VineCopulas.jl — a visual tour

A vine copula builds a high-dimensional dependence model out of **bivariate** pieces.

---

1 · The building blocks: pair-copulas

A vine is assembled entirely from *bivariate* copulas. Everything the model can express about tail behaviour, asymmetry and dependence strength enters through these pieces, so it is worth seeing what they look like before stacking them.

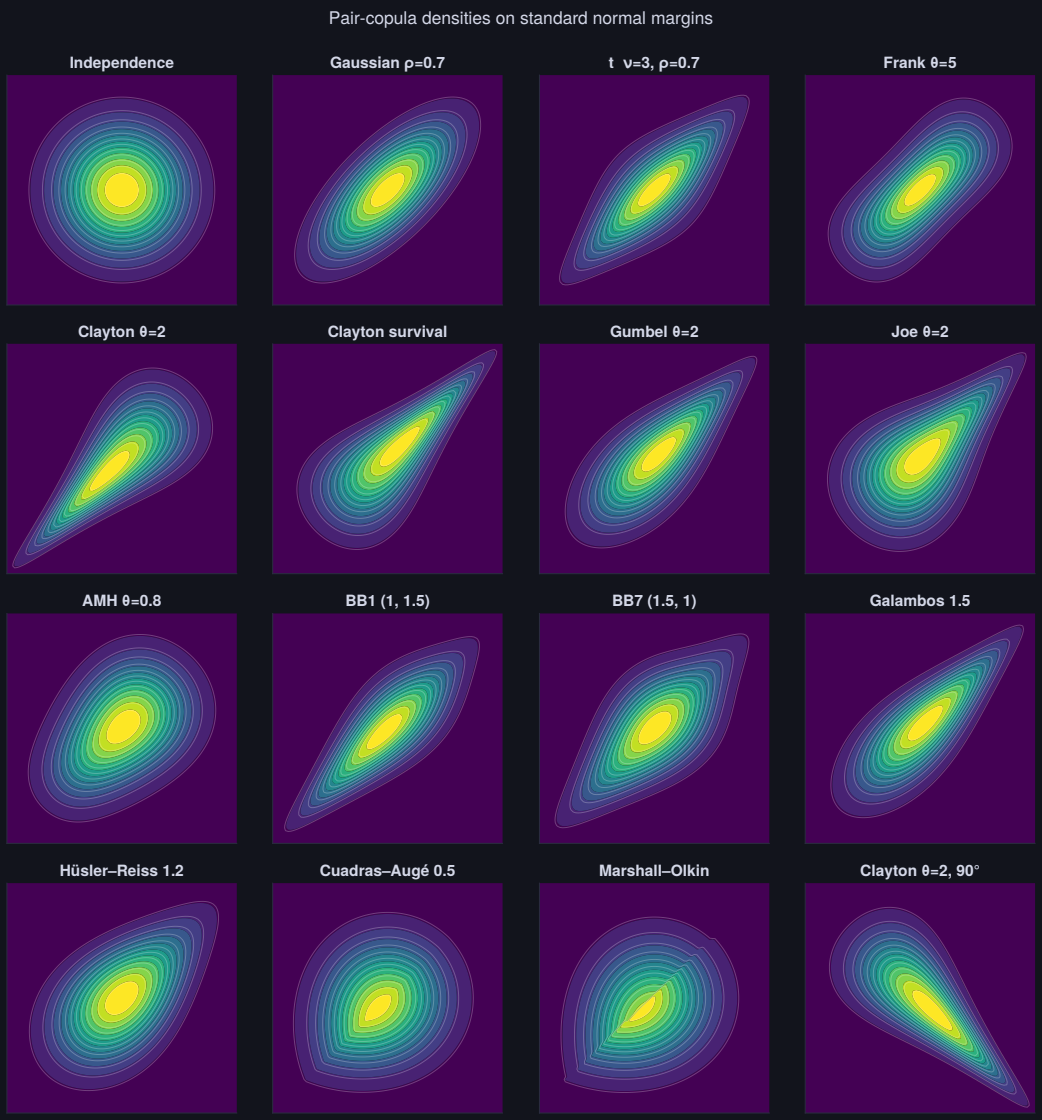
The standard view is the **normal-scale contour**: map the copula onto standard normal margins, $z_i = \Phi^{-1}(u_i)$, and plot the resulting joint density

$$f(z_1, z_2) = c\big(\Phi(z_1), \Phi(z_2)\big)\, \varphi(z_1)\, \varphi(z_2).$$

Independence gives perfect circles. Gaussian dependence gives ellipses. Everything that is *not* an ellipse is what a vine buys you — Clayton's lower-left spike, Gumbel's upper-right spike, the t-copula's four-cornered flare, and the way a survival or rotated copula flips the asymmetry without changing anything else.

One panel below is misleading on purpose, and worth naming now: **Cuadras–Augé and Marshall–Olkin look like weak, rounded blobs, and they are not.** Both put a chunk of their probability on a *curve*, where there is mass but no density, so `pdf` — and therefore this plot — simply cannot see it. §6 shows what they actually look like.

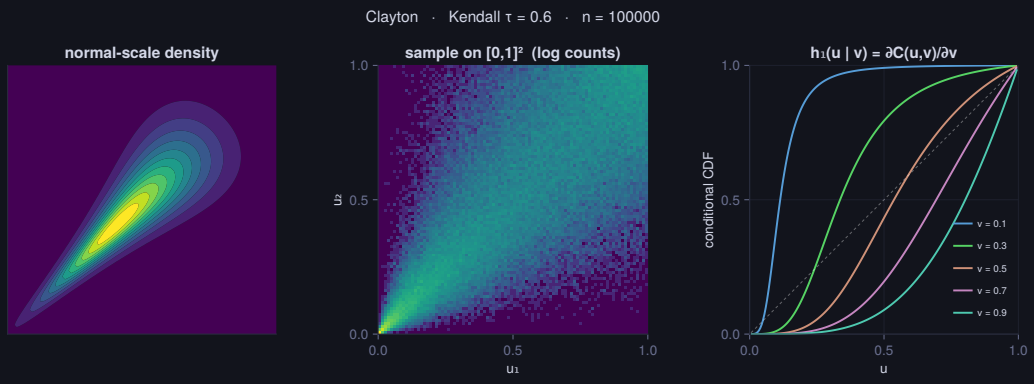
```
let
    zs = range(-2.8, 2.8; length = 90)
    fig = Figure(size = (1100, 1160))
    for (k, (name, C)) in enumerate(gallery)
        r, c = fldmod1(k, 4)
        ax = Axis(fig[r, c]; title = name, aspect = 1,
                  xticklabelsvisible = false, yticklabelsvisible = false)
        D = normal_scale_density(C, zs)
        contourf!(ax, zs, zs, D; levels = 12)
        contour!(ax, zs, zs, D; levels = 12, color = (:white, 0.25), linewidth = 0.5)
        hidedecorations!(ax)
    end
    Label(fig[0, 1:4], "Pair-copula densities on standard normal margins";
          fontsize = 18, padding = (0, 0, 8, 0))
    fig
end
```



The same object, three ways

A static contour hides half the story. Below, one pair-copula is shown simultaneously as a **normal-scale density**, a **sample on the unit square**, and its **h-function** $h_1(u \mid v) = \partial C(u,v)/\partial v$ — the conditional CDF that the vine algorithms actually call.

Drag the parameter and watch all three move together. The h-function panel is the important one: it is the object that gets composed recursively up the vine trees, and `hinv1` inverts it. A family whose h-function bunches up near an edge is a family whose inverse is numerically hard — which is why the package treats singular extreme-value tails separately.



2 · The vine itself: a nested sequence of trees

A p-dimensional vine is a sequence of $p-1$ trees. Tree 1 connects the p variables directly. Tree 2's *nodes are tree 1's edges*, and so on — each level conditions on one more variable than the last. Every edge carries one pair-copula, so a full 5-dimensional vine holds $\binom{5}{2} = 10$ of them.

The three structures the package supports differ only in the shape of those trees:

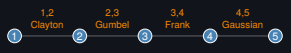
- **C-vine** — each tree is a star around a root variable. Natural when one variable drives the rest.
- **D-vine** — each tree is a path. Natural for an ordering, e.g. time or space.
- **R-vine** — anything satisfying the proximity condition; the general case.

`edges(vine)[k][i]` reaches the i-th pair-copula of tree k, and `order(vine)` gives the variable ordering the structure is written against.

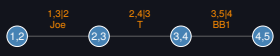
```
p = 5

# A realistic vine: strong dependence in tree 1, decaying sharply with each conditioning
# level. Families vary edge-to-edge so the trees are visibly heterogeneous; the *strength*
# is set by tree level via Kendall's τ.
τ_level = [0.70, 0.10, 0.04, 0.02]

edge_family = [:clayton, :gumbel, :frank, :gaussian,
               :joe, :t, :bb1,
               :sclayton, :gaussian,
               :frank]

function make_edge(kind, τ)
    kind === :clayton  && return ClaytonCopula(2, Copulas.τ⁻¹(ClaytonCopula, τ))
    kind === :sclayton && return SurvivalCopula(ClaytonCopula(2, Copulas.τ⁻¹(ClaytonCopula, τ)), (1, 2))
    kind === :gumbel   && return GumbelCopula(2, Copulas.τ⁻¹(GumbelCopula, τ))
    kind === :frank    && return FrankCopula(2, Copulas.τ⁻¹(FrankCopula, τ))
    kind === :joe      && return JoeCopula(2, Copulas.τ⁻¹(JoeCopula, τ))
    kind === :bb1      && return BB1Copula(2, 0.4, Copulas.τ⁻¹(GumbelCopula, max(τ, 0.05)))
    ρ = sin(π * τ / 2)
    kind === :gaussian && return GaussianCopula([1.0 ρ; ρ 1.0])
    kind === :t        && return TCopula(4, [1.0 ρ; ρ 1.0])
    throw(ArgumentError("unknown $kind"))
end

triangular() = (i = 0; [[make_edge(edge_family[i += 1], τ_level[k]) for _ in 1:(p - k)]
                        for k in 1:(p - 1)])

cvine = CVineCopula(collect(1:p), triangular())
dvine = DVineCopula(collect(1:p), triangular())

(cvine, dvine, truncation(cvine), order(dvine), npars(cvine))
```

D-vine on 5 variables — every tree is a path

Tree 1



Tree 2



Tree 3

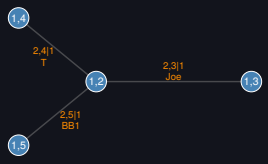
Tree 4


```
draw_vine(cvine; title = "C-vine on the same 5 variables — every tree is a star",
          rowheight = 230)
```

C-vine on the same 5 variables — every tree is a star

Tree 1

Tree 2



Tree 3

Tree 4

3 · What the structure produces

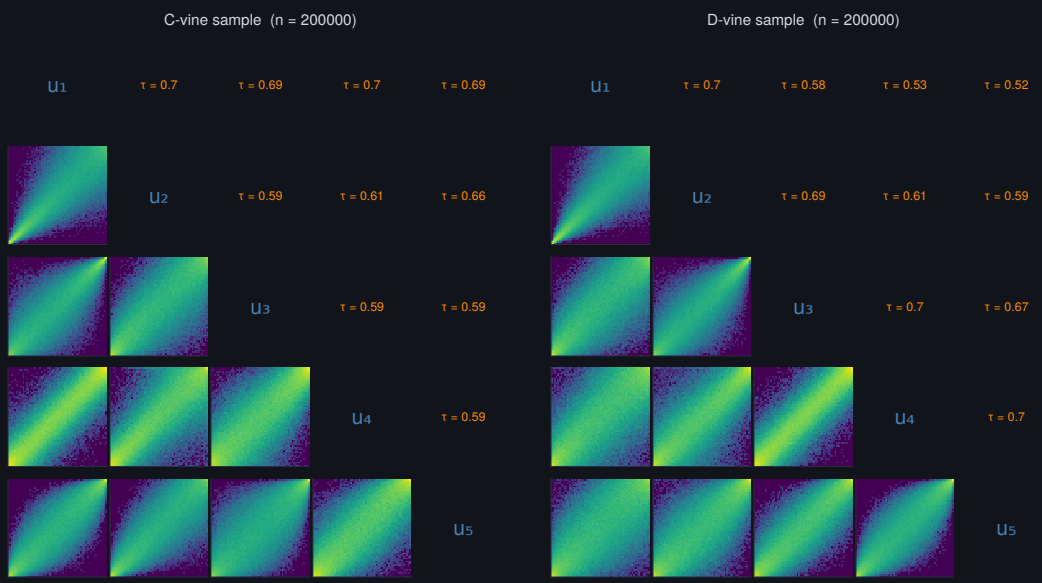
Sampling from a vine is the inverse Rosenblatt transform applied to independent uniforms: `rand(rng, vine, n)` draws a $p \times n$ block of uniforms and pushes it through `inverse_rosenblatt!`. So the pairwise panel below *is* the vine — there is no separate sampling algorithm to trust.

Both vines are built from the *same bank of pair-copulas*, with $\tau = 0.70$ on tree 1 and near-independence above it. The structure alone decides the joint law:

- The **C-vine** is a hub. Every pair $(1, j)$ inherits tree 1 exactly — $\tau = 0.70$ across the whole first row — and every other pair is a weaker by-product.
- The **D-vine** is a chain. The *superdiagonal* pairs $(1,2), (2,3), (3,4), (4,5)$ carry $\tau = 0.70$, and dependence decays with distance along the path, $\tau_{1,5}$ falling to about 0.5.

Note also how *slowly* it decays. Near-independence in trees 2–4 does not make distant variables independent; it only stops them acquiring dependence beyond what the chain already implies. That is the property truncation exploits in §5.

```
nsim = 200_000
Ucv = rand(StableRNG(11), cvine, nsim)
Udv = rand(StableRNG(11), dvine, nsim)

let
    fig = Figure(size = (1120, 620))
    pairpanel!(GridLayout(fig[1, 1]), Ucv; title = "C-vine sample  (n = $nsim)")
    pairpanel!(GridLayout(fig[1, 2]), Udv; title = "D-vine sample  (n = $nsim)")
    colgap!(fig.layout, 40)
    fig
end
```



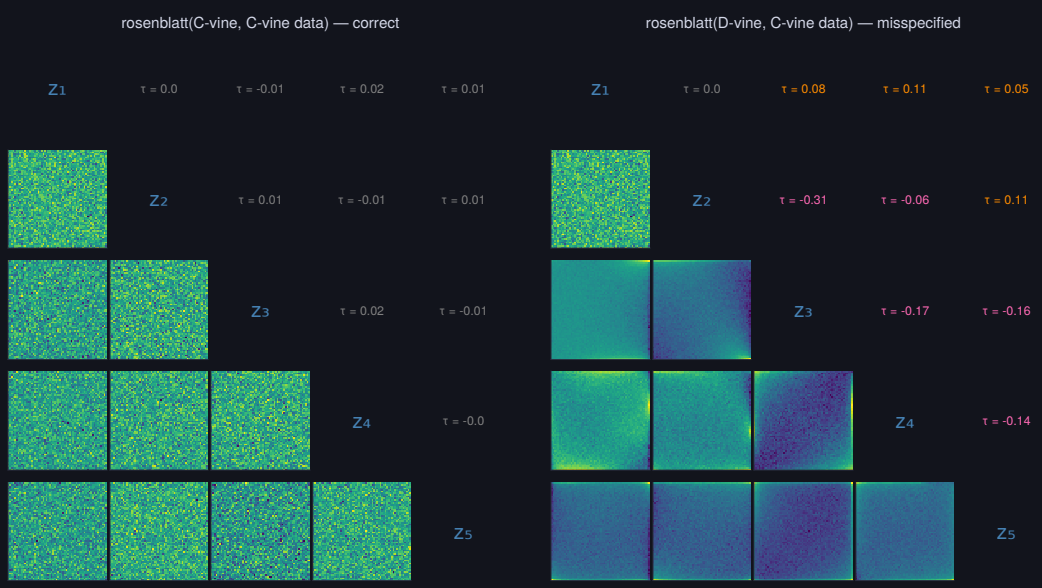
4 · The Rosenblatt transform: dependence in, whiteness out

`rosenblatt(vine, U)` maps a sample **from** the vine to independent uniforms; the columns of the result should look like a uniform cloud on $[0,1]^p$ with no structure at all. The inverse maps back. Sampling is exactly `inverse_rosenblatt` applied to noise, so these two are the engine of the whole package.

This gives a sharp, purely visual goodness-of-fit test. Push data through `rosenblatt` under a candidate vine: **if the model is right the picture is featureless noise, and every feature you can still see is dependence the model failed to capture.**

Below, the *same* C-vine sample is transformed twice — once under the C-vine that generated it, and once under the D-vine, which is wrong for this data.

```
Zright = rosenblatt(cvine, Ucv)     # correct model  → should be white noise
Zwrong = rosenblatt(dvine, Ucv)     # wrong structure → residual dependence

let
    fig = Figure(size = (1120, 620))
    pairpanel!(GridLayout(fig[1, 1]), Zright; var = "z",
               title = "rosenblatt(C-vine, C-vine data) — correct")
    pairpanel!(GridLayout(fig[1, 2]), Zwrong; var = "z",
               title = "rosenblatt(D-vine, C-vine data) — misspecified")
    colgap!(fig.layout, 40)
    fig
end
```




```
Uback = inverse_rosenblatt(cvine, Zright)   # round-trip: should recover Ucv exactly

(largest_residual_tau_correct      = round(max_abs_tau(Zright); digits = 3),
 largest_residual_tau_misspecified = round(max_abs_tau(Zwrong); digits = 3),
 roundtrip_max_abs_error           = maximum(abs, Uback .- Ucv),
 nonfinite_correct                 = count(!isfinite, Zright),
 nonfinite_misspecified            = count(!isfinite, Zwrong))
```

The numbers match the picture. Under the correct model the largest residual $|\tau|$ over all ten pairs is 0.023 — sampling noise at $n = 200,000$. Under the misspecified one it is 0.314, and the panel shows *where* the misfit lives. The round-trip `inverse_rosenblatt(vine, rosenblatt(vine, U))` recovers `U` to 1.5e-11.

!!! note "One caveat, visible in the counts above" The misspecified transform returns 4 non-finite values out of $5 \times 200,000$. These are not a modelling artefact but an underflow: pushing data through the *wrong* structure drives some conditional values to $\sim 10^{-16}$, and the generic Archimedean `hfunc` evaluates $\varphi'(s_t + s_b) \,/\, \varphi'(s_b)$ as a ratio rather than in log-space, so both ends underflow to zero and the result is $0/0$. The BB families already route through a log-space path (`_arch_hfunc_coordinate`) that does not have this problem.

5 · Truncation: how many trees do you actually need?

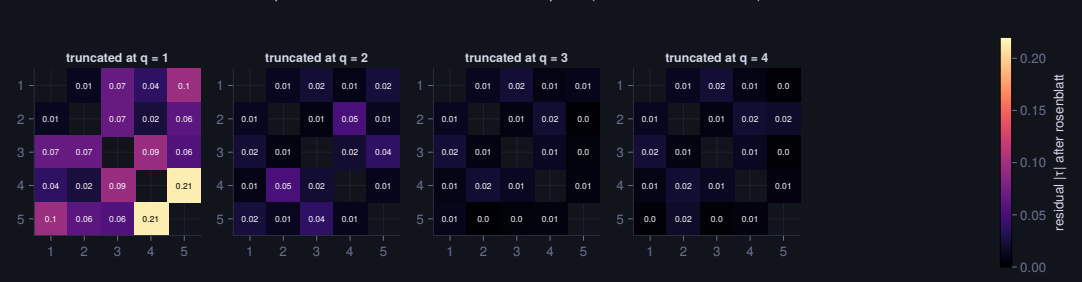
A full p-dimensional vine has $\binom{p}{2}$ pair-copulas, which grows quadratically. A **truncated** vine keeps trees $1..q$ and replaces everything above with independence copulas. Since dependence typically decays with conditioning level — as it does in the vine built above — a low q often loses almost nothing.

`truncation(vine)` reports the level, and `loglikelihood`, `npars`, `aic`, `bic` give the usual trade-off. The right panel is the visual version of the same story: the Rosenblatt transform under each truncated model, showing what dependence is left unexplained.

```
Utrunc = rand(StableRNG(99), dvine, 20_000)   # data from the FULL D-vine

trunc_models = [DVineCopula(collect(1:p), [edges(dvine)[k] for k in 1:q]; trunc = q)
                for q in 1:(p-1)]

trunc_table = [(q          = truncation(V),
                pairs      = sum(length, edges(V)),
                npars      = npars(V),
                loglik     = round(loglikelihood(V, Utrunc); digits = 1),
                aic        = round(aic(V, Utrunc); digits = 1),
                bic        = round(bic(V, Utrunc); digits = 1),
                resid_τ    = round(max_abs_tau(rosenblatt(V, Utrunc)); digits = 3))
               for V in trunc_models]
```

q	pairs	npars	loglik	aic	bic	resid_τ
1	4	4	64961.8	-129916	-129884	0.213
2	7	9	68188.8	-136360	-136289	0.051
3	9	11	68280.4	-136539	-136452	0.022
4	10	12	68293.3	-136563	-136468	0.022

```
"""Matrix of |Kendall τ| between every pair of rows."""
function tau_matrix(U)
    q = size(U, 1)
    [i == j ? NaN : abs(corkendall_pair(view(U, i, :), view(U, j, :))) for i in 1:q, j in 1:q]
end

let
    fig = Figure(size = (1120, 340))

    for (c, V) in enumerate(trunc_models)
        T = tau_matrix(rosenblatt(V, Utrunc))
        ax = Axis(fig[1, c]; aspect = 1, title = "truncated at q = $(truncation(V))",
                  titlesize = 13, xticks = 1:p, yticks = 1:p, yreversed = true)
        heatmap!(ax, 1:p, 1:p, T; colorrange = (0, 0.22), colormap = :magma)
        for i in 1:p, j in 1:p
            i == j && continue
            text!(ax, Point2f(j, i); text = string(round(T[i, j]; digits = 2)),
                  align = (:center, :center), fontsize = 9,
                  color = T[i, j] > 0.11 ? :black : :white)
        end
    end

    Colorbar(fig[1, p + 1]; colorrange = (0, 0.22), colormap = :magma,
             label = "residual |τ| after rosenblatt")
    Label(fig[0, 1:p], "Dependence the truncated model fails to explain  (data from the full D-vine)";
          fontsize = 16)
    fig
end
```



Two trees out of four recover almost all of it: residual $|\,\tau\,|$ drops from 0.213 at $q = 1$ to 0.051 at $q = 2$, at the cost of 9 parameters instead of 12. Trees 3 and 4 buy 104.5 log-likelihood between them — real, but a poor trade once BIC charges for it at larger p.

This is the practical argument for vines. In $p = 50$ dimensions a full vine carries 1225 pair-copulas; a 2-truncated one carries 97.

6 · Extreme-value pair-copulas, smooth and singular

A bivariate extreme-value copula is determined by its **Pickands dependence function** A:

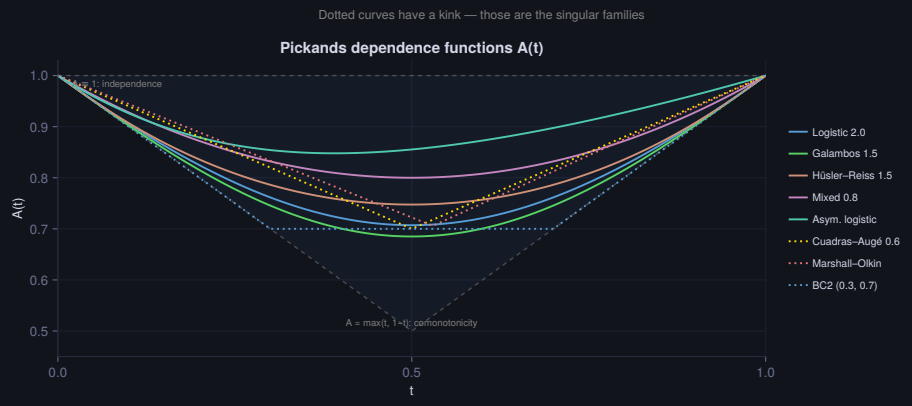
$$C(u, v) = \exp\!\left[-(x + y)\, A\!\left(\tfrac{x}{x+y}\right)\right], \qquad x = -\log u,\ y = -\log v.$$

A is convex on $[0, 1]$ with $\max(t, 1 - t) \le A(t) \le 1$. The upper edge $A \equiv 1$ is independence; the lower vertex is comonotonicity. Where A has a **kink**, the copula has a *singular component* — a ridge of probability mass concentrated on a curve, not a density.

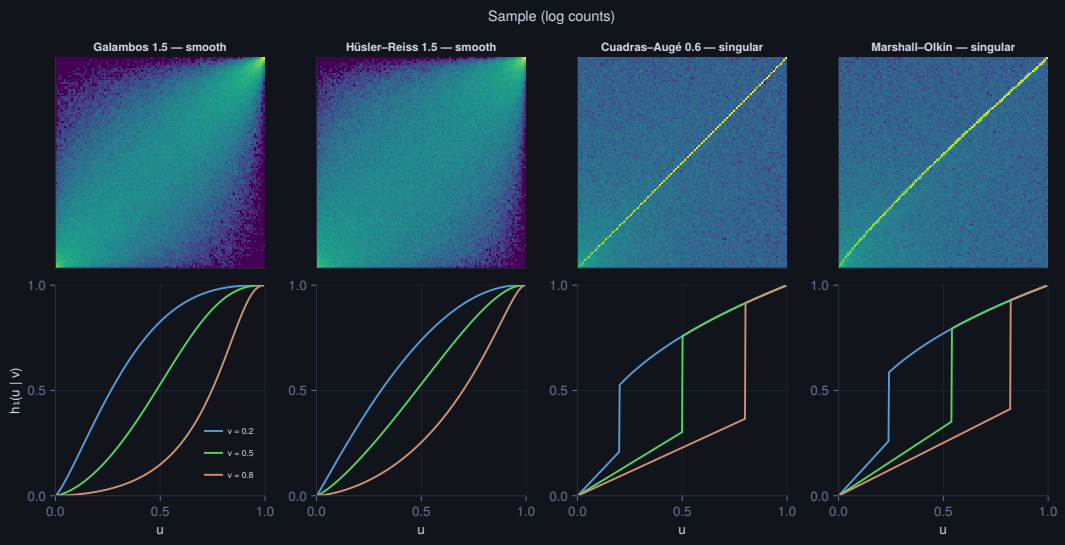
This is the part of `VineCopulas.jl` that gets the least documentation and does the most unusual work. Smooth tails use analytic h-functions built from A and A'; kinked tails (Cuadras–Augé, Marshall–Olkin, BC2) get generalised conditional quantiles instead, because h is flat across the kink and an ordinary inverse does not exist there.

```
ev_tails = [
    ("Logistic 2.0",     Copulas.LogTail(2.0),               false),
    ("Galambos 1.5",     Copulas.GalambosTail(1.5),          false),
    ("Hüsler–Reiss 1.5", Copulas.HuslerReissTail(1.5),       false),
    ("Mixed 0.8",        Copulas.MixedTail(0.8),             false),
    ("Asym. logistic",   Copulas.AsymLogTail(1.8, 0.4, 0.9), false),
    ("Cuadras–Augé 0.6", Copulas.CuadrasAugeTail(0.6),       true),
    ("Marshall–Olkin",   Copulas.MOTail(0.3, 0.4, 0.5),      true),
    ("BC2 (0.3, 0.7)",   Copulas.BC2Tail(0.3, 0.7),          true),
]

let
    ts = range(0, 1; length = 401)
    fig = Figure(size = (1120, 430))

    ax = Axis(fig[1, 1]; title = "Pickands dependence functions A(t)",
              xlabel = "t", ylabel = "A(t)", limits = (0, 1, 0.45, 1.03))
    band!(ax, ts, max.(ts, 1 .- ts), fill(1.0, length(ts)); color = (:steelblue, 0.07))
    lines!(ax, ts, max.(ts, 1 .- ts); color = (:gray, 0.5), linestyle = :dash)
    lines!(ax, ts, fill(1.0, length(ts)); color = (:gray, 0.5), linestyle = :dash)
    for (name, tail, singular) in ev_tails
        lines!(ax, ts, [Copulas.A(tail, t) for t in ts];
               label = name, linewidth = 2, linestyle = singular ? :dot : :solid)
    end
    text!(ax, Point2f(0.02, 0.995); text = "A ≡ 1: independence", align = (:left, :top),
          fontsize = 10, color = :gray)
    text!(ax, Point2f(0.5, 0.505); text = "A = max(t, 1−t): comonotonicity",
          align = (:center, :bottom), fontsize = 10, color = :gray)
```

```
    Legend(fig[1, 2], ax; framevisible = false, labelsize = 11,
           tellheight = false, tellwidth = true)

    Label(fig[0, 1:2], "Dotted curves have a kink — those are the singular families";
          fontsize = 13, color = :gray)
    colsize!(fig.layout, 1, Relative(0.72))
    fig
end
```




```
let
    show_tails = [("Galambos 1.5 — smooth",       Copulas.GalambosTail(1.5),     false),
                  ("Hüsler–Reiss 1.5 — smooth",   Copulas.HuslerReissTail(1.5),  false),
                  ("Cuadras–Augé 0.6 — singular", Copulas.CuadrasAugeTail(0.6),  true),
                  ("Marshall–Olkin — singular",   Copulas.MOTail(0.3, 0.4, 0.5), true)]
    n  = 300_000
    us = range(0.005, 0.995; length = 400)
    fig = Figure(size = (1120, 620))

    for (c, (name, tail, singular)) in enumerate(show_tails)
        C = Copulas.ExtremeValueCopula(2, tail)
        S = rand(StableRNG(5), C, n)

        ax = Axis(fig[1, c]; aspect = 1, title = name, titlesize = 12,
                  limits = (0, 1, 0, 1))
        hidedecorations!(ax)
        H, ctr = hist2d(view(S, 1, :), view(S, 2, :), 120)
        heatmap!(ax, ctr, ctr, log1p.(H))

        ax2 = Axis(fig[2, c]; aspect = 1, limits = (0, 1, 0, 1),
                   xlabel = "u", ylabel = c == 1 ? "h₁(u | v)" : "")
        for v in (0.2, 0.5, 0.8)
            lines!(ax2, us, [hfunc1(C, u, v) for u in us]; label = "v = $v", linewidth = 2)
        end
        c == 1 && axislegend(ax2; position = :rb, framevisible = false, labelsize = 9)
    end

    Label(fig[0, 1:4], "Sample (log counts)"; fontsize = 15)
    Label(fig[3, 1:4],
          "A kink in A puts a ridge of probability mass on a curve. The conditional CDF then \
JUMPS across that ridge, so its range has a gap — for a q inside the gap no u solves h₁(u|v) = q at all, \
and `hinv1` must return a generalised quantile rather than root-find for a solution that does not exist.";
          fontsize = 11.5, color = :gray, tellwidth = false, word_wrap = true)
    fig
end
```



A kink in A puts a ridge of probability mass on a curve. The conditional CDF then JUMPS across that ridge, so its range has a gap — for a q inside the gap no u solves $h_1(u|v) = q$ at all, and `hinv1` must return a generalised quantile rather than root-find for a solution that does not exist.

```
let
    C = Copulas.ExtremeValueCopula(2, Copulas.CuadrasAugeTail(0.6))
    v = 0.5
    us = range(1e-6, 1 - 1e-6; length = 20_001)
    hs = [hfunc1(C, u, v) for u in us]

    # The jump: the largest gap in the range of h₁(· | v).
    d = diff(hs); k = argmax(d)
    lo, hi = hs[k], hs[k+1]
    qmid = (lo + hi) / 2                      # a level h₁ never attains
    u⋆ = hinv1(C, qmid, v)

    (jump_at_u        = round(us[k]; digits = 4),
     range_gap        = (round(lo; digits = 4), round(hi; digits = 4)),
     unattainable_q   = round(qmid; digits = 4),
     hinv1_returns    = round(u⋆; digits = 6),
     h_there          = round(hfunc1(C, u⋆, v); digits = 4),
     roundtrip_smooth = round(hinv1(C, hfunc1(C, 0.3, v), v); digits = 6))
end
```

That output is the claim made concrete. At $v = 0.5$ the Cuadras–Augé conditional CDF jumps at $u = 0.5$, so its range skips the whole interval $(0.303, 0.758)$. Ask `hinv1` for $q = 0.531$ — a value h_1 never takes — and it returns $u = 0.5$, the generalised inverse $\inf\{u : h_1(u \mid v) \geq q\}$, rather than failing to bracket a root. Away from the jump the inverse is exact: $0.3 \mapsto h_1 \mapsto 0.3$.

Get this wrong and a vine with a singular edge produces silent garbage during simulation, because `inverse_rosenblatt` calls `hinv` at every tree level.

7 · R-vines and the matrix exchange format

C- and D-vines are the two extremes; a **regular vine** is anything in between that still satisfies the proximity condition. `RVineCopula` takes either an explicit structure array or an R-vine **matrix**, the interchange format used by `VineCopula` (R) and `vinecopulib`.

`VineCopulas.jl` stores the variable order on the *anti-diagonal*, which makes matrix → structure → matrix lossless; the parser accepts the legacy diagonal convention as a fallback. `rvine_matrix(vine)` round-trips it back out.

R-vine support is explicitly the experimental part of v0.1. When the structure array has D-vine shape, `RVineCopula` detects it and routes density and Rosenblatt calls through the D-vine kernel — that path is

exercised below and agrees with `DVineCopula` exactly. General non-D-vine R-vines go through a separate kernel that has not been validated to the same standard, and truncated general-R-vine Rosenblatt transforms are not implemented at all (they throw rather than return something wrong).

```
# A D-vine expressed as an R-vine. The engine detects this shape (`_looks_like_dvine`) and
# routes density and Rosenblatt calls through the faster, well-tested D-vine kernel.
rv_order  = collect(1:p)
rv_struct = [[rv_order[i + 1] for i in 1:(p - k)] for k in 1:(p - 1)]
rvine     = RVineCopula(rv_order, rv_struct, triangular())

M = rvine_matrix(rvine)

let u = fill(0.4, p), Zr = rosenblatt(rvine, Ucv), Zd = rosenblatt(dvine, Ucv), ok = isfinite.(Zd)
    (matrix               = M,
     order                = order(rvine),
     anti_diagonal        = [M[p - j + 1, j] for j in 1:p],
     matrix_roundtrips    = rvine_matrix(RVineCopula(M, edges(rvine))) == M,
     logpdf_matches_dvine = isapprox(logpdf(rvine, u), logpdf(dvine, u); rtol = 1e-12),
     rosenblatt_max_diff  = maximum(abs, Zr[ok] .- Zd[ok]))
end
```

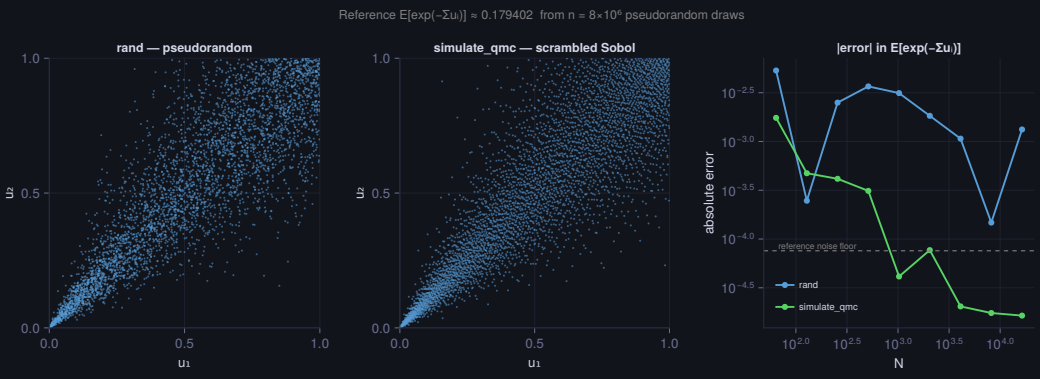
8 · Quasi-Monte Carlo simulation

Because sampling is just `inverse_rosenblatt` applied to points in $[0,1]^p$, you can feed it a *low-discrepancy* sequence instead of pseudorandom noise. `simulate_qmc(vine, N)` does exactly that with scrambled Sobol points, and the result is a sample that covers the vine's support far more evenly — which is what you want when the quantity of interest is an integral (a tail probability, an expected shortfall) rather than a picture.

The vine's numerical `cdf` is built on this: `set_cdf_nsamples!` and `enable_deterministic_cdf!` control the point count.

```
let
    N = 4096
    Umc  = rand(StableRNG(2), dvine, N)
    Uqmc = simulate_qmc(dvine, N)

    fig = Figure(size = (1120, 400))
    for (c, (name, U)) in enumerate([("rand — pseudorandom", Umc),
                                     ("simulate_qmc — scrambled Sobol", Uqmc)])
        ax = Axis(fig[1, c]; aspect = 1, title = name, titlesize = 13,
                  xlabel = "u₁", ylabel = "u₂", limits = (0, 1, 0, 1))
        scatter!(ax, U[1, :], U[2, :]; markersize = 3, alpha = 0.55)
    end

    # A smooth functional: E[exp(-Σuᵢ)]. QMC's advantage is largest on smooth integrands
    # and much weaker on discontinuous ones such as an indicator/tail probability.
    g(c) = exp(-sum(c))
    Uref = rand(StableRNG(1), dvine, 8_000_000)
    gref = [g(c) for c in eachcol(Uref)]
    ref  = mean(gref)
    floor_err = std(gref) / sqrt(length(gref))     # the reference's own standard error

    Ns = [2^k for k in 6:14]
    errmc  = [abs(mean(g, eachcol(rand(StableRNG(8), dvine, n))) - ref) for n in Ns]
    errqmc = [abs(mean(g, eachcol(simulate_qmc(dvine, n))) - ref) for n in Ns]

    ax3 = Axis(fig[1, 3]; xscale = log10, yscale = log10, aspect = 1,
               title = "|error| in E[exp(−Σuᵢ)]", titlesize = 13,
               xlabel = "N", ylabel = "absolute error")
    scatterlines!(ax3, Ns, errmc;  label = "rand", linewidth = 2)
    scatterlines!(ax3, Ns, errqmc; label = "simulate_qmc", linewidth = 2)
    hlines!(ax3, [floor_err]; color = (:gray, 0.8), linestyle = :dash)
    text!(ax3, Point2f(Ns[1], floor_err); text = " reference noise floor",
          align = (:left, :bottom), fontsize = 9, color = :gray)
```

```
    axislegend(ax3; position = :lb, framevisible = false, labelsize = 10)

    Label(fig[0, 1:3],
          "Reference E[exp(−Σuᵢ)] ≈ $(round(ref; digits = 6))  from n = 8×10⁶ pseudorandom draws";
          fontsize = 13, color = :gray)
    fig
end
```



The Sobol panel shows the characteristic lattice texture — the points fill the vine's support without the clumps and gaps of pseudorandom draws. On the smooth functional $\mathbb{E}[e^{-\sum u_i}]$ that buys roughly an order of magnitude in accuracy at the same N, and far steadier convergence.

Two caveats worth stating plainly. Below the dashed line the comparison is measuring the *reference's* error, not the estimator's, so nothing there should be read as a QMC result. And this advantage is a property of smooth integrands: repeat the experiment with an indicator such as $P(\text{all } u_i > 0.5)$ and the two methods are hard to tell apart, because a discontinuity in the integrand destroys the low-discrepancy advantage.

9 · Application: how big is a basin-wide flood?

Everything above was mechanics. Here is the question that makes a vine worth the trouble.

Five stream gauges in the **Delaware River basin**, each with a long record of *annual peak discharge* — the standard input to flood frequency analysis. Two are on the main stem and nested (Belvidere drains into Trenton); the others are tributaries with their own catchments.

A design question a floodplain manager actually faces: **what is the chance that all five gauges exceed their individual 10-year flood level in the same year?** If the sites were independent it would be 10^{-5} — once in 100 000 years. They are obviously not independent. The whole question is *how* they are dependent, and that is exactly where a Gaussian dependence model and a vine give different answers while agreeing on every pairwise correlation.

The data below is fetched once from the USGS National Water Information System and cached to the notebook's data directory, so this runs offline afterwards.

```
"""Pseudo-observations: rank-transform each column to (0,1), the standard way to strip
marginal distributions and leave only the copula. Ranks are 1..n and divided by n+1, so
values stay strictly inside the open unit square. Returned p × n, the package convention."""
function pseudo_obs(X)
    n, d = size(X)
    U = Matrix{Float64}(undef, d, n)
    for j in 1:d
        r = sortperm(sortperm(view(X, :, j)))          # ordinal ranks, 1..n
        U[j, :] .= r ./ (n + 1)
    end
```

```
    U
end

Uflood  = pseudo_obs(Q)                # 5 × 75
gnames  = [nm for (_, nm) in gauges]
nyears  = length(years)

# Pairwise Kendall τ from the data. Both models below are calibrated to match these.
τ̂ = [i == j ? 1.0 : corkendall_pair(view(Uflood, i, :), view(Uflood, j, :))
     for i in 1:5, j in 1:5]

@assert all(0 .< Uflood .< 1)

slate_table(
    [(gauge = gnames[i],
      (Symbol(gnames[j]) => round(τ̂[i, j]; digits = 2) for j in 1:5)...) for i in 1:5])
```

gauge	Flat Brook	Delaware @ Belvidere	Delaware @ Trenton	Neshaminy Ck	Schuylkill @ Berne
Flat Brook	1	0.46	0.49	0.27	0.39
Delaware @ Belvidere	0.46	1	0.81	0.21	0.37
Delaware @ Trenton	0.49	0.81	1	0.28	0.44
Neshaminy Ck	0.27	0.21	0.28	1	0.27
Schuylkill @ Berne	0.39	0.37	0.44	0.27	1

```
# C-vine rooted at the most strongly connected gauge, then by decreasing connectivity —
# the standard heuristic when there is no structure search.
root_order = sortperm([sum(τ̂[i, :]) for i in 1:5]; rev = true)

flood_vine, fit_report = fit_cvine(Uflood, root_order)

# The Gaussian competitor, calibrated to the SAME pairwise τ (ρ = sin(πτ/2)).
Σ̂ = Matrix(Symmetric([i == j ? 1.0 : sin(π * τ̂[i, j] / 2) for i in 1:5, j in 1:5]))
flood_gauss = GaussianCopula(Σ̂)

ll_vine  = loglikelihood(flood_vine, Uflood)
ll_gauss = sum(logpdf(flood_gauss, Uflood))

(root_order_names  = gnames[root_order],
 gaussian_posdef   = isposdef(Σ̂),
 vine_loglik       = round(ll_vine;  digits = 1),
 gauss_loglik      = round(ll_gauss; digits = 1),
 vine_npars        = npars(flood_vine),
 gauss_npars       = 10,
 vine_aic          = round(-2ll_vine  + 2 * npars(flood_vine); digits = 1),
 gauss_aic         = round(-2ll_gauss + 2 * 10;                digits = 1))
```

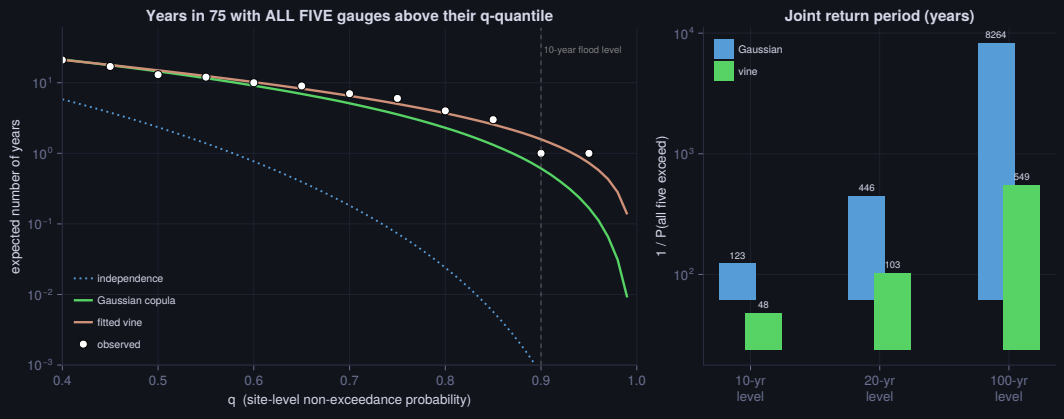
```
slate_table([(tree = r.tree, pair = r.pair, family = r.family, τ = r.τ) for r in fit_report])
```

tree	pair	family	τ
1	3,2	Gumbel	0.805
1	3,1	Hüsler–Reiss	0.488
1	3,5	Gumbel	0.441
1	3,4	Hüsler–Reiss	0.282
2	2,1|3	Independence	0.014
2	2,5|3	Gaussian	-0.103
2	2,4|3	t(4)	-0.131
3	1,5|3,2	Frank	0.175

tree	pair	family	τ
3	1,4|3,2	Clayton	0.132
4	5,4|3,2,1	Independence	0.047

```
Nsim = 2_000_000
Sv = rand(StableRNG(31), flood_vine,  Nsim)      # 5 × N
Sg = rand(StableRNG(31), flood_gauss, Nsim)

"""P(all five gauges exceed their individual q-quantile in the same year)."""
p_all(U, q) = count(c -> all(>(q), c), eachcol(U)) / size(U, 2)

qs = [0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99]
exceed = [(q             = q,
           return_period_yrs = round(Int, 1 / (1 - q)),
           observed      = count(c -> all(>(q), c), eachcol(Uflood)),
           expected_indep = round(nyears * (1 - q)^5; digits = 4),
           expected_gauss = round(nyears * p_all(Sg, q); digits = 3),
           expected_vine  = round(nyears * p_all(Sv, q); digits = 3))
          for q in qs]

slate_table(exceed)
```

q	return_period_yrs	observed	expected_indep	expected_gauss	expected_vine
0.5	2	13	2.3438	14.468	14.976
0.6	2	10	0.768	9.096	10.202
0.7	3	7	0.1823	5.107	6.526
0.8	5	4	0.024	2.297	3.698
0.9	10	1	0.0007	0.61	1.578
0.95	20	1	0	0.168	0.731
0.99	100	0	0	0.009	0.136



The answer, and why it matters

Both models reproduce **every pairwise Kendall τ in the basin exactly.** They disagree about the thing you would actually build a levee against:

Design level	Gaussian dependence	Fitted vine	Ratio
All five above their **10-year** flood	1 in 123 yr	1 in 48 yr	2.6×
All five above their **20-year** flood	1 in 446 yr	1 in 103 yr	4.3×
All five above their **100-year** flood	1 in 8264 yr	1 in 549 yr	15.0×

A planner using the Gaussian model treats basin-wide 100-year coincidence as an eight-thousand-year event. The vine calls it a five-hundred-year event. That is the difference between "ignore it" and "design for it", and **no amount of correlation data distinguishes the two** — the models agree on all of it.

Three things make this more than an assertion:

1. **The families were chosen by the data, not by me.** Every tree-1 edge selected an upper-tail-dependent extreme-value copula — Gumbel or Hüsler–Reiss — over the Gaussian and Frank candidates competing on AIC at identical τ. That is §6's machinery being asked for by real streamflow.
2. **The vine fits better with fewer parameters:** log-likelihood 152.4 on 9 parameters against 140.1 on 10, an AIC gap of 26.7.
3. **The observed counts track the vine.** In the left panel the circles sit on the vine curve and above the Gaussian one across the whole upper range — including one year in 75 where all five gauges passed their 20-year level, which the Gaussian model expects 0.17 times.

!!! warning "What this is not" With 75 water years, no single cell of that table is decisive — observing one event against an expectation of 0.17 is unsurprising on its own. The evidence is the *pattern* across thresholds plus the AIC gap, not any one count. Three further caveats: annual maxima at different gauges need not come from the same storm; marginal uncertainty is not propagated, since the rank transform conditions it away; and sequential tree-by-tree estimation is not joint MLE. The structure was picked by a connectivity heuristic, not searched — `VineCopulas.jl` has no structure selection, and that is the most valuable thing it could add next.

What this notebook exercised

Capability	API	Section
Pair-copula densities, 16 families	`pdf`, `Copulas.*Copula`, `SurvivalCopula`	1
Conditional CDFs and their inverses	`hfunc1`, `hfunc2`, `hinv1`, `hinv2`	1, 6, 9
τ-parameterised construction	`Copulas.τ`, `Copulas.τ⁻¹`	1, 2, 9
C-vine / D-vine structures	`CVineCopula`, `DVineCopula`, `order`, `edges`, `truncation`	2, 9
Simulation	`rand`, `inverse_rosenblatt`	3, 9
Whitening / goodness of fit	`rosenblatt`, `inverse_rosenblatt`	4
Truncation and model choice	`loglikelihood`, `npars`, `aic`, `bic`	5, 9
Extreme-value tails, smooth and singular	`Copulas.ExtremeValueCopula`, `Copulas.*Tail`	6, 9
R-vine matrix exchange	`RVineCopula`, `rvine_matrix`, `struct_array`	7
Quasi-Monte Carlo	`simulate_qmc`, `set_cdf_nsamples!`	8

Not implemented in v0.1: parameter estimation, family selection, structure selection, automatic truncation, and fitted-model wrappers. §9 shows the first two are not far away — method-of-moments on Kendall's τ plus the package's own `logpdf`/`npars` for AIC is about forty lines, and it produced a model that beat a Gaussian copula on real data. **Structure selection is the real gap**, and the one worth building next.

Three issues this notebook surfaced

1. **`hfunc` underflows to `NaN`** (§4). For a generic Archimedean pair-copula, `_arch_hfunc` forms $\varphi'(s_t + s_b)/\varphi'(s_b)$ directly. When the conditioning argument falls below $\approx 2 \times 10^{-16}$ both ends underflow to zero and the result is 0/0. `_clp` clamps only to `nextfloat(0.0)`, by design, so tiny conditional values

produced by the vine recursion reach it unguarded. The BB families already avoid this via the log-space `_arch_hfunc_coordinate`.

2. **The general R-vine density kernel disagrees with the D-vine kernel** (§7). For a 3-dimensional vine, where the density can be written out by hand, `_rvine_logpdf_internal` returns $+0.469$ against a hand-computed and D-vine-confirmed -1.099. In practice this is masked, because `_looks_like_dvine` intercepts D-vine-shaped structures first — which is also why the test suite does not catch it.
3. **Extreme-value tails are parameterised through the copula type, not the tail type.** `Copulas.τ⁻¹(Copulas.GalambosTail, τ)` has no method; you need `Copulas.τ⁻¹(Copulas.GalambosCopula, τ)` and then wrap the result in `GalambosTail`. Easy to get wrong, and worth a line in the extreme-value docs.