equation-based Modelica model of the heating, ventilation and control systems to EnergyPlus's envelope simulation, taking from the acausal side the flexibility to model novel heat-pump and storage systems that the sequential simulator's fixed loop topologies cannot express, and accepting in return longer translation and simulation times [10]. In both, the lumped model type is the frame that holds the rest together.

What it costs

Domain knowledge to write, and almost nothing to run. Writing the six lines above needed someone who knew that torque imbalance over inertia is an acceleration and that heat loss is proportional to a temperature difference; those are undergraduate physics, but they are physics, and a software engineer either learns them for the domain at hand or works with

someone who has. Running them costs microseconds a step. The model passes the time-budget criterion for every value pattern in Table 2, and it passes the evaluability criterion because its inputs are the SCADA stream. What it does not do, alone, is say how sure it is: it returns numbers, not spreads.

How it is kept current

Two things drift, and they are kept current by two different mechanisms. The **state** – the rotor speed and the oil temperature the model thinks it has – drifts from the truth with every step, because the inputs are measured imperfectly and the model is not the turbine; it is corrected from every reading by an estimator, which is the state and estimator chapter's whole subject. The **parameters** – J, C_{th}, k_{th} – drift as the turbine ages, a fouled cooler lowering k_{th} over months, and they are refit from accumulated residuals on a slow schedule, which is the Keeping a Model Current section's subject. Nothing about the structure of the six lines changes until the turbine itself does.

One Euler step gives the state-space form

Close by writing the step once more, as a table. Stack the two states into a vector $x = \begin{bmatrix} \omega_{\text{r}} \\ T_{\text{oil}} \end{bmatrix}$ and the two driving quantities into $u = \begin{bmatrix} Q_{\text{a}} - Q_{\text{g}} \\ P_{\text{loss}} + k_{\text{th}}\, T_{\text{amb}} \end{bmatrix}$, the second entry being the heat that would flow in with the oil at nacelle temperature. Both rate rules are then weighted sums of the states and inputs, and one Euler step of both at once is:

$$x_{k+1} = F\,x_k + G\,u_k, \qquad F = \begin{bmatrix} 1 & 0 \\ 0 & 1 - h\,k_{\text{th}}/C_{\text{th}} \end{bmatrix}, \qquad G = \begin{bmatrix} h/J & 0 \\ 0 & h/C_{\text{th}} \end{bmatrix} \tag{5}$$

Equation 5 is the discrete-time state-space form the appendix builds under *Many Quantities at Once* and the state and estimator chapter runs its filter on. The turbine's F has a one in the corner because the rotor speed, left alone, keeps its value, and $1 - h\,k_{\text{th}}/C_{\text{th}} = 0.9995$ in the other because the oil's excess decays half a thousandth per second; multiply x by F, add G times u, and you have the two numbers computed by hand above. That a six-line physics model and a two-by-two table are the same model, written twice, is the fact this section hands on: the physics gives the twin its expectation, and the tables are the form in which the next chapter corrects it.

Discrete-State and Discrete-Event Models

Which mode is the turbine in, and when will a vessel get to it? Neither half of that question is "how much". The first asks which of six words describes the turbine; the second asks when a sequence of things – a fault, a dispatch, a weather window, a sailing, a repair – will have happened. Both are questions about discrete states and the events that move between them, and this is the model type for them.

Home turf: a finite state machine with guards and actions

A software engineer has written this model many times without calling it one. A **finite state machine** is a finite set of modes, exactly one of them active, and a set of **transitions**, each

fired by an event, each with a **guard** – a condition on the state or the inputs that must hold for the transition to fire – and an **action**, what is done on firing. Figure 2 draws the turbine's operating modes as one. Idle becomes start-up when the wind rises above the cut-in speed; start-up becomes producing when the rotor reaches its rated speed; producing becomes curtailed on a grid command and returns when the command is lifted; any operating mode becomes fault on an alarm; fault becomes maintenance when the vessel arrives; maintenance becomes idle when the repair is done.

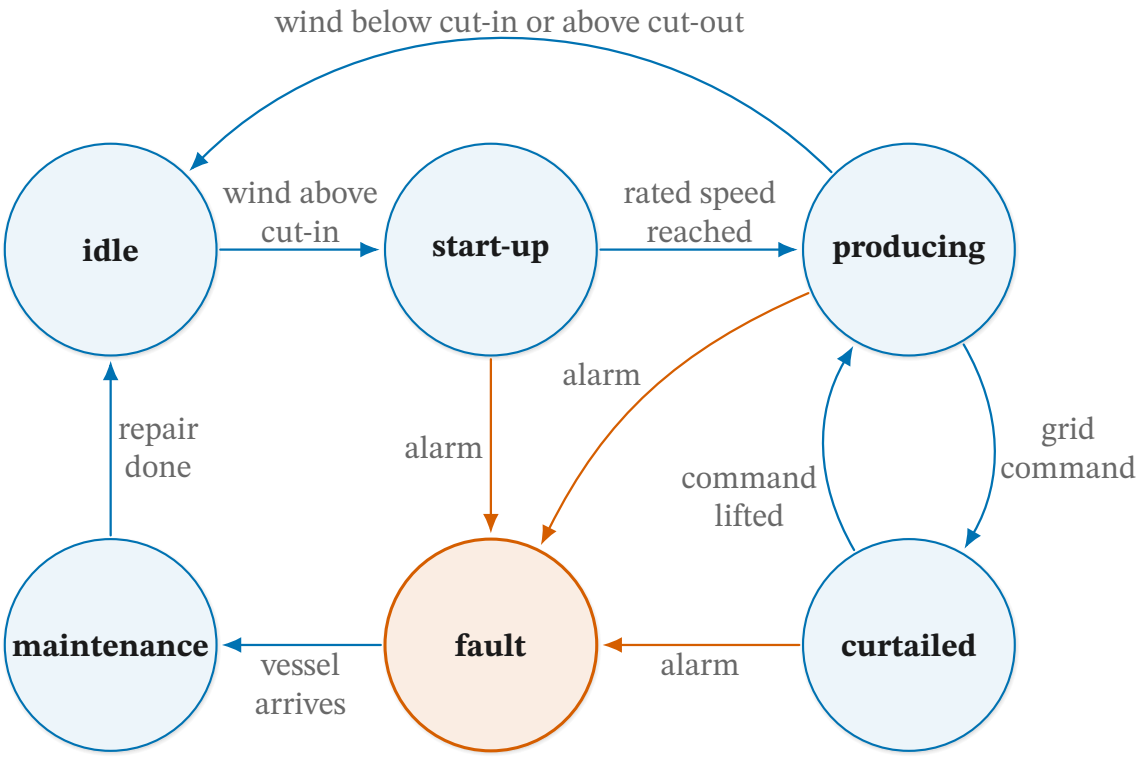


Figure 2: The turbine's operating modes as a finite state machine. Each arrow carries its guard; the actions – log the change, set the generator torque to zero, raise a work order – are not drawn. Alarms lead to the fault mode from every operating mode.

In the terms of the What Every Model Has section: the state record is one value from an enumeration of six; the inputs are the wind speed, the grid command, the alarm flags and the vessel's arrival; there are no parameters yet; the step function is "if the guard of any outgoing transition holds, move and perform its action"; and the output function is the mode itself, which is what the control room log records. The state is one word, which is why this model type answers "which" and not "how much".

Adding time, twice

A plain state machine has no clock. It moves when an event arrives and otherwise waits, and it cannot say how long start-up takes. Time enters in two steps.

Timed transitions. Give a transition a duration: start-up takes ninety seconds; a repair takes eight hours. The step function now has a clock, and a transition fires when its duration has elapsed as well as when its guard holds. That is enough for a single machine.

Timestamped events in a queue. The moment there are two machines – the turbine and

the vessel – or one machine and a world that sends things at its own pace, the clean way to run them is to put every pending event on a **priority queue** ordered by its timestamp, pop the earliest, advance the clock to it, apply it, and let the application schedule new events onto the queue. This is the **discrete-event** abstraction: the step is "pop, advance, apply", and the state of the whole model is the modes of its machines plus the queue. The appendix's *What a Solver and a Simulator Do* runs exactly this loop under *An event scheduler runs the discrete version*. It is also the view the discrete-event system specification (DEVS) formalises: each atomic model has a time-advance function that says how long it will stay in its state if nothing arrives, an internal transition for when that time runs out and an external transition for when an input arrives first, with the elapsed time since the last transition available so that an interrupted process can resume with the time it has left [7]. A twinning system's design passes through a stage of choosing such formalisms – differential equations, DEVS or Petri nets – to match its goals, before deployment [11]; the choice this section teaches is the second of those, with the third next.

Petri nets: the thing state machines do badly

A state machine has one active mode. Two vessels, four turbines, one spare gearbox and two weather windows have no single mode; they have counts, and the question is which activities can proceed at once and what they compete for. A **Petri net** answers it. Its state is a marking: a count of **tokens** in each **place**, where a place may stand for a process step ("waiting for a vessel") or a resource ("spare gearboxes in port") alike; a **transition** fires when every place feeding it holds enough tokens, consuming them and producing tokens in the places it feeds. Petri nets treat processes and resources as the same kind of thing, encode the state as a vector of natural numbers, and are built for concurrency and synchronisation – and, because they cover all possible behaviours rather than one run, they are used mostly for analysing properties, such as showing that two activities can never both hold the same crane, rather than for simulation [7]. A repair that needs a vessel *and* a spare *and* a window is three tokens consumed by one transition, and a state machine would need a mode for every combination.

Queues are the special case a software engineer meets most: things arrive, wait in a line, are served one at a time. Turbines with faults are the arrivals, the vessel is the server, and the waiting line is the lost production. The formalisms' own literature lists suitability for competition over shared resources – which is what leads to queueing – among the properties a modelling language is chosen for [7]. Any queue can be drawn as a small Petri net, and most are simulated as a discrete-event model instead.

The turbine: vessel dispatch with weather windows

Two machines and a queue. The turbine's mode machine is Figure 2. The vessel's machine has four modes: in port, waiting for a window, sailing, on site. The parameters this model type does have are durations and frequencies: the sailing time, the repair time, and the weather – modelled here as windows that open and close as events, with a mean gap between them that is the one number a planner most wants to know in winter. Table 7 pops the first five events from the queue for one fault.

Table 7: Five events popped from the queue for one gearbox fault, with the mode changes each causes. The waiting for a window, nine and a half hours here, is the part of the twenty hours no repair crew can shorten.

Clock	Event popped	Guard checked	Action, and events scheduled
02:10	gearbox alarm on turbine 7	turbine 7 is producing	turbine 7 to fault; generator torque to zero; schedule “dispatch requested” now
02:10	dispatch requested	vessel in port; a spare gearbox in port	vessel to waiting for a window; ask the weather model for the next window
11:40	weather window opens	vessel waiting	vessel to sailing; schedule “on site” at 11:40 plus the three-hour sailing time
14:40	vessel on site	window still open	turbine 7 to maintenance; schedule “repair done” at 14:40 plus eight hours
22:40	repair done	–	turbine 7 to idle; vessel to sailing home; log twenty hours and thirty minutes of lost production

Read Table 7 down the clock column: nine and a half of the twenty hours are the vessel waiting in port, and that is the number the model exists to produce. Run the same queue a thousand times with the weather windows drawn from their winter distribution and the lost-production hours per fault come out as a spread, which is the “when will a vessel get to it” question answered honestly. Run it with a second vessel, or a spare gearbox kept on the turbine, and the spread moves: that is the decision the operator is paying for.

Where else this model type appears

The manufacturing-twin literature’s own table of modelling techniques lists, under machine and process states, the finite state machine, the hybrid automaton and the hidden Markov model, with a hidden Markov model used for the reactive control of a robot cell’s components and a generic hybrid automaton written for the continuous and discrete dynamics of additive manufacturing [12]. In production, discrete-event simulation models abstract a group of machines into event-driven operations and can form a twin of the whole production process [13]; and the software engineering literature on twins records the same abstraction carried into a hospital, where a discrete-event simulation integrated with pervasive sensing forms the hospital’s twin [14]. Patients, parts and turbines with faults are the same arrivals to the same queue.

Hybrid systems are where this model type meets the previous one. A turbine in start-up obeys one set of rate rules, a turbine producing under torque control another, and a turbine in fault with the brake on a third: the mode machine chooses which rate rules run, and the rate rules’ states decide when a guard fires. A state machine whose modes each carry their own rate rules is a **hybrid automaton**, and the hybrid automaton of an additive process

cited above is one. The simulators half of this part takes up how the two clocks are kept in step; here, the design point is that the mode is state too, and belongs in the state record beside the speed and the temperature.

What it costs

Almost nothing to run: popping events from a queue is cheap, and a year of a turbine's mode changes is a few thousand events. The hard part is not computing but deciding. Which events matter? A model that includes every alarm code the controller can raise has a state machine nobody can read and durations nobody can estimate; a model that collapses them to "fault" answers the availability question and no other. The cost is the modeller's judgement about the decision the model serves, and it is paid once, up front.

How it is kept current

Its parameters are durations and frequencies – how long a start-up takes, how often a window opens, what fraction of faults need a jack-up vessel – and every one of them is re-estimated from the event log the control room already keeps. This is the easiest model type to keep current, because the log *is* the calibration data, and the Keeping a Model Current section's slow schedule can be "recompute the means from the last two winters". The structure – which modes exist, which transitions join them – changes only when the turbine or the operating rules do, and that is configuration's business.

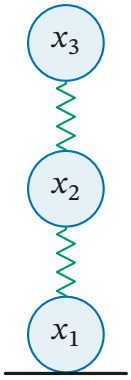
High-Fidelity Field Models

Has the tower's stiffness changed since commissioning? A crack at a weld, a scoured seabed around a monopile, a loosened flange: each changes how the tower bends, and none of them is a lumped quantity. The question is about the tower as a body, and the model type that answers it treats it as one.

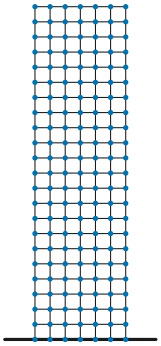
From a handful of states to a field

The previous two model types kept one number per component. A **field** model keeps a value at every point of the component: the stress at every point of the tower wall, the temperature at every point of a gearbox casing, the air speed at every point around a blade. The state is no longer a record of a few quantities but a function over space, and a computer cannot hold a function over space. So the body is cut into small cells – a **mesh** – and the field is stored at the cells' corners, the **nodes**, with a simple rule for how it varies inside each cell, the **element**. Figure 3 shows the two views of the same tower section side by side.

Now the thing to see. Once the mesh is drawn, every node's displacement is a state, and the physics at each node – force in minus force out equals mass times acceleration, the same conservation sentence as the previous section – is a rate rule relating it to its neighbours. A field model is therefore a lumped model with a million states, written automatically by a program that walks the mesh, rather than by hand. It has the same anatomy as everything in this chapter: a state vector, an input (the wind and wave loads at the nodes on the surface), parameters (the steel's stiffness and density, the wall thickness, the soil's stiffness at the base), a step function and an output function that reads the acceleration at the nodes where



(a) three lumps: three states stiffness and mass assumed



(b) a mesh: every node a state modes computed, not assumed

Figure 3: The same tower section as a lumped model and as a field model. In panel (a) three masses on two springs give three states and a stiffness somebody assumed; in panel (b) a mesh gives a state at every node and a stiffness computed from the wall thickness and the steel.

the accelerometers are. It is the size that changes, and with it the cost and what can be computed.

Modes are computed, not assumed. The appendix's *Many Quantities at Once* introduced a mode as a shape a structure prefers to move in, at a natural frequency, and gave the two-mass example whose frequencies followed from two stiffnesses somebody had to supply. A field model supplies them. From the mesh and the parameters it computes the tower's mode shapes and natural frequencies – the first sway, the second sway, the first twist – without anybody assuming a stiffness for the whole tower, because the stiffness is a consequence of the wall thickness and the material, element by element. That is what makes the field model the right tool for the stiffness question: a change in stiffness anywhere moves a natural frequency somewhere, and a natural frequency is something an accelerometer can measure.

Two families

Finite-element models are the field models of solids and structures: towers, blades, gearbox casings, bridge decks. The finite element method (FEM) is the mesh-walking program described above, and its output is displacements, stresses and modes. **Computational fluid dynamics** (CFD) is the field model of fluids: the air around a blade, the water around a foundation, the hot gas in a furnace. The same idea – mesh, conservation at every cell, a huge state vector – with the complication that the fluid moves through the mesh, so the equations at each cell involve what is flowing in from upstream. The two are often coupled, because the fluid's pressure is the structure's load, and that coupling is one of the reasons these models are expensive.

The turbine: computed frequencies against measured ones

The tower and blades are meshed; the mesh, the steel's properties, the wall thicknesses and an assumed soil or mooring stiffness are the parameters; and the model computes the

first few natural frequencies. Meanwhile the accelerometers of Table 3 record the tower's motion, and a procedure the appendix names – operational modal analysis, which extracts natural frequencies and mode shapes from vibration records without knowing the loads – produces the measured frequencies. The comparison is the decision: if the computed first sway frequency was 0.31 hertz at commissioning and the measured one is 0.31 hertz today, the tower's stiffness has not changed; if the measured one has drifted to 0.29, something has. The offshore twin literature describes exactly this loop for North Sea platforms, in five levels from screening and diagnostics through finite-element model updating and wave-load calibration to quantified uncertainty and accumulated fatigue [2], and lists the model parameters that move the frequencies – mass and centre of gravity, element stiffness, local joint stiffness, soil stiffness and damping – as the ones an update adjusts [5]. The diagnostic twin of the floating turbine sits on the same asset and reads the SCADA side of it [4]; the field model is what would read the structural side.

Where else this model type appears

The structural twin of a small unmanned aircraft is a finite-element model of its wing, calibrated so that the first two bending modes' natural frequencies match the ones extracted from strain measurements after an initial-condition test; once calibrated, the model computes the higher bending, torsional and skin-buckling modes that are hard to measure, and its structural-health parameters are percentage reductions in stiffness in regions of the wing – the aircraft's version of the tower question [15]. In construction, a twinning platform generates building twins automatically from the building information model, exchanged as Industry Foundation Classes, which carries the geometry a field model of the building starts from, so that the twin is instantiated from the building's own design documents rather than from a separate modelling effort [16]. Both are the same move as the turbine: the mesh comes from the design, and the parameters are what the twin has left to learn.

What it costs

Hours per run, specialist tooling, and somebody who knows how to mesh. A tower model of a few hundred thousand nodes takes minutes to compute its modes and hours to simulate a storm; a blade in CFD takes days. The tooling is a commercial finite-element or CFD package or a research code, and the meshing is a skill of its own – too coarse and the model misses the stress concentration at the weld, too fine and the run does not finish. The manufacturing-twin review reaches the conclusion this cost forces: high-fidelity numerical models are well suited to twin applications in principle, but may not be appropriate in practice because fidelity comes at the price of computational cost and simulation time, and the remedy the literature reaches for is reduced- order modelling or simplification [12]. Against the four criteria, the field model fails the time budget for every value pattern below certify, and the next section is how it buys that back. It passes the others: its inputs are the loads and the accelerometers, its parameters are updatable, and once its uncertainty is quantified it is the most honest model of the tower there is.

How it is kept current

By **model updating**: adjusting the model's parameters until the frequencies it computes match the frequencies the accelerometers measure. The published twin-platform case does this for a steel beam – vibration records are processed by operational modal analysis to

extract the natural frequencies, a detailed finite-element model is built, and the measured frequencies become the targets of an optimisation over the model's design variables, which is finite-element model updating [17]. The turbine's version adjusts the soil or mooring stiffness and the local joint stiffnesses until the computed sway frequencies match the measured ones, and records how far they had to move, which is the stiffness question's answer. It is done on a slow schedule – after commissioning, after a storm, once a season – because each update is hours of compute; the Keeping a Model Current section places it on that schedule beside the lumped model's parameter refit.

Reduced-Order and Surrogate Models

What load is the tower carrying right now, and how much fatigue did this gust cost? The field model of the previous section knows the answer, and takes hours to say it. The decision wants it every second. There are two ways to buy that time back, and they are different enough to deserve different names.

Reduce: keep a few combinations of the million states

The field model's million nodal displacements are not a million independent things. The appendix's *Many Quantities at Once* made the point for a two-mass system: any motion is a mix of the modes, and a handful of modes describes most of what a large linear system does. A tower sways mostly in its first and second sway modes and twists a little in its first twist mode; the other 999,997 combinations of its nodal displacements carry almost none of the motion. A **reduced-order model** keeps only the few combinations that matter – a **reduced basis**, typically the leading modes – and writes the rate rules for their amplitudes instead of for the nodes. The state vector shrinks from a million to five, the step function is a five-by-five table, and the loop runs in microseconds. The full field can be reconstructed at any moment by mixing the five mode shapes in the current amplitudes, which is how a five-state model still answers "what is the stress at the weld".

The model produced this way is a lumped model again, and everything the Lumped-Parameter First-Principles Models section said about steps, solvers and the state-space form applies to it unchanged. What is new is where it came from: its five states were chosen by the field model's modes, not by an engineer's judgement, and its parameters were computed, not assumed.

Replace: fit a cheap function from inputs to outputs

The second way does not shrink the model; it stops running it. If the decision needs one number – the fatigue damage a ten-minute interval of given wind speed, wave height and turbulence does to the tower – run the field model a few hundred times offline over the range of conditions the turbine will meet, record the input-output pairs, and fit a cheap function through them. That function is a **surrogate**. It can be a polynomial in the inputs, a **response surface**; a **Gaussian process**, which fits a distribution over functions and so returns a spread as well as a value; or a network. The twin literature describes surrogates in exactly these terms – response surfaces, kriging, support vector machines and neural networks that mimic computationally expensive physics models – and notes that they are built from training data taken from the original simulation, not from the asset [18]. Once

fitted, a surrogate is evaluated in microseconds, and the big model is never run again in operation.

The turbine: a reduced tower and a fatigue surrogate

The reduced tower. Project the finite-element tower onto its first three sway and twist modes. Feed the three-state model the tower accelerations and the SCADA power, pitch and rotor speed, and it estimates the load at every section of the tower every second – which is the load-monitoring decision. The published relative of this is the executable-twin literature's wind-turbine blade: a detailed finite-element model of the blade compressed into a reduced-order model and paired with a state estimator gave a virtual sensor that shows the full strain field of the blade in real time on a test rig instrumented with only a few strain gauges [19]. The same paper did the same for a truck anchorage whose critical locations could not be instrumented at all, which is the general point: a reduced model plus a few sensors reads what no sensor can reach. The state estimator that makes it work is the state and estimator chapter's; the reduced model is what it runs on.

The fatigue surrogate. Fatigue is the slow accumulation of damage from many small load cycles, and computing it from a load history is a standard but expensive calculation. Run the field model, or the reduced one, over a grid of wind speeds, wave heights and turbulence intensities; compute the ten-minute fatigue damage for each; fit a Gaussian process to the table. In operation, the twin reads the ten-minute SCADA statistics, evaluates the surrogate, and adds the result to the tower's running fatigue total, with a spread from the Gaussian process that says how well the surrogate knows this corner of the input space. That is "how much fatigue did this gust cost", every ten minutes, for microseconds.

Where else this model type appears

The aircraft twin above is built on a **library** of component-based reduced-order models – reduced models of the wing's components, built component by component – so that any asset state the library covers can be simulated in real time, and a classifier reads the sensor data and picks which library models best describe this aircraft now; sharing the library across many aircraft is what lets one modelling effort serve many twins [20]. In manufacturing, the argument has been made for reduced-order models that are interpretable, so that an operator can see what a reduced state means, and reusable across processes in a manufactory-as-a-service setting [21]. And a surrogate can sit inside a probabilistic update rather than replace one: a multi-fidelity neural network trained on a finite-element model of several thousand degrees of freedom maps damage and operating parameters to sensor readings, and a Markov chain Monte Carlo procedure calls it thousands of times to update the distribution of a structure's damage state from noisy observations – a use that would be impossible with the full model in the loop [22]. That last case is the bridge to the next section.

A surrogate mimics a computational model, not this asset

The first of two warnings, and the one the literature is most emphatic about. A surrogate is fitted to input-output pairs from a *model*. It knows what the finite-element tower does; it knows nothing about turbine seven. Its confusion with a twin comes from a surface similarity – both use data to update what they know – but a surrogate alone is not a twin, because it maintains no representation of the state of an instance of a physical system; it

mimics a computational model, and belongs *inside* a twin, where the original model is too slow for the decision interval, not in place of one [18]. In this chapter's terms: the surrogate is a fast output function for a model whose state and parameters the twin still has to hold and update. Drop the state and the update and you have a fast calculator, not a twin.

A surrogate is valid only where it was trained

The second warning. A response surface fitted over wind speeds from four to twenty-five metres per second says nothing about a thirty-metre-per- second storm; it will return a number, and the number will be confidently wrong. The Gaussian process is the honest choice among surrogates for exactly this reason – its spread grows away from the training points, so it says "I do not know" where a polynomial would not – but even it only knows that it does not know, not what the answer is. Every surrogate needs a check that its inputs are inside the region it was fitted on, and a fall-back – the reduced model, or the field model on a slower schedule – for when they are not.

What it costs

The big model must exist first. A reduced model is a projection of a field model and a surrogate is a fit to one; neither can be built without the hours of meshing and running the previous section priced. What they buy is the time budget: after the offline cost is paid, both run in microseconds and pass criterion three for every value pattern. Against the other criteria, the reduced model is as evaluable and updatable as the field model it came from, and honest about uncertainty only if the state estimator makes it so; the surrogate is honest only if it is a Gaussian process or carries an explicit validity check.

How it is kept current

When the field model is updated – the tower's stiffness re-fitted after a storm, as in the previous section – the reduced basis and the surrogate's training table are both stale, and both are rebuilt from the updated model, or the surrogate is refit on a fresh set of runs. That rebuild is hours, on the field model's schedule, not seconds; the twin runs the old reduced model until the new one is ready, and should know that it is doing so. The National Academies' report on research gaps for digital twins lists this as open: dynamic updating and adaptation must often happen on the fly under computational and time constraints, surrogates are what make such updating feasible, and yet the surrogates must themselves be updated and re-validated as the twin's virtual representation evolves [23]. A twin that cannot say which version of its surrogate produced a number fails the audit the Keeping a Model Current section asks for.

Probabilistic Models and Bayesian Updating

How long before the main bearing must be replaced, and with what confidence? The second half of that question is the point. The four model types so far return numbers, and a number for a bearing's remaining life is a guess wearing a suit. This model type returns a distribution, and it is the one that makes the fourth criterion – honest about uncertainty – structural rather than an afterthought.

Probabilities on the arrows: the Markov chain

Start where the Discrete-State and Discrete-Event Models section left off. Put a probability on every arrow of the turbine's mode machine – from producing, the chance of a fault in the next hour is 0.002, of a curtailment 0.05, of staying put the rest – and the model's state stops being one word. It becomes a **belief**: a row of six probabilities, one per mode, that sums to one. Each step multiplies that row by the table of arrow probabilities to get next hour's belief. This is the **Markov chain** the appendix builds under *State Machines with Probabilities*, and it is worth one paragraph here and no more: it is the smallest stochastic model there is, its step returns a distribution because its state *is* a distribution, and the state and estimator chapter is where a noisy reading is folded into the belief. Here the chain is the bridge from "which mode" to "how sure".

Wear as a random walk with drift

A bearing does not have modes; it has a wear level that only rises. Call it W, measured as the vibration level the condition-monitoring system reports, in millimetres per second. The simplest honest model of it says two things: on average it rises at a **drift** μ per month, and month to month it wanders around that average by an amount with spread σ. Written as a step:

$$W(t+h) = W(t) + \mu\, h + \sigma\sqrt{h}\,\xi, \qquad \xi \sim \mathcal{N}(0,1) \tag{6}$$

Equation 6 is a **Wiener process** with drift, and the one unfamiliar thing in it is the square root. The random part scales with the square root of the interval, not the interval, because independent random kicks add up in variance, not in size: two months of wandering is not twice as far as one, on average, but about 1.4 times as far. Everything else is Euler's step with a random term. The degradation literature that twins draw on models wear this way – as a stochastic process, with the Wiener, gamma, inverse-Gaussian and Markov processes the common choices when the physics of the damage is not known well enough to write a rate rule for it [24].

Remaining useful life is then a distribution, not a date. Fix a failure threshold W_{f}, the vibration level at which the bearing must be replaced. The remaining useful life (RUL) is the time until W first crosses W_{f}, and because each run of Equation 6 crosses at a different time, the RUL is the spread of those crossing times. Its centre is roughly the distance to the threshold over the drift, $(W_{\mathrm{f}} - W)/\mu$; its width grows with σ and shrinks as the threshold nears. The predictive-maintenance twin literature defines RUL prediction as estimating the time to a component's failure from the system's data and lists the Wiener process first among its non-deep-learning methods [25]. A twin that reports "replace in fourteen months" has thrown away the width; one that reports "between nine and twenty, most likely fourteen" has kept it.

Hidden health states, noisy readings

The vibration level is not the bearing's health; it is a reading that depends on it, with noise from the load, the speed and the sensor. The model's true state – the wear – is **hidden**, and the twin sees it only through $z = W +$ noise. The appendix's *A chain seen through a noisy sensor* is this situation for a Markov chain, the hidden Markov model, and the state and

estimator chapter is the machinery for estimating the hidden state from the readings. This section needs only the idea: the reading updates a belief about something unseen, and the next paragraph does that update for one parameter by hand.

Bayes' rule on one parameter, three candidates

Recall Bayes' rule from the appendix's *Probability and Distributions*: the belief in a hypothesis after a reading is proportional to the belief before it times how likely the reading was if the hypothesis were true. Prior times likelihood, normalised, gives posterior. Apply it to the drift μ, which is the parameter the RUL depends on most and the one nobody knows for this bearing.

Suppose fleet history says bearings of this type wear at one of three rates – slow, normal, fast – with μ of 0.5, 1.0 or 2.0 millimetres per second per month, and that half of bearings are slow, three in ten normal, two in ten fast. That is the prior. Last month the vibration level rose by 1.2 millimetres per second. Taking the month-to-month scatter of a one-month rise, sensor noise included, as a Gaussian with standard deviation 0.5, the likelihood of a 1.2 rise under each candidate is the Gaussian density at 1.2 with mean μ and standard deviation 0.5. Table 8 does the arithmetic.

Table 8: Bayes' rule on the bearing's drift, with three candidate values and one month's reading. The posterior column is the fourth column divided by its sum; one reading has moved the belief from "probably slow" to "probably normal".

Candidate μ	Prior	Likelihood of a 1.2 rise	Prior times likelihood	Posterior
0.5 (slow)	0.50	0.299	0.150	0.36
1.0 (normal)	0.30	0.737	0.221	0.53
2.0 (fast)	0.20	0.222	0.044	0.11
sum	1.00	–	0.415	1.00

Read Table 8 across the normal row: the prior 0.30 times the likelihood 0.737 gives 0.221, and dividing by the column sum 0.415 gives a posterior of 0.53. The reading was most consistent with normal wear, so normal wear gained belief; it was least consistent with fast wear, which lost most. Next month's reading multiplies the posterior by a new likelihood and normalises again, and after a few months the belief will sit almost entirely on one row. Three candidates is a teaching simplification – a real twin carries μ as a continuous distribution – but the mechanism is identical, and the appendix's worked examples do it twice on the same bearing.

Model-form discrepancy, in words

Fitting μ assumes the Wiener model is right and only its parameter is unknown. It is not right; no model is. The bearing's wear may accelerate near the end, or respond to load in a way a constant drift cannot follow. The honest move is to admit that the model is wrong by an unknown amount, assume that amount is smooth – it does not jump about from

month to month – and learn it from the residuals beside the parameters. The twin literature calls that amount **model-form uncertainty** or **model discrepancy**, and the standard way to learn it beside the parameters is the Kennedy and O'Hagan framework, in which the uncertain parameters are calibrated concurrently with a discrepancy term, which has been shown to improve prediction over calibrating the parameters alone [26]. In this chapter's terms: the model's output gains a second uncertain piece, and the update that moved μ moves it too. The credibility chapter is where the size of that piece becomes evidence for or against believing the twin.

The twin as a chain of belief updates

Step back and the whole twin has the shape this section has been building one piece at a time. Figure 4 draws it as a **probabilistic graphical model** (PGM): boxes for uncertain quantities, arrows for what depends on what. Along the top, the physical asset's true state moves forward in time on its own. Along the bottom, the **digital state** – the model's parameters and hidden states, held as distributions: the belief over modes, the wear level, the drift μ, the stiffness reduction in a tower section – moves forward too. Readings climb from the asset into the digital state and update it by Bayes' rule, exactly as Table 8 did; decisions computed from the digital state descend onto the asset. The figure is to be read, not derived: the published version defines the asset and its twin as two coupled dynamical systems, the upper path the physical state and the lower the digital state, coupled by observational data assimilated to update the digital state and by control inputs chosen from it [15], and the civil-structures twin built on the same idea encodes the repeating observations-to-decisions flow as a dynamic Bayesian network whose updated digital state drives prediction and maintenance planning [27].

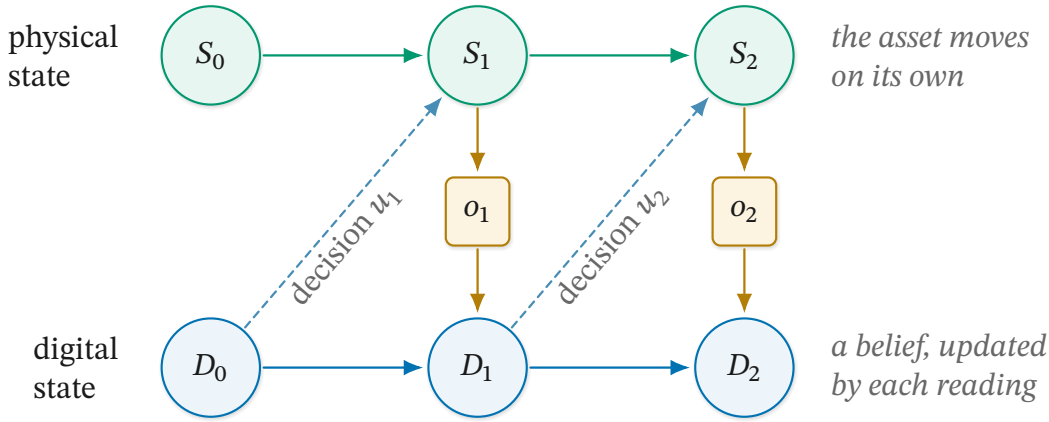


Figure 4: The twin as a probabilistic graphical model. The physical state advances along the top and the digital state along the bottom; each reading climbs into the digital state and updates it, and each decision descends from the digital state onto the asset.

The reason to draw Figure 4 here rather than in the architecture chapter is what it says about models. Every model type in this chapter is a way of writing the arrow along the bottom row – how the digital state moves from one step to the next – and every one of them is improved by being read as this section reads the Wiener model: with its parameters as beliefs, updated by readings. That reading is what the phrase "digital state" means in the literature: the set of variables that characterise a twin's models, often not directly measurable, estimated through noisy readings [24].

The turbine: a remaining-life distribution, not a date

Put the pieces together for the main bearing. The state is the wear level W, read through the vibration trend; the parameters are the drift μ and spread σ; the step is Equation 6; the update is Table 8, once a month. With the current level at 3.5 and the threshold W_{f} at 7.1 millimetres per second, the distance to failure is 3.6, and the posterior of the table says: with probability 0.36 the drift is 0.5 and the bearing has about seven months; with 0.53 it is 1.0 and about three and a half months; with 0.11 it is 2.0 and under two months. The RUL is that mixture, widened further by σ. The operator does not get a date. The operator gets "a one-in-ten chance this needs a vessel inside two months", which is the number a maintenance planner can act on – book a provisional weather window, or not – and the number no deterministic model could have produced.

What it costs

Where the parameter has three candidates, the update is a table. Where it is a continuous distribution over several parameters and the model is nonlinear, the posterior has no closed form and is drawn by **sampling**: Markov chain Monte Carlo, Gibbs sampling and slice sampling are the names the twin literature gives for computing the posterior of uncertain parameters, and they are costly enough that surrogates are routinely placed inside them [26]. This chapter names them and teaches none; the cost to know is that a Bayesian update of a rich model is thousands of model evaluations, which is why it is done on the slow schedule and why the Reduced-Order and Surrogate Models section matters to this one. Against the four criteria, this model type is the one built to pass the fourth, and it inherits the other three from whichever model it wraps.

How it is kept current

This model type *is* updating. Its parameters are beliefs and every reading moves them; there is no separate "keep current" step because the step function and the update are the same machinery seen from two sides. What the next section adds is the generalisation: the other four model types can be kept current the same way, on three different clocks.

Keeping a Model Current

Every model type above closed on the same sub-heading, and the answers had a shape. Something changed every reading, something changed on a slow schedule, and something changed only when the turbine itself did. This section names the three, says which chapter owns each, and gives the middle one – the one this chapter owns – its vocabulary.

Three rates, three homes

Table 9 is the section in one table. The **state** changes with every reading and is corrected by an estimator, which is the state and estimator chapter's subject; the **parameters** change on a slow schedule or on a trigger and are re-estimated, which is this section's; the **structure** – which models exist, which states they carry, how they connect – changes when the asset changes and is declared in the twin's configuration, which the configuration and manager chapter owns.

Table 9: The three rates at which a twin's model changes, with the mechanism and the owner of each. A change applied at the wrong rate is the commonest way a twin drifts from its asset.

What changes	How often	Mechanism	Turbine instance	Owned by
state x	every reading: seconds to minutes	estimation: predict with the step, correct with the reading	rotor speed, oil temperature, wear level, mode belief	the state and estimator chapter
parameters θ	slow schedule, or on a trigger: weeks to seasons	re-estimation from accumulated residuals, or a Bayesian update	k_{th} after cooler fouling; μ each month; soil stiffness after a storm	this chapter
structure	when the asset changes: years, or on an event	reconfiguration: a new model, a new state, a new connection, declared	a replaced gearbox with a different ratio; a new sensor; a repowered turbine	the configuration and manager chapter

The last column of Table 9 is a rule about responsibility, and the middle column is why it matters. An estimator that is allowed to move a parameter every reading will "explain" a faulty sensor by re-fitting the inertia; a parameter refit that changes the structure will silently re-map a replaced sensor onto the old model's state. The published approaches to the third row keep it apart deliberately: one twin architecture runs two feedback loops, a behavioural loop that adapts the running model's settings and an architectural loop above it that reconfigures the first when the asset moves to a new lifecycle stage declared in an asset model [28]; another observes that a twin's schema must change whenever the physical system evolves, and that schema, instances and stored time series then have to be evolved together, which the platforms surveyed did not support [29]. Both are the configuration chapter's material; here they fix the boundary of this one.

Verification, calibration, validation

Three words the literature uses about a model's fitness, and they are three different questions, worth keeping apart because a positive answer to one is routinely mistaken for a positive answer to another. Chapter 3 introduced them and cited the same source; here they are placed against the three rates.

- **Verification** asks: did we build the model we meant to? Does the code implement the six lines, does the mesh represent the drawing, does the state machine have the transitions the operating rules specify. It is a software question, answered by tests, and it is independent of the asset.
- **Calibration** asks: what parameter values make this model match this asset? It is the middle row of Table 9, and this section's business.

- **Validation** asks: does the calibrated model match this asset well enough for this decision? It is answered by comparing predictions against readings the calibration did not use, and because a twin's asset keeps changing, it is not a step but a continuous process, performed periodically or on demand, and possibly at more than one level of the system [30].

Validation's evidence – what was compared, how far apart it was, and whether that is close enough – is the credibility chapter's subject and is left to it. This section's claim is narrower: calibration is the activity, verification is its precondition, and validation is its test.

Parameter estimation, or discrepancy

When the residuals grow there are two things a calibration can do, and the Probabilistic Models and Bayesian Updating section met both. **Parameter estimation** assumes the model's form is right and moves its parameters until the residuals shrink: refit k_{th}, update the belief over μ. **Discrepancy** admits the form is wrong by a smooth unknown amount and learns that amount beside the parameters. The choice is a judgement about the residual's shape. A residual that is a constant offset, or that scales with an input the model already has, is a parameter; a residual with a shape no parameter can produce – the oil warming faster at high power than the linear loss rule allows, at every parameter value – is discrepancy, and moving parameters to chase it will make some other prediction worse. Recording which of the two a calibration did is part of the audit below.

Triggers

A slow schedule – refit each season – is the simplest trigger and often the right one for the field model, whose update costs hours. Two others are worth naming.

Drift indicators watch the residual for a change and start an update when one is seen. The twin literature's drift-based approaches compute the error between the model's predictions and recent measurements, track it over a window, and update the deployed model when the error trend crosses a threshold, so that an operational change in the asset is not left unmodelled until the next scheduled refit [24]. For the rotor-and-oil model, a drift indicator is a running mean of the oil-temperature residual; for the bearing, it is the monthly update itself.

Evolution triggers come from the asset's life rather than its data. A repair, a replaced component, a firmware change in the controller, a move to a new lifecycle stage: each says the model *may* need to change before any residual has had time to show it. The twin-engineering literature places evolution at the centre of the discipline and identifies the triggers of a twin's evolution as the events in the actual twin's lifecycle that the twin, as a software system with its own lifecycle, must follow [31]. Some evolution triggers call for a parameter refit; a gearbox swap with a different ratio calls for reconfiguration, and the trigger's job is to raise the question, not to answer it.

What to log

A model update that cannot be audited is a model that cannot be believed, and the credibility chapter will ask for the log. Write down, for every update, at least:

1. **Which model and which version** – of the code, the mesh, the surrogate's training table.
2. **Which parameters moved, from what to what**, and whether a discrepancy term was learned alongside them.
3. **What data the update used** – which readings, over which interval, with which quality flags from Chapter 5 – and what it deliberately excluded.
4. **What triggered it** – the schedule, a drift indicator's value, or an evolution event.
5. **What the residuals were before and after**, on data the update did not use.
6. **Who or what approved it**, if the update was not automatic.

The log is the middle row of Table 9 made inspectable, and it is what lets a reviewer six months later tell a model that learned something from a model that was quietly bent to fit a broken sensor.

Choosing a Model Type

Five model types, four criteria, and a decision to serve. This section is one table and four questions, and it closes with the turbine's answer.

The table

Table 10 puts the five types against the four criteria, the expertise each needs and the time each takes to run. Read it as a menu, not a ranking: every row is the right answer to some decision and the wrong answer to others, and the National Academies' report says the same in its own words – a twin's representation should be fit for purpose in model type, fidelity and resolution, chosen and often adapted to the decision and the computational constraints, which may mean high-fidelity, simplified or surrogate models or a mixture [23].

Table 10: The five model types against the four criteria, the expertise each needs and its run time. No row passes everything; the twin carries several.

Model type	Answers	Evaluable from readings	Updatable	Time budget	Honest about uncertainty	Expertise needed	Typical run time
lumped first-principles	how much, for a few quantities	yes, from the operational stream	parameters refit, state estimated	passes every pattern	no, unless wrapped	the domain's physics	microseconds a step
discrete-state and discrete-event	which mode; when	yes, from the event log	durations and frequencies re-estimated	passes every pattern	yes, once probabilities are on the arrows	the operation's rules	microseconds an event
high-fidelity field	how much, everywhere	yes, from loads and accelerometers	model updating	fails below certify	yes, once quantified; the most honest of the tower	meshing, a specialist package	minutes to days a run

Model type	Answers	Evaluable from readings	Updatable	Time budget	Honest about uncertainty	Expertise needed	Typical run time
reduced-order and surrogate	the field model's answer, fast	as the field model	rebuilt when the field model is	passes every pattern	reduced: via the estimator; surrogate: only a Gaussian process, and only in its training region	the field model's, plus projection or fitting	microseconds a step
probabilistic, Bayesian	how sure; how long, as a spread	as the model it wraps	it *is* updating	table: instant; sampling: slow schedule	yes, by construction	probability, and the model it wraps	a table, or thousands of evaluations

Four questions

Walk down Table 10 with these, in order; each one removes rows.

1. **How much, or which mode?** A question about quantities wants a rate rule or a field; a question about modes, sequences and waiting wants a state machine, an event queue or a Petri net. If both – which mode, and how much in each – it is a hybrid system, and the mode machine picks the rate rules.
2. **How many states does the decision need?** If a handful of lumped quantities can carry it, write the lumped model and stop; it is the cheapest to write, run and keep current. If the decision is about a body – where a stress is, whether a stiffness has changed – it needs a field, and the field model must at least be built.
3. **Does the time budget allow the model's run time?** Take the decision's latency from Chapter 1 and the model's run time from the table, and compare, with the real-time factor of the appendix. If the field model does not fit, reduce it or replace it, and accept that the field model still has to exist first.
4. **Does the decision need a spread, or will a number do?** Monitoring and diagnosis usually take a number and a threshold. Prediction, decision and anything with the word "when" in it needs a spread, and the answer is to wrap whatever model the first three questions chose in the probabilistic type: parameters as beliefs, readings as updates.

The fourth question is the one most often skipped, and skipping it is how a twin comes to report a date for a bearing that fails four months early with no warning the twin's own numbers could have given.

The turbine's answer

Table 4 promised the reasons. The power and thermal expectation is a how-much question about two quantities on a ten-minute budget: lumped, and the diagnostic residual falls

out of it. Availability and repair planning is a which-mode-and-when question: the mode machine and the vessel's event queue, with weather windows as random events so that the lost-production answer is a spread. Structural integrity is a question about the tower as a body on a seasonal budget: the field model, updated from the accelerometers. Load monitoring every second is the field model's question on a one-second budget: the reduced tower, with the fatigue surrogate for the ten-minute totals. Bearing replacement is a "how long" question with money riding on the confidence: the Wiener model with Bayesian updating of the drift. Five decisions, five model types, one turbine – and the published twins split the same way, the structural twin running a Kalman filter over a linear model of the structure to estimate loads that no sensor measures [3] while the diagnostic twin of the same class of turbine watches its gearbox temperatures for anomalies [4]. A twin is not a model. It is the models a set of decisions needs, kept current at three rates.

Modelling the Plant Demonstrator

The one section where the rig of Chapter 2 appears. It is here to show that the five model types are not reserved for turbines: the rig's own equations already fall onto the first two, the third and fourth are declined for a stated reason, and the fifth is what the rig's residual has been waiting for. Chapter 2's symbols are used unchanged.

A lumped model, already in sampled time. Chapter 2 wrote the pot as two equations: with no dose the moisture falls at the drying rate, and a dose raises it by the dose response times the millilitres delivered. Read them as this chapter reads a model. The state record is one number, the moisture m. The input record is the dose V the twin asked for. The parameter record is the drying rate d and the dose response g. The step function is the prediction rule Chapter 2 already wrote, from one check to the next T_c hours later:

$$m_{k+1} = m_k + g\,V_k - d\,T_c \tag{7}$$

Equation 7 is Equation 1 with the rig's names in it, and it is already an Euler step: Chapter 2's drying rule "moisture falls by d per hour" is a rate rule, and multiplying d by the interval T_c and subtracting is Euler's step with $h = T_c$. The rig's controller, in other words, was running a lumped first-principles model in sampled time from the first chapter that described it; this chapter has only named the parts. The output function is the identity, $z = m$, because the probe reads the state directly – which is the rig's great simplification and the turbine's great complication.

The controller as a state machine. The watering strategies of Chapter 2 are a state machine with four modes: waiting for the next check, reading the probe, deciding a dose, pumping. The guard on the transition from deciding to pumping is $m < m_t$; its action is to compute V by the dose rule and hold the relay for the pump time the stored calibration gives. The transition from pumping back to waiting is timed, by that pump time. Nothing about this needed a queue, because there is one plant and one pump; the moment the rig has two plants sharing one pump, the shared pump is a resource and the Petri net's tokens are the right picture, which is the exercise this chapter declines to set on the rig and sets on a lift instead.

Why no field model of the soil. Moisture in a pot is a field: wetter at the bottom, drier at the surface, drier again near the roots that drink. A field model of the soil would compute

it at every point, and the sensor reads one point. The decision – dose or do not dose, once every six hours, from one probe – cannot use a field's worth of information, and the field model's parameters, the soil's conductivity and the roots' uptake, are not things the rig can measure to calibrate. By the second and third of the four questions the field model is declined: one state carries the decision, and the cost of the model would buy nothing the decision can spend. The published greenhouse twin that is this rig's closest relative makes the same choice, running a simulation model of the greenhouse that calibrates the frequency or duration of watering to how quickly the soil in the pots dries, with an asset model and a time-series database beside it, rather than a model of the soil as a body [32].

Why no surrogate. A surrogate stands in for a model too slow for the decision interval. Equation 7 runs in a microsecond on a six-hour interval. There is nothing to replace, and building a surrogate would add the validity-region warning of the Reduced-Order and Surrogate Models section for no gain.

The dose response as an uncertain parameter. Chapter 2's residual, $r = \hat{m} - m_{\text{obs}}$, was left as a number to watch. This chapter says what to do with it. The dose response g is not known for this pot; it depends on the pot's volume, the soil and the season. Give it three candidates, 0.08, 0.10 and 0.12 percentage points per millilitre, equally likely to begin with. Deliver an 80-millilitre dose and predict the rise under each: 6.4, 8.0 and 9.6 points. Observe a rise of 8.9 points, with the check-to-check scatter, drying included, taken as a Gaussian of standard deviation 1.0. The likelihoods, from the Gaussian density at 8.9, are in the ratio 0.044 to 0.667 to 0.783, and with a flat prior the posterior is 0.03, 0.45 and 0.52: two doses in, the twin already believes g is 0.10 or above and has nearly ruled out 0.08. That posterior is what Chapter 3's Stage 2 – the polling fault detector whose residual a technician acts on – should feed: not a threshold on one residual, but a belief over g that a run of residuals sharpens, and that says, when the belief drifts downward over weeks, that the pot is responding less to each millilitre than it did, which is either roots filling the pot or a pump delivering less than it claims. Which of the two it is, the pump-time model of Chapter 2 and the two candidates for where the residual appears – between doses or at them – can tell apart; the belief is what tells the technician to look.

Faded Examples

Five examples, one per model type, each on the second system chosen for that type, and each with less of the work done for you than the one before: the first is worked in full, the next two leave one step to you, and the last two are posed as problems. None of them is the rig.

Example 1, worked in full: a battery cell as an equivalent circuit

Lumped first-principles. A lithium-ion cell in a bus, whose twin must report how much charge is left and how healthy the cell is. The cheapest model with the physics in it is the **equivalent circuit**: the cell as a voltage source whose voltage depends on the state of charge, in series with a resistor for the instantaneous losses, in series with a resistor-capacitor pair for the slower polarisation that makes the terminal voltage sag over tens of seconds under load. Model-based battery prognostics work on exactly such equivalent-circuit models of the cell voltage, or on physics-based degradation models, and both generalise across cells of one chemistry because they combine known physics with voltage, current and resistance

data [33]. Table 11 names the parts.

Table 11: The battery cell's equivalent-circuit model. The two parameters marked "and health" are the ones a battery twin's digital state carries.

Symbol	Meaning	Unit	Role	Value used
SoC	state of charge, fraction of capacity remaining	–	state	0.800
V_1	voltage across the polarisation pair	volts	state	0
I	current drawn, positive on discharge	amperes	input	3.0
Q	capacity	coulombs	parameter, and health	10800 (three ampere-hours)
R_0	series resistance	ohms	parameter, and health	0.020
R_1, C_1	polarisation resistance and capacitance	ohms, farads	parameters	0.015, 2000
OCV(SoC)	open-circuit voltage at this state of charge	volts	a lookup table, part of the parameters	3.95 at 0.800
V_t	terminal voltage, what the pack's sensor reads	volts	output	–
h	step	seconds	the loop's	1

Three rules from conservation. Charge out of the cell is charge no longer stored, so the state of charge falls at the current over the capacity:

$$\frac{d\,SoC}{dt} = -\frac{I}{Q}$$

The polarisation capacitor stores charge that the current pushes in and its resistor lets leak away, so its voltage rises with the current and decays with its own time constant:

$$\frac{dV_1}{dt} = \frac{I}{C_1} - \frac{V_1}{R_1 C_1}$$

And the output function is the circuit read from its terminals: the open-circuit voltage less the two drops,

$$V_{\mathrm{t}} = \mathrm{OCV}(SoC) - R_0\, I - V_1 \tag{8}$$

One step. With the values of Table 11, the state of charge falls by $3.0/10800 = 0.000278$ in one second, to 0.79972; the polarisation voltage rises by $3.0/2000 = 0.0015$ volts from zero, its decay term being zero at the start; and the terminal voltage by Equation 8 is $3.95 - 0.020 \times 3.0 - 0.0015 = 3.8885$ volts. Take another step and the decay term is no longer zero: V_1 rises by $0.0015 - 0.0015/30$, a little less, and will level off at $I R_1 = 0.045$ volts after a few time constants of thirty seconds. That sag, from 3.89 to about 3.845 volts under a steady three amperes, is what the pack's voltage sensor will show, and the model says why.

Why the parameters are the twin's digital state. Nothing in the step moves Q or R_0; they are parameters. But over months the cell fades and Q falls, and it ages and R_0 rises, and the twin's whole purpose is to know by how much. The twin literature names exactly these – the capacity and the internal resistance as state-of-health parameters of a lithium-ion cell – as the digital state a battery twin estimates through noisy voltage and current readings [24]. In this chapter's terms: the parameters of the lumped model are the beliefs of the probabilistic one, and the Keeping a Model Current section's middle row is where the battery's value lives.

Example 2, one step left to you: a 3D printer's modes

Discrete-state and discrete-event. A desktop 3D printer in a shared workshop, whose twin must report what the printer is doing and how long the queue of jobs will take. Its modes have been published as a Unified Modeling Language (UML) state machine: the printer starts in **StandBy**, becomes **Connected** when a device connects, becomes **Operational** on receiving a print file and preheats before printing, can be paused and resumed or cancelled while printing, returns to StandBy when the job finishes, can enter **Error** from any state, and is reset from Error back to StandBy [34]. That is Figure 2 with different words on the boxes, and drawing it is the worked half: six modes, transitions with guards ("file received", "preheat temperature reached", "job finished", "error raised", "reset").

The step left to you. Extend it in two directions. First, add the queue: jobs arrive at random times with a print duration each, and the printer serves them one at a time. Write the events – job submitted, print started, print finished, error raised, reset – as timestamped entries in a priority queue in the manner of Table 7, and say what the model's state is when the queue holds three jobs and the printer is in Error. Second, make it hybrid: inside the printing mode, the extruder's temperature obeys a rate rule of the gearbox-oil shape – heater power in, loss to the air out – and the guard "preheat temperature reached" fires when that rate rule's state crosses a threshold. The manufacturing-twin review records a generic hybrid automaton of the continuous and discrete-event dynamics of additive processes as precisely this construction [12]. Write the rate rule, name the mode in which it runs, and say which transition its state fires.

Example 3, one step left to you: an industrial oven with a conveyor

High-fidelity field. Steel bars ride a conveyor through a furnace and must leave at a target temperature. A twin of such a furnace has been published for testing: the furnace is divided into five zones of induction heaters, four coils each, with a pyrometer between coils reading the bar temperature, and when the line downstream stalls the furnace drops into a holding mode with reduced coil power and the bars oscillating slowly to hold their temperature [35]. The engineering-design literature describes the model such a furnace's remaining-life twin needs: a computational fluid dynamics model of the temperature distribution in the furnace chamber, a finite-element model of the stress in its weld seams, qualified reduced-order versions of both, and real temperature sensors near the hot parts feeding virtual sensors inside them [36].

Worked half. The field is the temperature at every point of the chamber's gas and the bars' steel; the mesh covers both; the inputs are the coil powers and the belt speed; the parameters are the steel's conductivity and heat capacity and the gas's flow properties; the output function reads the temperature at the pyrometers' spots.

The step left to you. Two judgements a meshing engineer makes, made here in words. First, where must the mesh be fine? Consider where the temperature changes fastest in space – at a bar's surface as it enters a zone, around each coil – and where the decision needs the value – at the pyrometer spots and at the weld seams – and say where you would spend the nodes and where a coarse mesh will do. Second, what does the belt speed do to the field? A faster belt gives each bar less time in each zone, so the same coil power produces a lower exit temperature and a steeper gradient from a bar's surface to its core; say what the holding mode, with its slow oscillation, does to that gradient, and why the twin's operators might prefer it to stopping the belt.

Example 4, posed as a problem: reduced models and a surrogate for capacity fade

Reduced-order and surrogate. Return to the battery. Over hundreds of charge-discharge cycles its capacity Q fades. The battery prognostics literature's earliest models of that fade are **algebraic reduced-order models**: a formula for capacity against cycle number with two or three fitted parameters, assumed to follow an algebraic trajectory such as a power law, which, paired with sampling over its parameters, captures model-form and parameter uncertainty in a way a network ensemble does not [37]. A recent framework goes one step further and fits a machine-learning model that predicts the fade model's parameters from **early-life features** – a few numbers computed from the first hundred cycles – so that a cell's whole capacity trajectory is predicted before most of it has happened, with under two per cent error on cells like the training set and under four on cells unlike it [38].

The problem. (a) Write a two-parameter algebraic fade model, $Q(n) = Q_0\,(1 - a\,n^b)$, with n the cycle number, and say which of this chapter's model types it is and what it reduced – what is the "field model" it stands in for, and what did it keep. (b) The machine-learning model from early-life features to a and b is a surrogate. Say what computational model it mimics, and then apply both warnings of the Reduced-Order and Surrogate Models section to it: what is it *not*, and where is it valid. (c) State what the twin must still hold and update for the pair to count as part of a twin rather than as a calculator.

Example 5, posed as a problem: capacity fade as a Wiener process

Probabilistic, Bayesian updating. The same fade, modelled the way the Probabilistic Models and Bayesian Updating section modelled bearing wear. Capacity falls on average by a fade rate μ per cycle and wanders by σ per square-root cycle; the cell is retired when Q falls below eighty per cent of its rated value.

The problem. (a) Write the step, in the form of Equation 6 with the battery's names. (b) Give μ three candidate values from a fleet – say 0.010, 0.015 and 0.025 per cent of rated capacity per cycle – and a prior, observe a fade of 0.8 per cent over the last fifty cycles with a scatter you choose, and fill in a table of the shape of Table 8. (c) From the posterior, describe the remaining-life distribution in cycles for a cell now at ninety per cent: three centres, their weights, and what σ does to each. (d) Say in one sentence why the answer to (c) is more useful to the bus operator than the single number Example 4's fade model returns, and in a second sentence what Example 4's model gives that this one does not.

Summary

Six items, returning to the six objectives.

1. **Every model is four records and two functions** (objective 1). A state – the smallest record that, with the future inputs, fixes the future – an input, a parameter set and an output, a step function from state, input and interval to next state, and an output function that reads what a sensor would see. Time enters the step three ways, continuous, sampled and event-driven; a step returns a value or a distribution; and a model is white, grey or black by how much physics is written in it. Four criteria judge a model inside a twin: evaluable from this asset's readings, updatable, inside the decision's time budget, honest about uncertainty.
2. **Five model types, chosen per decision** (objective 2). Lumped first-principles for how much about a few quantities; discrete-state and discrete-event for which mode and when; high-fidelity field for how much everywhere; reduced-order and surrogate for the field model's answer inside the time budget; probabilistic with Bayesian updating for how sure. The turbine needed all five, one per decision, and the choosing table's four questions walk from a decision to a row.
3. **A rate rule becomes a loop by Euler's step, and the step size is set by the fastest state** (objective 3). The rotor's speed and the gearbox oil's temperature share one step, so the oil is stepped six hundred times per reading to move a hundredth of a degree; that is what a small step costs, and the appendix's better solvers are the remedy. One Euler step of a linear lumped model is the discrete state-space form, handed to the state and estimator chapter.
4. **A mesh turns a body into a million states; a mode is a shape the body prefers to move in, now computed rather than assumed; a reduced basis is the few modes that carry the motion** (objective 4). The field model is worth paying for when the decision is about the body – a changed stiffness, a stress at a weld – and the reduced model or the surrogate is how its answer fits a one-second budget, with the two warnings: a surrogate mimics a model, not the asset, and is valid only where it was trained.
5. **Degradation is a random walk with drift, remaining life is a distribution, and a reading updates a parameter by Bayes' rule** (objective 5). Prior times likelihood,

normalised: one month's vibration rise moved the bearing's drift from probably slow to probably normal, and the remaining life became a weighted mixture of three centres rather than a date. Model-form discrepancy is the admission that the form is wrong by a smooth unknown amount, learned beside the parameters; the probabilistic graphical model is the whole twin drawn as a chain of such updates over a digital state.

6. **Three rates, three homes** (objective 6). State changes every reading and belongs to the estimator in the state and estimator chapter; parameters change on a slow schedule or a trigger and belong to this chapter, under calibration, with drift indicators and evolution triggers to start an update and a six-line log to audit it; structure changes when the asset changes and belongs to configuration. Verification, calibration and validation are three questions, and the credibility chapter takes the third.

The sentence to carry forward: **a model gives the twin its expectation; the estimator, next, is how the twin corrects that expectation with what the sensors say.**

Exercises

Solutions or hints follow the exercises; each says which. The objective each exercise serves and its difficulty are given in brackets. None of the exercises uses the plant demonstrator.

Exercise 1 (Objective 1, easy). A water tower is filled by a pump and drained by a town's demand. Name its state, inputs, parameters and output, and write its step function for an interval h in the style of the turbine's `step` function. Say which quantity you were tempted to call state and is not, and why.

Exercise 2 (Objective 1, medium). A lift in a ten-storey building serves calls from the floors. Name the state record of a model that answers "how long will I wait", and say which of the three notions of time its step should use and why. Then name the state record of a model that answers "how hot is the motor", and say why the two records share nothing.

Exercise 3 (Objective 3, easy). The gearbox oil's rate rule has a time constant $C_{\text{th}}/k_{\text{th}} = 2000$ seconds. Step it with $h = 1$, $h = 600$ and $h = 3000$ seconds from $T_{\text{oil}} = 48$ towards its steady value of 52 degrees, for one step each, and say what goes wrong at the largest step. Which appendix section explains it?

Exercise 4 (Objective 3, medium). Rewrite the turbine's two-state step as the matrix form Equation 5 with $h = 600$ seconds, compute F, and apply it once to $x = \begin{bmatrix} 1.70 \\ 48.0 \end{bmatrix}$ with $u = \begin{bmatrix} 6.0 \times 10^4 \\ 4.0 \times 10^4 + 1.0 \times 10^3 \times 12 \end{bmatrix}$. Compare the oil temperature you get with six hundred one-second steps of the worked example, and say which is closer to the truth and why.

Exercise 5 (Objective 2, medium). For each decision below, choose a model type and defend it against the four criteria in one sentence each: (a) a data centre's cooling plant must decide every minute whether to start a second chiller; (b) a bridge owner must decide, once a year, whether a deck can carry a new heavier vehicle class; (c) a hospital must estimate how many beds will be free at 08:00 tomorrow.

Exercise 6 (Objective 2, hard). Draw the Petri net for one lift serving two floors whose doors cannot both be open while the car is moving. Name the places, the transitions and the tokens, and say what the marking is when the car is at floor 1 with its door open and a

call is pending at floor 2. Then say, in one sentence, why a finite state machine of the same system needs more modes than the Petri net has places.

Exercise 7 (Objective 4, easy). Explain to a colleague, in three sentences and no equations, what a mesh, a mode and a reduced basis are, using the tower of Figure 3 as the instance.

Exercise 8 (Objective 4, medium). A twin's fatigue surrogate was fitted on runs with wind speeds from 4 to 25 metres per second and wave heights up to 6 metres. A storm arrives with 32 metres per second and 8-metre waves. Say what the surrogate will return, what a Gaussian process surrogate would return that a polynomial would not, and what the twin should do for the duration of the storm.

Exercise 9 (Objective 5, medium). A pump's seal leaks at a rate modelled by Equation 6 with drift μ in millilitres per hour per week and a replacement threshold of 60 millilitres per hour. The current leak is 24 millilitres per hour. Fleet history gives μ three candidates, 3, 6 and 12, with prior 0.4, 0.4 and 0.2. Last week the leak rose by 7 millilitres per hour; take the weekly scatter as a Gaussian of standard deviation 3. Fill in a table of the shape of Table 8 and describe the remaining-life distribution in weeks.

Exercise 10 (Objective 5, hard). Repeat Exercise 9 for a second week in which the leak rose by 2 millilitres per hour, starting from the posterior you found. Then say, in two sentences, why the two-week result is not the same as a single update with a two-week rise of 9, and what that says about the value of frequent readings.

Exercise 11 (Objective 6, easy). Sort the following into the three rows of Table 9, naming the owning chapter for each: the rotor speed; the gear ratio after a gearbox swap; the heat-loss coefficient after a cooler is cleaned; the belief over the turbine's mode; a new accelerometer added at mid-height; the bearing's drift.

Exercise 12 (Objective 6, medium). A colleague proposes that the state estimator be allowed to adjust the rotor inertia J at every reading "so the model always fits". Using the three rates and the verification-calibration-validation distinction, say what will go wrong, which trigger and schedule you would propose instead, and which of the six log entries would let a reviewer catch the problem after the fact.

Solutions and hints

Exercise 1 (solution). State: the water level, or equivalently the volume stored. Inputs: the pump's flow when on, and the town's demand. Parameters: the tank's cross-section, which turns a volume into a level, and the pump's rated flow. Output: the level the tank's float sensor reports. Step: level $+\ h\,(\text{pump flow} - \text{demand})/\text{area}$. The temptation is to call the pump's on-or-off status state; it is an input, decided outside the tank, unless the model includes the controller, in which case it becomes a mode in a hybrid model.

Exercise 2 (hint). For waiting time the state is the car's floor and direction, its door state, and the set of pending calls – discrete things – so the step is event-driven, popping calls and arrivals from a queue. For the motor's heat the state is a temperature and the step is a rate rule in continuous or sampled time. They share nothing because the decisions share nothing; a twin of the lift holds both.

Exercise 3 (solution). The rate is $(52 - 48)/2000 = 0.002$ degrees per second. With $h = 1$ the oil reaches 48.002; with $h = 600$, 49.2, still short of 52 and plausible; with $h = 3000$

it reaches 54, *overshooting* the steady value it can never physically exceed, and the next step would swing back below it. A step longer than the time constant makes Euler's step unstable; the appendix's *From an Equation to a Loop*, under *Worked example 2* and *Better Solvers*, shows it and the implicit step that cures it.

Exercise 4 (hint). With $h = 600$ seconds the two tables are:

$$F = \begin{bmatrix} 1 & 0 \\ 0 & 0.7 \end{bmatrix}, \qquad G = \begin{bmatrix} 1.0 \times 10^{-4} & 0 \\ 0 & 3.0 \times 10^{-4} \end{bmatrix}$$

The second diagonal entry of F is $1 - 600 \times 1000/(2.0 \times 10^6) = 0.7$. One step gives $\omega_r = 1.70 + 600 \times 0.010 = 7.70$ – which is absurd, because the torque imbalance would not stay constant for ten minutes; the controller would act – and $T_{oil} = 0.7 \times 48 + 3.0 \times 10^{-4} \times 5.2 \times 10^4 = 49.2$. Six hundred small steps give about 49.1, and the smaller-step answer is closer because it follows the decaying rate rather than holding the initial rate for the whole interval. The rotor row is the lesson: a step must be short compared with how fast the *inputs* change, too.

Exercise 5 (hint). (a) A lumped thermal model of the plant, possibly wrapped in a mode machine for chiller states: evaluable from the plant's sensors, refit as the plant fouls, microseconds against a one-minute budget; honest about uncertainty only if wrapped. (b) A field model of the deck, updated from accelerometers on a yearly schedule: the decision is about the body, the budget is a year, and the field model's quantified uncertainty is what a certification needs. (c) A discrete-event model of admissions, discharges and transfers with durations from the log, run many times for a spread: which-mode-and- when, cheap, and honest once the durations are distributions.

Exercise 6 (hint). Places: car at 1, car at 2, car moving, door 1 open, door 1 closed, door 2 open, door 2 closed, call pending at 1, call pending at 2. Transitions: open door 1 (needs car at 1 and door 1 closed), close door 1, depart 1 for 2 (needs car at 1, door 1 closed and door 2 closed), arrive at 2, and their mirrors; serve call 2 consumes the pending-call token when the car arrives with the door open. The marking asked for has one token in car at 1, door 1 open, door 2 closed and call pending at 2. A state machine needs one mode per reachable *combination* of car position, both doors and both calls, which multiplies where the Petri net adds.

Exercise 7 (hint). A mesh cuts the tower into small pieces and keeps a number at each corner, so that the tower's motion is described by many numbers instead of one. A mode is a way the tower likes to sway or twist, with its own tempo, and a tall tower has a few slow ones that carry almost all its motion. A reduced basis keeps only those few, so the twin tracks a handful of tempos instead of a million corners and can still rebuild the whole picture from them.

Exercise 8 (solution). A polynomial surrogate will return a number extrapolated from the edge of its fitted region, with no sign that it is outside it; the number is untrustworthy in either direction. A Gaussian process will return a number with a spread that has grown large, signalling that it does not know. Either way the twin should flag the inputs as out of the validity region, stop adding the surrogate's fatigue increments to the running total for the storm, and fall back to the reduced model driven by the accelerometers, or to a field-model run afterwards on the recorded loads, to fill in the storm's damage.

Exercise 9 (solution). Likelihoods of a rise of 7 with standard deviation 3: for $\mu = 3$, the

reading is 1.33 deviations away, density about 0.055; for 6, 0.33 away, about 0.126; for 12, 1.67 away, about 0.033. Prior times likelihood: 0.022, 0.050, 0.007; sum 0.079; posterior 0.28, 0.64, 0.08. Distance to the threshold is 36 millilitres per hour, so the centres are 12, 6 and 3 weeks with weights 0.28, 0.64 and 0.08: most likely about six weeks, a real chance of twelve, a small chance of three, each centre widened by the scatter.

Exercise 10 (hint). Start from the posterior 0.28, 0.64, 0.08 as the new prior; a rise of 2 favours the slow candidate. The two-week result differs from one update on a rise of 9 because the two-week scatter is not 3 but $3\sqrt{2}$, and because two readings constrain the *path* – a fast week then a slow one is evidence against a steady fast drift that a single two-week total cannot see. Frequent readings buy shape, not just total.

Exercise 11 (solution). State, the estimator's chapter: the rotor speed; the belief over the mode. Parameters, this chapter: the heat-loss coefficient after cleaning; the bearing's drift. Structure, the configuration and manager chapter: the gear ratio after a gearbox swap, because the constraint changes and so may the model; the new accelerometer, because the output function gains a row.

Exercise 12 (hint). An inertia that moves every reading will absorb sensor noise, a stuck speed counter and controller transients into a physical constant that cannot change on that time scale, so the model "fits" and predicts nothing; the state should be estimated every reading and J refit on a schedule, or on a drift indicator over the speed residual, with the residual checked on held-out data before the new value is accepted. Log entries 2 and 5 – which parameters moved, and the residuals before and after on data the update did not use – are what a reviewer would use to see that J had been chasing noise.

Further Reading

Grouped by model type, saying where a worked example is the book's own construction rather than a source's.

- *What every model has:* [1] for the white, grey and black box sorting as the twin literature uses it; [2] for reading the twin itself as the grey position between model-based and vibration-based monitoring. The four-records-two-functions view, the three notions of time and the four criteria are the book's own framing.
- *The turbine:* [3] for the physics-based twin of a floating turbine validated on a full-scale prototype, whose estimator and virtual sensing the state and estimator chapter will return to; [4] for the diagnostic twin of an operational floating turbine and the SCADA signals it reads; [5] for the offshore-structures programme of predictive modelling; [6] for what SCADA data looks like at farm scale. The six-line rotor-and-oil model, its numbers and the vessel-dispatch table are the book's own.
- *Lumped first-principles:* [7] for lumped-parameter models, the bond-graph analogy across domains and the acausal style, at book length; [8] for Modelica models reaching a twin as compiled units in a DevOps pipeline; [9] for a whole system decomposed into lumped component models joined by constraints; [10] for an equation-based building model coupled to a sequential simulator and what each side gives up.
- *Discrete-state and discrete-event:* [7] again, for DEVS and Petri nets with worked code; [11] for choosing a formalism as a design stage; [12] for hidden Markov models and hybrid automata of machine states in manufacturing twins; [13] and [14] for discrete-event twins of production and of a hospital.

- *High-fidelity field:* [5] and [2] for finite-element model updating of offshore structures from measured modal parameters; [15] for the aircraft wing calibrated on its first two bending modes; [16] for building twins generated from the building information model; [17] for model updating from operational modal analysis on a twin platform; [12] for the finding that fidelity's computational price often makes such models unsuitable inside a twin.
- *Reduced-order and surrogate:* [18] for what a surrogate is and the argument that a surrogate alone is not a twin; [19] for reduced models with a state estimator as virtual sensors, including the turbine blade; [20] for a library of component reduced models; [21] for interpretable, reusable reduced models in manufacturing; [22] for a multi-fidelity surrogate inside Bayesian updating; [23] for fit-for-purpose fidelity and the open problem of adapting surrogates under time constraints. The fatigue surrogate is the book's own construction.
- *Probabilistic and Bayesian:* [24] for the digital state and the stochastic-process degradation models; [26] for Bayesian updating, the Kennedy and O'Hagan discrepancy framework and the sampling methods named here and taught nowhere in this book; [25] for remaining useful life in predictive-maintenance twins; [15] and [27] for the probabilistic graphical model of a twin, the figure this chapter redrew in its own terms. The three-candidate table and the bearing's numbers are the book's own.
- *Keeping a model current:* [28] for architectural self-adaptation when the asset changes stage; [29] for evolving a twin's schema and data together; [30] for verification and validation as a continuous process; [24] for drift-based updating; [31] for evolution triggers. The three-rates table and the six-line log are the book's own.
- *The faded examples:* [33] and [24] for the battery's equivalent circuit and its health parameters as digital state; [34] for the 3D printer's published state machine; [36] and [35] for the furnace's field model and its zones and coils; [37] and [38] for algebraic capacity-fade models and the early-life surrogate. The battery's circuit values and the worked step are the book's own.
- *The rig:* [32] for the published greenhouse twin whose watering calibration is the rig's closest relative.

References

[1] O. J. Pinon Fischer, S. Sabri, and Y. Chen, "Fundamentals of Digital Twins, Modeling Approaches, and Governance," in *Digital Twin: Fundamentals and Applications*, S. Sabri, K. Alexandridis, and N. Lee, Eds., Cham: Springer Nature Switzerland, 2024, pp. 13–29. doi: 10.1007/978-3-031-67778-6_2.

[2] H. Pezeshki, H. Adeli, D. Pavlou, and S. C. Siriwardane, "State of the art in structural health monitoring of offshore and marine structures," *Proceedings of the Institution of Civil Engineers - Maritime Engineering*, vol. 176, no. 2, pp. 89–108, Apr. 2023, doi: 10.1680/jmaen.2022.027.

[3] E. Branlard, J. Jonkman, C. Brown, and J. Zhang, "A digital twin solution for floating offshore wind turbines validated using a full-scale prototype," *Wind Energy Science*, vol. 9, no. 1, pp. 1–24, Jan. 2024, doi: 10.5194/wes-9-1-2024.

[4] F. Stadtmann and A. Rasheed, "Diagnostic Digital Twin for Anomaly Detection in Floating Offshore Wind Energy." arXiv, Jun. 2024. doi: 10.48550/arXiv.2406.02775.

[5] U. T. Tygesen, K. Worden, T. Rogers, G. Manson, and E. J. Cross, "State-of-the-Art and Future Directions for Predictive Modelling of Offshore Structure Dynamics Using Machine Learning," in *Dynamics of Civil Structures, Volume 2*, S. Pakzad, Ed., Cham: Springer International Publishing, 2019, pp. 223–233. doi: 10.1007/978-3-319-74421-6_30.

[6] L. A. Bull *et al.*, "Data-Centric Monitoring of Wind Farms: Combining Sources of Information," in *Data Driven Methods for Civil Structural Health Monitoring and Resilience*, CRC Press, 2023.

[7] P. Carreira, V. Amaral, and H. Vangheluwe, Eds., *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer Nature, 2020. doi: 10.1007/978-3-030-43946-0.

[8] J. Hugues, J. Yankel, J. Hudak, and A. Hristozov, "Twinops: Digital twins meets devops," *CARNEGIE-MELLON UNIV PITTSBURGH PA, Tech. Rep.*, 2022, Accessed: Jan. 08, 2025. [Online]. Available: https://apps.dtic.mil/sti/trecms/pdf/AD1168471.pdf

[9] D. Shangguan, L. Chen, C. Su, J. Ding, and C. Liu, "A Triple Human-Digital Twin Architecture for Cyber-Physical Systems," *Computer Modeling in Engineering & Sciences*, vol. 131, no. 3, pp. 1557–1578, 2022, doi: 10.32604/cmes.2022.018979.

[10] M. Wetter, K. Benne, H. Tummescheit, and C. Winther, "Spawn: Coupling Modelica Buildings Library and EnergyPlus to enable new energy system and control applications," *Journal of Building Performance Simulation*, vol. 17, no. 2, pp. 274–292, Mar. 2024, doi: 10.1080/19401493.2023.2266414.

[11] G. Lugaresi and H. Vangheluwe, *From Digital Twins to Twinning Systems*. 2025.

[12] T. Böttjer *et al.*, "A review of unit level digital twin applications in the manufacturing industry," *CIRP Journal of Manufacturing Science and Technology*, vol. 45, pp. 162–189, Oct. 2023, doi: 10.1016/j.cirpj.2023.06.011.

[13] A. Hakiri, A. Gokhale, S. B. Yahia, and N. Mellouli, "A comprehensive survey on digital twin for future networks and emerging Internet of Things industry," *Computer Networks*, vol. 244, p. 110350, May 2024, doi: 10.1016/j.comnet.2024.110350.

[14] M. Dalibor *et al.*, "A Cross-Domain Systematic Mapping Study on Software Engineering for Digital Twins," *Journal of Systems and Software*, vol. 193, p. 111361, Nov. 2022, doi: 10.1016/j.jss.2022.111361.

[15] M. G. Kapteyn, J. V. R. Pretorius, and K. E. Willcox, "A probabilistic graphical model foundation for enabling predictive digital twins at scale," *Nature Computational Science*, vol. 1, no. 5, pp. 337–347, May 2021, doi: 10.1038/s43588-021-00069-0.

[16] P. Zech, C. Nardin, S. Ristov, M. Flora, and R. Breu, "Digital-Twins-as-a-Service in Construction Engineering," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, Aug. 2024, pp. 3004–3010. doi: 10.1109/CASE59546.2024.10711409.

[17] N. Wagner and D. Tcherniak, *Data-Driven Updating of Digital Twins through Experimental Measurements and Parametric Finite Element Model Optimization*. 2025.

[18] E. VanDerHorn and S. Mahadevan, "Digital Twin: Generalization, characterization and implementation," *Decision Support Systems*, vol. 145, p. 113524, Jun. 2021, doi: 10.1016/j.dss.2021.113524.

[19] D. Hartmann and H. Van der Auweraer, *The Executable Digital Twin: Merging the digital and the physics worlds*. 2022.

[20] M. G. Kapteyn, D. J. Knezevic, and K. Willcox, "Toward predictive digital twins via component-based reduced-order models and interpretable machine learning," in *AIAA Scitech 2020 Forum*, in AIAA SciTech Forum., American Institute of Aeronautics; Astronautics, 2020. doi: 10.2514/6.2020-0418.

[21] V. Zambrano *et al.*, *Interpretable and Reusable Reduced Order Models for Digital Twins in Manufactory as a Service*. 2023. doi: 10.23967/admos.2023.070.

[22] M. Torzoni, A. Manzoni, and S. Mariani, "A Deep Neural Network, Multi-fidelity Surrogate Model Approach for Bayesian Model Updating in SHM," in *European Workshop on Structural Health Monitoring*, P. Rizzo and A. Milazzo, Eds., Cham: Springer International Publishing, 2023, pp. 1076–1086. doi: 10.1007/978-3-031-07258-1_108.

[23] *Foundational Research Gaps and Future Directions for Digital Twins*. Washington, D.C.: National Academies Press, 2024. doi: 10.17226/26894.

[24] A. Thelen *et al.*, "A comprehensive review of digital twin — part 1: Modeling and twinning enabling technologies," *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, p. 354, Nov. 2022, doi: 10.1007/s00158-022-03425-4.

[25] D. Zhong, Z. Xia, Y. Zhu, and J. Duan, "Overview of predictive maintenance based on digital twin technology," *Heliyon*, vol. 9, no. 4, p. e14534, Apr. 2023, doi: 10.1016/j.heliyon.2023.e14534.

[26] A. Thelen *et al.*, "A comprehensive review of digital twin—part 2: Roles of uncertainty quantification and optimization, a battery digital twin, and perspectives," *Structural and Multidisciplinary Optimization*, vol. 66, no. 1, p. 1, Dec. 2022, doi: 10.1007/s00158-022-03410-x.

[27] M. Torzoni, M. Tezzele, S. Mariani, A. Manzoni, and K. E. Willcox, "A digital twin framework for civil engineering structures," *Computer Methods in Applied Mechanics and Engineering*, vol. 418, p. 116584, Jan. 2024, doi: 10.1016/j.cma.2023.116584.

[28] E. Kamburjan, N. Bencomo, S. L. Tapia Tarifa, and E. B. Johnsen, "Declarative Lifecycle Management in Digital Twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, Linz Austria: ACM, Sep. 2024, pp. 353–363. doi: 10.1145/3652620.3688248.

[29] D. Lehner, A. Garmendia, and M. Wimmer, "Towards Flexible Evolution of Digital Twins with Fluent APIs," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021, pp. 1–4. doi: 10.1109/ETFA45728.2021.9613222.

[30] Z. Ali, R. Biglari, J. Denil, J. Mertens, M. Poursoltan, and M. K. Traoré, "From modeling and simulation to Digital Twin: Evolution or revolution?" *SIMULATION*, vol. 100, no. 7, pp. 751–769, Jul. 2024, doi: 10.1177/00375497241234680.

[31] T. Alskaif *et al.*, "Evolution at the Core of Digital Twin Engineering," Grand Rapids, USA: IEEE, Jul. 2025. doi: 10.5283/epub.77656.

[32] E. Kamburjan *et al.*, "GreenhouseDT: An Exemplar for Digital Twins," in *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, in SEAMS '24. New York, NY, USA: Association for Computing Machinery, Jun. 2024, pp. 175–181. doi: 10.1145/3643915.3644108.

[33] A. Thelen, M. Li, C. Hu, E. Bekyarova, S. Kalinin, and M. Sanghadasa, "Augmented model-based framework for battery remaining useful life prediction," *Applied Energy*, vol. 324, p. 119624, Oct. 2022, doi: 10.1016/j.apenergy.2022.119624.

[34] D. Lehner, S. Sint, M. Eisenberg, and M. Wimmer, “A pattern catalog for augmenting Digital Twin models with behavior,” *at - Automatisierungstechnik*, vol. 71, no. 6, pp. 423–443, Jun. 2023, doi: 10.1515/auto-2022-0144.

[35] Y. Ma *et al.*, “Automated and Systematic Digital Twins Testing for Industrial Processes.” arXiv, Feb. 2023. doi: 10.48550/arXiv.2302.13198.

[36] N. Anwer, R. Stark, F. Tao, and J. Erkoyuncu, “Developing and leveraging digital twins in engineering design,” *CIRP Annals*, vol. 2025, Jun. 2025, doi: 10.1016/j.cirp.2025.05.002.

[37] A. Thelen, X. Huan, N. Paulson, S. Onori, Z. Hu, and C. Hu, “Probabilistic machine learning for battery health diagnostics and prognostics—review and perspectives,” *npj Materials Sustainability*, vol. 2, no. 1, pp. 1–33, Jun. 2024, doi: 10.1038/s44296-024-00011-1.

[38] T. Li *et al.*, “Coupling a capacity fade model with machine learning for early prediction of the battery capacity trajectory,” *Applied Energy*, vol. 389, p. 125703, Jul. 2025, doi: 10.1016/j.apenergy.2025.125703.