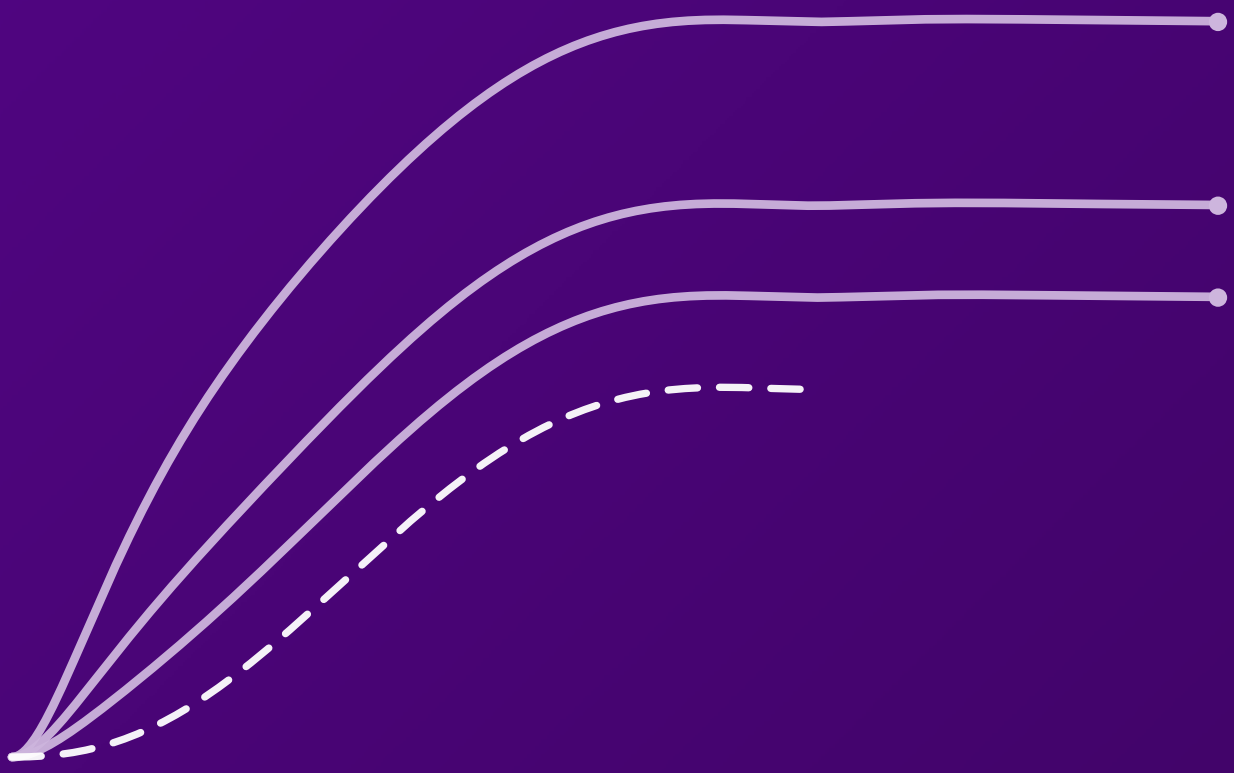
DMP 101

A friendly, intuition-first guide to dynamic movement primitives



Fares J. Abu-Dakka
Assistant Professor of Mechanical Engineering
Robot Learning & Control Lab (REACH Lab), New York University Abu Dhabi

REACH
ROBOT LEARNING AND CONTROL LAB

جامعة نيويورك أبوظبي
NYU | ABU DHABI

DMP 101

A friendly, intuition-first guide to dynamic movement primitives

Fares J. Abu-Dakka

Assistant Professor of Mechanical Engineering
Robot Learning & Control Lab (REACH Lab), New York University Abu Dhabi

Table of contents

Welcome 1

Chapter 1 · Meet the DMP: a recipe for movement 3

Chapter 2 · The spring that always gets home 8

Chapter 3 · A clock that runs down 14

Chapter 4 · Bumps that bend the path: the forcing term 19

Chapter 5 · Learning from one demonstration 25

Chapter 6 · New goal, new speed 33

Chapter 7 · Round and round: the periodic DMP 39

Chapter 8 · Teaching a rhythm 48

Chapter 9 · Where to go next 54

Sources 58

Welcome

Imagine showing a robot how to wave just once, and then watching it wave again somewhere new, quicker or slower, larger or smaller, without ever needing a second lesson. That small bit of magic is what a **dynamic movement primitive** (DMP) makes possible, and along the way through this book, you will discover how it works.

Bring your Python and your curiosity; you can join us even if you have never taken a control course. We will meet each new word as it comes along, starting every chapter beside a robot and exploring every idea in a notebook you can run and break.

- **Read online**
  Begin the journey with Chapter 1.
- **Download the PDF**
  The same book, for reading offline or printing.
- **Open the notebooks**
  Explore them right in your browser, with nothing to install.
- **Download the notebooks as a zip**
  The three notebooks and the `dmpkit` code, to run on your own computer.

As you explore, your notebook work stays in your browser on this computer; the server does not keep it.
Download your notebooks before you clear your browser data or switch computers, or you will lose your changes.

The three notebooks

- 01 · The discrete playground: begin with a motion you load or draw, and once you have fitted it, explore what happens when you move the goal and the start or change the speed.
- 02 · The periodic playground: teach the robot a rhythm, and then watch how the beat changes with its frequency while the swing grows or shrinks with its size.
- 03 · Create your own: give the robot something of your own to do, from writing your initials to waving, and along the way, break a DMP on purpose.

How to read this book

As we walk through each chapter, the same four kinds of box will keep you company:

- **Intuition** gives you a picture to hold in your head before the symbols arrive.
- **The math** puts that picture into equations, with each one followed by a sentence that reads it aloud.
- **Try it** takes you to a notebook and the exact cell to open, with a little guidance on what to look for.
- **Gotcha** walks you through the chapter's one trap, showing what goes wrong, why it happens, and how to find your way out.

To help you find your bearings in equations, each idea keeps one color everywhere: the goal g, the spring–damper term $\alpha_z(\beta_z(g - y) - z)$, the forcing term $f(x)$, the phase x, the basis functions Ψ_i, the demonstration y_d and the reproduction y. When we move from equations to plots, a second cue, such as a dashed line for the demonstration, helps you follow the story, so you never have to rely on color alone.

The model equations you will meet come from one survey paper and three older ones, all listed on the Sources page, so you can follow their numbers back to where the story began.

01

CHAPTER ONE

Meet the DMP: a recipe for movement

You will meet the spring, push and clock that let a robot repeat one motion toward new goals.

Imagine guiding a robot arm by hand as it lifts a cup onto a shelf, showing it a graceful curve just once. Then someone moves the shelf 20 cm to the left, and the arm carries the cup to the new spot with the same smooth motion, without another lesson. How does it do that?

The answer is a small recipe called a *dynamic movement primitive*, or DMP, whose three ingredients we will meet along the way. By the end of this chapter, you will be able to describe that recipe in one sentence.

The picture

Picture yourself walking a dog home, with the leash gently drawing it toward your front door. That pull is the **spring**: it always points home, and the farther the dog wanders, the harder it pulls.

Along the way, a puddle spreads across the sidewalk, so you give the leash a little sideways tug to steer the dog around it. That tug is the **push**, shaping the route you take while leaving your destination alone.

Because the puddle is always a few minutes into the walk, a glance at your watch tells you when the tug is coming. Your watch is the **clock**.

Now swap the dog for a robot hand and the front door for a shelf, and you have the picture behind a DMP. In Figure 1, those same three parts become boxes that feed one moving dot.

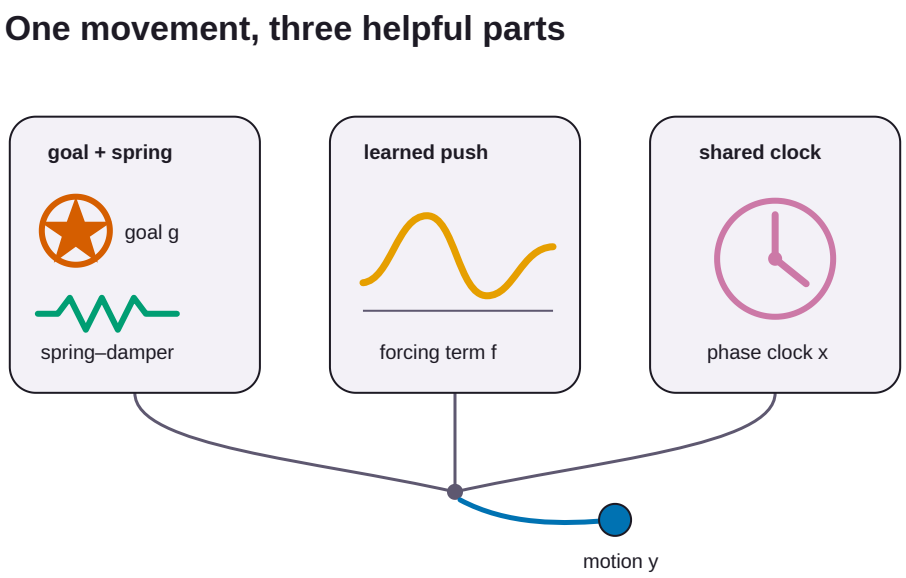


Figure 1. The three parts of a DMP: a goal with its spring, a learned push, and a clock, all feeding one moving dot.

The idea

With that walk in mind, we can give each part a job in the robot's motion. The **spring** pulls the hand toward a **goal**, the place where the motion should end, and makes sure it arrives. The **push** is the learned nudge you taught by hand, bending the path into the shape you showed.

Meanwhile, the **clock** slowly runs down from 1 toward 0, telling the push when to act and letting it fade out as the motion comes to an end.

Put those ingredients together and you have your one-sentence recipe: a DMP is a spring that pulls toward a goal, plus a learned push that shapes the path, plus a clock that tells the push when to act.

> ☼ **Intuition: the goal lives in the spring**
> The push stores the shape of the motion, while the spring stores the place where it ends, which means you can move the goal without touching what you taught. The spring pulls somewhere new, and the push keeps bending the path in the way you showed.

Now the shelf puzzle has an answer: once the robot has learned the push, moving the shelf only changes the goal.

You can follow that story in Figure 2, where the dashed demonstration ends at a goal of 1.0. The three solid lines replay it toward three new goals, each marked with a star, and every path reaches its star even though the robot was never shown those destinations.

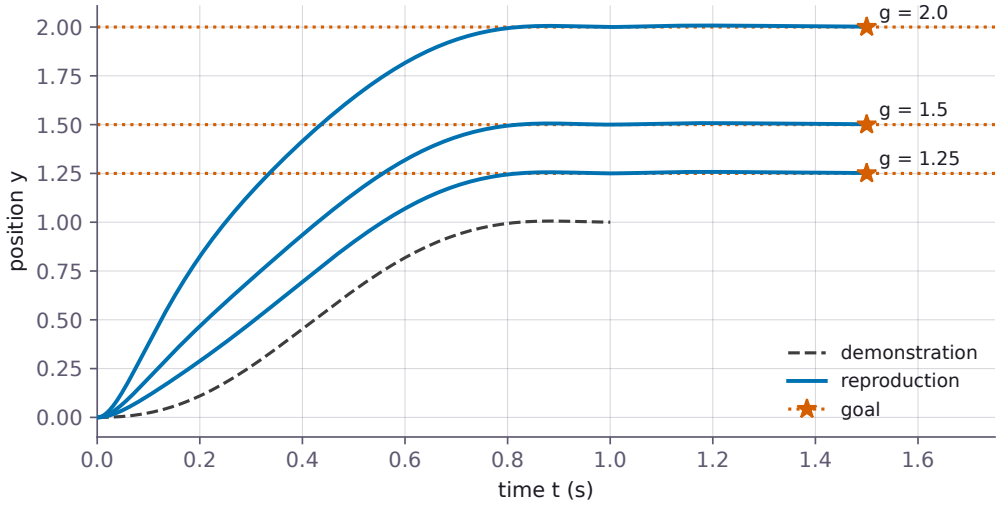


Figure 2. One demonstration (dashed) and three reproductions (solid) to the new goals 1.25, 1.5 and 2.0, each marked with a star.

The math

Each of our three ingredients has a short equation, and here you get a first glimpse of them together. We will open these boxes one by one in the coming chapters, so you can let the details wait.

> ☼ **Intuition: a spring with a nudge**
> Picture a ball attached to a spring hooked to the goal: when you let go, it swings home. Add a gentle hand to nudge it along the way, and you have the picture that the three equations below put into words and symbols.

Before we look inside, let's give those moving parts their names: the robot's position is y. Its helper variable z is the velocity scaled by the time constant τ, which we will explore in Chapter 2. The destination is our goal g, while the clock value is the *phase* x, starting at 1 and decaying toward 0. Along the route, the push $f(x)$ shapes the motion.

> ∑ **The math: eq. (1)–(3)**
>
> $$\tau\,\dot{z} = \underbrace{\alpha_z(\beta_z(g - y) - z)}_{\text{spring–damper term}} + \underbrace{f(x)}_{\text{forcing term}} \tag{1}$$

This reads: the change of the scaled velocity is a spring pull toward the goal, plus the push.

$$\tau\,\dot{y} = z \tag{2}$$

This reads: the position changes at a rate given by the scaled velocity.

$$\tau\,\dot{x} = \underbrace{-\,\alpha_x\,x}_{\text{the clock runs down}} \tag{3}$$

This reads: the clock runs down on its own, a little slower each moment, from 1 toward 0.

When you read about DMPs, you will meet equations (1) and (2) together as the *transformation system*, and equation (3) as the *canonical system*. These are the standard names in the DMP literature: the survey that this book follows uses them (S23 10.1177/02783649231201196, section 2.1.1, page 1137), and they go back to Ijspeert and colleagues (IJ13 10.1162/NECO_a_00393, pp. 332–333).

The colored chips in eq. (1) give you a trail to follow: green marks the spring, orange the push, and pink on τ the clock, as in eq. (3). The chip on g marks the goal, and the blue ones mark the motion itself, y and z. You can keep following those same colors through every figure and equation in the book.

Before we move on, picture how this recipe fits a whole robot: a DMP drives one joint or one coordinate, called one *degree of freedom*. When the robot has several, you run several DMPs that share one clock (S23 p. 1137).

Try it

The notebook already holds a trained DMP, so you can step into the shelf story yourself by moving the goal and watching the path follow.

▶ Try it: 01_discrete_playground.ipynb › Simulate 1.4 - Goal, start, tau and number of basis functions
As you drag the goal slider g, watch the path follow its new destination without another lesson. The printed line tells you where the motion ends, within a few thousandths of g each time. Leave the other sliders for now; we will meet them as the story unfolds.

Gotcha

> **⚠ Gotcha: a DMP is not a motor command**
> Symptom: you plug the DMP output into a motor and the robot jerks, or nothing moves. Cause: the DMP plans a motion by producing a desired position, velocity and acceleration; it has no motor commands to send. This trips up everyone the first time. Fix: pass those three numbers to a separate controller, which turns them into motor commands (IJ13 p. 332, 10.1162/NECO_a_00393).

◆

02

CHAPTER TWO

The spring that always gets home

You will see why a robot glides home without wobbling, and how three knobs shape its arrival.

Your robot arm is holding a pose when you give its hand a push and let go. The hand glides home and stops, without wobbling, bouncing or creeping toward its destination. That neat homecoming comes from the part of a DMP we will meet in this chapter: a spring with a brake.

The picture

Think of the little arm at the top of a heavy door, drawing it shut after you walk through. Inside that door closer, a spring pulls while a damper, a piston full of oil, slows the movement. Without the spring the door would hang open; without the damper it would slam, bounce back and rattle in its frame.

Carry that picture over to the robot hand: the “door” is its position y, and “shut” is the goal g. As a spring pulls y toward g, a damper fights the speed and softens the arrival.

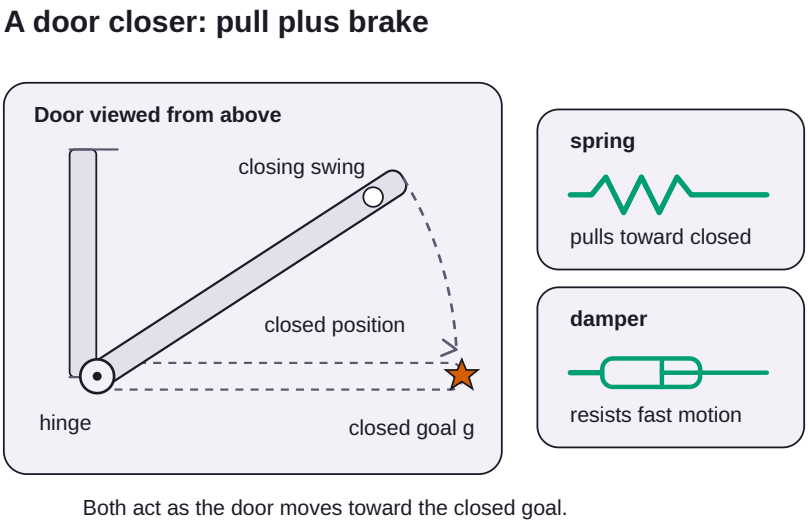


Figure 1. A door with a spring that pulls it shut and a damper that slows it down, each part labeled.

The idea

As the hand travels home, the spring's pull changes with the distance still to go: strong far away, gentle close by, and zero at the goal. The damper responds to speed instead, braking hard when the hand moves quickly and offering no brake when it stands still.

Working together, they let the hand start quickly, slow as it nears the goal, and settle into place. How much brake you add decides the style of that arrival.

- With **too little brake**, called *under-damped*, the hand overshoots the goal, swings back, and rings like a diving board.
- With **just enough brake**, called *critically damped*, the hand slides into place and stops without overshooting.
- With **too much brake**, called *over-damped*, the hand creeps toward the goal and takes a very long time to arrive.

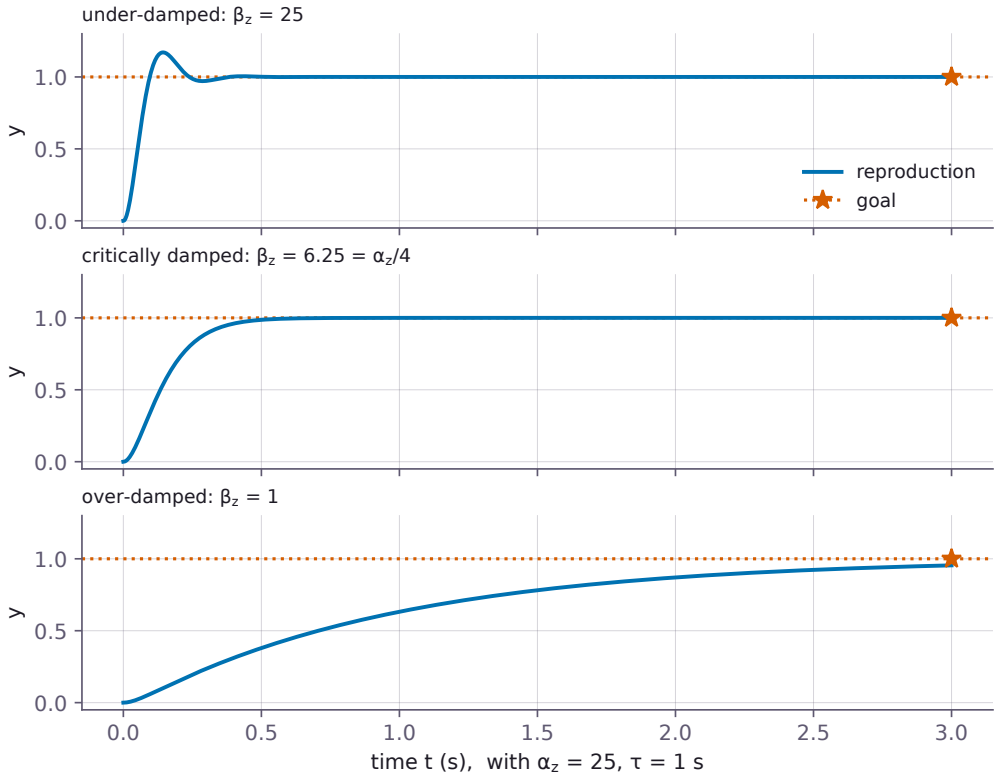


Figure 2. Three stacked panels with the spring alone: an under-damped path that overshoots and rings, a critically damped path that slides in, and an over-damped path that is still short of the goal after 3 seconds.

The math

> ☼ **Intuition: a spring with a brake**
> Imagine the spring saying "go to the goal, harder when far", while the damper answers "slow down, harder when fast". Critical damping is the balance that lets the hand arrive as fast as it can without crossing the goal.

In Chapter 1, you met the full DMP, with a push $f(x)$ on top of the spring. For now we switch that push off, setting $f = 0$, so we can watch the spring and damper on their own. Together they form the *spring–damper term*, the middle piece of eq. (1), and the robot's state is described by two numbers: position y and an auxiliary variable z.

Σ The math: eq. (1)–(2)

$$\tau \dot{z} = \underbrace{\alpha_z(\beta_z(g - y) - z)}_{\text{spring–damper term}} + \underbrace{f(x)}_{\text{forcing term}} \tag{1}$$

Read it aloud: the clock-scaled rate of change of z equals the spring pull toward the goal, minus the damper brake on z, plus the forcing push (which is zero in this chapter).

$$\tau\,\dot{y} = z \tag{2}$$

Read it aloud: the clock-scaled rate of change of the position is z, so z is the speed, scaled by τ.

To get a feel for z, look back at equation (2), where z is the speed multiplied by τ. That is why you will often hear z called the *scaled velocity*.

∑ The math: eq. (E2)

$$z = \tau\,\dot{y} \quad \text{(discrete)}, \qquad z = \frac{\dot{y}}{\Omega} \quad \text{(periodic)} \tag{E2}$$

Read it aloud: z is just τ times the speed of the hand.

We can find the right amount of brake by multiplying out the spring–damper term of eq. (1). The pull toward the goal carries the factor $\alpha_z\beta_z$, the spring's stiffness, while the brake on z carries α_z, the damper's strength. This means that changing α_z changes both spring and damper, whereas changing β_z changes only the spring. To get the critically damped balance, the two gains must satisfy $\alpha_z = 4\beta_z$. The kit achieves this with defaults $\alpha_z = 25$ and $\beta_z = 6.25$, or $25/4$.[1]

Follow the hand through the three-panel plot, where α_z stays at 25, keeping the damper the same while only the spring changes. In the top panel, $\beta_z = 25$ is four times the balanced value $\alpha_z/4$, making the spring too stiff for the damper, so the hand overshoots and rings. In the bottom panel, $\beta_z = 1$ makes the spring so soft that the damper wins, leaving the path short of the goal after 3 seconds.

Now imagine turning a slow-motion knob: that is the parameter τ. Double τ and the same path takes twice as long; halve it and the robot moves twice as fast, keeping the shape of its motion throughout.

[1] The defaults $\alpha_z = 25$ and $\beta_z = \alpha_z/4$ are the values IJ13 uses throughout (Table 1, p. 349, 10.1162/NECO_a_00393). S23 states the condition $\alpha_z = 4\beta_z$ (p. 1137, citing IJ13) but gives no numbers.

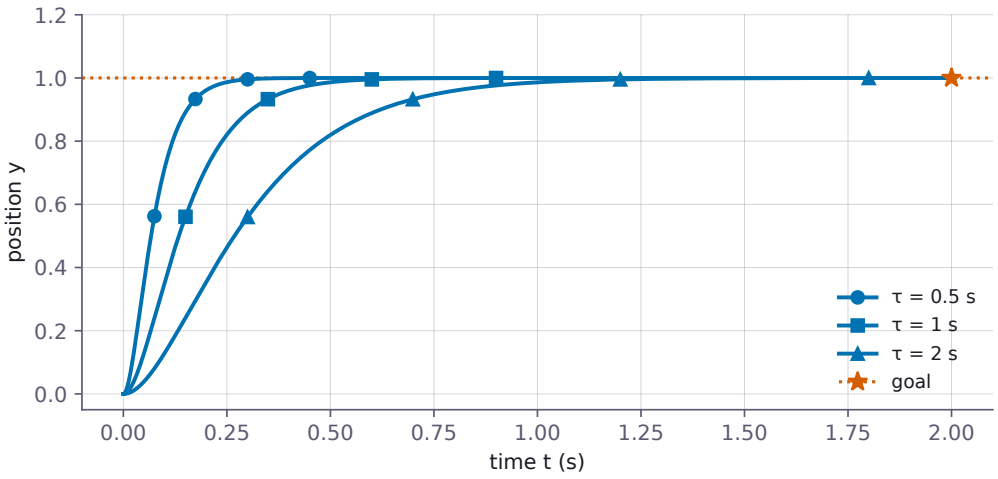


Figure 3. The critically damped spring alone for τ equal to 0.5, 1 and 2 seconds: the same path, stretched in time.

☼ **Intuition: a computer takes small steps**

A computer follows the smooth motion by taking tiny steps in time, asking at each one, "how fast is this value changing right now?" It then moves a little in that direction, building the path step by step.

Σ The math: eq. (E4)

$$s(t + \Delta t) = s(t) + \Delta t \cdot \dot{s}(t), \qquad s \in \{y, z, x, \phi\} \tag{E4}$$

Read it aloud: the new value equals the old value plus the time step Δt times the current rate of change.

This way of advancing the motion is called the explicit Euler step: first use (1) and (2) to find the rates of change of z and y, then apply the step above to both. In the figures and most notebook cells, $\Delta t = 0.001$ s makes one second of motion into 1,000 steps. You can follow every one of them in the kit's loop, written out in plain NumPy.

Try it

▶ Try it: 03_create_your_own.ipynb › Create 3.6 - Break it

Start with `BREAK = dict(beta_z_run=6.25)` and run the cell to watch the replay follow the demonstration at the balanced value it learned with. The printed overshoot is a fraction of the trip: 0.0079, below 1 %, and that small step past the goal is already in the demonstration.

Then try 25, followed by 100, and watch the hand's arrival change. At 25 the overshoot grows only a little, to 0.0108, but at 100 it reaches 0.3945: about 40 % past the

goal. The weights stay the same throughout; only β_z changes, making the spring stiffer while the damper stays the same.

Gotcha

⚠ Gotcha: break the 4-to-1 balance and the robot overshoots
Symptom: the hand swings past the goal and comes back, perhaps making the trip more than once. Cause: β_z is larger than $\alpha_z/4$, giving the spring more strength than the damper can handle. This trips up everyone, because a bigger number can feel like "more precise". Fix: keep $\alpha_z = 4\beta_z$ after learning, and remember that on a real robot an overshoot can mean hitting the shelf.

◆

03

CHAPTER THREE

A clock that runs down

You will discover why a DMP keeps its own clock and how to make it run faster or slower.

Picture a robot halfway through a motion when someone gently holds its hand, then lets go. You want it to carry on from where it is, without leaping ahead to wherever the wall clock says it should be. That calls for a clock of its own, one it can read whenever it needs to know how far through the motion it has come. In this chapter we build that clock, the smallest piece of a DMP and the one everything else hangs on.[1]

The picture

Imagine a candle with marks painted along its side, starting tall and burning down quickly before its descent slows. In our picture, its height keeps shrinking toward the table without ever quite reaching it.

[1] How a DMP can wait for a held hand is described in 10.1177/02783649231201196, section 2.1.1.2. This book only points to it.

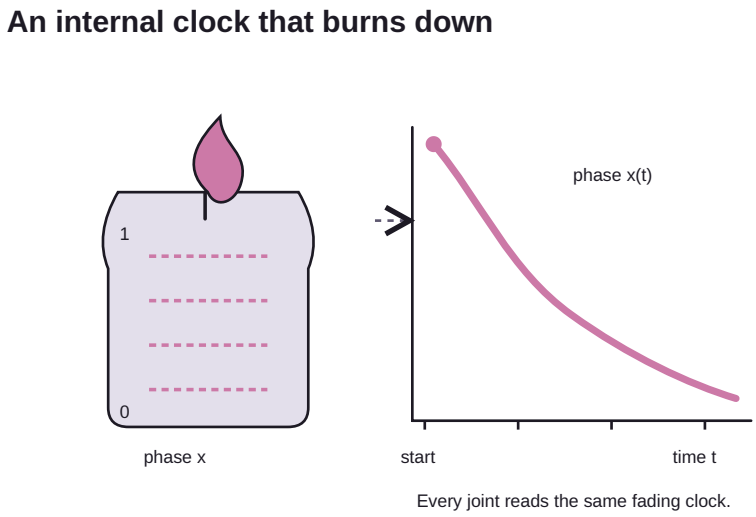


Figure 1. A candle burning down beside the curve of the phase, which starts at 1 and slides toward 0.

The candle's height is the *phase*, written x, a clock that runs from 1 down toward 0. At the start of the motion $x = 1$, and by the time the motion is nearly over, x is tiny. To find your place in the movement, you read the candle.

The idea

The candle offers something the wall clock t does not: it belongs to the robot. Whenever another part of the DMP needs to know "how far along are we?", the candle gives the answer x.

You have two knobs to control this candle: α_x sets how fast it burns, with a bigger α_x giving a faster, hungrier flame. The other knob, τ, stretches everything in time, so doubling τ makes the candle burn twice as slowly and the whole motion take twice as long.

Along the way, this clock brings a useful bonus: each joint of a robot can have its own spring and push while reading the same candle as all the others. Sharing the clock keeps them in step, like dancers following one drummer.

The math

> ☼ **Intuition: a candle that burns fast, then slow**
> At each moment, the candle loses a fixed *fraction* of its height rather than a fixed amount, which means a tall candle loses a lot and a short one loses a little. That shrinking loss is why its height slides smoothly toward zero without quite arriving.

We can capture that candle in one small equation, saying that the speed of the burn is proportional to the height still left.

∑ The math: eq. (3)

$$\tau\,\dot{x} = \underbrace{-\,\alpha_x\,x}_{\text{the clock runs down}} \tag{3}$$

Read aloud: the time scale τ times the speed of the phase equals minus α_x times the phase, so the taller the candle, the faster it burns down.

Once we set the starting phase to $x(0) = 1$, the equation has a known solution that you can use without having to derive it.

∑ The math: eq. (E1)

$$x(t) = \exp\!\left(-\,\alpha_x\,t/\tau\right) \tag{E1}$$

Read aloud: the phase at time t is e raised to minus α_x times t divided by τ, which is a smooth slide from 1 toward 0.

In the notebooks you do not run the clock from (E1); instead you *step* equation (3) forward in small time steps, using the explicit Euler step from chapter 2, which takes the following form when applied to the phase.

∑ The math: eq. (E4)

$$s(t+\Delta t) = s(t) + \Delta t \cdot \dot{s}(t), \qquad s \in \{y, z, x, \phi\} \tag{E4}$$

Read aloud: the new value of the state is its old value plus the time step Δt times its rate of change. For the clock, s is the phase x and its rate is minus α_x times x, divided by τ.

As you build the clock in code, the exact answer (E1) gives you a useful companion: with $\Delta t = 0.001$ s, the stepped values should agree with it to about 0.002 all along. A larger gap tells you something in the code needs attention.

Where is the candle at the end?

When the motion reaches its end at $t = \tau$, equation (E1) gives $x(\tau) = e^{-\alpha_x}$, which means the candle's remaining height depends only on α_x.

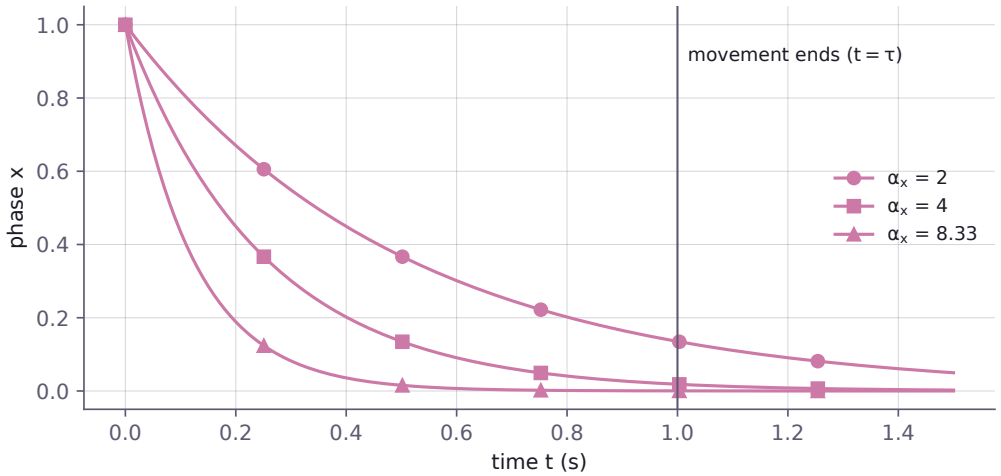


Figure 2. The phase for three values of the gain, with a vertical line at the movement end $t = \tau$: a larger gain leaves a much shorter candle.

Follow the figure to the end of the motion, using $\tau = 1$ s and three values of α_x. With $\alpha_x = 2$ the candle still stands at about 0.14, while $\alpha_x = 4$ leaves about 0.018 and $\alpha_x = 8.33$ leaves about 0.0002.

The kit's default is $\alpha_x = 25/3 \approx 8.33$.[2] With that default, the candle is almost out by the time the robot reaches the goal, letting the push fade so the spring alone can finish the job.

Try it

▶ Try it: 01_discrete_playground.ipynb › Simulate 1.4 - Goal, start, tau and number of basis functions
Open the cell and move the slider for τ, watching the clock slow as you make it bigger and the same path take longer to finish. Move it back down and the path fits into less time, keeping its shape as the speed changes. Along the way, follow the printed line that tells you where the motion ends.

[2]The defaults $\alpha_z = 25$, $\beta_z = \alpha_z/4$ and $\alpha_x = \alpha_z/3$ are the values IJ13 uses throughout (Table 1, p. 349, 10.1162/NECO_a_00393). S23 only asks for $\alpha_x > 0$ (p. 1137).

Gotcha

> **⚠ Gotcha: the candle burns too fast**
> Symptom: the path starts well, but its final stretch looks lazy or wrong. Cause: a large α_x leaves x tiny long before $t = \tau$, and the forcing term in eq. (4) is multiplied by x, leaving almost no push for the last part of the motion. This trips up everyone, because a plot of x alone can still look “fine”. Fix: keep α_x moderate, or give the late bumps of the forcing term larger weights, as the kit does. You can read about the effect in 10.1177/02783649231201196, section 2.1.1.3, p. 1139.

◆

04

CHAPTER FOUR

Bumps that bend the path: the forcing term

Build a smooth push from weighted bumps, and predict where the path bends when you move one slider.

Picture a robot hand carrying a cup toward its goal, with a bowl sitting in the way: a straight path would knock the bowl off the table. To carry the cup safely over the rim, the hand has to lift and then come down, while a spring alone would pull it straight toward the goal. That lift needs another source, and in this chapter you build it: the push that bends the path.

The picture

Imagine listening to a song while you move the equalizer on a music player: bass, mids, and treble each have their own slider. As you push one slider up, that part of the sound grows stronger while the other sliders stay where you left them.

A DMP gives you that same feeling of shaping something one part at a time, because it splits the movement into overlapping stretches with a slider for each one. Raise a slider, and the robot pushes harder during that stretch of the motion. Those sliders are the weights w_i, and the bands they control are the bumps, which we call basis functions.

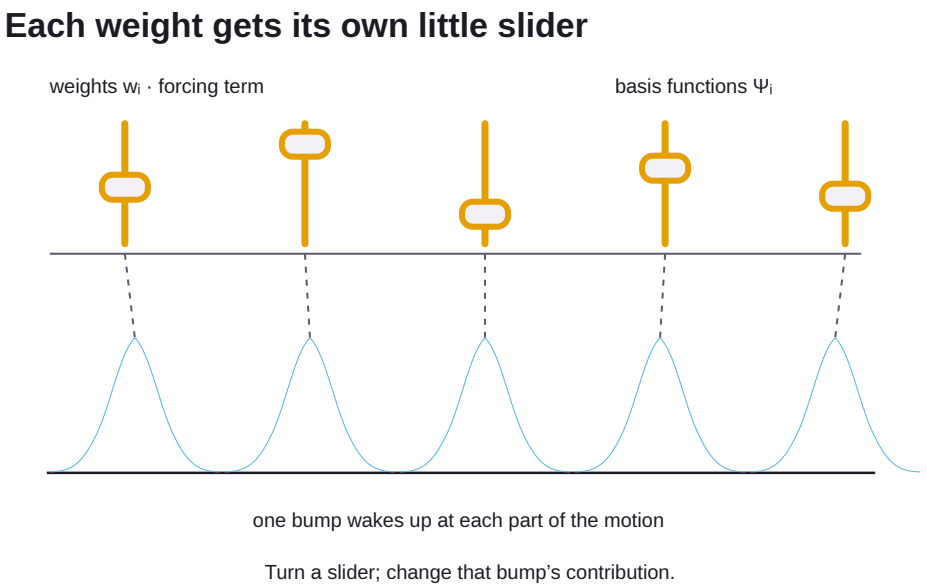


Figure 1. An equalizer: each slider is one weight, and each slider is linked to one bump that is awake during its own part of the movement.

The idea

Let us build that push together, starting with the clock and adding the pieces as we go.

In chapter 3 you met the phase x, which runs from 1 down to 0 as the movement unfolds. That gives us a clock to place the bumps along, so each one has its own moment in the movement.

Along that phase, imagine laying out N little hills: each *basis function* is a bump-shaped curve Ψ_i, large near its own center c_i and almost zero far away. We can think of each bump as "awake" near its center and "asleep" elsewhere, which gives it just one part of the motion to look after.

Now give each bump a weight, like setting its slider on the equalizer: a large weight brings a big push while the bump is awake, a negative weight pushes the other way, and a weight of zero contributes nothing.

Once we blend those weighted bumps into a smooth curve, we have the forcing term $f(x)$. The bumps overlap along the way, so the blend has no jumps and can take almost any shape you like.

To bring the hand home at the end, we multiply the blend by the phase x. Because x shrinks to nearly zero, the push fades away on its own, leaving the spring to have the last word as the hand lands on the goal.

☼ Intuition: a smooth push built from sliders
Picture the bumps sitting side by side along the clock, each awake for a short while as the motion passes through. Multiply every bump by its slider and take a weighted average, then let the clock fade the result toward zero. You get a gentle push that is strongest where you set the sliders high and fades away as the movement ends.

The math

☼ Intuition: a weighted average of bumps
At any moment, only a few bumps are awake, and each casts a vote with its weight to make the average push. As the movement draws to a close, the clock turns the volume down.

Two equations bring this picture together: the first builds the push from the weighted bumps, and the second describes the shape of a single bump. The centers c_i come from the clock: the kit places the first at $x = 1$ and the last at $x = e^{-\alpha_x}$, spreading the bumps evenly in time.[1] Neighboring bumps need to overlap, so their widths are chosen through h_i; a larger h_i makes a narrower bump.

∑ The math: eq. (4)–(5)

$$f(x) = \frac{\sum_{i=1}^{N} w_i \, \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} \, x \tag{4}$$

Read it aloud: take each bump Ψ_i at the current phase, multiply it by its weight w_i, add them up, divide by the sum of the bumps (a weighted average), and then multiply by the phase x.

$$\Psi_i(x) = \exp\left(- h_i \, (x - c_i)^2 \right) \tag{5}$$

Read it aloud: bump i is 1 when the phase sits exactly at its center c_i, and it falls away smoothly on both sides, faster when h_i is large.

Where several bumps overlap, adding them without dividing would make the push stronger just because more bumps happened to be awake there. Dividing by their sum turns that crowd into a *weighted average*, with the awake bumps having the most say. The weights then give the heights you ask for, and the average never exceeds the biggest of them.

[1] S23 prints the centers without brackets (p. 1137, 10.1177/02783649231201196). The formula it follows is U10's: $c_i = e^{-\alpha_x (i-1)/(N-1)}$, $i = 1, \ldots, N$ (p. 803, footnote 1, 10.1109/TRO.2010.2065430). These are the values of the phase at N equally spaced times from 0 to τ. IJ13 says the same in words: the kernels have "equal spacing in time, which corresponds to an exponential spacing in x" (p. 334, 10.1162/NECO_a_00393).

To see how the clock arranges the bumps, follow the same ten across the two panels below: on the left they sit along the phase x, and on the right they sit along time t.

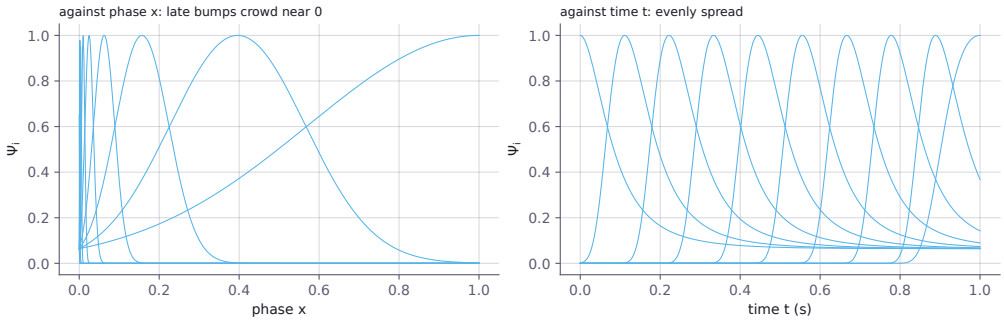


Figure 2. The same ten bumps drawn against phase x on the left, where they are crowded near 0, and against time t on the right, where they are evenly spread.

With the bumps in place, we can watch ten weights join forces to make one forcing term. These weights were learned from the reach demonstration in the notebook; chapter 5 will walk you through how that learning happens.

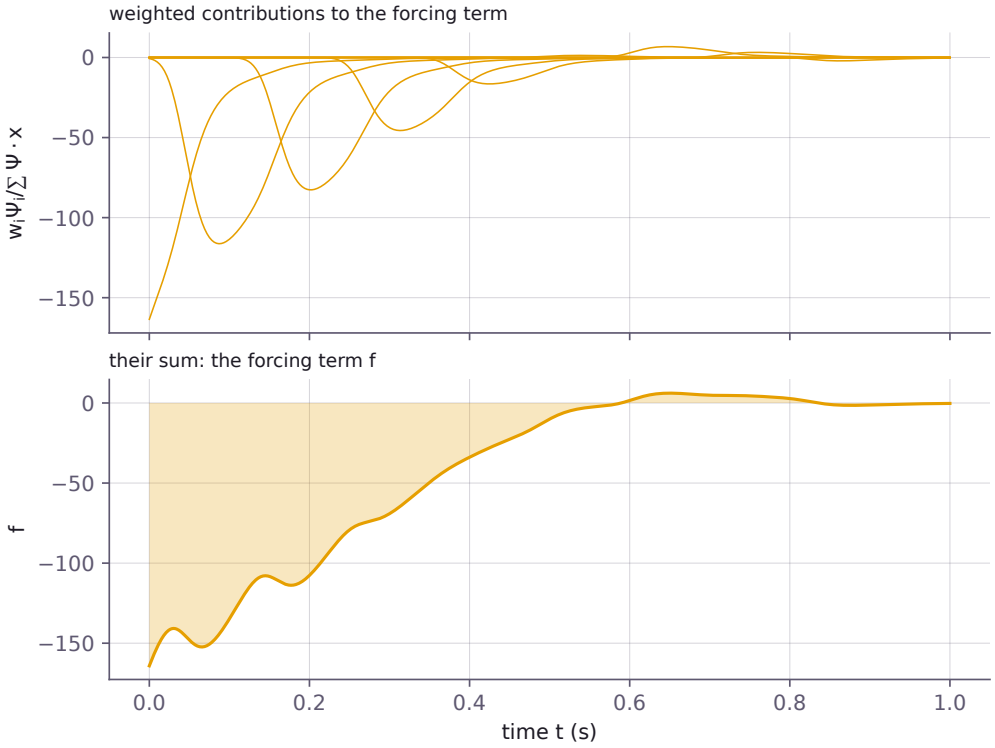


Figure 3. Ten weights learned from the reach demonstration. Top: each bump's part of the push, its weight times its share of all the bumps, times the phase x. Bottom: their sum, the forcing term f, shaded.

Now bring the cup and bowl back into the picture and watch what one big weight does:

with $f = 0$ the spring slides straight to the goal. Give bump 4 a large weight, and the path rises above the goal before returning, making the bulge our cup needed.

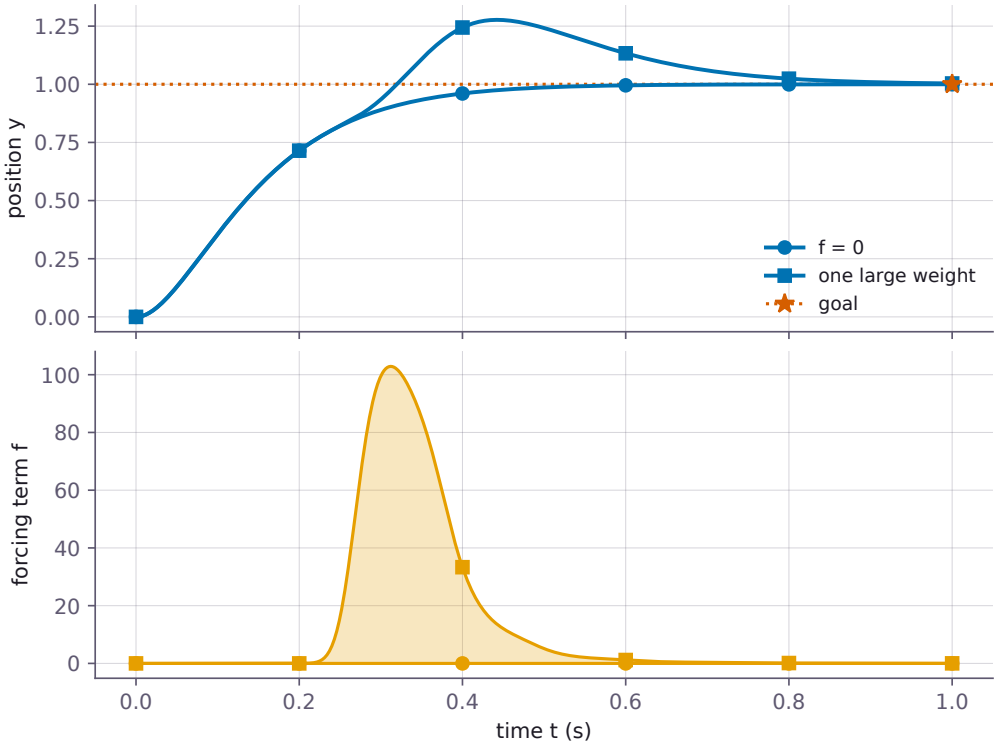


Figure 4. The path with the spring only (f = 0), and with one large weight (w_4 = 2500), which makes the path rise above the goal and come back. Below, the forcing term f over time.

Follow the bend back to the slider that made it: one slider moved, and one stretch of the path bent. That local influence gives you something useful to hold onto, because you can predict where a weight change will show up.

Try it

▶ Try it: 01_discrete_playground.ipynb › Create 1.5 - Challenge
Open the table of the gap for N = 2, 5, 10, 20, 30 and 60 bumps, and follow the shrinking gap as N grows. The gap levels off at about 0.3 % from N = 20 on, because that last bit comes from the time step rather than the bumps. As you explore, find the smallest N that keeps the path within 2 % and type it into `MY_N`. Each bump adds another slider to learn, so you are looking for enough to bend the path without keeping more than you need.

Gotcha

> **⚠ Gotcha: bumps evenly spaced in time look crowded in x**
> Symptom: when you plot the bumps against the phase, the last ones look squeezed together near 0 while the first ones look wide, as though something has gone wrong. Cause: the centers are spread evenly in time, and the phase decays like $e^{-\alpha_x t/\tau}$, so equal steps in time become tiny steps in x. This trips up everyone. Fix: nothing needs fixing, and this spacing also explains why the widths h_i differ from bump to bump. Follow them along time in the right panel of Figure 2, and you will see the evenly spread bumps again.

◆

05

CHAPTER FIVE

Learning from one demonstration

Show the robot a motion once, turn it into weights, and check how closely the robot can replay it.

Standing beside the robot arm, you take hold of its wrist as the motors go soft, then draw one smooth curve through the air from the shelf toward the box. The robot records every wobble, and when you let go, it replays that curve on its own, as smoothly as you drew it. This is kinesthetic teaching, a name for "teaching by moving the robot's body," and here you build the part that turns one recording into a motion the robot owns.

The picture

Imagine laying tracing paper over a drawing and following its lines once, until the shape belongs to your own sheet. You can return to that copy whenever you like and reproduce the drawing, even with your eyes closed.

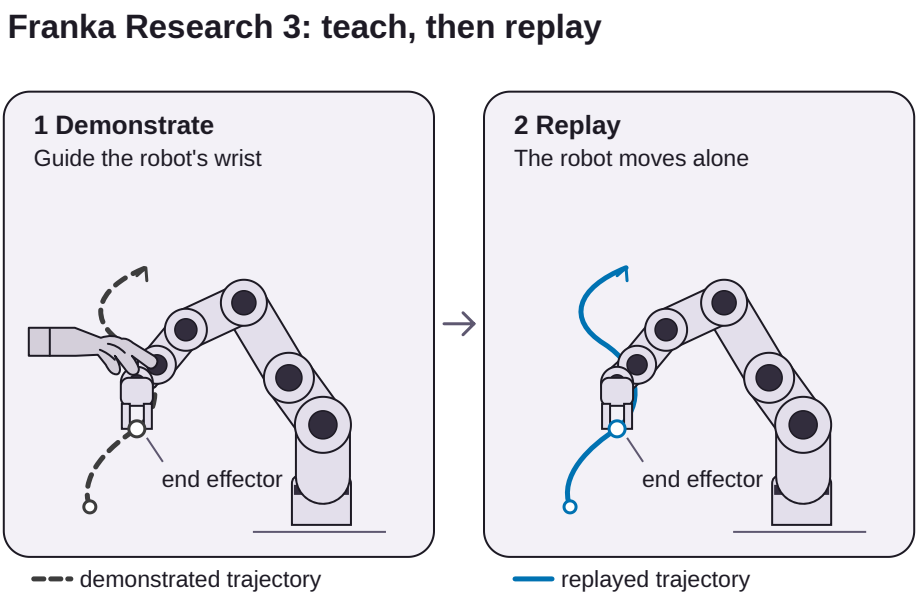


Figure 1. A hand traces a curve once, then a robot arm replays the traced curve by itself.

A DMP keeps its own kind of tracing, storing a short list of numbers, the weights w_i, from which it can rebuild the curve whenever you ask. The recording becomes those weights rather than a point-by-point copy.

The idea

We already have a spring to pull the robot toward the goal and a clock that fades from 1 to 0, so two parts of our machine are ready. What remains is the push f, which has to bend the path into the shape you drew.

To find it, imagine asking the spring what push would have made it follow this exact demonstration. We can answer that question in two steps.

1. At every recorded moment, work out the push the spring would have needed, giving us the target push f_d.
2. Then choose the weights so that the bumps, added up, match that target as closely as possible.

The math

> ☼ **Intuition: run the spring backwards**
> Usually you choose a push and watch the spring move; this time, you watch the motion and ask which push could have caused it. It is like following a ball through the air and working out how hard it was thrown.

Your demonstration gives us positions y_d recorded at sample times t_J, and we also need its velocity and acceleration: the speed and the rate at which that speed changes. Differencing the positions gives us those quantities, although the built-in reach already comes with exact ones.[1]

> **∑ The math: eq. (6)**
>
> $$f_d(t_J) = \tau^2\, \ddot{y}_d(t_J) - \alpha_z\Big(\beta_z\big(g - y_d(t_J)\big) - \tau\, \dot{y}_d(t_J)\Big) \tag{6}$$
>
> Read it aloud: the push needed at each sample is the acceleration (scaled by τ^2) minus whatever the spring–damper term already supplies.

Think of equation (6) as equation (1) turned inside out: in (1) you put a push in and get a motion, while in (6) you put the motion in and get the push out.

You can follow the three signals going into (6) in the figure below, using the built-in reach as our demonstration.

[1] g is the last demonstrated position, y_0 the first, and τ the duration of the demonstration, as IJ13 states (p. 342, 10.1162/NECO_a_00393). S23 sets τ "equal to the total duration of the desired movement" (p. 1137), and U10 uses the same τ and g (p. 803, 10.1109/TRO.2010.2065430). The defaults $\alpha_z = 25$, $\beta_z = \alpha_z/4$ and $\alpha_x = \alpha_z/3$ follow IJ13 Table 1 (p. 349).

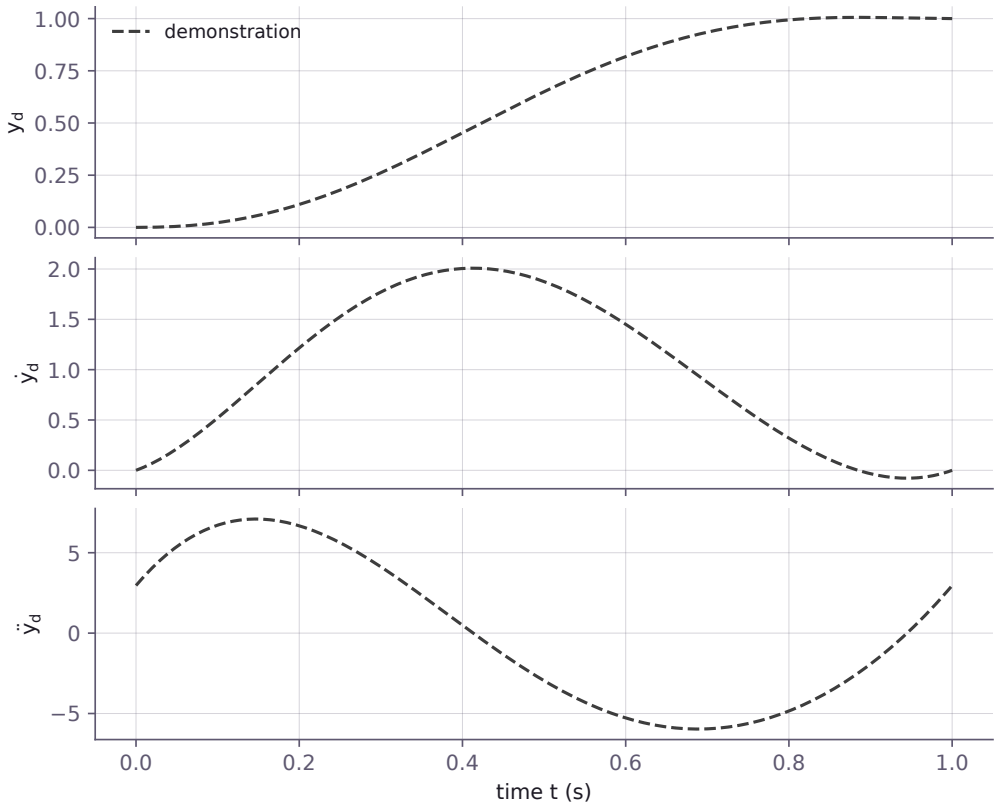


Figure 2. The position, velocity and acceleration of the built-in reach demonstration, stacked, with the demonstration drawn dashed.

By now we have a target value f_d for every sample, each one a number that our model for f, the weighted sum of bumps in eq. (4), should match. Gathering all those samples together gives us one linear system for the weights.

> ☼ **Intuition: a recipe with N ingredients**
> Imagine each bump as an ingredient with a fixed shape, and its weight as the amount you pour into the mixture. We are looking for the amounts that make the finished mixture taste like the target.

> **∑ The math: eq. (7)–(8)**
>
> $$\Phi\, w = \mathfrak{F} \tag{7}$$
>
> Read it aloud: the matrix of bump values times the weights should equal the stack of target pushes.

$$\boldsymbol{\Phi} = \begin{bmatrix} \dfrac{\Psi_1(x_1)}{\sum_{i=1}^{N} \Psi_i(x_1)} x_1 & \cdots & \dfrac{\Psi_N(x_1)}{\sum_{i=1}^{N} \Psi_i(x_1)} x_1 \\ \vdots & \ddots & \vdots \\ \dfrac{\Psi_1(x_T)}{\sum_{i=1}^{N} \Psi_i(x_T)} x_T & \cdots & \dfrac{\Psi_N(x_T)}{\sum_{i=1}^{N} \Psi_i(x_T)} x_T \end{bmatrix} \tag{8}$$

Read it aloud: row J of the matrix holds the normalized bump values at sample J, each multiplied by the phase x_J.

Picture the matrix as a table with one row per sample and one column per bump, with far more samples than weights to work with. No set of weights fits every sample exactly, so we ask NumPy for the least-squares answer: the weights with the smallest total squared miss. In `dmpkit`, one call to `np.linalg.lstsq` does that work.

Before we send the spring on its way, let us compare the learned push with the target it was trying to match.

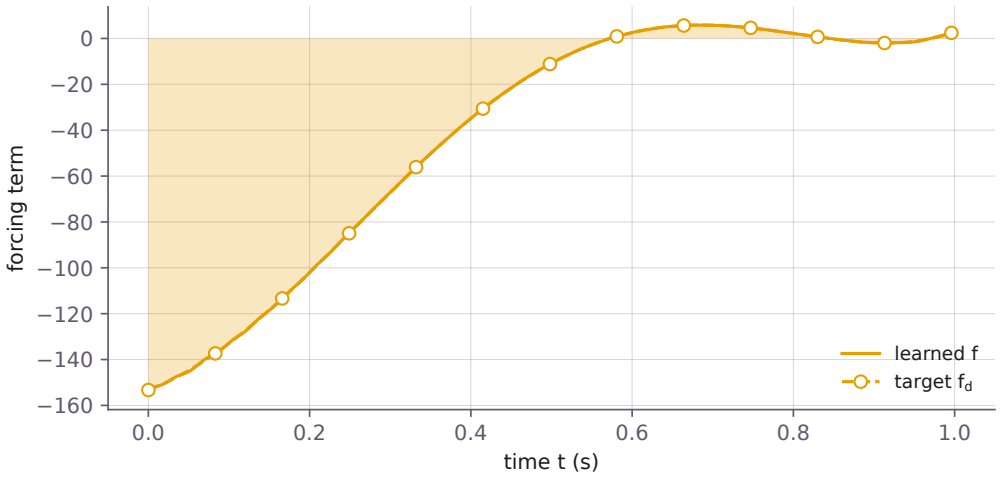


Figure 3. The target push f_d drawn dashed over the learned push f drawn solid and shaded, with N = 30 bumps; the two coincide.

Then comes the check that matters for the robot: run the spring and learned push forward in time with explicit Euler (E4), and compare the motion with your demonstration.

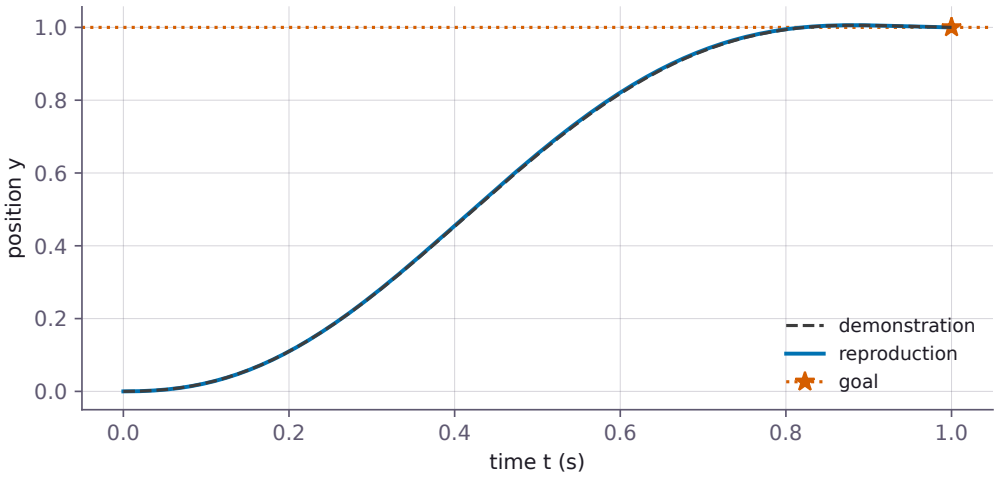


Figure 4. The dashed demonstration drawn over the solid reproduction with the goal marked by a star; the two coincide, with a largest gap of 0.3 percent.

The curves lie together, and the reach you drew is now a motion the robot can call its own.

Learning on the fly

The least-squares method we just used waits until the whole demonstration is available, which is why we call it a batch. But imagine the samples arriving one by one, with the weights improving as each new piece of the movement comes in.

> ☼ **Intuition: a running average, but for weights**
> Think of keeping a running average as new numbers arrive: instead of rebuilding the whole sum, you nudge the old answer by a fraction of the surprise. Recursive least squares carries that same idea over to the weights.

> **∑ The math: eq. (9)–(10)**

$$\mathbf{P}_J = \frac{1}{\lambda}\left(\mathbf{P}_{J-1} - \frac{\mathbf{P}_{J-1}\boldsymbol{\varphi}_J\boldsymbol{\varphi}_J^{\top}\mathbf{P}_{J-1}}{\lambda + \boldsymbol{\varphi}_J^{\top}\mathbf{P}_{J-1}\boldsymbol{\varphi}_J}\right) \tag{9}$$

Read it aloud: after each sample, update the matrix P, which tracks how sure the learner is about each direction, dividing by the forgetting factor λ.

$$\mathbf{w}_J = \mathbf{w}_{J-1} + \left(f_d(t_J) - \boldsymbol{\varphi}_J^{\top}\mathbf{w}_{J-1}\right)\mathbf{P}_J\boldsymbol{\varphi}_J \tag{10}$$

Read it aloud: the new weights are the old weights plus the surprise, the target minus the current prediction, steered by the matrix and the basis values at that sample.

We begin the recursion with P equal to the identity matrix and all weights zero, then let each arriving sample shape the answer. With a forgetting factor of $\lambda = 1$, nothing is forgotten along the way.

After one sweep through the demonstration, though, the weights still differ from those found by batch least squares, as the paired bars below show.

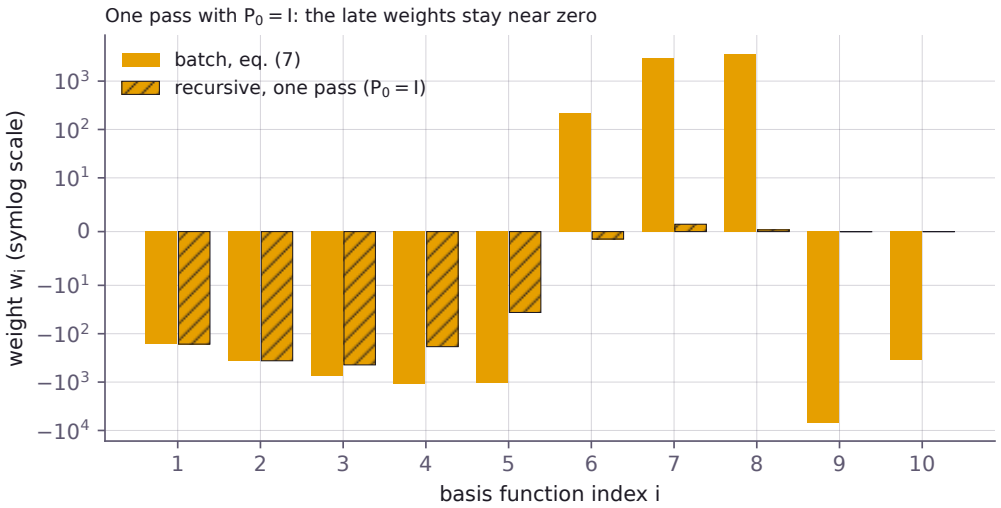


Figure 5. Ten weights as paired bars on a symmetric-log axis, comparing batch least squares with one pass of recursive least squares; after one pass the first recursive weights match the batch ones, and the later ones stay near zero.

The difference begins with the start value $P_0 = I$, which acts like a gentle pull of every weight toward zero. One pass of (9)–(10) therefore gives exactly the least-squares answer with that extra pull included. In eq. (8), every row carries the phase x_J, and by the time the late bumps wake up, that phase is tiny. For those bumps the pull toward zero wins over the data, leaving their weights near zero while the first ones match the batch weights. More passes help, although slowly; with a much larger start, such as $P_0 = 10^6 I$, we get the batch replay in one pass. This explains why the discrete notebooks use the batch version, and why the periodic DMP of Chapter 8 returns to the demonstration over many laps.

Try it

▶ Try it: 01_discrete_playground.ipynb › Simulate 1.2 - Fit the demonstration and replay it
When you run this cell, it fits $N = 30$ bumps to the demonstration and replays the movement. Follow the dashed demonstration and solid replay toward the star marking the goal, and look for the two curves lying on top of each other.

▶ Try it: 01_discrete_playground.ipynb › Test 1.3 - How close is the replay?
Run this cell to put a number on what you saw: it prints the largest gap in percent of the motion's size, with PASS appearing when the gap is under 5 %. Once you have checked the built-in motion, return to *Create 1.1 - Choose a demonstration* and set `DEMO` to `"waypoints"` with your own list of numbers. Now you can see whether your own curve passes too.

Gotcha

⚠ Gotcha: noisy demonstrations give wild accelerations
Symptom: the push f_d looks like static, leaving the replay jittery or wrong. Cause: eq. (6) needs the acceleration, so taking differences of positions twice turns even tiny measurement noise into a much larger signal (compare S23 Fig. 7, p. 1143, 10.1177/02783649231201196). Fix: smooth the demonstration before you differentiate it, so the noise does not overwhelm the motion you meant to teach. This trips up everyone the first time a real sensor joins the story.

◆

06

CHAPTER SIX

New goal, new speed

Same weights, two new numbers: send the motion to a new goal, and play it faster or slower.

The robot arm learned its reach yesterday, and today the boss walks in: the shelf moved 20 centimeters, and the motion needs to finish in half the time. You can keep the weights from that one demonstration and give the arm its new task by changing two numbers: the goal g and the time constant τ. Then press go; the robot needs no second lesson.

Let us follow what happens when you turn those two knobs, because one changes the motion just as you might expect, while the other holds a small surprise.

The picture

Picture signing your name across a whiteboard and then squeezing it onto a small form: the size changes, yet the handwriting stays recognizably yours. The same is true when you sign quickly at the checkout or take your time over a contract.

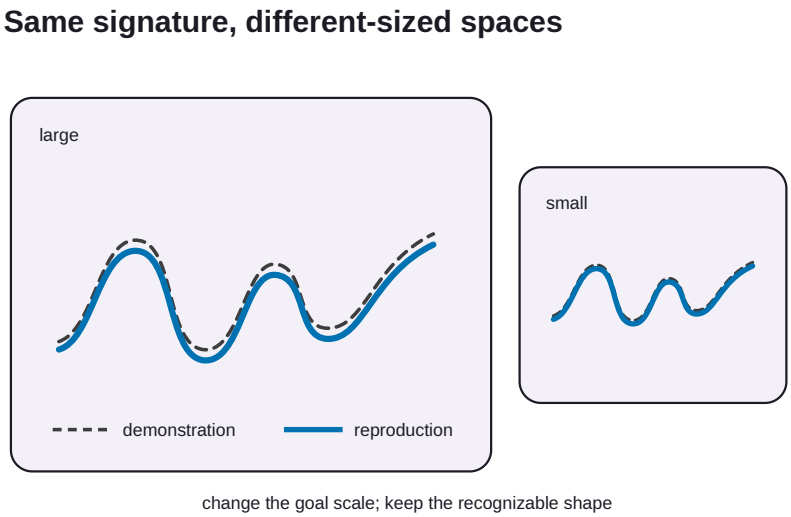


Figure 1. A signature written big and small: the same shape, scaled up or down.

The weights hold the DMP's version of that familiar "style," the way your hand moves, while the goal chooses where the motion ends and the time constant sets how quickly it gets there.

Keep one detail of the signature picture in mind: your hand adjusts size and speed together without much thought, while a DMP needs a choice from you. Changing speed comes with the model; making the shape scale in size takes an extra ingredient.

The idea

Speed first. Let us start with the time constant τ and the phase x, the clock you met running from 1 down to 0. Because τ sets how fast that clock ticks, a larger τ slows it down, and the rest of the DMP follows along. The whole motion then plays back more slowly or quickly along exactly the same path in space.

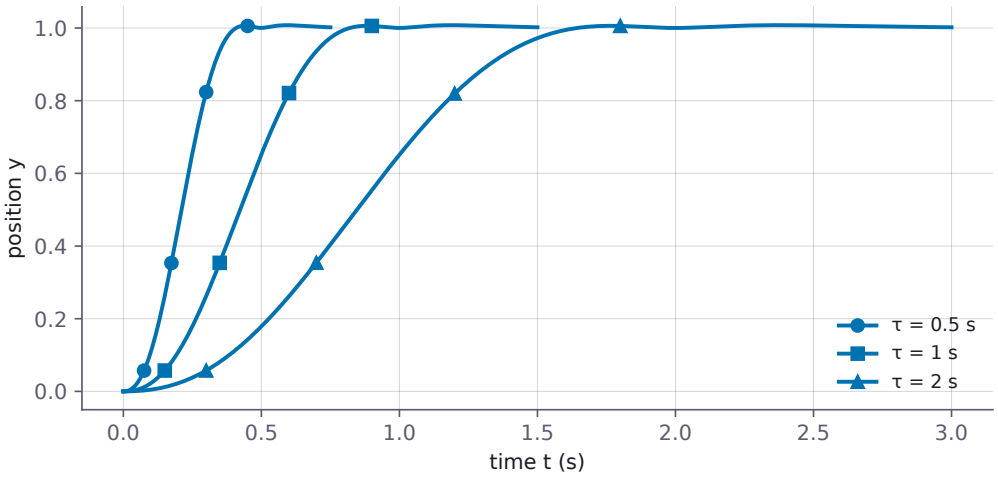


Figure 2. The learned motion replayed for τ = 0.5, 1 and 2 seconds: the same path, traveled at different speeds.

Goal second. Changing the destination asks something subtler of the model, because in the survey form, eq. (4), the forcing term is a fixed learned push whose formula never mentions the goal. The spring–damper term does know the destination and pulls toward the new g, so the familiar push now travels with a different pull.

Imagine moving the finish farther away: the spring has more work to do, while the learned push stays the same size. The path still arrives because the spring wins in the end, but its shape looks flatter, with the same bump of push spread over a longer trip.

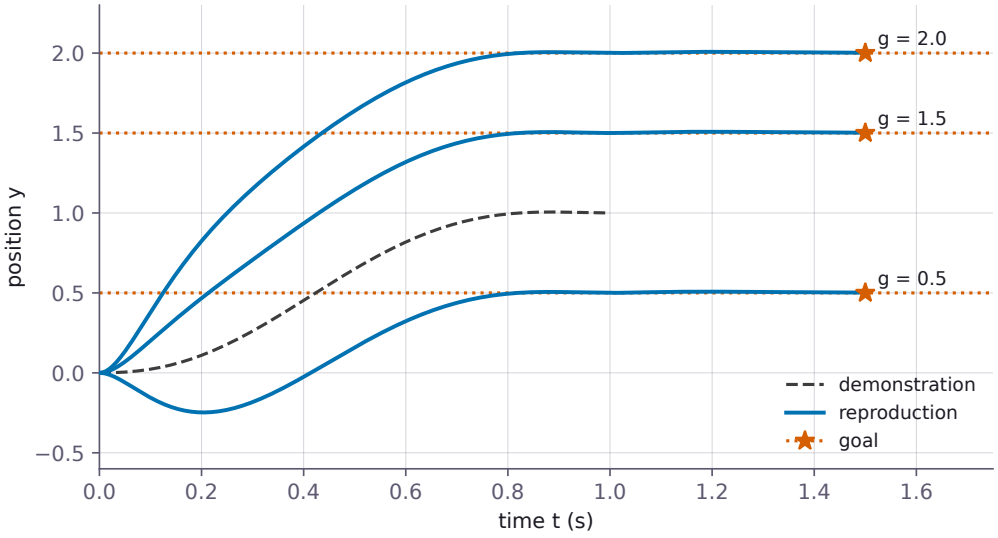


Figure 3. One demonstration (dashed) and reproductions to goals 0.5, 1.5 and 2.0 with the survey form; the push keeps its size, so the shape does not stretch, and for g = 0.5 the path first dips below zero.

To make the shape grow with the trip, we can choose variant D1, which multiplies the

push by the trip length, $(g - y_0)$, where y_0 marks the start. A longer trip then brings a bigger push, stretching the shape like your signature across the whiteboard.

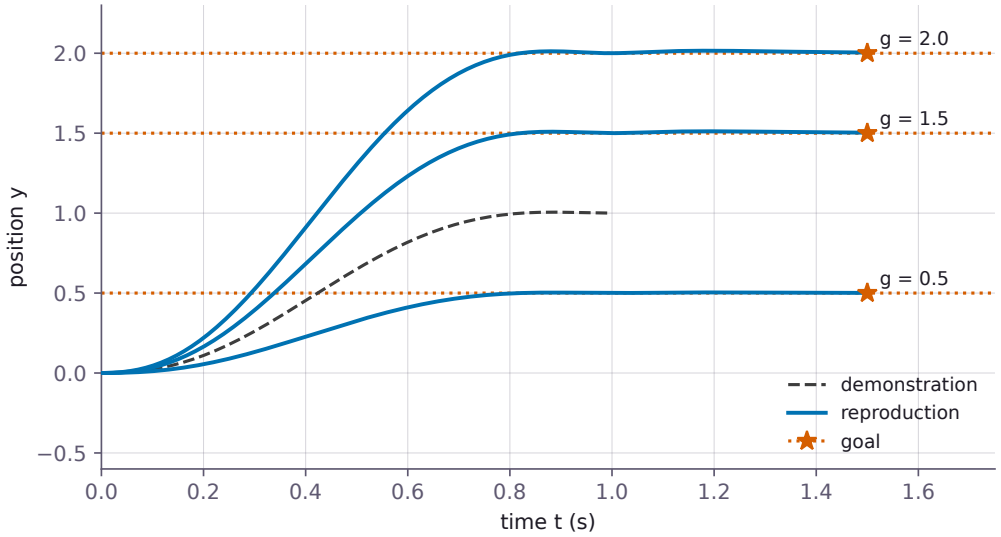


Figure 4. The same goals with the scaled variant: the whole shape stretches with the goal.

> ☼ **Intuition: two knobs, two behaviors**
> Turning τ feels like moving a video's playback slider, watching the same picture unfold at a different speed. Turn g in the survey form, and the finish line moves while the runner keeps the same running style. Turn g in the scaled variant, and the runner's stride stretches too, to fit the new track.

The math

The same equations are ready to take us to the new destination; we just put the new g and τ into the model we already know.

> ☼ **Intuition: same machine, new settings**
> Picture the equations as a familiar machine with two dials on its side: you can turn those dials to change the motion without rebuilding what is inside.

> **Σ The math: eq. (1)–(3)**
>
> $$\tau\,\dot{z} = \underbrace{\alpha_z(\beta_z(g - y) - z)}_{\text{spring–damper term}} + \underbrace{f(x)}_{\text{forcing term}} \tag{1}$$

Read aloud: the speed of z, scaled by τ, is the spring pulling toward the goal plus the learned push.

$$\tau\,\dot{y} = z \tag{2}$$

Read aloud: the speed of the position, scaled by τ, is z.

$$\tau\,\dot{x} = \underbrace{-\,\alpha_x\,x}_{\text{the clock runs down}} \tag{3}$$

Read aloud: the phase shrinks at a rate set by α_x, and τ stretches that clock.

Look at τ on the left of all three equations, and nowhere else: making τ twice as big makes every change take twice as long. That stretches the journey in time while keeping its path in space. Meanwhile, the goal g sits only inside the spring term of eq. (1), so changing the destination leaves the weights untouched.

> ☼ **Intuition: the push is a fixed recipe**
> Think of the survey form as a recipe for the push, written once from the demonstration with no goal among its ingredients. A new destination changes the spring pulling the robot home, while the recipe keeps making the same push.

∑ The math: eq. (4)

$$f(x) = \frac{\sum_{i=1}^{N} w_i\,\Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)}\,x \tag{4}$$

Read aloud: the push is a weighted blend of bumps, faded out by the phase, and no goal appears anywhere in it.

When you want the push to grow with the trip, the scaled variant D1 gives you that extra ingredient, as described by Ijspeert and colleagues (10.1162/NECO_a_00393, p. 333).

∑ The math: eq. (IJ-2.3)

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x)\,w_i}{\sum_{i=1}^{N} \Psi_i(x)}\,x\,(g - y_0) \tag{IJ-2.3}$$

Read aloud: the same blend as in (4), now multiplied by the trip length $g - y_0$, so a longer trip means a bigger push.

Choosing D1 also changes how we learn: the weights are fitted with the demonstration's trip length, so each row of the learning matrix in eq. (8) is multiplied by $(g - y_0)$.

When you replay the motion toward a new goal, that factor changes along with the destination.[1]

Try it

▶ Try it: 01_discrete_playground.ipynb › Simulate 1.4 - Goal, start, tau and number of basis functions
Open this cell and move the goal slider g, following the path as it reaches the new destination without another lesson. Then drag τ and watch the same journey take more or less time, before ticking “scale by (g − y0)” to switch from the survey form (4) to the scaled variant (IJ-2.3). As you move the goal, look for the shape stretching with it, and try placing the goal below the start to see what each form does.

Gotcha

⚠ Gotcha: under the scaled variant, a goal near the start switches the push off
Symptom: after you tick “scale by (g − y0)” and place the goal very close to the start, the robot does almost nothing or moves in an odd way. Cause: the factor $(g - y_0)$ in (IJ-2.3) approaches zero when start and goal almost coincide, taking the push with it. Picture a tap on a table that begins and ends at the same height: there is no “trip” left to scale. IJ13’s fix, eq. (IJ-2.18), addresses a demonstration whose start and goal coincide by dividing by the trip length used in fitting, keeping the push on during learning. A goal moved near the start afterward still leaves $(g - y_0)$ as a factor, so this fix cannot help and can instead magnify the motion hugely (IJ13 pp. 346–347). Fix: for motions that begin and end at nearly the same place, choose the survey form (4), whose push has no factor $(g - y_0)$ to switch it off.

◆

[1] The kit takes the goal as the last demonstrated position, y_0 as the first, and τ as the demonstration duration, as IJ13 does when it fits the weights (p. 342, 10.1162/NECO_a_00393). These fitted values are the g_{fit} and $y_{0,fit}$ of eq. (IJ-2.18).

07

CHAPTER SEVEN

Round and round: the periodic DMP

Keep a robot's rhythm going as you discover the three knobs that change its speed, its size and its center.

Picture your robot with a sponge in its hand and a dirty table in front of it, ready to wipe left, right, left, right until you say stop. Perhaps you would rather have it drum a beat; either way, there is no goal where the motion should settle and rest. It needs to keep going. As you build the pattern behind that movement, you will discover three knobs that let you choose how fast it goes, how big it swings, and where its middle sits.

The picture

Imagine climbing onto a merry-go-round and watching the horses pass as the ride turns in circles. The angle tells you where you are in the turn, and we call that angle the *phase* ϕ. Around the rim, each horse stands at a fixed spot and represents a *basis function*, a bump that is "awake" only when the ride is nearby. Every time you go round, the same horses greet you in the same order.

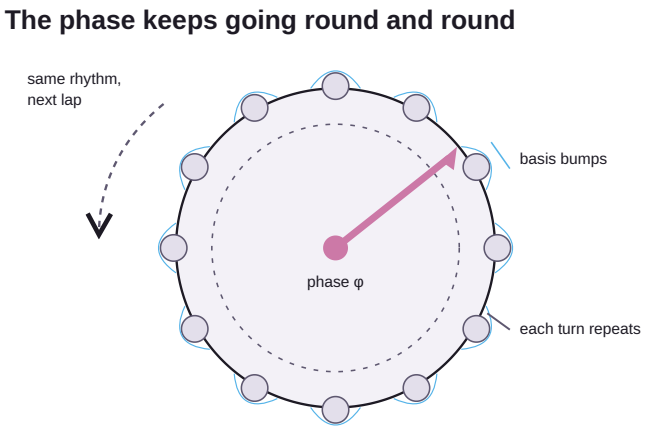


Figure 1. A merry-go-round drawn as a clock face: the bumps stand around the rim, and a hand sweeps round and round to show the phase.

Carry that picture into a periodic DMP: because the horses return on every lap, the motion can repeat along with them.

The idea

In chapter 3 you watched the phase burn down like a candle, from 1 toward 0, but a candle only ever burns lower and never comes back up. To keep a rhythm alive, we replace it with a clock hand that can travel round and round without stopping.

Once the bumps sit on the clock face, each wakes up at the same moment of every lap. The spring from chapter 2 still pulls the robot toward a fixed center, which is 0 for now, while the bumps supply the pushes that keep it moving. You can steer the rhythm with two numbers: the frequency Ω tells you how fast the hand turns, and the amplitude r makes the swing bigger or smaller.

> ☼ **Intuition: a hand on a clock, and bumps on its face**
> As the hand turns at a steady speed, each bump it passes pushes the robot a little, while the spring pulls back toward the center. Lap after lap, the pushes arrive in the same order, and so the motion repeats like a familiar dance. Turn the hand faster to speed up that dance, or scale the pushes to make it bigger.

The math

> ☼ **Intuition: the same spring, a new clock**
> You can bring along the spring from chapter 2 and the push from chapter 4, because only two things change here. The candle becomes a clock hand, and the spring pulls toward 0 because there is no goal.

Here is the periodic DMP of the survey S23 (10.1177/02783649231201196, section 2.2.1, p. 1142). As you read, follow the first two equations into the robot's motion, then look to the third for the clock that keeps the rhythm.

∑ The math: eq. (34)–(36)

$$\dot{z} = \Omega\Big(\underbrace{\alpha(\beta(-y) - z)}_{\text{spring–damper term}} + \underbrace{f(\phi)}_{\text{forcing term}} \Big) \tag{34}$$

Read it aloud: the speed of the scaled velocity z is the frequency times the spring pull toward 0 plus the push f.[1]

$$\dot{y} = \Omega\, z \tag{35}$$

Read it aloud: the position y moves at z times the frequency, so z is how far y moves for each radian the clock hand turns.

$$\tau\, \dot{\phi} = 1 \tag{36}$$

Read it aloud: the clock hand turns at a constant rate, one full turn for each period.

Because the clock never stops, ϕ keeps growing, but the bumps return every 2π, which means you never need to wrap the phase back around. The kit uses $\Omega = 1/\tau$.[2] With that connection in place, you can follow the clock through its solution, (E3):

∑ The math: eq. (E3)

$$\phi(t) = \phi(0) + \Omega\, t, \qquad \Omega = 1/\tau \tag{E3}$$

Read it aloud: the phase at time t is where it started plus the frequency times the time.

[1] The survey prints the gains in (34) as α and β. Its learning rule (39) for the same system uses α_z and β_z, and so do G09 (eq. (8), p. 9, 10.1007/s10514-009-9118-y) and U10 (eq. (28), p. 813, 10.1109/TRO.2010.2065430). The notebook defaults $\alpha_z = 8$ and $\beta_z = 2$ are G09's (p. 9).

[2] S23 writes the clock with τ in (36) and the motion with Ω in (34)–(35), without relating the two. U10 states it: "The formula $\tau = 1/\Omega$ is used in the case of periodic movements" (p. 803, 10.1109/TRO.2010.2065430). The clock then turns at rate Ω, as in G09 eq. (6), p. 9.

Now picture the bumps themselves, much like the discrete ones you already know, arranged around a circle so that the pattern has no end.

∑ The math: eq. (37)–(38)

$$f(\phi) = \frac{\sum_{i=1}^{N} \Psi_i(\phi)\, w_i\, r}{\sum_{i=1}^{N} \Psi_i(\phi)} \tag{37}$$

Read it aloud: the push is a weighted average of the weights of the awake bumps, scaled by the amplitude r.

$$\Psi_i(\phi) = \exp\big(h\,(\cos(\phi - c_i) - 1)\big) \tag{38}$$

Read it aloud: bump i is largest, equal to 1, when the hand points at its center c_i, and it fades smoothly away from there.

The centers c_i are spread evenly around the circle, and a larger width parameter h makes narrower bumps.[3] Because of the cosine, a bump near the end of one lap joins the start of the next without a seam.

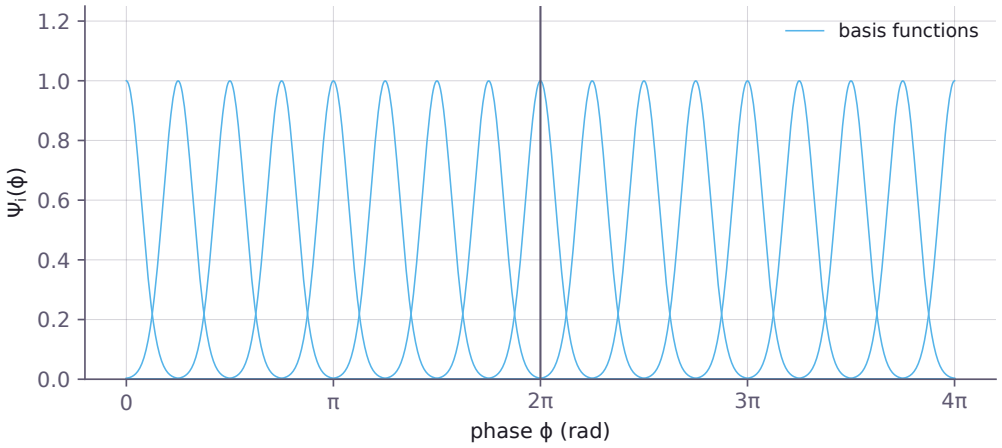


Figure 2. Eight periodic bumps over two periods, 0 to 4π. The second lap copies the first.

Learning the weights works as before: the recursive update (40)–(42) fits the weights one kernel at a time, and you will meet it in chapter 8.

[3] S23 gives no values for c_i and h. The kit follows G09: h "is set to $h = 2.5N$", and the c_i "are equally spaced between 0 and 2π in N steps" (p. 9, 10.1007/s10514-009-9118-y).

Speed and size

With the rhythm running, you can start turning the knobs: doubling r doubles the push, and the swing doubles along with it. Doubling Ω sends the clock round twice as fast, so each lap takes half the time while keeping its shape.

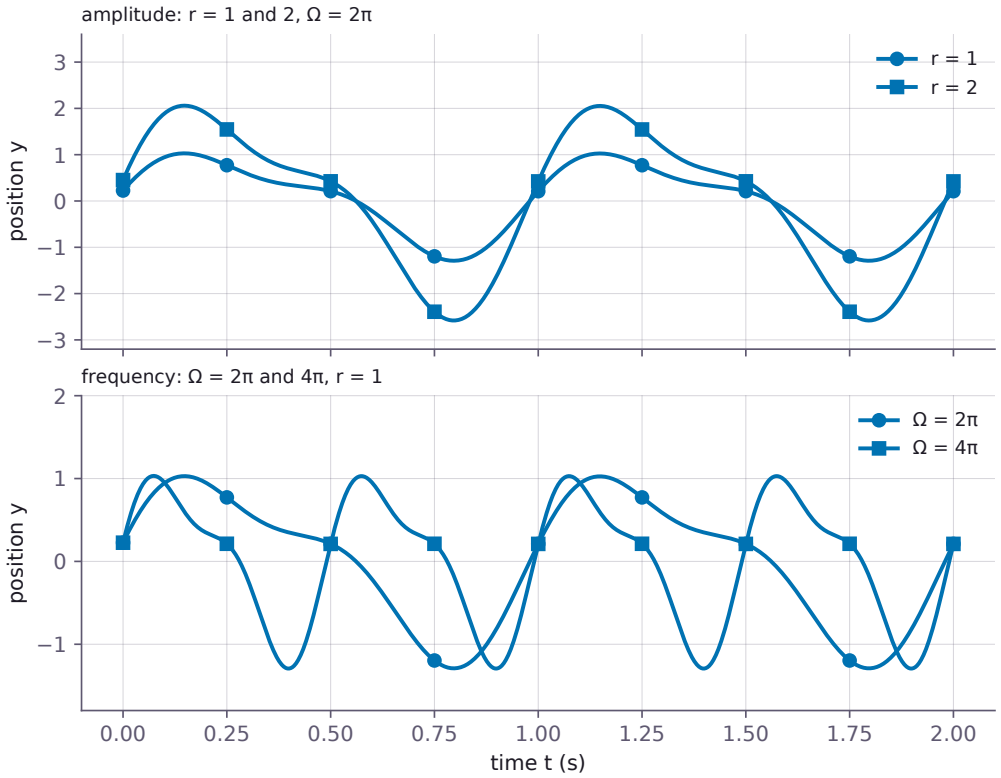


Figure 3. One learned rhythm replayed. Top: r = 1 and r = 2 at the same frequency, so the swing doubles. Bottom: Ω = 2π and Ω = 4π at r = 1, so the same shape runs twice as fast.

Moving the center

So far you have watched the rhythm swing around 0, because the spring in (34) has no goal and pulls toward $y = 0$. That suits a wiper swinging around zero, but imagine an arm whose wave should be centered at 30°, rather than 0°.

> ☼ **Intuition: a spring that needs an anchor**
> Picture the arm tied to a post by a rubber band: in (34), that post stands at 0. For a wave centered at 30°, you need to tie the band to a post at 30°, so that moving the post carries the whole wave along. This post is the anchor, which we write as g.

Σ The math: eq. (G09-6), (G09-8)–(G09-10)

$$\dot{\Phi} = \Omega \tag{G09-6}$$

Read it aloud: the phase advances at the frequency, as before.

$$\dot{z} = \Omega\Big(\underbrace{\alpha_z(\beta_z(g - y) - z)}_{\text{spring–damper term}} + \underbrace{\frac{\sum_{i=1}^{N} \Psi_i w_i r}{\sum_{i=1}^{N} \Psi_i}}_{\text{forcing term}} \Big) \tag{G09-8}$$

Read it aloud: the spring now pulls toward the anchor g, and the weighted push is scaled by r.

$$\dot{y} = \Omega z \tag{G09-9}$$

Read it aloud: the position follows z, scaled by the frequency.

$$\Psi_i = \exp\big(h\,(\cos(\Phi - c_i) - 1)\big) \tag{G09-10}$$

Read it aloud: the bumps are the same circular bumps as before.

You have now met variant D2 from G09, and setting $g = 0$ brings you back to (34).[4] You may wonder whether the plain form can still learn a rhythm centered at 30°; it can, because the weights absorb the offset. That hidden offset becomes a problem later, once you change r.

[4] Put $g = 0$ in (G09-8) and you get (34), with $\alpha = \alpha_z$ and $\beta = \beta_z$. U10 writes the same anchored form (eq. (28), p. 813, 10.1109/TRO.2010.2065430), and IJ13 calls g "an anchor point (or set point) for the oscillatory trajectory" (p. 336, 10.1162/NECO_a_00393).

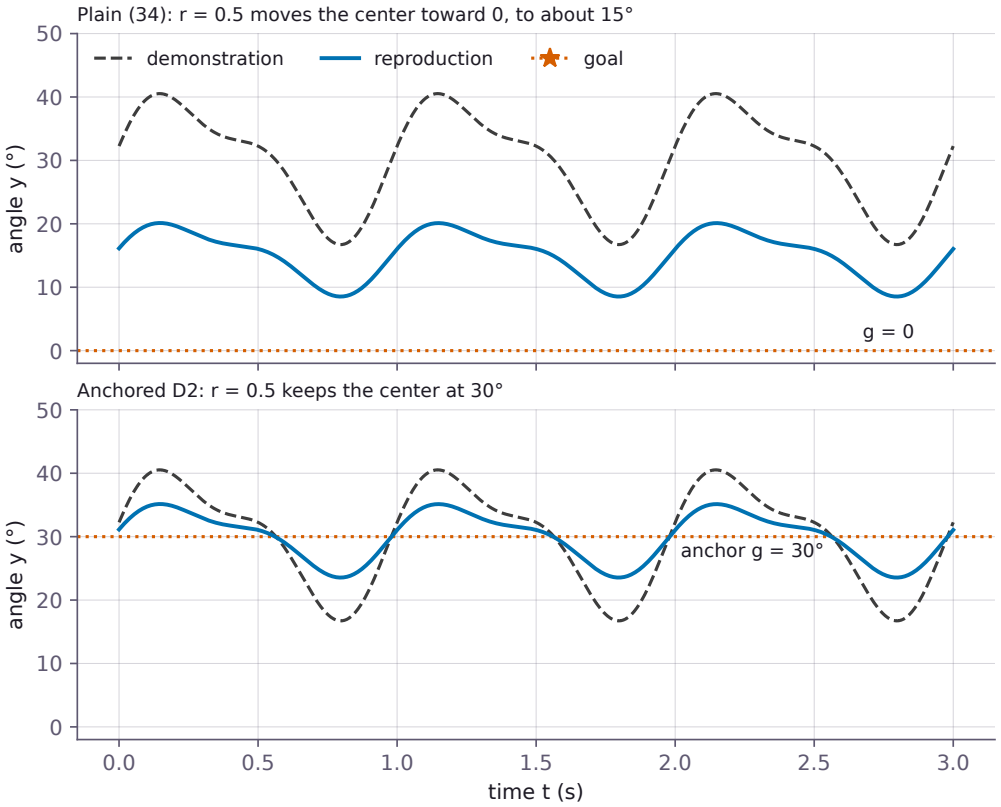


Figure 4. A rhythm centered at 30° replayed at r = 0.5. Top, plain (34): the whole offset shrinks, so the center falls to 15°. Bottom, anchored (G09-8) with g = 30°: the center stays at 30° and only the swing halves.

In the plain form, the offset travels inside the push, so r scales it along with everything else. Giving the offset its own home in g lets r change only the swing.

Starting from rest

Before you rely on that steady rhythm, watch what happens when the robot starts at rest: its first lap looks different from the ones that follow.

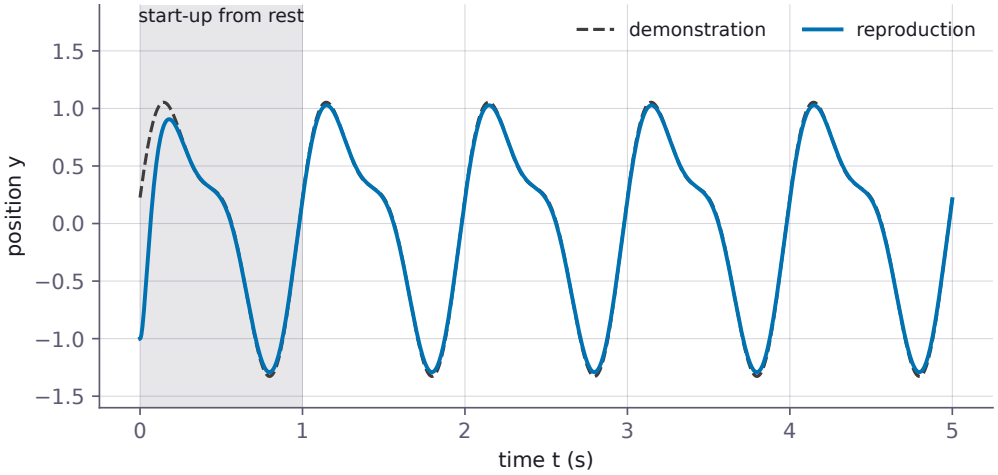


Figure 5. The reproduction started from rest over five periods. The first period, shaded as start-up, differs from the demonstration. Then the reproduction settles onto it.

Try it

▶ Try it: 02_periodic_playground.ipynb › Simulate 2.4 - Omega, r and number of basis functions
As you drag Ω, the beat speeds up while its shape stays; try r for a bigger or smaller swing. Explore N to find how few bumps still give a good wave, then tick "start from rest" to watch its first lap wobble before settling. There is no slider for the center g here.

▶ Try it: 03_create_your_own.ipynb › Create 3.4 - Define your wave
Here the arm waves around 30°, so you can move its anchor and watch the whole wave slide up or down while keeping its shape.

Gotcha

That wobble is a clue about what the robot learned: a rhythm to keep repeating, with no lesson for getting into it from rest.

⚠ Gotcha: the first laps are not taught
Symptom: the robot's first cycles from rest differ from your demonstration. Cause: a classical periodic DMP learns the rhythm; the dynamics decide how it starts. S23 says it "does not encode the transient motion needed to start the periodic one" (p. 1143). This trips up everyone: a walking robot's first step cannot be taught this

> way. Fix: start on the demonstration, the kit's default, or allow a few warm-up laps before relying on the robot.

◆

08

CHAPTER EIGHT

Teaching a rhythm

Learn a rhythm from a noisy recording, one lap at a time, and judge whether the robot really got it.

Take the robot's hand and shake it up and down a few times, showing it the wave you would like it to learn. A camera catches every wiggle, including the noise, leaving the robot with a messy recording from which to find a rhythm it can wave back on its own.

In Chapter 7 you met the periodic DMP with weights the kit had already learned; now you will walk through the learning itself, from that noisy recording to the numbers behind the wave.

The picture

Think of learning a song by ear, humming along as the chorus comes round and missing most of the notes on your first attempt. When the chorus returns, the part that caught you out a moment ago sounds a little better, and after a few laps the tune begins to fall into place.

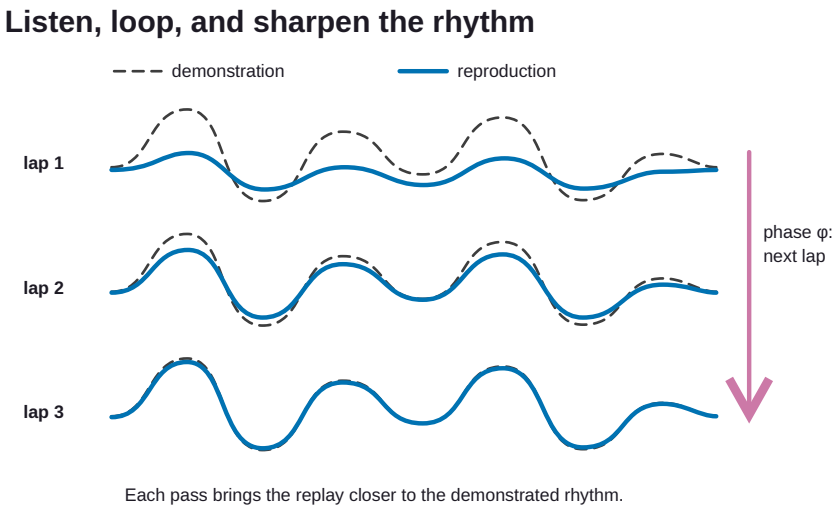


Figure 1. A looping melody line whose parts get sharper on every lap, like a song learned by ear.

Along the way, you have to decide how much to trust your memory of the first lap compared with what you heard just now. Hold too tightly to the old laps and you never adapt; trust only the newest and one bad note can throw you off. A number called the *forgetting factor* λ sets that balance.

The idea

As in Chapter 5, we follow the recording with a question in mind: “What push would have produced this demonstration?” The answer is the desired forcing value f_d, which we work out at every moment using the demonstrated position y_d, its velocity and its acceleration.

The acceleration is where that innocent-looking camera noise becomes troublesome, because taking differences of the positions twice makes the noise much larger. You can see the result in the figure below.

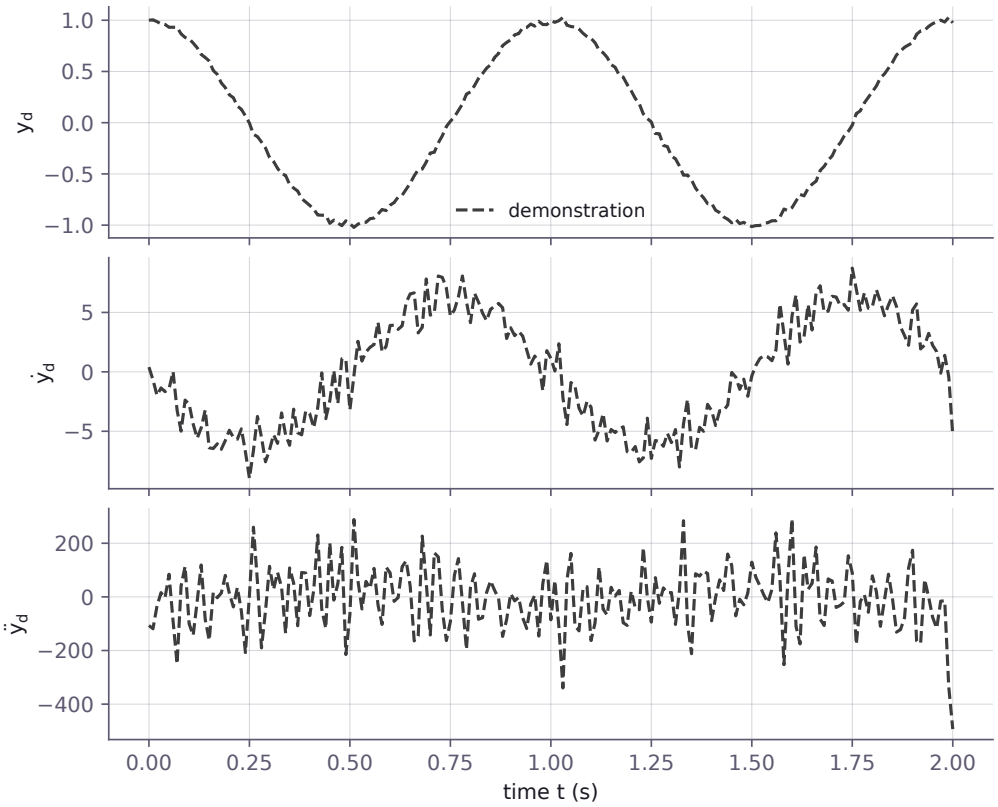


Figure 2. A noisy cosine demonstration, sampled every 0.01 s with noise of 0.02, and its raw derivatives: the acceleration is a mess of values in the hundreds.

You do not have to chase every spike: each bump only needs to find the average push that fits its own part of the cycle.

> ☼ **Intuition: each bump keeps its own running guess**
> Imagine every bump Ψ_i carrying its own number, the weight w_i, along with a small gauge of how sure it is. As the phase passes over, the bump wakes up, compares the push the demonstration needs with the push it currently gives, and nudges its weight to close the gap. Far from the phase it sleeps and barely changes; it is the waking moments, lap after lap, that let each weight settle.

The math

> ☼ **Intuition: a target, an error, a nudge**
> Once you know the push the demonstration needs, subtract the push the bump gives now and use a fraction of the difference to nudge its weight. That is the learning step you will repeat along the recording.

Let's begin with the push we need to explain: it comes from the demonstrated acceleration after accounting for what the spring alone would give.

Σ The math: eq. (39)

$$f_d(t_J) = \frac{\ddot{y}_d(t_J)}{\Omega^2} - \alpha_z \left(\beta_z(- y_d(t_J)) - \frac{\dot{y}_d(t_J)}{\Omega} \right) \tag{39}$$

Read aloud: the push needed at sample t_J is the demonstrated acceleration, divided by the frequency squared, minus the pull the spring and damper already supply.

Now you can give each bump i its own weight and gain P_i, the number that tells it how fast to learn. Before the recording begins, the gain starts at 1 and the weight starts at 0.

Σ The math: eq. (41), (42), (40)

$$e_r(t_J) = f_d(t_J) - w_i(t_J)\, r \tag{41}$$

Read aloud: the error of bump i is the push we want, minus the push its weight gives now.

$$P_i(t_{J+1}) = \frac{1}{\lambda} \left(P_i(t_J) - \frac{P_i(t_J)^2\, r^2}{\frac{\lambda}{\Psi_i} + P_i(t_J)\, r^2} \right) \tag{42}$$

Read aloud: the new gain is the old gain, shrunk a little by what was just learned, and divided by the forgetting factor λ so that old samples fade.

$$w_i(t_{J+1}) = w_i(t_J) + \Psi_i\, P_i(t_{J+1})\, r\, e_r(t_J) \tag{40}$$

Read aloud: the new weight is the old weight, plus the error, scaled by how awake the bump is (Ψ_i), by the new gain and by the amplitude r.

Keep the order close at hand: equation (40) needs the *new* gain $P_i(t_{J+1})$ from (42), which means you must compute (42) before (40).[1]

As you move through the recording, repeat this learning step at every sample for every bump. You can find it in S23 §2.2.1, p. 1143 (10.1177/02783649231201196). Let the demonstration cover whole periods, so that every bump receives the same number of visits.

[1] The order, and the reading of Ψ_i in (40) and (42) as $\Psi_i(\phi(t_J))$, follow transcription note P4 in the kit's notation file. The source prints the updates without spelling this out.

Watch the weights below as twelve bumps listen to four periods of a noisy cosine in one pass, each finding its part of the rhythm.

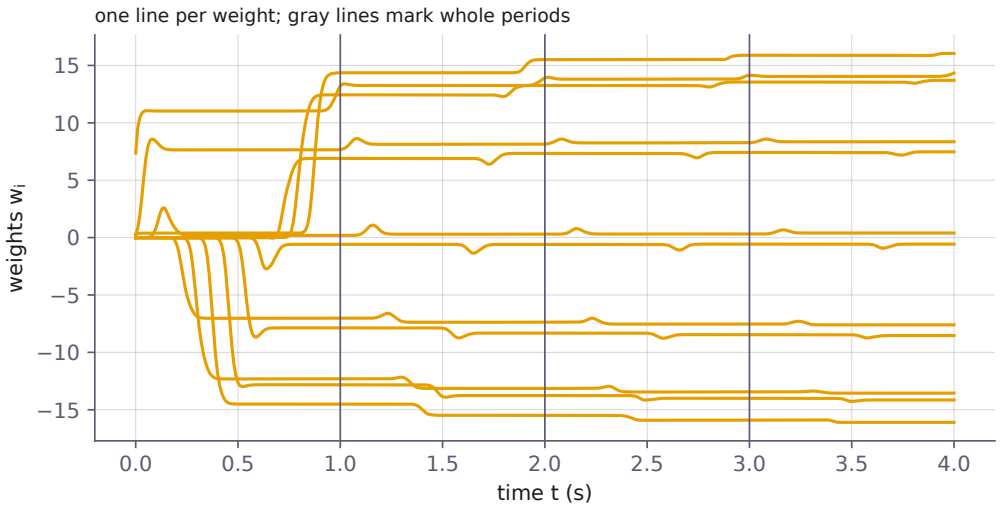


Figure 3. Twelve weights over four periods of a noisy cosine in one pass: each weight settles after its bump has been awake, and gray lines mark whole periods.

Once learning ends, you can put the recording aside and let the robot replay the rhythm from its learned weights alone.

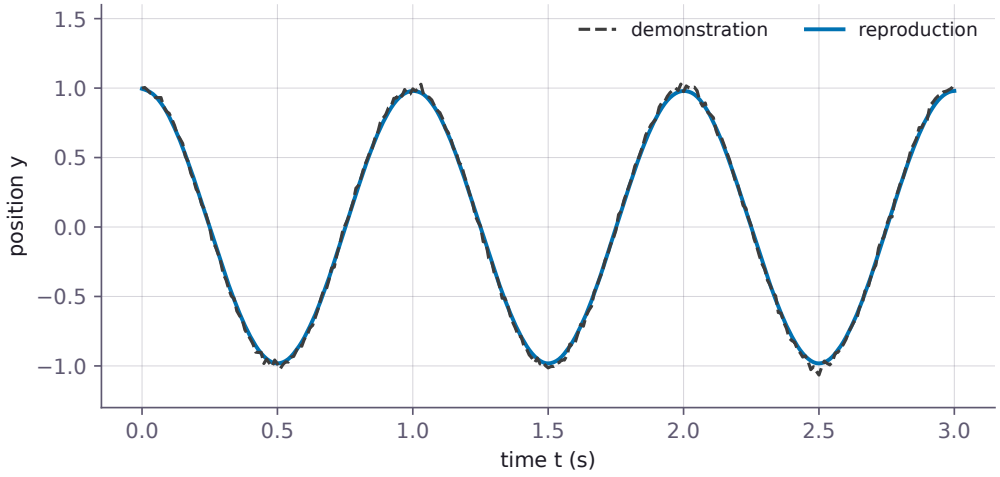


Figure 4. The dashed noisy demonstration over the solid reproduction for three periods, with N = 20 bumps.

Follow the smooth solid path against the noisy dashed demonstration: for these two figures, the kit blurred the recording a little before taking derivatives, as the Gotcha of Chapter 5 advises. The bumps then averaged away the noise that remained, giving you a smooth replay through a learned weighted sum rather than a copy of every sample.

Try it

Now bring a rhythm of your own to the robot, and see how closely it can wave it back.

▶ Try it: 02_periodic_playground.ipynb › Create 2.1 - Choose a rhythm
With `RHYTHM` set to `"mine"`, you can write your own `my_rhythm(t)`, a NumPy expression that repeats once per second. A sum of two sines gives you a place to begin; once it is ready, run the next cells and watch the robot replay your rhythm.

▶ Try it: 02_periodic_playground.ipynb › Test 2.3 - How close is the replay?
When you run this cell on your rhythm, it prints the largest gap over three periods as a percent of the swing, and says PASS under 5 %. You can explore what the bumps can capture by giving the rhythm sharp corners and watching the gap grow, because a small collection of bumps cannot follow a sharp corner.

For another turn at the controls, try *Create 2.5 - Challenge* in the same notebook, where you can ask the robot to wave three times per second at twice the swing.

Gotcha

⚠ Gotcha: sleeping bumps can blow up the update
Symptom: with many narrow bumps, Python raises a `ZeroDivisionError`, or NumPy prints "divide by zero encountered" warnings. Cause: equation (42) divides by Ψ_i, and far from a bump's center Ψ_i becomes so tiny that the computer rounds it to exactly 0. The term λ/Ψ_i then blows up to infinity, a trap that trips up everyone. Fix: once Ψ_i falls below a tiny threshold, skip that bump's update for the sample, because even a successful division would produce only a vanishingly small change. The kit follows this rule: a kernel whose Ψ_i is below 10^{-6} sleeps.

◆

09

CHAPTER NINE

Where to go next

Discover what the classical DMP cannot do yet, and find a path into the papers that explain each next step.

Your robot can now reach for a goal and keep a wiping rhythm, so you hand it a screwdriver and wait for it to turn its wrist to the right angle. It cannot do that yet. A chair moved into its path leaves it plowing straight on, and when a colleague grabs its arm mid-motion, it keeps pulling as though nothing happened. Those three encounters mark the edge of what you have built, and this chapter offers road signs toward the ideas that address them.

The picture

Imagine the moment after learning to ride a bike, when balancing, steering and braking begin to feel familiar and you look up at the roads ahead. Some climb hills, some carry traffic, and some lead toward cities whose names you have never heard.

The classical DMP is the bike you now know how to ride, and this chapter is a road map you can unfold beside it. You will find names and directions here, with a place to read about each route when you are ready to follow it.

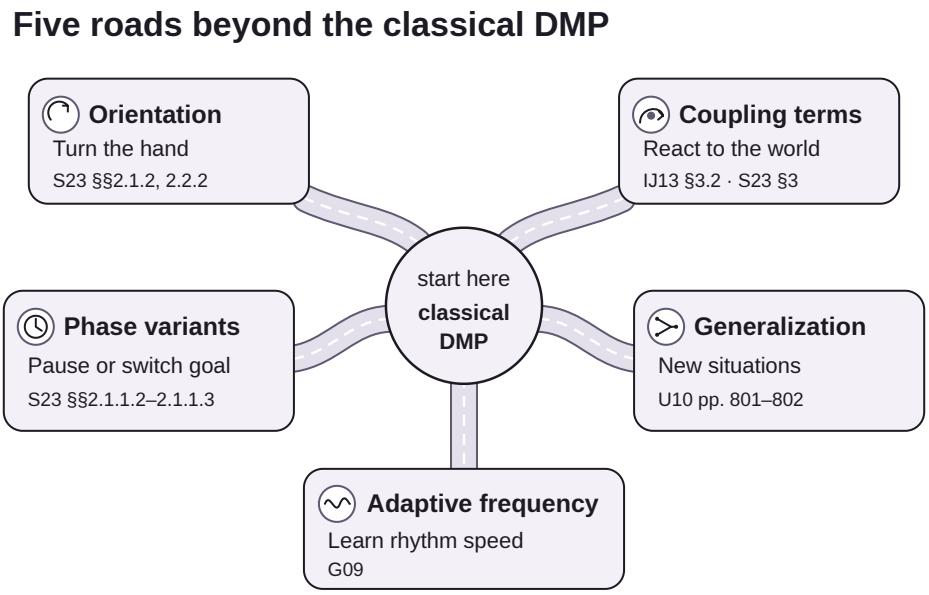


Figure 1. A road map that leads from the classical DMP to orientation, coupling, phase variants, generalization and adaptive frequency, each sign labeled with its reference.

The idea

You can carry the book's central picture with you in two sentences: a spring pulls the robot to a goal, and a learned push shapes the path along the way. A clock tells the push when to act. As you explore the roads ahead, watch for the piece each one changes.

> ☼ **Intuition: one piece changes per road**
> Keep the spring, the push and the clock in view as you look over the map. Each road leaves two parts alone and changes the third, or adds a fourth part alongside them, so knowing which part changes gives you a foothold when you open a paper.

The signs below give each road a name and a place to begin reading when a question draws you that way.

Orientation. Return to the screwdriver and picture how the robot's hand must turn: a position is a list of numbers that add up nicely, but two rotations do not just add. A DMP for that turning hand needs different tools, which you will find in S23 §2.1.2 for discrete motions and §2.2.2 for periodic ones.

Coupling terms. When the chair appears or someone pushes the arm, an extra push added to the DMP can help the robot react to the world. This is a coupling term, and you can read about using it to dodge obstacles or give way in IJ13 §3.2, then S23 §3.

Phase stopping, goal switching and other phase variables. So far the clock has run on its own, with the goal fixed during each motion. To explore what happens when you pause that clock, change the goal mid-motion, or choose a different kind of clock, follow S23 §2.1.1.2 and §2.1.1.3.

Generalizing to new situations. One demonstration gives you one motion, but imagine collecting several throws toward baskets in different places and wanting the robot to aim a throw toward a new basket. U10 labels each demonstration with its situation and uses regression to work out a new DMP for a situation it has never seen (U10 pp. 801–802).

Learning the frequency automatically. In chapter 7 you chose the periodic DMP's frequency yourself; G09 explores how the robot can find it by watching a rhythm. Adaptive-frequency oscillators placed in front of the DMP discover that frequency and pass it along, leaving the DMP to learn the shape.

The survey itself. S23 has been your companion throughout this book, and it offers a place to begin exploring. Along the way, its list of open-source DMP implementations in §6.2 lets you see how other people bring the same ideas into code.

Further down the road, the survey points to probabilistic and neural variants, and to learning the weights by reinforcement learning. You can begin with its overview and follow the citations toward whichever question interests you.

When you are ready to set out, these four sources and their DOIs will help you find your way:

- S23, the tutorial survey (IJRR): 10.1177/02783649231201196
- IJ13, Ijspeert et al., Neural Computation: 10.1162/NECO_a_00393
- G09, Gams et al., periodic movements: 10.1007/s10514-009-9118-y
- U10, Ude et al., generalization: 10.1109/TRO.2010.2065430

The math

You have already gathered every equation you need to run a classical discrete or periodic DMP in chapters 1 to 8, so this chapter adds no new ones. Each road above brings its own equations, which are best read in the papers alongside the explanations that give them meaning.

Keep this book's equations beside you as a dictionary when you read, because most extensions begin with the spring and the push you already know.

Try it

You can also keep exploring with your hands on the controls: the notebook below has both the discrete and the periodic DMP ready for something you imagine yourself.

There is no separate sandbox notebook yet, so this is the place to start shaping a new motion.

> **▶ Try it: 03_create_your_own.ipynb › Create 3.1 - Pick your letters**
> Choose your initials and watch two DMPs, one per direction, write them across the page; then let a different shape, goal or speed lead you into changing the code. See where you can take the robot beyond the motions this book has shown you.

Gotcha

> **⚠ Gotcha: papers use different conventions**
> You copy an equation from a paper into your code, and the robot does something strange, often because a convention changed along the way. This trips up everyone. Some papers scale the push f by $(g - y_0)$ while others leave it unscaled; some use τ for the time scale, and some use Ω for the frequency. Even a bump's width may arrive as h_i in one paper and σ_i in another. The fix is to match every symbol against the equations of chapters 1 to 8 before you type a single line of code.

◆

Sources

Every model equation in this book comes from one of four papers; the helper relations (E1)–(E3) are worked out from them, and (E4), the Euler step, is the kit's own. We cite them by DOI, with the section, equation and page where it helps.

Key	Reference	DOI
S23	Saveriano, Abu-Dakka, Kramberger, Peternel (2023). *Dynamic movement primitives in robotics: A tutorial survey.* International Journal of Robotics Research 42(13): 1133–1184.	10.1177/02783649231201196
IJ13	Ijspeert, Nakanishi, Hoffmann, Pastor, Schaal (2013). *Dynamical movement primitives: Learning attractor models for motor behaviors.* Neural Computation 25(2): 328–373.	10.1162/NECO_a_00393
G09	Gams, Ijspeert, Schaal, Lenarčič (2009). *On-line learning and modulation of periodic movements with nonlinear dynamical systems.* Autonomous Robots 27(1): 3–23.	10.1007/s10514-009-9118-y
U10	Ude, Gams, Asfour, Morimoto (2010). *Task-specific generalization of discrete and periodic dynamic movement primitives.* IEEE Transactions on Robotics 26(5): 800–815.	10.1109/TRO.2010.2065430

The main discrete and periodic equations are transcribed from S23 exactly as printed, quirks included. Where the printed form is ambiguous or leaves a value open, this book says so in a footnote and cites the paper that settles it.