

Package: PhysioTwin (via r-universe)

October 3, 2026

Title Digital-Twin Simulation and Data Assimilation for Physiological and Movement Systems

Version 0.1.2

Description A digital-twin engine that couples mechanistic simulation of movement and physiology with realistic sensor emulation and data assimilation, so a subject-specific model can be personalised from real measurements and then used for in-silico experiments. It ships a parameterised limb model (forward dynamics) and mechanistic physiological and network generators -- a Jansen-Rit neural-mass EEG column with an optional 1/f aperiodic background, a McSharry limit-cycle electrocardiogram, a Windkessel arterial blood-pressure model with a closed-loop baroreflex, a respiration waveform and respiratory-sinus-arrhythmia heart-rate-variability generator, and Kuramoto coupled-oscillator and Matsuoka central-pattern-generator rhythms. A forward sensor-emulation layer turns clean signals into realistic recordings (inertial-measurement-unit, optical-marker occlusion, surface-electromyography electrode and electrocardiogram lead), with richer chains for multi-channel electromyography crosstalk and amplifier characteristics, multi-lead electrocardiogram and analogue-to-digital conversion. A data-assimilation layer personalises the twin to recorded data with a family of state-space estimators (unscented, extended, particle and ensemble Kalman filters), likelihood-free Bayesian calibration (rejection and sequential Monte-Carlo approximate Bayesian computation), Markov-chain Monte- Carlo, a Gaussian-process surrogate and GP-accelerated (Bayesian-optimisation) calibration -- including personalising the closed-loop cardiovascular- respiratory twin to measured heart-rate and blood-pressure variability -- as well as particle Markov-chain Monte-Carlo and particle Gibbs (with ancestor sampling) for whole-waveform Bayesian fitting -- including a whole-record fit of the closed-loop cardiovascular model that identifies the baroreflex and respiratory gains a summary fit cannot, with either a resonant Mayer-oscillator or a broadband

first-order-autoregressive low-frequency component for records without a distinct Mayer peak -- and Fisher-information optimal experimental design for choosing the most informative experimental condition. Applications include single- and population-level in-silico intervention (perturb parameters and predict the change, with effect size, responder rate and statistical power), multi-modal coupling, and labelled synthetic training-data generation with sensor domain randomisation, all backed by a simulation validity-and-verification harness (identifiability, global sensitivity and validation against real recordings). A clinical decision loop ties these together as research tooling: a personalised twin is certified by a validation gate (goodness of fit and parameter identifiability) before it may advise, candidate interventions are scored with their predictive uncertainty and ranked on multiple criteria, mechanistic predictions are reconciled with external evidence rather than overriding it, and a follow-up measurement is checked against the predicted interval. A closed-loop cardiovascular-respiratory model couples a delayed baroreflex with respiration to produce emergent Mayer waves and respiratory sinus arrhythmia, and an optional bridge holds the twin's predictions to account against a real single-case (SCED/MCID/ICF) clinical evaluation.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Maintainer Yusuke Matsui <mail.to.matsui@gmail.com>

URL <https://github.com/x-biosignal/PhysioTwin>,
<https://x-biosignal.r-universe.dev/PhysioTwin>

BugReports <https://github.com/x-biosignal/PhysioTwin/issues>

Imports stats

Suggests testthat (>= 3.0.0), PhysioRehab, PhysioAppKit, knitr,
 rmarkdown

VignetteBuilder knitr

Additional_repositories <https://x-biosignal.r-universe.dev>

Config/testthat/edition 3

Repository <https://x-biosignal.r-universe.dev>

Date/Publication 2026-10-03 13:11:50 UTC

RemoteUrl <https://github.com/x-biosignal/PhysioTwin>

RemoteRef HEAD

RemoteSha f98025227960bb7ea910dd0083a4b8c81dcf64c2

Contents

abcCalibration	4
abcSMC	5
adcQuantize	7
adcSample	7
aperiodicNoise	8
bandPower	9
baroreflex	10
cardioRespiratory	11
cardioStateSpace	13
cardioSummary	14
clinicalDecision	15
cpgMatsuoka	16
crosstalkMatrix	17
defaultInterventions	18
detectRpeaks	18
ecgLead	19
ecgLeadSet	20
ecgMeasure	21
ecgMeasureLeads	22
ecgsyn	22
emgArray	24
emgArrayMeasure	25
emgElectrode	26
emgEnvelope	27
emgMeasure	27
ensembleKalmanFilter	28
evaluateIntervention	29
extendedKalmanFilter	30
fillMarkerGaps	31
generateTrainingData	32
gpCalibrate	33
gpEmulator	34
imuMeasure	35
imuSensor	36
insilicoIntervention	37
insilicoInterventionMultimodal	38
intervention	38
jansenRit	39
kuramoto	41
limbTwin	42
markerMeasure	43
markerSensor	44
metropolis	45
multimodalTwin	46
optimalDesign	47
particleFilter	48

particleGibbs	49
particleMCMC	50
personalizeCardio	52
personalizeCardioWaveform	53
personalizeTwin	54
populationIntervention	56
predict.gpEmulator	57
profileLikelihood	58
quantizationNoise	59
reconcileEvidence	60
rehabEvaluate	60
respiration	62
rsaTachogram	63
scoreInterventions	64
simulateMultimodal	65
simulateTwin	66
sobolIndices	67
twinReadiness	68
unscentedKalmanFilter	69
validateTwin	70
windkessel	72

Index 74

abcCalibration	<i>Approximate Bayesian Computation (likelihood-free parameter calibration)</i>
----------------	---

Description

Calibrates a simulator's parameters to observed data without a likelihood: parameters are drawn from prior, passed through `simulate` to produce summary statistics, and the draws whose summaries fall closest to observed (the smallest-distance accept fraction) are retained as posterior samples. Distances are Euclidean on summaries scaled by their simulated SD, so statistics on different scales contribute comparably. With `regression = TRUE` the accepted draws are corrected by a distance-weighted local-linear regression onto the observed summaries.

Usage

```
abcCalibration(
  simulate,
  observed,
  prior,
  n = 5000,
  accept = 0.02,
  regression = TRUE,
  seed = NULL
)
```

Arguments

<code>simulate</code>	function(theta) returning a numeric vector of summary statistics for a parameter vector theta.
<code>observed</code>	Observed summary statistics (same length as <code>simulate</code> 's output).
<code>prior</code>	function() returning one prior draw (a named numeric vector).
<code>n</code>	Number of prior simulations.
<code>accept</code>	Fraction of draws to retain (the ABC tolerance as a quantile).
<code>regression</code>	If TRUE, apply the Beaumont local-linear regression adjustment.
<code>seed</code>	Optional RNG seed.

Value

an `abc_posterior`: samples (accepted, adjusted parameter draws), raw (accepted draws before adjustment), mean, sd, quantiles (2.5/50/ 97.5%), threshold (accepted distance) and `accept_rate`.

References

Beaumont MA, Zhang W, Balding DJ (2002) *Genetics* 162:2025-2035; Pritchard JK et al. (1999).

See Also

[personalizeTwin\(\)](#), [profileLikelihood\(\)](#), [gpEmulator\(\)](#)

Examples

```
# recover the mean/SD of a Gaussian from its sample mean and SD
set.seed(1); obs <- c(mean = 2, sd = 1.5)
sim <- function(th) { x <- rnorm(200, th[1], th[2]); c(mean(x), sd(x)) }
pri <- function() c(m = runif(1, -3, 7), s = runif(1, 0.2, 4))
post <- abcCalibration(sim, obs, pri, n = 2000, accept = 0.02, seed = 1)
post$mean
```

abcSMC

Sequential Monte-Carlo ABC (likelihood-free posterior by tolerance annealing)

Description

A sequential Monte-Carlo approximate Bayesian computation sampler. A population of parameter particles is carried through a schedule of shrinking tolerances: at each level a particle is drawn from the previous population, perturbed by a Gaussian kernel (covariance twice the population's), simulated, and accepted if its summary distance is below the current tolerance; importance weights correct for the kernel. The tolerance for each level is an adaptive quantile of the previous population's distances. This concentrates on the posterior with far fewer simulations than one-shot rejection ABC.

Usage

```
abcSMC(
  simulate,
  observed,
  prior,
  prior_density = NULL,
  n_particles = 200,
  n_populations = 5,
  quantile = 0.5,
  max_tries = 2000L,
  seed = NULL
)
```

Arguments

<code>simulate</code>	function(theta) returning summary statistics.
<code>observed</code>	Observed summary statistics.
<code>prior</code>	function() returning one prior draw (named).
<code>prior_density</code>	Optional function(theta) prior density (for the weights); NULL assumes a (bounded) uniform prior.
<code>n_particles</code>	Particles per population.
<code>n_populations</code>	Number of tolerance levels.
<code>quantile</code>	Tolerance schedule: each level's tolerance is this quantile of the previous population's distances.
<code>max_tries</code>	Per-particle acceptance attempts before giving up (guards a too-tight tolerance).
<code>seed</code>	Optional RNG seed.

Value

an `abc_smc`: weighted posterior samples, weights, the tolerance schedule, and the weighted mean/sd.

References

Toni T et al. (2009) J R Soc Interface 6:187-202; Beaumont MA et al. (2009) Biometrika 96:983-990.

See Also

[abcCalibration\(\)](#), [metropolis\(\)](#)

Examples

```
set.seed(1); obs <- c(mean = 2, sd = 1.5)
sim <- function(th) { x <- rnorm(200, th[1], th[2]); c(mean(x), sd(x)) }
pri <- function() c(m = runif(1, -3, 7), s = runif(1, 0.2, 4))
post <- abcSMC(sim, obs, pri, n_particles = 100, n_populations = 4, seed = 1)
post$mean
```

 adcQuantize

Quantise a signal to a finite bit depth

Description

Rounds a signal to bits resolution over range, as an analogue-to-digital converter does. Optional (subtractive) dither randomises the quantisation error, decorrelating it from the signal.

Usage

```
adcQuantize(signal, bits = 12, range = NULL, dither = FALSE)
```

Arguments

signal	Numeric signal.
bits	Bit depth (e.g. 8, 12, 16).
range	c(lo, hi) full-scale range; defaults to the signal's range.
dither	If TRUE, add uniform dither of one LSB peak-to-peak before rounding.

Value

the quantised signal (clipped to range).

See Also

[adcSample\(\)](#), [quantizationNoise\(\)](#)

Examples

```
q <- adcQuantize(sin(seq(0, 6, length.out = 500)), bits = 8)
length(unique(round(q, 6))) <= 256
```

 adcSample

Resample a signal to a new rate with optional sampling jitter

Description

Resamples signal from fs_in to fs_out by linear interpolation, optionally perturbing each output sample time by Gaussian timing jitter – the aperture jitter of a real converter's clock.

Usage

```
adcSample(signal, fs_in, fs_out, jitter = 0)
```

Arguments

signal	Numeric signal sampled at fs_in.
fs_in, fs_out	Input and output sampling rates (Hz).
jitter	Sampling-time jitter SD, in input samples.

Value

a list: signal (resampled), fs (= fs_out), time.

See Also

[adcQuantize\(\)](#)

Examples

```
down <- adcSample(sin(seq(0, 12, length.out = 2000)), fs_in = 1000, fs_out = 250)
down$fs
```

aperiodicNoise	<i>Aperiodic (1/f) background noise</i>
----------------	---

Description

Generates zero-mean noise whose power spectrum falls as $1 / f^{\text{exponent}}$ – the aperiodic ("1/f") background that underlies real electrophysiological recordings and on which oscillatory rhythms sit. It is made by shaping white noise in the frequency domain. Added to a mechanistic oscillator (e.g. [jansenRit\(\)](#)), it turns an over-concentrated single-rhythm spectrum into a realistic broadband one.

Usage

```
aperiodicNoise(n, exponent = 1.5, sd = 1, seed = NULL)
```

Arguments

n	Number of samples.
exponent	Spectral slope: power scales as $1 / f^{\text{exponent}}$ (1 = pink, 2 = brown; a resting scalp EEG is typically ~1-2).
sd	Target standard deviation of the output.
seed	Optional RNG seed.

Value

a length-n numeric vector.

References

He BJ (2014) Scale-free brain activity. Trends Cogn Sci 18:480-487; Donoghue T et al. (2020) Nat Neurosci 23:1655-1665 (aperiodic/periodic).

See Also

[jansenRit\(\)](#)

Examples

```
x <- aperiodicNoise(1024, exponent = 1.5, seed = 1)
round(sd(x), 3)
```

bandPower

Relative band power of a simulated EEG

Description

Convenience helper: the fraction of spectral power of a jansenRit EEG (after dropping the transient) that falls in each conventional band.

Usage

```
bandPower(x)
```

Arguments

x a jansenRit object.

Value

a named numeric vector of relative power in delta, theta, alpha, beta and gamma.

See Also

[jansenRit\(\)](#)

Examples

```
bandPower(jansenRit(duration = 6, seed = 1))["alpha"]    # alpha-dominant
```

baroreflex

*Baroreflex closed-loop blood-pressure regulation***Description**

Closes the short-term baroreflex around the [windkessel\(\)](#) arterial model: each beat the mean arterial pressure is compared to a set-point and the next heart period (RR interval) is set to $RR_0 + \text{gain} * (\text{MAP} - \text{setpoint})$ – a higher pressure lengthens RR (slows the heart), lowering cardiac output and buffering the pressure (negative feedback; gain is the baroreflex sensitivity, s/mmHg). A step change in peripheral resistance at `perturb_time` (e.g. vasodilation) challenges the loop; with the reflex engaged the mean pressure is actively regulated back toward the set-point.

Usage

```
baroreflex(
  duration = 60,
  setpoint = 92,
  gain = 0.02,
  hr0 = 70,
  stroke_volume = 80,
  R = 1,
  C = 1.9,
  Zc = 0.05,
  systolic_frac = 0.3,
  perturb_time = 30,
  perturb_R = 0.7,
  reflex = TRUE,
  sfp = 250
)
```

Arguments

<code>duration</code>	Recording length (s).
<code>setpoint</code>	Regulated mean-arterial-pressure set-point (mmHg).
<code>gain</code>	Baroreflex sensitivity, RR change per unit pressure error (s/mmHg).
<code>hr0</code>	Baseline heart rate (bpm).
<code>stroke_volume, R, C, Zc, systolic_frac</code>	Windkessel parameters (see windkessel()).
<code>perturb_time</code>	Time of the resistance step (s); Inf for no perturbation.
<code>perturb_R</code>	Multiplier applied to R from <code>perturb_time</code> on (e.g. 0.7 = 30% vasodilation).
<code>reflex</code>	If FALSE, the RR interval is held at baseline (open loop) – the control for demonstrating regulation.
<code>sfp</code>	Output sampling rate (Hz).

Value

a baroreflex object: continuous time/pressure; per-beat beat_time, rr, hr, beat_map; the setpoint; and regulated (the post-perturbation steady-state absolute MAP error).

References

Ursino M (1998) Am J Physiol 275:H1733-H1747. deBoer RW et al. (1987) Am J Physiol 253:H680-H689.

See Also

[windkessel\(\)](#), [rsaTachogram\(\)](#)

Examples

```
br <- baroreflex(duration = 60, perturb_time = 30, perturb_R = 0.7)
br$regulated      # small residual MAP error -- pressure is buffered
```

cardioRespiratory	<i>Closed-loop cardiovascular-respiratory twin</i>
-------------------	--

Description

A beat-to-beat model in which arterial pressure, heart period and peripheral resistance evolve under a delayed, two-arm baroreflex, coupled to respiration. A Windkessel map sets each beat's systolic and diastolic pressure; the baroreflex adjusts the next heart period (fast vagal + delayed sympathetic) and the peripheral resistance (delayed sympathetic) from the sensed systolic pressure; respiration modulates stroke volume (a mechanical blood-pressure effect) and vagal tone (respiratory sinus arrhythmia). The delayed resistance/heart-period feedback, excited by small noise, produces an EMERGENT ~0.1 Hz Mayer wave in pressure and heart rate; respiration adds the respiratory-frequency oscillations. None of these rhythms is imposed.

Usage

```
cardioRespiratory(
  duration = 300,
  hr0 = 70,
  resp_rate = 12,
  S0 = 120,
  D0 = 80,
  R0 = 1,
  g_vagal = 0.01,
  g_symp = 0.007,
  symp_delay = 2L,
  g_resist = 0.01,
  resist_delay = 4L,
  rsa_amp = 0.035,
```

```

    resp_bp_amp = 2.5,
    starling = 0.1,
    noise = 0.02,
    respiration = TRUE,
    seed = NULL
)

```

Arguments

<code>duration</code>	Recording length (s).
<code>hr0</code>	Baseline heart rate (bpm).
<code>resp_rate</code>	Breathing rate (breaths/min).
<code>S0, D0</code>	Target systolic/diastolic pressure (mmHg).
<code>R0</code>	Baseline peripheral resistance (relative).
<code>g_vagal, g_symp</code>	Baroreflex heart-period gains (s/mmHg): fast vagal (current beat) and delayed sympathetic.
<code>symp_delay, resist_delay</code>	Sympathetic heart-period and resistance baroreflex delays (beats).
<code>g_resist</code>	Baroreflex resistance gain (per mmHg).
<code>rsa_amp</code>	Respiratory vagal modulation of heart period (s).
<code>resp_bp_amp</code>	Respiratory mechanical modulation of stroke volume (mmHg).
<code>starling</code>	Frank-Starling sensitivity of stroke volume to the preceding heart period (fractional).
<code>noise</code>	SD of the beat-to-beat neural noise (fraction) that excites the loop.
<code>respiration</code>	If FALSE, respiration is switched off (to show the Mayer wave is intrinsic to the baroreflex loop, not driven by breathing).
<code>seed</code>	Optional RNG seed.

Value

a `cardio_respiratory` object: per-beat time, `rr`, `hr`, systolic, diastolic, resistance; the HRV and systolic-pressure LF/HF band summaries `hrv` and `bpv` (each with `lf_power/hf_power/lf_peak/hf_peak`); and the baroreflex sensitivity `brs` (ms/mmHg) recovered from the beat series.

References

de Boer RW, Karemaker JM, Strackee J (1987) Am J Physiol 253:H680-H689; Julien C (2006) Cardiovasc Res 70:12-21 (Mayer waves).

See Also

[windkessel\(\)](#), [baroreflex\(\)](#), [rsaTachogram\(\)](#)

Examples

```
cr <- cardioRespiratory(duration = 240, resp_rate = 12, seed = 1)
cr$bpv$lf_peak      # ~0.1 Hz Mayer wave in blood pressure
cr$hrv$hf_peak      # respiratory-frequency RSA in heart rate
```

cardioStateSpace	<i>Closed-loop cardiovascular state-space model (for whole-record fitting)</i>
------------------	--

Description

A linear-Gaussian state-space representation of the RR-interval series: a damped second-order Mayer oscillator (the delayed-baroreflex resonance) drives the heart period through a gain g , and respiration adds a high-frequency (respiratory sinus arrhythmia) term of amplitude A_{rsa} . Suitable as the `build_model` argument of [particleMCMC\(\)](#) / [particleGibbs\(\)](#).

Usage

```
cardioStateSpace(
  g = 8,
  rho = 0.9,
  omega_m = 0.53,
  A_rsa = 20,
  resp,
  rr0 = 850,
  q_mayer = 1,
  r_obs = 9,
  lf_model = c("oscillator", "ar1"),
  phi = 0.9
)
```

Arguments

<code>g</code>	Low-frequency gain onto the RR interval (ms per unit state).
<code>rho</code>	Mayer-oscillator damping (0..1; nearer 1 = sharper low-frequency peak; used when <code>lf_model = "oscillator"</code>).
<code>omega_m</code>	Mayer angular frequency (radians per beat; "oscillator" only).
<code>A_rsa</code>	Respiratory (RSA) amplitude on the RR interval (ms).
<code>resp</code>	Respiratory regressor sampled at the beats (length = number of beats).
<code>rr0</code>	Mean RR interval (ms).
<code>q_mayer, r_obs</code>	Low-frequency process-noise and RR observation-noise variances.
<code>lf_model</code>	Low-frequency model: "oscillator" (a resonant, second-order damped Mayer oscillator, for records with a distinct Mayer peak) or "ar1" (a first-order autoregressive process giving BROADBAND low-frequency power without a peak, for records whose low-frequency variability is not a sharp rhythm).
<code>phi</code>	First-order autoregressive coefficient (0..1; used when <code>lf_model = "ar1"</code>); nearer 1 = more low-frequency power.

Value

a state-space model list f, h (`function(x, k)`), Q, R, x_0, P_0 .

See Also

[personalizeCardioWaveform\(\)](#), [particleMCMC\(\)](#), [cardioRespiratory\(\)](#)

Examples

```
resp <- sin(2 * pi * 0.25 * cumsum(rep(0.85, 200)))
m <- cardioStateSpace(g = 8, rho = 0.9, omega_m = 0.53, A_rsa = 20, resp = resp)
m$h(c(1, 0), 1)
```

cardioSummary

Summarise a closed-loop cardiovascular-respiratory recording

Description

The heart-rate/blood-pressure variability summaries used to personalise the closed-loop twin: baroreflex sensitivity (brs, ms/mmHg), the Mayer-wave low-frequency power of blood pressure (bp_lf) and the respiratory high-frequency power of heart rate (hr_hf).

Usage

```
cardioSummary(cr)
```

Arguments

`cr` a `cardio_respiratory` object from [cardioRespiratory\(\)](#).

Value

a named numeric vector `c(brs, bp_lf, hr_hf)`.

See Also

[cardioRespiratory\(\)](#), [personalizeCardio\(\)](#)

Examples

```
cardioSummary(cardioRespiratory(duration = 120, seed = 1))
```

clinicalDecision	<i>Run the clinical decision loop end-to-end</i>
------------------	--

Description

Personalises a twin to a patient's IMU data, gates it with [twinReadiness\(\)](#), and – only reporting through the gate's confidence – scores and ranks the candidate interventions. A one-call demonstration of the loop; the returned object holds the fit, the readiness verdict and the ranked recommendations.

Usage

```
clinicalDecision(  
  imu,  
  prior,  
  estimate = "strength",  
  candidates = defaultInterventions(),  
  outcome = "rom",  
  method = c("optim", "ukf"),  
  dt = 0.005,  
  n_mc = 200,  
  weights = c(efficacy = 1, certainty = 1, risk = 1, burden = 0.5),  
  seed = NULL  
)
```

Arguments

imu	Patient IMU data (as from imuMeasure()).
prior	A movement_twin prior/initial guess for personalizeTwin() .
estimate	Parameters to personalise.
candidates	Candidate interventions (default defaultInterventions()).
outcome	Outcome to optimise.
method	Personalisation method ("optim"/"ukf").
dt, n_mc, weights, seed	As in scoreInterventions() .

Value

a clinical_decision: fit, readiness, and scores.

See Also

[personalizeTwin\(\)](#), [twinReadiness\(\)](#), [scoreInterventions\(\)](#)

Examples

```
truth <- limbTwin(strength = 6, damping = 0.6, stiffness = 3)
imu <- imuMeasure(simulateTwin(truth, 6, 0.01), imuSensor(gyro_noise = 0.03), seed = 1)
dec <- clinicalDecision(imu, limbTwin(strength = 4), estimate = "strength",
                        dt = 0.01, n_mc = 40, seed = 1)
dec$scores$ranking$intervention[1]
```

cpgMatsuoka

Matsuoka central pattern generator (half-centre oscillator)

Description

Simulates a Matsuoka two-neuron half-centre oscillator: two mutually-inhibiting neurons with fatigue/adaptation produce a self-sustained anti-phase rhythm – the mechanistic building block of locomotor central pattern generators. The output is the difference of the two rectified activities.

Usage

```
cpgMatsuoka(
  tau = 0.25,
  tau2 = 0.5,
  beta = 2.5,
  gamma = 2.5,
  u = 1,
  duration = 20,
  dt = 0.005
)
```

Arguments

tau, tau2	Rise and adaptation time constants (s).
beta	Adaptation (self-inhibition) strength.
gamma	Mutual-inhibition strength.
u	Tonic drive.
duration, dt	Simulation length and step (s).

Value

a cpg object: time, output (rhythmic drive $y_1 - y_2$), y_1 , y_2 (the two neuron outputs), and frequency (Hz, estimated from the output).

References

Matsuoka K (1985) Biol Cybern 52:367-376.

See Also

[kuramoto\(\)](#), [jansenRit\(\)](#)

Examples

```
cpg <- cpgMatsuoka(duration = 20)
cpg$frequency      # a sustained locomotor-like rhythm
```

crosstalkMatrix	<i>Electrode crosstalk mixing matrix</i>
-----------------	--

Description

A symmetric, energy-normalised mixing matrix: each channel keeps most of its own signal and picks up crosstalk of an immediate neighbour, crosstalk^2 of the next, and so on (rows sum to one). $\text{crosstalk} = 0$ gives the identity.

Usage

```
crosstalkMatrix(channels, crosstalk = 0.15)
```

Arguments

channels	Number of channels.
crosstalk	Per-neighbour crosstalk fraction (0..1).

Value

a channels x channels mixing matrix.

See Also

[emgArray\(\)](#), [emgArrayMeasure\(\)](#)

Examples

```
round(crosstalkMatrix(3, 0.2), 2)
```

defaultInterventions *A default library of movement-twin interventions*

Description

A small library of candidate interventions for the single-joint movement twin: strengthening, tone reduction, stretching, an orthosis, and a no-intervention control.

Usage

```
defaultInterventions()
```

Value

a list of twin_intervention objects.

See Also

[intervention\(\)](#), [scoreInterventions\(\)](#)

Examples

```
names(defaultInterventions())
```

detectRpeaks *Remove baseline wander and detect R-peaks in an ECG*

Description

High-passes the signal (subtracting a ~1 s moving-average trend) and returns the indices of R waves – local maxima above a fraction of the signal range, separated by a physiological refractory gap.

Usage

```
detectRpeaks(x, fs, refractory = 0.3, thresh = 0.5)
```

Arguments

x	An ECG signal (numeric).
fs	Sampling rate (Hz).
refractory	Minimum spacing between R-peaks (s).
thresh	Peak threshold as a fraction of the (de-trended) signal range.

Value

the integer indices of detected R-peaks.

See Also[ecgMeasure\(\)](#)**Examples**

```
e <- ecgsyn(duration = 10, seed = 1)
length(detectRpeaks(e$ecg, e$sfecg))
```

ecgLead	<i>Specify ECG lead / electrode characteristics</i>
---------	---

Description

Bundles the ECG recording chain's artefacts: baseline wander, powerline interference, muscle noise, motion artefact, gain and offset.

Usage

```
ecgLead(
  gain = 1,
  noise = 0.01,
  powerline_hz = 50,
  powerline_amp = 0.05,
  baseline_amp = 0.15,
  baseline_hz = 0.25,
  motion_amp = 0,
  offset = 0,
  fs = 250
)
```

Arguments

gain	Electrode/amplifier gain.
noise	High-frequency muscle-noise SD (mV).
powerline_hz	Mains frequency (Hz).
powerline_amp	Powerline interference amplitude (mV).
baseline_amp, baseline_hz	Baseline-wander amplitude (mV) and frequency (Hz, typically respiratory ~0.25 Hz).
motion_amp	Motion-artefact amplitude (mV; a slow random walk).
offset	Electrode DC offset (mV).
fs	Sampling rate (Hz).

Value

an `ecg_lead` object.

See Also

[ecgMeasure\(\)](#), [detectRpeaks\(\)](#), [ecgsyn\(\)](#)

Examples

```
ecgLead(baseline_amp = 0.2, powerline_hz = 60)
```

ecgLeadSet

Specify a multi-lead ECG lead set

Description

Bundles a set of named ECG leads derived from one cardiac source. Each lead has a projection gain and a morphology (derivative-mixing) weight – so leads differ in shape, not only amplitude – plus baseline wander, shared mains interference and muscle noise.

Usage

```
ecgLeadSet(
  leads = c("II", "V2", "V5"),
  gains = NULL,
  baseline_amp = 0.15,
  baseline_hz = 0.25,
  powerline_hz = 50,
  powerline_amp = 0.05,
  noise = 0.01,
  fs = 250
)
```

Arguments

leads	Character vector of lead names (e.g. "II", "V2", "V5").
gains	Optional per-lead projection gains; defaults to standard-ish values.
baseline_amp, baseline_hz	Baseline-wander amplitude (mV) and frequency (Hz).
powerline_hz, powerline_amp	Mains frequency (Hz) and amplitude (mV).
noise	Muscle-noise SD (mV).
fs	Sampling rate (Hz).

Value

an `ecg_lead_set` object.

See Also

[ecgMeasureLeads\(\)](#), [ecgLead\(\)](#), [ecgsyn\(\)](#)

Examples

```
ecgLeadSet(leads = c("II", "V2", "V5"))
```

ecgMeasure

Measure a clean ECG through a lead/electrode (forward model)

Description

Adds the electrode chain's artefacts – baseline wander, powerline interference, muscle noise, optional motion artefact, gain and offset – to a clean ECG.

Usage

```
ecgMeasure(ecg, sensor = ecgLead(), seed = NULL)
```

Arguments

ecg	Clean ECG signal (mV), e.g. from ecgsyn() .
sensor	An <code>ecg_lead</code> from ecgLead() .
seed	Optional RNG seed.

Value

an `ecg_measurement`: signal (recorded lead, mV), clean, time, fs, sensor.

See Also

[ecgLead\(\)](#), [detectRpeaks\(\)](#)

Examples

```
e <- ecgsyn(duration = 10, seed = 1)
rec <- ecgMeasure(e$ecg, ecgLead(baseline_amp = 0.2, fs = e$fsfecg), seed = 1)
sd(rec$signal)
```

ecgMeasureLeads	<i>Measure a clean ECG as a multi-lead recording (forward model)</i>
-----------------	--

Description

Projects one clean ECG onto a set of leads: each lead is a gain- and morphology-weighted mix of the source and its derivative, plus independent baseline wander, shared mains interference and independent muscle noise.

Usage

```
ecgMeasureLeads(ecg, sensor = ecgLeadSet(), seed = NULL)
```

Arguments

ecg	Clean ECG signal (mV), e.g. from ecgsyn() .
sensor	An <code>ecg_lead_set</code> from ecgLeadSet() .
seed	Optional RNG seed.

Value

an `ecg_multilead`: signals (`n_time` x `n_leads`, columns named by lead), leads, time, fs, sensor.

See Also

[ecgLeadSet\(\)](#), [ecgMeasure\(\)](#)

Examples

```
e <- ecgsyn(duration = 10, seed = 1)
ml <- ecgMeasureLeads(e$ecg, ecgLeadSet(fs = e$fsfecg), seed = 1)
dim(ml$signals)
```

ecgsyn	<i>ECGSYN mechanistic ECG generator (McSharry dynamical model)</i>
--------	--

Description

Generates a realistic ECG from the ECGSYN 3-state limit-cycle model: P/Q/R/S/T waves arise as the trajectory passes Gaussian events, and heart-rate variability is injected via a bimodal RR tachogram. The output has physiological morphology AND HRV.

Usage

```
ecgsyn(
  duration = 10,
  hr = 60,
  sfecg = 256,
  hrv_sd = 0.04,
  lfhratio = 0.5,
  anoise = 0,
  scale_mv = c(-0.4, 1.2),
  theta = c(-60, -15, 0, 15, 90) * pi/180,
  a = c(1.2, -5, 30, -7.5, 0.75),
  b = c(0.25, 0.1, 0.1, 0.1, 0.4),
  seed = NULL
)
```

Arguments

duration	Length (s).
hr	Mean heart rate (bpm).
sfecg	Sampling frequency (Hz).
hrv_sd	RR-interval standard deviation (s) – the amount of HRV (0 = fixed rate).
lfhratio	LF/HF power ratio of the RR spectrum (autonomic balance).
anoise	Additive measurement-noise SD (mV).
scale_mv	Length-2 physiological output range <code>c(min, max)</code> in mV that the raw model output is affinely mapped onto (McSharry default <code>c(-0.4, 1.2)</code>); NULL keeps the raw model units.
theta, a, b	P,Q,R,S,T event angles (rad), amplitudes and widths (McSharry defaults).
seed	Optional RNG seed.

Value

an ecgsyn object: time, ecg, the rr intervals, beats (R-peak times), sfecg.

References

McSharry PE, Clifford GD, Tarassenko L, Smith LA (2003) IEEE Trans Biomed Eng 50:289-294.

See Also

[jansenRit\(\)](#), [windkessel\(\)](#), [rsaTachogram\(\)](#), [kuramoto\(\)](#), [validateTwin\(\)](#)

Examples

```
e <- ecgsyn(duration = 6, hr = 60, sfecg = 256, seed = 1)
range(e$ecg)      # P/Q/R/S/T morphology
length(e$beats)   # ~ 6 beats
```

emgArray

*Specify a surface-EMG array (crosstalk + amplifier)***Description**

Bundles a multi-channel surface-EMG measurement chain: inter-electrode crosstalk, amplifier gain, a common-mode-rejected mains interference, a DC-blocking high-pass, a passband and thermal noise.

Usage

```
emgArray(
  channels = 4,
  crosstalk = 0.15,
  gain = 1000,
  noise = 0.005,
  powerline_hz = 50,
  powerline_amp = 0.05,
  cmrr_db = 80,
  hp_cutoff = 10,
  offset = 0,
  bandwidth = c(20, 450),
  fs = 1000
)
```

Arguments

channels	Number of electrode channels.
crosstalk	Per-neighbour crosstalk fraction (see crosstalkMatrix()).
gain	Amplifier gain.
noise	Thermal-noise SD (referred to the electrode).
powerline_hz, powerline_amp	Mains frequency (Hz) and its (common-mode) amplitude.
cmrr_db	Common-mode rejection ratio (dB) attenuating the mains interference.
hp_cutoff	DC-blocking high-pass corner (Hz).
offset	Electrode DC offset (a constant the DC-blocking high-pass removes).
bandwidth	Passband c(low, high) (Hz).
fs	Sampling rate (Hz).

Value

an emg_array object.

See Also

[emgArrayMeasure\(\)](#), [crosstalkMatrix\(\)](#), [emgElectrode\(\)](#)

Examples

```
emgArray(channels = 4, crosstalk = 0.2, cmrr_db = 90)
```

emgArrayMeasure	<i>Measure per-muscle activations with a surface-EMG array (forward model)</i>
-----------------	--

Description

Turns a matrix of per-muscle activation envelopes into the multi-channel raw sEMG an electrode array records: each channel's activation-modulated Gaussian sEMG is band-limited, the channels are MIXED through the crosstalk matrix, and the amplifier adds CMRR-rejected mains interference, a DC-blocking high-pass, thermal noise and gain.

Usage

```
emgArrayMeasure(activations, sensor = emgArray(), seed = NULL)
```

Arguments

activations	An <code>n_time</code> x <code>channels</code> matrix of non-negative activation envelopes (one column per muscle/channel).
sensor	An <code>emg_array</code> from emgArray() .
seed	Optional RNG seed.

Value

an `emg_array_measurement`: raw (`n_time` x `channels`), clean (uncontaminated, un-mixed, amplified), time, fs, the crosstalk matrix, sensor.

See Also

[emgArray\(\)](#), [crosstalkMatrix\(\)](#)

Examples

```
act <- matrix(pmax(0, sin(outer(seq(0, 6, len = 1200), 1:3))), ncol = 3)
m <- emgArrayMeasure(act, emgArray(channels = 3, crosstalk = 0.2), seed = 1)
dim(m$raw)
```

emgElectrode

Specify surface-EMG electrode/amplifier characteristics

Description

Bundles the surface-EMG measurement chain: passband, powerline interference, baseline/motion wander, thermal noise, amplifier gain and clipping.

Usage

```
emgElectrode(
    gain = 1000,
    noise = 0.005,
    powerline_hz = 50,
    powerline_amp = 0.05,
    baseline_amp = 0.02,
    baseline_hz = 1,
    bandwidth = c(20, 450),
    fs = 1000,
    clip = Inf
)
```

Arguments

gain	Amplifier gain.
noise	Thermal (white) noise SD, referred to the electrode.
powerline_hz	Mains frequency (Hz); odd harmonics are added.
powerline_amp	Powerline interference amplitude.
baseline_amp, baseline_hz	Baseline/motion-wander amplitude and frequency (Hz).
bandwidth	Passband c(low, high) (Hz).
fs	Sampling rate (Hz).
clip	Saturation level (loutputl clipped to this; Inf = none).

Value

an emg_electrode object.

See Also

[emgMeasure\(\)](#), [emgEnvelope\(\)](#), [imuSensor\(\)](#)

Examples

```
emgElectrode(powerline_hz = 60, powerline_amp = 0.08)
```

emgEnvelope	<i>EMG linear envelope (rectify + low-pass)</i>
-------------	---

Description

The standard sEMG envelope: full-wave rectification followed by a low-pass at cutoff Hz (FFT-based), recovering the activation.

Usage

```
emgEnvelope(x, fs, cutoff = 5)
```

Arguments

x	A raw or clean EMG signal.
fs	Sampling rate (Hz).
cutoff	Low-pass cut-off (Hz).

Value

the linear envelope (same length as x).

See Also

[emgMeasure\(\)](#)

Examples

```
emgEnvelope(rnorm(1000) * abs(sin(seq(0, 6, length.out = 1000)))), 1000)
```

emgMeasure	<i>Measure muscle activation with a surface-EMG electrode (forward model)</i>
------------	---

Description

Turns an activation envelope into a realistic raw sEMG: an activation-modulated Gaussian process band-passed to the electrode's passband (the clean EMG), then contaminated with powerline interference, baseline/motion wander and thermal noise and amplified.

Usage

```
emgMeasure(activation, sensor = emgElectrode(), seed = NULL)
```

Arguments

activation	Non-negative activation envelope (0..1), length n.
sensor	An emg_electrode from emgElectrode() .
seed	Optional RNG seed.

Value

an emg_measurement: raw (contaminated, amplified sEMG), clean (uncontaminated band-limited EMG, amplified), time, activation, fs, sensor.

See Also

[emgElectrode\(\)](#), [emgEnvelope\(\)](#)

Examples

```
act <- pmax(0, sin(seq(0, 4 * pi, length.out = 2000)))
emg <- emgMeasure(act, emgElectrode(fs = 1000), seed = 1)
cor(emgEnvelope(emg$clean, 1000), act)
```

ensembleKalmanFilter *Ensemble Kalman filter (Monte-Carlo covariance state estimator)*

Description

A stochastic ensemble Kalman filter for $x_k = f(x_{k-1}, k) + w$, $y_k = h(x_k, k) + v$. An ensemble of members carries the state distribution; its sample cross- and innovation-covariances form the Kalman gain, and the members are updated with perturbed observations. Scales to high-dimensional states where a full covariance is impractical. Same interface as [unscentedKalmanFilter\(\)](#).

Usage

```
ensembleKalmanFilter(f, h, y, Q, R, x0, P0, n_ensemble = 100, seed = NULL)
```

Arguments

f, h	Process and measurement models function(x, k).
y	Measurement matrix (n_steps x m).
Q, R	Process- and measurement-noise covariances.
x0, P0	Initial state mean and covariance.
n_ensemble	Ensemble size.
seed	Optional RNG seed.

Value

an enkf_result: state (n_steps x L ensemble means), cov (final ensemble covariance), innovation.

References

Evensen G (1994) J Geophys Res 99:10143-10162; Burgers et al. (1998).

See Also

[unscentedKalmanFilter\(\)](#), [extendedKalmanFilter\(\)](#), [particleFilter\(\)](#)

Examples

```
set.seed(1); y <- matrix(sin(seq(0, 6, length.out = 50)) + rnorm(50, 0, .1), 50, 1)
fit <- ensembleKalmanFilter(function(x, k) x, function(x, k) x, y,
                           Q = matrix(0.05), R = matrix(0.01), x0 = 0, P0 = matrix(1), n_ensemble = 80)
fit$state[50]
```

evaluateIntervention	<i>Evaluate a delivered intervention against its prediction (close the loop)</i>
----------------------	--

Description

After an intervention is delivered and a follow-up outcome is measured, check whether the observed change fell within the twin's predicted interval – the empirical test that closes the personalise -> predict -> act -> evaluate loop and calibrates trust in the twin.

Usage

```
evaluateIntervention(
  twin,
  chosen,
  observed_delta,
  readiness = NULL,
  outcome = "rom",
  outcome_sd = 0,
  n_mc = 200,
  duration = 6,
  dt = 0.005,
  theta0 = NULL,
  seed = NULL
)
```

Arguments

twin	The personalised movement_twin.
chosen	The delivered twin_intervention.
observed_delta	The measured outcome change at follow-up.
readiness	A twin_readiness (for the parameter posterior).

outcome, n_mc, duration, dt, theta0, seed
 As in [scoreInterventions\(\)](#).
 outcome_sd Outcome observation/session variability (SD). The check uses a PREDICTION interval combining the parameter (epistemic) uncertainty with this (aleatoric) noise, so a realistic follow-up is covered – a parameter-only interval is over-narrow for a real measurement.

Value

an intervention_evaluation: predicted effect and prediction ci, the observed change, and whether it was within_interval.

See Also

[scoreInterventions\(\)](#), [clinicalDecision\(\)](#)

Examples

```
truth <- limbTwin(strength = 6)
imu <- imuMeasure(simulateTwin(truth, 6, 0.01), imuSensor(gyro_noise = 0.03), seed = 1)
fit <- personalizeTwin(imu, limbTwin(strength = 4), estimate = "strength", dt = 0.01)
rdy <- twinReadiness(fit, imu, dt = 0.01)
evaluateIntervention(fit$twin, defaultInterventions()$strengthening, 1.5, rdy,
  outcome_sd = 0.1, n_mc = 50)
```

extendedKalmanFilter *Extended Kalman filter (Jacobian-linearised state estimator)*

Description

A first-order extended Kalman filter for $x_k = f(x_{k-1}, k) + w$, $y_k = h(x_k, k) + v$. The process and measurement Jacobians are obtained by central finite differences unless analytic versions are supplied. Same interface and return shape as [unscentedKalmanFilter\(\)](#); for a linear-Gaussian model it reproduces the exact Kalman filter.

Usage

```
extendedKalmanFilter(f, h, y, Q, R, x0, P0, Fjac = NULL, Hjac = NULL)
```

Arguments

f, h	Process and measurement models function(x, k).
y	Measurement matrix (n_steps x m).
Q, R	Process- and measurement-noise covariances.
x0, P0	Initial state mean and covariance.
Fjac, Hjac	Optional analytic Jacobians function(x, k) (rows = outputs); finite differences are used when NULL.

Value

an ekf_result: state (n_steps x L), cov (final), innovation.

References

Jazwinski AH (1970) Stochastic Processes and Filtering Theory.

See Also

[unscentedKalmanFilter\(\)](#), [particleFilter\(\)](#), [ensembleKalmanFilter\(\)](#)

Examples

```
set.seed(1); y <- matrix(cumsum(rnorm(40, 0, .1)) + rnorm(40, 0, .1), 40, 1)
fit <- extendedKalmanFilter(function(x, k) x, function(x, k) x, y,
  Q = matrix(0.05), R = matrix(0.01), x0 = 0, P0 = matrix(1))
fit$state[40]
```

fillMarkerGaps	<i>Fill occlusion gaps in a measured marker trajectory</i>
----------------	--

Description

Linear-interpolates the NA runs (occlusions) of a [markerMeasure\(\)](#) result, column by column, holding the endpoints flat where a gap reaches the boundary.

Usage

```
fillMarkerGaps(m)
```

Arguments

m a marker_measurement from [markerMeasure\(\)](#), or a numeric matrix with NA rows.

Value

the trajectory matrix with gaps interpolated.

See Also

[markerMeasure\(\)](#)

Examples

```
pos <- matrix(cumsum(rnorm(200)), 200, 1)
filled <- fillMarkerGaps(markerMeasure(pos, markerSensor(dropout_prob = 0.05), seed = 1))
```

generateTrainingData *Generate a labelled synthetic training set from the twin*

Description

Samples n virtual subjects: physical parameters are drawn from the given ranges, the IMU sensor is randomised (domain randomisation for robustness), each twin is simulated and measured, and the sensor features are returned with the generating parameters as labels. Use it to build ML training data where the labels are true mechanistic quantities.

Usage

```
generateTrainingData(
  n = 100,
  strength = c(3, 10),
  damping = c(0.1, 0.8),
  stiffness = c(1.5, 4),
  duration = 5,
  dt = 0.02,
  randomize_sensor = TRUE,
  features = TRUE,
  seed = NULL
)
```

Arguments

n	Number of virtual subjects.
strength, damping, stiffness	Length-2 ranges (uniform) for each parameter.
duration, dt	Simulation settings.
randomize_sensor	If TRUE, draw a different IMU noise/bias/scale per subject (domain randomisation).
features	If TRUE (default) return summary features; if FALSE return the raw IMU streams alongside the labels.
seed	Optional RNG seed.

Value

a twin_dataset: a data frame of features + label columns (strength, damping, stiffness), or (when features = FALSE) a list with labels and streams.

See Also

[limbTwin\(\)](#), [imuMeasure\(\)](#)

Examples

```
d <- generateTrainingData(20, duration = 4, dt = 0.02, seed = 1)
head(d)
```

gpCalibrate	<i>GP-accelerated calibration (Bayesian optimisation of an expensive loss)</i>
-------------	--

Description

Minimises an expensive loss over a bounded parameter box by Bayesian optimisation: it fits a [gpEmulator\(\)](#) surrogate to the evaluated (parameters, loss) points and repeatedly evaluates the loss at the point of maximum expected improvement. Far fewer loss evaluations than grid search or a local optimiser – the tool for calibrating a slow simulator to data.

Usage

```
gpCalibrate(
  loss,
  ranges,
  n_init = 12,
  n_iter = 20,
  xi = 0.01,
  n_cand = 500,
  seed = NULL
)
```

Arguments

loss	function(theta) returning the scalar loss to MINIMISE (e.g. the distance between simulated and observed summary statistics).
ranges	Named list of c(lower, upper) bounds, one per parameter.
n_init	Number of initial random design points.
n_iter	Number of Bayesian-optimisation iterations (loss evaluations after the initial design).
xi	Expected-improvement exploration margin.
n_cand	Candidate points scored by expected improvement each iteration.
seed	Optional RNG seed.

Value

a gp_calibration: the best par and value, the full design X/y, the best-so-far trace, and the final gp surrogate.

References

Jones DR, Schonlau M, Welch WJ (1998) J Glob Optim 13:455-492 (efficient global optimisation / expected improvement).

See Also

[gpEmulator\(\)](#), [abcSMC\(\)](#), [personalizeCardio\(\)](#)

Examples

```
# recover the minimum of a 2-D loss in a few evaluations
loss <- function(p) (p[1] - 0.3)^2 + (p[2] - 0.7)^2
cal <- gpCalibrate(loss, list(x = c(0, 1), y = c(0, 1)), n_init = 8, n_iter = 15, seed = 1)
round(cal$par, 2)
```

gpEmulator

Gaussian-process emulator (surrogate model)

Description

Fits a Gaussian-process regression surrogate with a squared-exponential kernel to training inputs X and outputs y – typically a design of simulator runs, so the emulator stands in for an expensive twin. Hyperparameters (length-scale, signal and noise SD) are fitted by maximising the marginal likelihood. [predict.gpEmulator\(\)](#) returns the posterior mean and standard deviation at new inputs; the SD is small near training points and grows away from them.

Usage

```
gpEmulator(
  X,
  y,
  optimize = TRUE,
  lengthscale = NULL,
  sigma_f = NULL,
  sigma_n = NULL
)
```

Arguments

<code>X</code>	Training inputs: a matrix ($n \times d$) or length- n vector.
<code>y</code>	Training outputs (length n).
<code>optimize</code>	If TRUE, fit hyperparameters by marginal likelihood; otherwise use the initial values.
<code>lengthscale, sigma_f, sigma_n</code>	Initial (or fixed) kernel length-scale, signal SD and noise SD; NULL picks data-driven defaults (median pairwise distance, $sd(y)$, a small nugget).

Value

a gpEmulator object (with a `predict.gpEmulator()` method): the fitted lengthscale, `sigma_f`, `sigma_n` and the training data.

References

Rasmussen CE, Williams CKI (2006) Gaussian Processes for Machine Learning.

See Also

`predict.gpEmulator()`, `sobolIndices()`, `abcCalibration()`

Examples

```
set.seed(1); x <- runif(20, 0, 6); y <- sin(x) + rnorm(20, 0, 0.05)
gp <- gpEmulator(x, y)
p <- predict(gp, seq(0, 6, length.out = 5)); round(p$mean, 2)
```

imuMeasure

Measure a movement twin with an IMU (forward sensor model)

Description

Turns a simulated twin trajectory into a realistic IMU data stream by applying the forward sensor model and the sensor's imperfections.

Usage

```
imuMeasure(trajecotory, sensor = imuSensor(), seed = NULL, noise = TRUE)
```

Arguments

trajecotory	A twin_trajecotory from <code>simulateTwin()</code> .
sensor	An imu_sensor from <code>imuSensor()</code> (default: mild noise).
seed	Optional RNG seed.
noise	If FALSE, return the noise-free forward IMU (the ground truth).

Value

a data frame time, gyro, ax, ay (angular velocity rad/s, body-frame accelerations m/s²).

See Also

`imuSensor()`, `personalizeTwin()`

Examples

```
sim <- simulateTwin(limbTwin(), duration = 2, dt = 0.01)
head(imuMeasure(sim, imuSensor(gyro_noise = 0.04), seed = 1))
```

`imuSensor`*Specify IMU sensor characteristics*

Description

Bundles the imperfections of an inertial measurement unit – bias, white noise, random-walk drift and scale error – so the same sensor can be reused, and randomised across a population (see [generateTrainingData\(\)](#)).

Usage

```
imuSensor(  
  gyro_bias = 0,  
  gyro_noise = 0.03,  
  gyro_drift = 0,  
  gyro_scale = 1,  
  accel_bias = 0,  
  accel_noise = 0.15,  
  accel_scale = 1  
)
```

Arguments

`gyro_bias`, `gyro_noise`, `gyro_drift`, `gyro_scale`
Gyroscope constant bias (rad/s), white-noise SD (rad/s), random-walk drift SD per step (rad/s), and multiplicative scale error (1 = perfect).

`accel_bias`, `accel_noise`, `accel_scale`
Accelerometer bias (m/s², length 1 or 2), white-noise SD (m/s²) and scale error.

Value

an `imu_sensor` object.

See Also

[imuMeasure\(\)](#)

Examples

```
imuSensor(gyro_bias = 0.02, gyro_noise = 0.05)
```

insilicoIntervention *Predict the effect of an in-silico intervention*

Description

Applies parameter changes to a (personalised) twin – multiplicatively via scale and/or by absolute values via set – re-simulates, and reports how the movement outcomes change. This is the digital twin used as an experiment: "if this subject's muscle were 30% stronger, how would the movement change?".

Usage

```
insilicoIntervention(
  twin,
  scale = list(),
  set = list(),
  duration = 6,
  dt = 0.005,
  theta0 = 0.1,
  omega0 = 0
)
```

Arguments

twin	A movement_twin (typically a personalised one).
scale	Named list of MULTIPLICATIVE changes, e.g. list(strength = 1.3, stiffness = 0.7).
set	Named list of ABSOLUTE parameter values to set.
duration, dt, theta0, omega0	Simulation settings (as simulateTwin()).

Value

an insilico_intervention: before, after (trajectories), outcomes (before/after/relative-change table) and the modified twin.

See Also

[personalizeTwin\(\)](#), [simulateTwin\(\)](#)

Examples

```
tw <- limbTwin(strength = 5, damping = 0.6, stiffness = 3)
iv <- insilicoIntervention(tw, scale = list(strength = 1.3, damping = 0.7))
iv$outcomes
```

insilicoInterventionMultimodal

Multi-modal in-silico intervention

Description

Applies a fitness-raising intervention (e.g. a rehabilitation / conditioning programme) to a multi-modal twin and predicts the coherent change across modalities: movement range of motion up, resting/mean heart rate down.

Usage

```
insilicoInterventionMultimodal(mt, fitness_gain = 0.2, ...)
```

Arguments

mt	a multimodal_twin from multimodalTwin() .
fitness_gain	Increase in latent fitness (0..1, clamped).
...	Passed to simulateMultimodal() .

Value

an mm_intervention: the before/after cross-modal summaries and their change.

See Also

[multimodalTwin\(\)](#), [populationIntervention\(\)](#)

Examples

```
insilicoInterventionMultimodal(multimodalTwin(fitness = 0.4), fitness_gain = 0.3, seed = 1)$change
```

intervention

Define a candidate in-silico intervention

Description

A candidate intervention for the clinical loop: a named set of parameter changes (multiplicative scale and/or absolute set, as [insilicoIntervention\(\)](#)) plus its clinical burden and risk, and optionally an evidence-based effect.

Usage

```
intervention(
  name,
  scale = list(),
  set = list(),
  burden = 0,
  risk = 0,
  evidence = NULL
)
```

Arguments

name	Label.
scale, set	Parameter changes (see insilicoIntervention()).
burden, risk	Clinical burden and risk, each on a 0..1 scale.
evidence	Optional list(effect, se): an external (e.g. literature) estimate of the outcome change and its standard error, for reconciliation.

Value

a twin_intervention.

See Also

[defaultInterventions\(\)](#), [scoreInterventions\(\)](#)

Examples

```
intervention("strengthening", scale = list(strength = 1.4), burden = 0.4, risk = 0.1)
```

jansenRit

Jansen-Rit neural-mass EEG generator

Description

Simulates a single cortical column with the Jansen-Rit neural-mass model: three interacting neural populations (pyramidal, excitatory and inhibitory interneurons) whose average post-synaptic potentials evolve as second-order ("alpha-function") systems coupled through a sigmoidal rate-to-potential nonlinearity, driven by a background firing-rate input $p(t)$. With the default parameters the column produces a self-organised ~10 Hz **alpha** rhythm; the scalp EEG is taken as the pyramidal post-synaptic potential $y_1 - y_2$.

Usage

```
jansenRit(
  duration = 8,
  sfeeg = 256,
  A = 3.25,
  B = 22,
  a = 100,
  b = 50,
  C = 135,
  p_mean = 220,
  p_sd = 30,
  oversample = 4,
  aperiodic = 0,
  aperiodic_exp = 1.4,
  seed = NULL
)
```

Arguments

duration	Recording length (s).
sfeeg	Output sampling rate (Hz).
A, B	Excitatory and inhibitory average synaptic gains (mV).
a, b	Excitatory and inhibitory rate constants (inverse time constants, 1/s).
C	Global connectivity constant; the four intra-column connectivities are $C1 = C$, $C2 = 0.8 C$, $C3 = C4 = 0.25 C$.
p_mean, p_sd	Mean and SD of the background firing-rate input $p(t)$ (pulses/s), drawn independently each integration step. The default $p_mean = 220$ sits in the middle of Jansen & Rit's classic 120-320 input range, where the column settles into its alpha limit cycle.
oversample	Internal integration steps per output sample (accuracy margin for the fast a-dynamics).
aperiodic	Amplitude of an added $1/f$ background, as a multiple of the oscillation's SD (0 = the pure neural-mass rhythm; a real scalp channel needs aperiodic around 1, which spreads power off the alpha peak so the relative band powers match a real EEG). See aperiodicNoise() .
aperiodic_exp	Spectral slope of the $1/f$ background (see aperiodicNoise()).
seed	Optional RNG seed.

Details

The excitation/inhibition balance and the drive select the regime: reducing the mean input p_mean below the oscillation threshold collapses the alpha limit cycle to a low-amplitude resting fixed point, while changing the gains A/B or the connectivity C shifts the rhythm.

Value

a jansenRit object: time, eeg (the y1 - y2 scalp potential, mV), dominant_hz (the dominant rhythm above 3 Hz, robust to the 1/f floor), band (its conventional EEG band), sfeeg and params.

References

Jansen BH, Rit VG (1995) Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns. Biol Cybern 73:357-366. Grimbert F, Faugeras O (2006) Neural Comput 18:3052-3068.

See Also

[ecgsyn\(\)](#), [kuramoto\(\)](#), [cpgMatsuoka\(\)](#)

Examples

```
eeg <- jansenRit(duration = 6, seed = 1)
eeg$dominant_hz      # ~10 Hz alpha rhythm
eeg$band             # "alpha"
```

`kuramoto`

Kuramoto coupled-oscillator network

Description

Simulates n phase oscillators with natural frequencies ω and all-to-all coupling K ($\theta_i' = \omega_i + K/n \sum_j \sin(\theta_j - \theta_i)$). The order parameter measures collective synchronisation: near 0 (incoherent) below the critical coupling, near 1 (synchronised) above it.

Usage

```
kuramoto(n = 20, K = 1, omega = NULL, duration = 20, dt = 0.01, seed = NULL)
```

Arguments

<code>n</code>	Number of oscillators.
<code>K</code>	Coupling strength.
<code>omega</code>	Natural frequencies (length n); default <code>rnorm(n)</code> .
<code>duration, dt</code>	Simulation length and step (s).
<code>seed</code>	Optional RNG seed.

Value

a kuramoto object: time, phase ($nt \times n$), order (order parameter per step), signals ($\cos(\text{phase})$), mean_order.

References

Kuramoto Y (1975); Strogatz SH (2000) Physica D 143:1-20.

See Also

[cpgMatsuoka\(\)](#), [ecgsyn\(\)](#), [jansenRit\(\)](#)

Examples

```
kuramoto(n = 15, K = 3, duration = 20, seed = 1)$mean_order # synchronised
kuramoto(n = 15, K = 0, duration = 20, seed = 1)$mean_order # incoherent
```

limbTwin	<i>Build a parameterised movement twin (single-joint limb)</i>
----------	--

Description

Constructs a mechanistic single-joint limb model whose parameters map to physical/clinical quantities (segment inertia, active muscle strength, passive damping, joint stiffness). It is the object simulated, personalised and intervened on by the rest of *PhysioTwin*.

Usage

```
limbTwin(
  inertia = 0.15,
  strength = 5,
  damping = 0.3,
  stiffness = 2.5,
  freq = 1,
  rest = 0,
  mass = 2.5,
  com = 0.2,
  g = 9.81,
  sensor_r = 0.25
)
```

Arguments

<code>inertia</code>	Segment moment of inertia about the joint (kg m^2).
<code>strength</code>	Active-drive amplitude (N m) – the muscle strength.
<code>damping</code>	Passive viscous damping (N m s / rad).
<code>stiffness</code>	Joint stiffness toward rest (N m / rad).
<code>freq</code>	Rhythm of the active drive (Hz).
<code>rest</code>	Neutral joint angle (rad).
<code>mass, com, g</code>	Segment mass (kg), centre-of-mass distance from the joint (m) and gravity (m/s^2); set $g = 0$ for a horizontal-plane movement.
<code>sensor_r</code>	Distance of the IMU from the joint along the segment (m), used by the sensor layer.

Value

a movement_twin object (a named parameter list with class).

See Also

[simulateTwin\(\)](#), [personalizeTwin\(\)](#), [insilicoIntervention\(\)](#)

Examples

```
tw <- limbTwin(strength = 6, damping = 0.4, stiffness = 3)
sim <- simulateTwin(tw, duration = 4, dt = 0.005)
range(sim$theta)
```

markerMeasure

Measure a marker trajectory with an optical mocap system

Description

Applies the forward marker sensor model to a clean trajectory: adds Gaussian noise, carves contiguous occlusion gaps (visible/occluded Markov process), optionally quantises and injects ghost samples. Occluded frames are returned as NA rows.

Usage

```
markerMeasure(position, sensor = markerSensor(), seed = NULL)
```

Arguments

position	Clean marker trajectory: an $n \times d$ matrix (or length- n vector) of marker positions over time.
sensor	A marker_sensor from markerSensor() .
seed	Optional RNG seed.

Value

a marker_measurement: position (measured, NA rows where occluded), visible (logical per frame), gap_fraction, and the sensor.

See Also

[markerSensor\(\)](#), [fillMarkerGaps\(\)](#)

Examples

```
pos <- cbind(sin(seq(0, 6, length.out = 300)), cos(seq(0, 6, length.out = 300)))
m <- markerMeasure(pos, markerSensor(dropout_prob = 0.02, dropout_len = 12), seed = 1)
m$gap_fraction
```

markerSensor	<i>Specify motion-capture marker characteristics</i>
--------------	--

Description

Bundles the imperfections of an optical marker: positional noise, occlusion dropout (contiguous gaps), quantisation and ghost samples.

Usage

```
markerSensor(
  noise = 0.002,
  dropout_prob = 0.02,
  dropout_len = 10,
  ghost_prob = 0,
  quantization = 0
)
```

Arguments

noise	Gaussian position-noise SD (same units as the trajectory, e.g. m).
dropout_prob	Per-frame probability of STARTING an occlusion while visible.
dropout_len	Mean occlusion length (frames); gaps are geometric.
ghost_prob	Per-frame probability of a spurious (ghost) sample.
quantization	Position resolution (round to this grid; 0 = off).

Value

a marker_sensor object.

See Also

[markerMeasure\(\)](#), [fillMarkerGaps\(\)](#), [imuSensor\(\)](#)

Examples

```
markerSensor(noise = 0.001, dropout_prob = 0.03, dropout_len = 8)
```

metropolis*Adaptive random-walk Metropolis-Hastings sampler*

Description

Samples from a log-posterior by random-walk Metropolis-Hastings. The Gaussian proposal covariance adapts to the empirical covariance of the chain during the burn-in (Haario et al. 2001), then is frozen, so correlated targets are sampled efficiently while the stationary distribution is preserved.

Usage

```
metropolis(  
  logpost,  
  init,  
  n_iter = 6000,  
  proposal_sd = 0.5,  
  burn_in = 0.4,  
  seed = NULL  
)
```

Arguments

logpost	function(theta) returning the log-posterior (up to a constant); may return $-\infty$ outside the support.
init	Initial parameter vector (its names label the output).
n_iter	Number of iterations.
proposal_sd	Initial per-parameter proposal SD.
burn_in	Fraction discarded (and over which the proposal adapts).
seed	Optional RNG seed.

Value

an `mcmc_result`: posterior samples, acceptance rate, mean, sd, and 2.5/50/97.5% quantiles.

References

Metropolis et al. (1953); Haario H, Saksman E, Tamminen J (2001) *Bernoulli* 7:223-242 (adaptive Metropolis).

See Also

[abcSMC\(\)](#), [abcCalibration\(\)](#), [gpCalibrate\(\)](#)

Examples

```
# posterior of a normal mean: 20 obs, N(0,10) prior
set.seed(1); y <- rnorm(20, 2, 1)
lp <- function(m) sum(dnorm(y, m, 1, log = TRUE)) + dnorm(m, 0, 10, log = TRUE)
fit <- metropolis(lp, init = c(mu = 0), n_iter = 4000, seed = 1)
fit$mean
```

multimodalTwin	<i>Build a multi-modal (movement + cardiorespiratory) twin</i>
----------------	--

Description

Couples a latent fitness and effort to both a [limbTwin\(\)](#) movement model (higher fitness -> more strength, less damping) and cardiorespiratory set-points (higher fitness -> lower resting heart rate; higher effort -> higher heart rate and breathing rate).

Usage

```
multimodalTwin(
  fitness = 0.5,
  effort = 0.5,
  base_strength = 5,
  base_damping = 0.5
)
```

Arguments

fitness	Latent fitness (0..1).
effort	Latent effort/activity level (0..1).
base_strength, base_damping	Reference movement parameters at fitness = 0.5.

Value

a multimodal_twin: the derived movement twin, resting_hr, resp_rate and the latent fitness/effort.

See Also

[simulateMultimodal\(\)](#), [insilicoInterventionMultimodal\(\)](#), [rsaTachogram\(\)](#)

Examples

```
mt <- multimodalTwin(fitness = 0.7)
mt$resting_hr
```

Description

Chooses the experimental condition that will best identify a movement twin's parameters. For each candidate value of a design variable (e.g. the driving frequency), it forms the Fisher information matrix of the noise-weighted IMU observable about the parameters and scores the design by an optimality criterion – D-optimality (the log-determinant of the information, maximising overall information) or A-optimality (minimising the total Cramer-Rao variance). The most informative condition is the experiment worth running to pin the parameters down.

Usage

```
optimalDesign(
  twin,
  estimate = "strength",
  designs,
  design_var = "freq",
  sensor = imuSensor(),
  duration = 6,
  dt = 0.01,
  theta0 = 0.1,
  criterion = c("D", "A")
)
```

Arguments

<code>twin</code>	A movement_twin giving the assumed (prior) parameter values.
<code>estimate</code>	Parameter(s) whose identifiability is optimised.
<code>designs</code>	Numeric vector of candidate values for the design variable.
<code>design_var</code>	The twin field the design varies (e.g. "freq", "duration").
<code>sensor</code>	An imuSensor (its gyro_noise/accel_noise weight the channels).
<code>duration, dt, theta0</code>	Simulation settings for the observable.
<code>criterion</code>	"D" (max log-det Fisher information) or "A" (min total CRLB variance).

Value

an optimal_design: a table (design, logdet_fim, score, and a crlb_<param> Cramer-Rao SD per estimated parameter), the best design, and the estimate/design_var/criterion.

References

Chaloner K, Verdinelli I (1995) Bayesian experimental design: a review. *Statist Sci* 10:273-304;
Fedorov (1972) D-optimality.

See Also

[personalizeTwin\(\)](#), [validateTwin\(\)](#), [profileLikelihood\(\)](#)

Examples

```
od <- optimalDesign(limbTwin(), estimate = "strength",
                   designs = seq(0.4, 2.4, by = 0.2), design_var = "freq",
                   duration = 5, dt = 0.02)
od$best      # the most informative driving frequency
```

particleFilter

Bootstrap particle filter (sequential Monte-Carlo state estimator)

Description

A sequential-importance-resampling (bootstrap) particle filter for $x_k = f(x_{k-1}, k) + w$, $y_k = h(x_k, k) + v$. Particles are propagated through f with process noise Q , weighted by the Gaussian measurement likelihood, and resampled (systematic) when the effective sample size falls below `resample_thresh` of the population. Handles strong nonlinearity and multi-modal posteriors that a Kalman filter cannot. Same interface as [unscentedKalmanFilter\(\)](#).

Usage

```
particleFilter(
  f,
  h,
  y,
  Q,
  R,
  x0,
  P0,
  n_particles = 1000,
  resample_thresh = 0.5,
  seed = NULL
)
```

Arguments

<code>f, h</code>	Process and measurement models function(x , k).
<code>y</code>	Measurement matrix ($n_steps \times m$).
<code>Q, R</code>	Process- and measurement-noise covariances.
<code>x0, P0</code>	Initial state mean and covariance.
<code>n_particles</code>	Number of particles.
<code>resample_thresh</code>	Resample when effective sample size $< thresh * N$.
<code>seed</code>	Optional RNG seed.

Value

a pf_result: state (n_steps x L posterior means), cov (final weighted covariance), innovation, and ess (effective sample size per step).

References

Gordon NJ, Salmond DJ, Smith AFM (1993) IEE Proc F 140:107-113.

See Also

[unscentedKalmanFilter\(\)](#), [extendedKalmanFilter\(\)](#), [ensembleKalmanFilter\(\)](#)

Examples

```
set.seed(1); y <- matrix(sin(seq(0, 6, length.out = 50)) + rnorm(50, 0, .1), 50, 1)
fit <- particleFilter(function(x, k) x, function(x, k) x, y,
  Q = matrix(0.05), R = matrix(0.01), x0 = 0, P0 = matrix(1), n_particles = 500)
fit$state[50]
```

particleGibbs

Particle Gibbs with ancestor sampling

Description

Samples the joint posterior of a state-space model's latent trajectory and parameters by particle Gibbs: each iteration draws a new state trajectory with a conditional SMC (ancestor-sampling) sweep given the current parameters, then moves the parameters with a Metropolis step using the tractable complete-data likelihood of that trajectory. Ancestor sampling gives good mixing even for long series where plain particle Gibbs degenerates.

Usage

```
particleGibbs(
  build_model,
  y,
  prior_logd,
  init,
  n_particles = 100,
  n_iter = 2000,
  proposal_sd = 0.1,
  burn_in = 0.3,
  seed = NULL
)
```

Arguments

<code>build_model</code>	function(theta) returning the state-space model: a list f, h (function(x, k)), Q, R, x0, P0.
<code>y</code>	Observation matrix (n_steps x m) or vector.
<code>prior_logd</code>	function(theta) log prior density (-Inf off the support).
<code>init</code>	Initial parameter vector (its names label the output).
<code>n_particles</code>	Particles in the conditional SMC.
<code>n_iter</code>	Iterations.
<code>proposal_sd</code>	Initial per-parameter proposal SD.
<code>burn_in</code>	Fraction discarded (and over which the proposal adapts).
<code>seed</code>	Optional RNG seed.

Value

a pgibbs_result: parameter posterior samples, acceptance, mean, sd, 2.5/50/97.5% quantiles, and the posterior-mean state trajectory.

References

Andrieu C, Doucet A, Holenstein R (2010) J R Stat Soc B 72:269-342; Lindsten F, Jordan MI, Schon TB (2014) J Mach Learn Res 15:2145-2184.

See Also

[particleMCMC\(\)](#), [metropolis\(\)](#), [particleFilter\(\)](#)

Examples

```
set.seed(1); a <- 0.8; x <- numeric(50); for (k in 2:50) x[k] <- a * x[k-1] + rnorm(1, 0, 0.3)
y <- matrix(x + rnorm(50, 0, 0.3), 50, 1)
bm <- function(th) list(f = function(x, k) th[1] * x, h = function(x, k) x,
                        Q = matrix(0.09), R = matrix(0.09), x0 = 0, P0 = matrix(1))
fit <- particleGibbs(bm, y, function(th) dunif(th[1], -1, 1, log = TRUE),
                    init = c(a = 0.4), n_particles = 60, n_iter = 600, seed = 1)
fit$mean
```

particleMCMC

Particle marginal Metropolis-Hastings (particle MCMC)

Description

Samples the parameter posterior of a state-space model from a WHOLE observation series, when the likelihood is intractable. Each proposed parameter is scored by a bootstrap particle filter's unbiased estimate of the marginal likelihood $p(y_{1:T} \mid \theta)$, which enters the Metropolis-Hastings acceptance ratio; the resulting chain targets the exact parameter posterior (Andrieu-Doucet-Holenstein). The proposal covariance adapts during burn-in.

Usage

```
particleMCMC(
  build_model,
  y,
  prior_logd,
  init,
  n_particles = 200,
  n_iter = 3000,
  proposal_sd = 0.1,
  burn_in = 0.3,
  seed = NULL
)
```

Arguments

<code>build_model</code>	function(theta) returning the state-space model for parameters theta: a list f, h (function(x, k)), Q, R, x0, P0.
<code>y</code>	Observation matrix (n_steps x m) or vector.
<code>prior_logd</code>	function(theta) returning the log prior density (-Inf off the support).
<code>init</code>	Initial parameter vector (its names label the output).
<code>n_particles</code>	Particles in the likelihood-estimating filter.
<code>n_iter</code>	MCMC iterations.
<code>proposal_sd</code>	Initial per-parameter proposal SD.
<code>burn_in</code>	Fraction discarded (and over which the proposal adapts).
<code>seed</code>	Optional RNG seed.

Value

a pmcmc_result: posterior samples, acceptance, mean, sd and 2.5/50/97.5% quantiles.

References

Andrieu C, Doucet A, Holenstein R (2010) J R Stat Soc B 72:269-342.

See Also

[particleFilter\(\)](#), [metropolis\(\)](#), [personalizeCardio\(\)](#)

Examples

```
# recover the AR coefficient of a linear-Gaussian state-space model from the series
set.seed(1); a <- 0.7; x <- numeric(60); for (k in 2:60) x[k] <- a * x[k-1] + rnorm(1, 0, 0.3)
y <- matrix(x + rnorm(60, 0, 0.3), 60, 1)
bm <- function(th) list(f = function(x, k) th[1] * x, h = function(x, k) x,
  Q = matrix(0.09), R = matrix(0.09), x0 = 0, P0 = matrix(1))
fit <- particleMCMC(bm, y, function(th) dunif(th[1], -1, 1, log = TRUE),
  init = c(a = 0.4), n_particles = 100, n_iter = 800, seed = 1)
fit$mean
```

personalizeCardio	<i>Personalise the closed-loop CV-respiratory twin to real HRV/BPV summaries</i>
-------------------	--

Description

Fits the closed-loop `cardioRespiratory()` model's baroreflex gains and respiratory modulation to a subject's measured variability summaries (baroreflex sensitivity, blood-pressure Mayer power, heart-rate respiratory power) by GP-accelerated calibration (`gpCalibrate()`). Because the summaries have very different scales, the discrepancy is relative. The returned twin, driven with the fitted parameters, reproduces the subject's HRV/BPV.

Usage

```
personalizeCardio(
  target,
  ranges = list(g_vagal = c(0.004, 0.02), g_resist = c(0.003, 0.02), rsa_amp = c(0.01,
    0.06)),
  fixed = list(),
  duration = 300,
  n_init = 12,
  n_iter = 25,
  n_rep = 3,
  seed = 1
)
```

Arguments

target	Named numeric vector of the observed summaries to match; any of brs, bp_lf, hr_hf (see <code>cardioSummary()</code>).
ranges	Named list of <code>c(lower, upper)</code> bounds for the parameters to fit (default <code>g_vagal</code> , <code>g_resist</code> , <code>rsa_amp</code>).
fixed	Named list of other <code>cardioRespiratory()</code> parameters held fixed.
duration	Simulation length per evaluation (s).
n_init, n_iter	Bayesian-optimisation initial design and iterations.
n_rep	Simulations averaged per evaluation to stabilise the (stochastic) variability summaries – the blood-pressure Mayer power is noisy, so averaging makes the inverse problem well-posed.
seed	RNG seed (fixes the stochastic loss so the calibration is deterministic).

Value

a `cardio_personalization`: the fitted estimates, the personalised twin, the target and achieved summaries, the relative error `rel_error`, and the calibration object.

See Also

[cardioRespiratory\(\)](#), [cardioSummary\(\)](#), [gpCalibrate\(\)](#), [personalizeTwin\(\)](#)

Examples

```
subject <- cardioSummary(cardioRespiratory(duration = 240, g_vagal = 0.013,
                                           g_resist = 0.008, rsa_amp = 0.045, seed = 3))
fit <- personalizeCardio(subject, n_init = 8, n_iter = 15, seed = 1)
fit$achieved           # reproduces the subject's brs / bp_lf / hr_hf
```

personalizeCardioWaveform

Personalise the closed-loop cardiovascular twin from the whole RR record

Description

Fits the [cardioStateSpace\(\)](#) parameters jointly to a whole RR-interval record. Because the closed-loop model is linear-Gaussian its likelihood is exact (a Kalman filter), so the posterior is sampled by [metropolis\(\)](#) – the whole-waveform counterpart to the summary-based [personalizeCardio\(\)](#). The Mayer gain/damping and the respiratory amplitude are identifiable here because the record separates the low- and high-frequency rhythms, resolving the ridge that confounds the summary fit. (For an intractable, nonlinear model use [particleMCMC\(\)](#) / [particleGibbs\(\)](#) with [cardioStateSpace\(\)](#)'s `f/h`.)

Usage

```
personalizeCardioWaveform(
  rr,
  resp_freq = 0.25,
  estimate = NULL,
  lf_model = c("oscillator", "ar1"),
  fixed = list(omega_m = 0.53, rr0 = NULL, q_mayer = 1, r_obs = 9),
  n_iter = 4000,
  seed = NULL
)
```

Arguments

<code>rr</code>	RR-interval series (ms).
<code>resp_freq</code>	Respiratory frequency (Hz) for the RSA regressor.
<code>estimate</code>	Parameters to fit (any of "g", "rho", "A_rsa", "phi"); NULL picks the default for the <code>lf_model</code> (g/rho/A_rsa for the oscillator, g/phi/A_rsa for "ar1").
<code>lf_model</code>	Low-frequency model (see cardioStateSpace()): "oscillator" (a resonant Mayer peak) or "ar1" (broadband low-frequency power, for records without a distinct Mayer rhythm, e.g. a regular healthy sinus rhythm).

fixed	Named list of held-fixed <code>cardioStateSpace()</code> parameters (<code>omega_m</code> , <code>rr0</code> , <code>q_mayer</code> , <code>r_obs</code> , and any low-frequency parameter not fitted).
n_iter	Metropolis iterations.
seed	Optional RNG seed.

Value

a `cardio_waveform_fit`: the posterior (an `mcmc_result`), the fitted estimates, and the estimated names.

See Also

`cardioStateSpace()`, `metropolis()`, `particleMCMC()`, `personalizeCardio()`

Examples

```
truth <- cardioStateSpace(g = 8, rho = 0.9, A_rsa = 20,
  resp = sin(2 * pi * 0.25 * cumsum(rep(0.85, 250))))
set.seed(1); x <- c(0, 0); rr <- numeric(250)
for (k in seq_len(250)) {
  x <- truth$f(x, k) + c(rnorm(1), 0); rr[k] <- truth$h(x, k) + rnorm(1, 0, 3)
}
fit <- personalizeCardioWaveform(rr, resp_freq = 0.25, estimate = c("g", "A_rsa"),
  n_iter = 3000, seed = 1)
fit$estimates
```

personalizeTwin

Personalise a movement twin to recorded IMU data

Description

Estimates a subject's twin parameters by assimilating their IMU stream with an unscented Kalman filter over an augmented state [`theta`, `omega`, `log(params)`]. Parameters are estimated in log-space (keeping them positive) and the fitted values are read from the converged filter.

Usage

```
personalizeTwin(
  imu,
  prior,
  estimate = "strength",
  method = c("optim", "ukf"),
  dt = 0.005,
  param_rw = 0.02,
  gyro_var = 0.03^2,
  accel_var = 0.15^2,
  burn_in = 0.5,
  n_starts = 25L
)
```

Arguments

<code>imu</code>	A data frame with <code>gyro</code> , <code>ax</code> , <code>ay</code> (as from <code>imuMeasure()</code>), sampled at <code>dt</code> .
<code>prior</code>	A <code>movement_twin</code> giving the fixed parameters and the initial guess for the estimated ones.
<code>estimate</code>	Character vector of parameters to personalise (any of "strength", "damping", "stiffness", "inertia").
<code>method</code>	"optim" (default) fits the whole record by minimising the simulated-vs-observed IMU error (a batch/variational fit, robust to strong non-linearity); "ukf" runs the recursive unscented Kalman filter (online, best in the mildly non-linear regime).
<code>dt</code>	Sample period (s).
<code>param_rw</code>	Random-walk SD (log-space) for the estimated parameters (their assumed drift; default 0.02).
<code>gyro_var, accel_var</code>	Measurement-noise variances for the gyro and accel channels.
<code>burn_in</code>	Fraction of the record discarded before averaging the converged estimate (default 0.5).
<code>n_starts</code>	Number of coarse-grid starting points scanned before the "optim" fit (multi-start). The objective is evaluated on a multiplicative grid around the prior (one cheap simulation each) and the fit is started from the best point, making multi-parameter fits robust to spurious co-estimation minima at negligible cost; 1 reproduces a single Nelder-Mead start from the prior.

Value

a `personalized_twin`: `twin` (the fitted twin), `estimates` (named fitted parameters), the co-estimated `gyro_bias`, and – for `method = "ukf"` – the per-step trajectory and the ukf result, or – for `method = "optim"` – the fitted `theta0` and final objective.

See Also

`unscentedKalmanFilter()`, `extendedKalmanFilter()`, `particleFilter()`, `abcCalibration()`, `insilicoIntervention()`

Examples

```
truth <- limbTwin(strength = 7, damping = 0.5)
sim <- simulateTwin(truth, duration = 6, dt = 0.01)
imu <- imuMeasure(sim, imuSensor(gyro_noise = 0.03, accel_noise = 0.1), seed = 1)
fit <- personalizeTwin(imu, limbTwin(strength = 4, damping = 0.2),
                      estimate = c("strength", "damping"), dt = 0.01)
fit$estimates
```

populationIntervention

In-silico intervention trial over a population of twins

Description

Samples a cohort of movement twins from a parameter distribution, applies the same intervention to each (via [insilicoIntervention\(\)](#)), and summarises the response as a paired in-silico trial: the mean effect and its paired effect size (Cohen's d_z), the responder rate against a minimal clinically important difference, the statistical power at this sample size and the sample size needed for 80% power.

Usage

```
populationIntervention(
  n = 200,
  params = list(strength = c(5, 1), damping = c(0.7, 0.2), stiffness = c(2.5, 0.6)),
  intervention = list(scale = list(strength = 1.4)),
  outcome = "rom",
  mcid = NULL,
  duration = 6,
  dt = 0.005,
  theta0 = 0.1,
  omega0 = 0,
  seed = NULL
)
```

Arguments

n	Number of subjects (twins) in the cohort.
params	Named list of $c(\text{mean}, \text{sd})$ population distributions for limbTwin() parameters (sampled Gaussian, truncated positive).
intervention	List with scale and/or set (as insilicoIntervention()).
outcome	Outcome to track: one of "rom", "peak_velocity", "mean_speed", "peak_accel", or a function(trajecory) returning a scalar.
mcid	Minimal clinically important difference (responder threshold); defaults to 0.1 x the baseline outcome SD.
duration, dt, theta0, omega0	Simulation settings (as simulateTwin()).
seed	Optional RNG seed.

Value

a population_intervention: subjects (per-subject params, pre, post, delta), effect (mean delta, SD, Cohen's d_z), responder_rate, mcid, power and n_for_80.

See Also

[insilicoIntervention\(\)](#), [multimodalTwin\(\)](#), [generateTrainingData\(\)](#)

Examples

```
pop <- populationIntervention(n = 100, intervention = list(scale = list(strength = 1.4)),
                             seed = 1)
pop$effect
```

predict.gpEmulator	<i>Predict from a Gaussian-process emulator</i>
--------------------	---

Description

Predict from a Gaussian-process emulator

Usage

```
## S3 method for class 'gpEmulator'
predict(object, newdata, ...)
```

Arguments

object	a gpEmulator from gpEmulator() .
newdata	New inputs: a matrix (m x d) or a vector (for 1-D emulators).
...	Unused.

Value

a list with mean (posterior mean) and sd (posterior standard deviation) at each new input.

See Also

[gpEmulator\(\)](#)

profileLikelihood	<i>Profile-likelihood practical identifiability</i>
-------------------	---

Description

Assesses whether a parameter is *practically* identifiable from a recording: it is fixed across a grid of values and the remaining nuisance parameters (plus a start angle and gyro bias) are re-optimised, tracing the fit cost. A clear minimum bounded by the chi-square threshold gives a confidence interval; a flat profile means the parameter cannot be pinned down.

Usage

```
profileLikelihood(
  imu,
  prior,
  param,
  over = c(0.4, 2),
  nuisance = character(0),
  n_grid = 15L,
  dt = 0.02,
  gyro_var = 0.03^2,
  accel_var = 0.12^2
)
```

Arguments

imu	An IMU recording (gyro, ax, ay), real or simulated.
prior	A movement_twin giving the fixed parameters and start point.
param	The parameter to profile.
over	Length-2 multiplicative range around prior[[param]] (default c(0.4, 2.0)), or an explicit numeric grid.
nuisance	Other parameters re-optimised at each grid point.
n_grid	Grid size (default 15).
dt, gyro_var, accel_var	Sample period and measurement variances.

Value

a profile_likelihood: the profile (grid value, cost), the cost minimum, the 95% ci (or NA if open), and identifiable (logical).

References

Raue A, et al. (2009) Bioinformatics 25:1923-1929.

See Also

[validateTwin\(\)](#), [sobolIndices\(\)](#)

Examples

```
tw <- limbTwin(strength = 6, damping = 0.5, stiffness = 6)
imu <- imuMeasure(simulateTwin(tw, 8, 0.02, theta0 = 0.2),
                  imuSensor(gyro_noise = 0.03, accel_noise = 0.12), seed = 1)
profileLikelihood(imu, tw, "strength", nuisance = "damping", n_grid = 9)$ci
```

quantizationNoise

Quantisation noise of an analogue-to-digital conversion

Description

The theoretical quantisation noise (least-significant-bit squared over twelve) and the empirical error of quantising signal to bits over range.

Usage

```
quantizationNoise(signal, bits, range = NULL)
```

Arguments

signal	Numeric signal.
bits	Bit depth.
range	c(lo, hi) full-scale range; defaults to the signal's range.

Value

a named numeric vector: lsb, theoretical_sd(= lsb/sqrt(12)), empirical_sd, theoretical_power(= lsb^2/12).

See Also

[adcQuantize\(\)](#)

Examples

```
quantizationNoise(sin(seq(0, 20, length.out = 4000))), bits = 12)
```

reconcileEvidence	<i>Reconcile a mechanistic prediction with external evidence</i>
-------------------	--

Description

Combines the twin's mechanistic effect estimate with an external (e.g. literature) estimate by inverse-variance (precision) weighting – the Bayesian conjugate-normal update – and flags whether the two AGREE. Evidence complements the mechanistic prediction; it is never overridden.

Usage

```
reconcileEvidence(mech_est, mech_se, ev_est, ev_se)
```

Arguments

mech_est, mech_se	Mechanistic effect estimate and its SE.
ev_est, ev_se	Evidence-based effect estimate and its SE.

Value

an evidence_reconciliation: combined estimate/se, the discrepancy z, and agreement (agree/disagree, at $|z| < 2$).

See Also

[scoreInterventions\(\)](#), [abcCalibration\(\)](#)

Examples

```
reconcileEvidence(mech_est = 0.30, mech_se = 0.05, ev_est = 0.26, ev_se = 0.04)
```

rehabEvaluate	<i>Evaluate a twin's intervention prediction against a real single-case series</i>
---------------	--

Description

Connects the digital-twin decision loop to the PhysioRehab clinical-reasoning framework. A real single-case series – outcome measurements in a baseline phase then an intervention phase – is assembled into a rehab_episode and analysed with PhysioRehab's single-case experimental design engine: the NAP and Tau effect sizes, whether the change exceeds the mcid, and the ICF category of the measure. If the twin's predicted_delta is supplied, it is reconciled with the clinically-observed change (do they agree in sign, and on the MCID verdict?), closing the predict -> deliver -> evaluate loop against real evidence.

Usage

```
rehabEvaluate(
  baseline,
  intervention,
  predicted_delta = NULL,
  measure = "gait_speed",
  mcid = 0.16,
  direction = "increase",
  condition = "in-silico twin",
  patient_id = "twin",
  start_date = "2026-01-01"
)
```

Arguments

baseline	Numeric outcome measurements in the baseline phase.
intervention	Numeric outcome measurements in the intervention phase.
predicted_delta	Optional: the twin's predicted change in the outcome (e.g. from scoreInterventions() / evaluateIntervention()), for reconciliation.
measure	Outcome name (a <i>PhysioRehab</i> measure, e.g. "gait_speed", "cadence"; drives the ICF mapping).
mcid	Minimal clinically important difference (threshold, outcome units).
direction	"increase" or "decrease" – the improving direction.
condition, patient_id	Episode metadata.
start_date	First session date (Date or a string coercible by as.Date()).

Value

a *rehab_evaluation*: the built episode, the *PhysioRehab* sced result, the icf code(s), and – when *predicted_delta* is given – a reconciliation (predicted, observed, sign_agree, mcid_agree, error).

See Also

[evaluateIntervention\(\)](#), [clinicalDecision\(\)](#)

Examples

```
if (requireNamespace("PhysioRehab", quietly = TRUE)) {
  set.seed(1)
  base <- rnorm(5, 0.80, 0.03); post <- rnorm(6, 1.00, 0.03)
  rehabEvaluate(base, post, predicted_delta = 0.18, measure = "gait_speed", mcid = 0.16)
}
```

respiration	<i>Respiratory waveform generator</i>
-------------	---------------------------------------

Description

Simulates a breathing (chest-volume) waveform: within each breath the volume rises smoothly on inspiration and falls on expiration, with the inspiration fraction set by the inspiration:expiration (ie_ratio) ratio (physiological breathing is expiration-dominant, ~1:2).

Usage

```
respiration(duration = 60, rate = 15, ie_ratio = 0.5, amplitude = 1, sfp = 100)
```

Arguments

duration	Recording length (s).
rate	Breathing rate (breaths/min).
ie_ratio	Inspiration-to-expiration duration ratio (0.5 = 1:2).
amplitude	Peak-to-trough volume amplitude (arbitrary units).
sfp	Sampling rate (Hz).

Value

a respiration object: time, volume (chest expansion, zero-mean), flow (its time derivative), and rate.

See Also

[rsaTachogram\(\)](#), [windkessel\(\)](#)

Examples

```
rp <- respiration(duration = 30, rate = 15)
rp$rate
```

rsaTachogram	<i>Respiratory sinus arrhythmia tachogram (heart-rate variability generator)</i>
--------------	--

Description

Generates a heart-rate-variability tachogram by integral pulse frequency modulation: the instantaneous heart rate is modulated by respiration (faster on inspiration – respiratory sinus arrhythmia) and by a ~0.1 Hz Mayer wave, and a beat is emitted whenever the integral of the instantaneous rate crosses an integer. The resulting RR-interval series reproduces the high-frequency (respiratory) and low-frequency (Mayer) peaks of the HRV spectrum.

Usage

```
rsaTachogram(
  duration = 300,
  hr0 = 70,
  resp_rate = 15,
  rsa = 0.12,
  mayer = 0.06,
  mayer_freq = 0.1,
  ie_ratio = 0.5,
  noise = 0.01,
  seed = NULL
)
```

Arguments

duration	Recording length (s).
hr0	Mean heart rate (bpm).
resp_rate	Breathing rate (breaths/min); sets the HF peak.
rsa	Depth of the respiratory heart-rate modulation (fraction of hr0).
mayer	Depth of the Mayer-wave modulation (fraction of hr0).
mayer_freq	Mayer-wave frequency (Hz).
ie_ratio	Inspiration:expiration ratio of the driving respiration.
noise	SD of a white heart-rate jitter (fraction of hr0).
seed	Optional RNG seed.

Value

an rsaTachogram object: beat_time, rr (RR intervals, s), hr (instantaneous, bpm), the driving resp waveform; and the HRV summaries hf_peak (Hz of the HF spectral peak), lf_power, hf_power and lf_hf (the LF/HF ratio).

References

Berger RD et al. (1989) IEEE Trans Biomed Eng 36:1049-1055 (IPFM). Task Force (1996) Circulation 93:1043-1065 (HRV bands).

See Also

[respiration\(\)](#), [ecgsyn\(\)](#)

Examples

```
hrv <- rsaTachogram(duration = 300, resp_rate = 15, seed = 1)
hrv$hf_peak          # ~ 0.25 Hz, the respiratory rate
```

scoreInterventions	<i>Score and rank candidate interventions with predictive uncertainty</i>
--------------------	---

Description

Runs each candidate intervention over the twin's PARAMETER POSTERIOR (sampled from the [twinReadiness\(\)](#) Fisher covariance), so every candidate gets a predicted effect WITH a predictive interval and the probability that the effect exceeds a minimal clinically important difference (MCID) – not a point estimate. Candidates are ranked by a multi-criteria score combining efficacy, certainty (probability of benefit), risk and burden, scaled by the twin's readiness confidence; optional per-candidate evidence is reconciled in, and low-confidence or evidence-disagreeing recommendations are flagged.

Usage

```
scoreInterventions(
  twin,
  candidates,
  readiness = NULL,
  outcome = "rom",
  mcid = NULL,
  n_mc = 200,
  weights = c(efficacy = 1, certainty = 1, risk = 1, burden = 0.5),
  duration = 6,
  dt = 0.005,
  theta0 = NULL,
  seed = NULL
)
```

Arguments

twin	A personalised movement_twin (e.g. fit\$twin).
candidates	A list of twin_intervention (see defaultInterventions()).

readiness	A twin_readiness; supplies the parameter posterior and the confidence. If NULL, a point estimate is used and the result is marked unvalidated.
outcome	Outcome name ("rom", "peak_velocity", ...).
mcid	Minimal clinically important difference; defaults to 10% of the baseline outcome.
n_mc	Posterior samples for uncertainty propagation.
weights	Named weights c(efficacy, certainty, risk, burden).
duration, dt, theta0	Simulation settings; theta0 defaults to the fit's.
seed	Optional RNG seed.

Value

an intervention_scores: a ranked ranking data frame (effect, ci_lo/ci_hi, p_benefit, risk, burden, reconciled evidence, score, flag), the mcid, and the readiness_confidence/verdict.

See Also

[twinReadiness\(\)](#), [reconcileEvidence\(\)](#), [clinicalDecision\(\)](#)

Examples

```
truth <- limbTwin(strength = 6, damping = 0.6, stiffness = 3)
imu <- imuMeasure(simulateTwin(truth, 6, 0.01), imuSensor(gyro_noise = 0.03), seed = 1)
fit <- personalizeTwin(imu, limbTwin(strength = 4), estimate = "strength", dt = 0.01)
rdy <- twinReadiness(fit, imu, dt = 0.01)
scoreInterventions(fit$twins, defaultInterventions(), rdy, n_mc = 50, seed = 1)$ranking
```

simulateMultimodal	<i>Simulate a multi-modal twin (movement + cardiorespiratory summary)</i>
--------------------	---

Description

Runs the movement twin and the cardiorespiratory generators and returns a small cross-modal summary: movement range of motion and peak velocity, and the mean heart rate.

Usage

```
simulateMultimodal(
  mt,
  duration_move = 6,
  duration_hr = 120,
  dt = 0.005,
  theta0 = 0.1,
  seed = NULL
)
```

Arguments

mt	a multimodal_twin from multimodalTwin() .
duration_move	Movement simulation length (s).
duration_hr	Heart-rate simulation length (s).
dt, theta0	Movement simulation settings.
seed	Optional RNG seed.

Value

a named numeric vector: rom, peak_velocity, mean_hr, resp_rate.

See Also

[multimodalTwin\(\)](#), [insilicoInterventionMultimodal\(\)](#)

Examples

```
simulateMultimodal(multimodalTwin(fitness = 0.6), seed = 1)
```

simulateTwin	<i>Simulate a movement twin (forward dynamics)</i>
--------------	--

Description

Integrates the twin's equation of motion with a fixed-step 4th-order Runge-Kutta scheme, returning the true joint state trajectory (angle, angular velocity and acceleration).

Usage

```
simulateTwin(twin, duration = 5, dt = 0.005, theta0 = 0, omega0 = 0)
```

Arguments

twin	A movement_twin from limbTwin() .
duration	Simulation time (s).
dt	Time step (s).
theta0, omega0	Initial angle and angular velocity.

Value

a twin_trajectory list: time, theta, omega, alpha (angular acceleration), and twin.

See Also

[limbTwin\(\)](#), [imuMeasure\(\)](#)

Examples

```
simulateTwin(limbTwin(), duration = 2, dt = 0.005)$theta[1:5]
```

sobolIndices

*Variance-based global sensitivity (Sobol indices)***Description**

Apportions the variance of a model output among the parameters using the Saltelli/Jansen estimator: first-order indices S_i (each parameter's own contribution) and total-effect indices ST_i (including interactions). A near- zero total index means the parameter is non-influential (and typically unidentifiable).

Usage

```
sobolIndices(
  ranges,
  output = NULL,
  base = limbTwin(),
  n = 256L,
  duration = 6,
  dt = 0.02,
  theta0 = 0.2,
  seed = NULL
)
```

Arguments

ranges	A named list of length-2 parameter ranges (uniform sampling).
output	A function function(twin) returning a scalar output metric; default the movement range of motion.
base	A movement_twin for the parameters NOT in ranges.
n	Base sample size (total model evaluations = $n * (k + 2)$; default 256).
duration, dt, theta0	Simulation settings for the default output.
seed	Optional RNG seed.

Value

a sobol_indices data frame: param, first-order S_i , total-effect ST_i .

References

Saltelli A, et al. (2010) Comput Phys Commun 181:259-270; Jansen MJW (1999).

See Also

[validateTwin\(\)](#), [profileLikelihood\(\)](#)

Examples

```
sobolIndices(list(strength = c(3, 9), damping = c(0.2, 1.0), stiffness = c(2, 6)),
             n = 64, seed = 1)
```

twinReadiness	<i>Certify a personalised twin before it is allowed to advise (validation gate)</i>
---------------	---

Description

The trust gate of the clinical loop. A personalised twin may only inform decisions once it is shown to (1) FIT the patient's data and (2) have IDENTIFIABLE parameters. This recomputes the personalisation residual (fit quality) and a Fisher-information parameter covariance (identifiability), and returns a verdict (ready / provisional / not_ready) with a $[0, 1]$ confidence that downstream scoring uses to scale and flag its recommendations.

Usage

```
twinReadiness(
  fit,
  imu,
  dt = 0.005,
  gyro_var = 0.03^2,
  accel_var = 0.15^2,
  cv_warn = 0.25,
  chi2_ready = 3,
  chi2_bad = 10
)
```

Arguments

fit	A personalized_twin from personalizeTwin() .
imu	The IMU data the twin was fitted to.
dt	Sample period (s).
gyro_var, accel_var	Measurement-noise variances (match the fit).
cv_warn	Log-space parameter SD above which a parameter is deemed poorly identified.
chi2_ready, chi2_bad	Reduced-chi-square thresholds: below chi2_ready the fit is good (a candidate for ready); above chi2_bad it is a misfit (not_ready).

Value

a twin_readiness: verdict, reduced_chi2 (goodness of fit across ALL channels), gyro_nrmse, the (misfit-inflated) parameter cov, per- parameter cv, confidence, and the estimates/theta0/gyro_bias.

References

Sargent RG (2013) J Simul 7:12-24 (model validation); Bevington & Robinson (reduced chi-square goodness of fit); Cramer-Rao identifiability.

See Also

[personalizeTwin\(\)](#), [scoreInterventions\(\)](#), [validateTwin\(\)](#)

Examples

```
truth <- limbTwin(strength = 7, damping = 0.5)
imu <- imuMeasure(simulateTwin(truth, 6, 0.01), imuSensor(gyro_noise = 0.03), seed = 1)
fit <- personalizeTwin(imu, limbTwin(strength = 4), estimate = "strength", dt = 0.01)
twinReadiness(fit, imu, dt = 0.01)$verdict
```

unscentedKalmanFilter *Unscented Kalman filter (general nonlinear state estimator)*

Description

A sigma-point (Julier-Uhlmann) Kalman filter for a nonlinear state-space model $x_k = f(x_{k-1}, k) + w$, $y_k = h(x_k, k) + v$. Used here to assimilate a mechanistic twin with sensor data and to estimate augmented parameters, but it is a stand-alone engine for any such model.

Usage

```
unscentedKalmanFilter(
  f,
  h,
  y,
  Q,
  R,
  x0,
  P0,
  alpha = 0.001,
  beta = 2,
  kappa = 0
)
```

Arguments

- f Process model function(x, k) returning the next state.
- h Measurement model function(x, k) returning the predicted measurement.
- y Measurement matrix (n_steps x m).
- Q Process-noise covariance (state x state).
- R Measurement-noise covariance (m x m).
- x0, P0 Initial state mean and covariance.
- alpha, beta, kappa Sigma-point spread parameters (defaults 1e-3, 2, 0).

Value

a ukf_result: state (n_steps x L filtered means), cov (final covariance), innovation (n_steps x m).

References

Julier SJ, Uhlmann JK (1997) SPIE 3068:182-193; Wan & van der Merwe (2000).

See Also

[personalizeTwin\(\)](#)

Examples

```
# track a noisy sine with a random-walk state
set.seed(1); y <- matrix(sin(seq(0, 6, length.out = 60)) + rnorm(60, 0, .1), 60, 1)
fit <- unscentedKalmanFilter(function(x, k) x, function(x, k) x, y,
                             Q = matrix(0.05), R = matrix(0.01), x0 = 0, P0 = matrix(1))
fit$state[60]
```

validateTwin	<i>Validity assessment of a movement twin (verification & validation)</i>
--------------	---

Description

Runs the applicable simulation-validity checks and returns a structured report: verification (energy conservation, integrator order), identifiability (can parameters be recovered in principle), Monte-Carlo recovery (bias / SD / coverage), sensitivity screening (which parameters matter), cross-condition prediction + face validity, and the domain of applicability. This is verification and internal validation on the model itself; validation against a subject’s REAL data is done separately by personalising and predicting held-out recordings.

Usage

```
validateTwin(
  twin,
  estimate = c("strength", "damping"),
  sensor = imuSensor(gyro_noise = 0.03, accel_noise = 0.12),
  duration = 6,
  dt = 0.02,
  theta0 = 0.2,
  n_mc = 15L,
  n_traj = 8L,
  method = "optim",
  checks = c("verification", "identifiability", "recovery", "sensitivity", "prediction",
    "domain")
)
```

Arguments

twin	A movement_twin (the ground-truth model to assess).
estimate	Parameters whose identifiability/recovery are assessed.
sensor	The imu_sensor used for the recovery / prediction checks.
duration, dt, theta0	Simulation settings for the checks.
n_mc, n_traj	Monte-Carlo recovery repetitions and Morris trajectories.
method	Assimilation method for the recovery check ("optim"/"ukf").
checks	Which layers to run (subset of c("verification", "identifiability", "recovery", "sensitivity", "prediction", "domain")).

Value

a twin_validity report.

References

Sargent RG (2013) J Simulation 7:12-24; ASME V&V 10/40; Oberkampf & Roy (2010) Verification and Validation in Scientific Computing.

See Also

[personalizeTwin\(\)](#), [insilicoIntervention\(\)](#)

Examples

```
rep <- validateTwin(limbTwin(strength = 6, damping = 0.5, stiffness = 6),
  estimate = c("strength", "damping"), n_mc = 8, n_traj = 5)
rep$verification
```

windkessel

Windkessel arterial blood-pressure generator

Description

Simulates arterial pressure with the classic Windkessel model: a half-sine ventricular ejection each beat charges the arterial compliance C , which drains through the peripheral resistance R ; the 3-element variant adds a proximal characteristic impedance Z_c . During diastole (no inflow) the pressure decays exponentially with time constant $R * C$. With physiological parameters it reproduces the systolic/diastolic/mean/pulse pressures and the diastolic run-off.

Usage

```
windkessel(
  duration = 10,
  hr = 70,
  stroke_volume = 80,
  R = 1,
  C = 1.9,
  Zc = 0.05,
  systolic_frac = 0.3,
  elements = 3,
  sfp = 250,
  p0 = 80
)
```

Arguments

duration	Recording length (s).
hr	Heart rate (bpm).
stroke_volume	Stroke volume ejected each beat (mL).
R	Peripheral (systemic vascular) resistance (mmHg*s/mL).
C	Arterial compliance (mL/mmHg).
Zc	Proximal characteristic impedance (mmHg*s/mL); used when elements = 3.
systolic_frac	Fraction of the cardiac cycle occupied by ejection.
elements	2 or 3 (2-element = pure RC; 3-element adds Zc).
sfp	Output sampling rate (Hz).
p0	Initial compliance-node pressure (mmHg).

Value

a windkessel object: time, pressure (arterial pressure, mmHg), flow (aortic inflow, mL/s), and the derived systolic, diastolic, map (mean arterial pressure) and pp (pulse pressure), measured over the last settled beats.

References

Westerhof N, Lankhaar J-W, Westerhof BE (2009) The arterial Windkessel. *Med Biol Eng Comput* 47:131-141.

See Also

[baroreflex\(\)](#), [ecgsyn\(\)](#), [respiration\(\)](#)

Examples

```
bp <- windkessel(duration = 10, hr = 70)
round(c(bp$systolic, bp$diastolic))    # ~ 120 / 80 mmHg
```

Index

abcCalibration, 4
abcCalibration(), 6, 35, 45, 55, 60
abcSMC, 5
abcSMC(), 34, 45
adcQuantize, 7
adcQuantize(), 8, 59
adcSample, 7
adcSample(), 7
aperiodicNoise, 8
aperiodicNoise(), 40
as.Date(), 61

bandPower, 9
baroreflex, 10
baroreflex(), 12, 73

cardioRespiratory, 11
cardioRespiratory(), 14, 52, 53
cardioStateSpace, 13
cardioStateSpace(), 53, 54
cardioSummary, 14
cardioSummary(), 52, 53
clinicalDecision, 15
clinicalDecision(), 30, 61, 65
cpgMatsuoka, 16
cpgMatsuoka(), 41, 42
crosstalkMatrix, 17
crosstalkMatrix(), 24, 25

defaultInterventions, 18
defaultInterventions(), 15, 39, 64
detectRpeaks, 18
detectRpeaks(), 20, 21

ecgLead, 19
ecgLead(), 20, 21
ecgLeadSet, 20
ecgLeadSet(), 22
ecgMeasure, 21
ecgMeasure(), 19, 20, 22

ecgMeasureLeads, 22
ecgMeasureLeads(), 20
ecgsyn, 22
ecgsyn(), 20–22, 41, 42, 64, 73
emgArray, 24
emgArray(), 17, 25
emgArrayMeasure, 25
emgArrayMeasure(), 17, 24
emgElectrode, 26
emgElectrode(), 24, 28
emgEnvelope, 27
emgEnvelope(), 26, 28
emgMeasure, 27
emgMeasure(), 26, 27
ensembleKalmanFilter, 28
ensembleKalmanFilter(), 31, 49
evaluateIntervention, 29
evaluateIntervention(), 61
extendedKalmanFilter, 30
extendedKalmanFilter(), 29, 49, 55

fillMarkerGaps, 31
fillMarkerGaps(), 43, 44

generateTrainingData, 32
generateTrainingData(), 36, 57
gpCalibrate, 33
gpCalibrate(), 45, 52, 53
gpEmulator, 34
gpEmulator(), 5, 33, 34, 57

imuMeasure, 35
imuMeasure(), 15, 32, 36, 55, 66
imuSensor, 36
imuSensor(), 26, 35, 44
insilicoIntervention, 37
insilicoIntervention(), 38, 39, 43, 55–57, 71
insilicoInterventionMultimodal, 38

insilicoInterventionMultimodal(), 46, 66
intervention, 38
intervention(), 18

jansenRit, 39
jansenRit(), 8, 9, 17, 23, 42

kuramoto, 41
kuramoto(), 17, 23, 41

limbTwin, 42
limbTwin(), 32, 46, 56, 66

markerMeasure, 43
markerMeasure(), 31, 44
markerSensor, 44
markerSensor(), 43
metropolis, 45
metropolis(), 6, 50, 51, 53, 54
multimodalTwin, 46
multimodalTwin(), 38, 57, 66

optimalDesign, 47

particleFilter, 48
particleFilter(), 29, 31, 50, 51, 55
particleGibbs, 49
particleGibbs(), 13, 53
particleMCMC, 50
particleMCMC(), 13, 14, 50, 53, 54
personalizeCardio, 52
personalizeCardio(), 14, 34, 51, 53, 54
personalizeCardioWaveform, 53
personalizeCardioWaveform(), 14
personalizeTwin, 54
personalizeTwin(), 5, 15, 35, 37, 43, 48, 53, 68–71
populationIntervention, 56
populationIntervention(), 38
predict.gpEmulator, 57
predict.gpEmulator(), 34, 35
profileLikelihood, 58
profileLikelihood(), 5, 48, 68

quantizationNoise, 59
quantizationNoise(), 7

reconcileEvidence, 60
reconcileEvidence(), 65

rehabEvaluate, 60
respiration, 62
respiration(), 64, 73
rsaTachogram, 63
rsaTachogram(), 11, 12, 23, 46, 62

scoreInterventions, 64
scoreInterventions(), 15, 18, 30, 39, 60, 61, 69
simulateMultimodal, 65
simulateMultimodal(), 38, 46
simulateTwin, 66
simulateTwin(), 35, 37, 43, 56
sobolIndices, 67
sobolIndices(), 35, 59

twinReadiness, 68
twinReadiness(), 15, 64, 65

unscentedKalmanFilter, 69
unscentedKalmanFilter(), 28–31, 48, 49, 55

validateTwin, 70
validateTwin(), 23, 48, 59, 68, 69

windkessel, 72
windkessel(), 10–12, 23, 62