Understanding and Using RAMP for Stochastic Electricity Demand Modelling

Table of Contents

Conceptual overview: how RAMP uses parameters

At its core, RAMP is a **hierarchical stochastic process**:

- **User categories** define *how many statistically independent agents exist*.
- **Appliances** define *what discrete stochastic processes those agents own*.
- **Time windows & duty cycles** define *admissible support of random variables*.
- **Probabilistic rules** generate *minute-level realizations* subject to constraints.

Mathematically, each appliance-day is a **constrained marked point process**:

- *Points*: switch-on events (random times),
- *Marks*: duration, power, coincidence,
- *Support*: windows,
- *Stopping rule*: total daily time.

All parameters control **either the support, the distribution, or the stopping conditions** of this process. Each appliance-day is generated by: sampling a total energy budget in time, restricting admissible switch-on locations, sequentially placing events without overlap, conditioning coincidence on peak behavior, marking events with power or duty cycles and summing across users and appliances.

Key concepts and terminology

Statistically independent agents

What it means: Two users are statistically independent if what one does does not directly determine what the other does. In RAMP each household, person, or facility is simulated independently, one user turning on a TV does not force another user to do the same.

Why this matters: this allows RAMP to represent diversity of behavior, avoid artificial synchronization and rely on aggregation to produce smooth community-level demand.

People share habits and constraints, but they do not coordinate minute by minute.

Random variables

A random variable is simply a quantity whose value is not fixed in advance, but instead drawn from a range of possible values (e.g. the exact time you turn on a light in the evening, how long you use your phone today, whether you cook at home or not etc.)

In RAMP, random variables include:

- Switch-on time of an appliance
- Total daily usage time
- Power level (within a range)
- Number of appliances used simultaneously

Each parameter in the Excel file shapes how these random values are drawn.

Realizations

A realization is one concrete outcome of all these random choices.

One realization = one simulated day

Multiple realizations = many possible days

Important point: the model does not produce "the" daily load profile, it produces a family of plausible profiles. This is why RAMP is especially suitable for planning, robustness analysis, and scenario comparison.

Events: "points in time"

A switch-on event is the moment when an appliance starts operating. From a modeling perspective:

- An event is a point in time (e.g. minute 18:42).
- Each event has consequences that last for some time (minutes or hours).

RAMP builds a day by placing multiple switch-on events for each appliance.

Marks: what happens during an event

A mark is additional information attached to an event. For each switch-on event, RAMP assigns:

- Duration (how long it stays on)
- Power (how much electricity it consumes)
- Coincidence (how many identical appliances are on at the same time)

Same event time + different marks → very different load profiles.

This allows RAMP to distinguish, for example, a short phone charge, from a long cooking session, from a fridge cycle.

Support: when events are allowed to happen

The support of a random variable is simply the set of values it is allowed to take.

In RAMP appliances are not allowed to turn on at any time, they are restricted to time windows (e.g. morning, evening). So instead of "An appliance can turn on at any minute of the day" we have "An appliance can turn on only inside these specific windows, possibly with some flexibility." This is how social routines are embedded in the model.

Stopping rule: when the process ends

A stopping rule defines when no more events are added. In RAMP, the stopping rule is the total daily usage time of the appliance. Once the allocated daily time is exhausted no more switch-on events are generated for that appliance. This ensures physical realism, energy conservation at appliance level and consistency across days.

Putting it all together: how a daily profile is built

For each appliance, for each user, for each day, RAMP does the following:

1. Decide whether the appliance is used today (based on probability, preferences, weekday/weekend rules).
2. Draw a total daily usage time (e.g. "this appliance will be used for about 90 minutes today").
3. Define when it could be used (time windows, slightly randomized).
4. Place switch-on events sequentially. Each event has a start time, a duration and a power level, without overlapping previous events.
5. Adjust behavior during peak hours (more appliances tend to be on simultaneously).
6. Repeat until the daily usage time is fully allocated.

7. Sum across appliances → user load
8. Sum across users → community load

Each run of the model gives one realization. Repeating the run gives a distribution of possible load profiles.

The statistical layer behind RAMP

Although RAMP is implemented as an algorithm, its behavior is entirely driven by well-defined statistical building blocks. These are often implicit in the code but correspond to standard concepts in probability theory. The key idea is: RAMP does not use one single distribution, but a **combination of simple probabilistic mechanisms**, each chosen to represent a specific type of real-world uncertainty.

Below, we introduce these mechanisms one by one.

Bernoulli trials: "Does this happen today?"

A Bernoulli trial is the simplest possible random experiment:

- it has two outcomes: yes / no, true / false, success / failure,
- it is controlled by a single probability $p \in [0,1]$.

RAMP uses Bernoulli logic whenever the model must decide whether something happens at all. Examples:

- occasional_use → "Is this appliance used today?"
- pref_index vs user_preference → "Does today match the user's daily preference pattern?"
- wd_we_type → "Is this appliance allowed on this type of day?"

This introduces day-to-day variability, even if all other parameters are fixed.

Uniform distributions: "Anywhere within a range"

A uniform distribution means: every value inside a given range is equally likely. If we say: "Choose a number uniformly between 10 and 20" then 11, 15, and 19 are all equally probable. Uniform randomness is used when there is no strong reason to favor one value over another, and variability is bounded by physical or behavioral limits. Main uses:

- Random shift of time windows (random_var_w)
- Random variation of total usage time
- Random variation of duty-cycle durations (r_c1, r_c2, r_c3)
- Random power variation for thermal appliances (thermal_p_var)

Uniform distributions model imprecision, not preference: "Usage can happen anywhere in this window", "Duration fluctuates within ±10%", "Power depends on conditions but stays within bounds" etc.

Uniform distributions are also applied when the model must choose among a finite set of options, rather than a continuous range (e.g. Start minute of a switch-on event, Which duty cycle to use etc.). This is especially important for placing switch-on events without bias inside allowed windows and choosing among alternative duty-cycle patterns.

Gaussian (normal) distributions: "Typical behavior with variability"

A Gaussian (normal) distribution describes quantities that have a typical value (mean) but fluctuate around it with extreme values being rare. This is the familiar "bell-shaped curve".

Gaussian distributions are used in RAMP only where collective behavior matters, not for individual timing. Main use is coincidence of appliance switch-ons during peak hours. During peak hours people tend to act more synchronously but not perfectly. The Gaussian distribution captures a typical number of simultaneous appliances, with natural dispersion. This is why peaks emerge smoothly rather than as sharp spikes.

Conditional probability: "Behavior depends on context"

In RAMP, probabilities are often conditional, meaning they depend on the situation.

Examples:

- Coincidence depends on whether we are inside the peak window
- Appliance use depends on weekday vs weekend
- Duty cycle selection depends on where the event falls

Conceptually, the same appliance behaves differently depending on when and under which conditions it is used.

Sequential sampling with constraints (no overlap)

Unlike many textbook random processes, RAMP does not sample all events at once. Instead, it:

- Samples one event
- Removes the occupied time
- Samples the next event from the remaining free time
- Stops when the daily budget is exhausted

This is called sequential constrained sampling. This matters because events cannot overlap physically, total daily usage must be respected and windows must not be violated. This mechanism replaces complex mathematics with a procedural but rigorous implementation of constraints.

The process stops when: total allocated time ≥ daily usage time. This enforces a hard constraint, not a probabilistic one.

RAMP Parameters

User-level parameters (population scaling & behavioral conditioning)

- **user_name**: identifier only (used for grouping, exporting, aggregation).
- **num_users**: controls how many independent people are being simulated and how smooth the total demand becomes. Each individual user is simulated independently, each time with different random choices, and treats, and profiles are then summed linearly. It affects how many identical appliances may be on at the same time (More users → higher potential coincidence during peak hours). Where used: *User.generate_aggregated_load_profile()* and *Appliance.calc_coincident_switch_on()*
- **user_preference**: controls which appliances tend to appear together on the same day, representing different types of daily routines. Technically it introduces an upper bound for daily preference states (categorical latent variable) via *pref_index*: each day is assigned a hidden label representing the user's behavioral mode. Where used: *User.generate_single_load_profile()*

  In practice:

  1. Choose how many daily routines exist (e.g. 1 → All days are identical, 2 → Two types of days (e.g. normal / special), 3 → Three routines (e.g. light-use / normal / intensive-use))
  2. Assign each appliance a pref_index: each appliance is tagged with one preference index, an integer between 1 and user_preference ("On which type of day is this appliance typically used?")

3. For each simulated day, RAMP randomly selects one daily preference *rand_daily_pref = random.randint(1, user_preference)* which represents today's behavioral mode, each appliance then checks if its pref_index matches today's preference → it may be used, if not → it is completely skipped for that day. Importantly: this happens before any stochastic timing or duration is simulated and appliances that are excluded do not consume any energy that day.

   Appliances with the same pref_index tend to appear together on the same days and disappear together on other days.

Appliance existence & deterministic constraints

This group of parameters defines whether an appliance exists, how strong it is, and under which high-level conditions it is allowed to operate, before any detailed stochastic timing is applied. They answer questions like: "How many of these appliances does each user have?", "How much power do they draw?", "Are they always active, sometimes active, or conditionally active?"

- **name**: Identifier only.
- **Number:** appliance multiplicity per user, used for coincident switch-on and power scaling. Each appliance type is treated as a set of identical units, when the appliance is active, the model decides how many of those units are on at the same time.
- **Power:** power random variable baseline. Types supported:
  - Constant → deterministic mean
  - Time series → deterministic seasonal forcing
  - Time series + thermal_p_var → stochastic perturbation: $P_t = \overline{P}_t \cdot U(1-\delta, 1+\delta)$
- **Flat**: Disable stochasticity. It's used to early exit in *generate_load_profile()* and, as effect, the appliance becomes a deterministic rectangular process over windows.
- **occasional_use**: Bernoulli trial per day. Where used: *if random.uniform(0,1) > occasional_use: return.* Statistics: $X_d \sim Bernoulli(p = occasional\ use)$. This is day-level thinning of the process.
- **pref_index:** preference matching. Appliance runs only if: pref_index=Pd. This is categorical conditioning, not probability scaling.

Note: differences between occasional use and user preferences

They operate at different conceptual levels and solve different modelling problems, even though both affect whether an appliance is active on a given day. In practice:

- occasional_use answers "*does this appliance appear today at all?*". It's a measure of *frequency of use* and models how often an appliance is used across days, acting independently for each appliance. It introduces *sporadic appearance*.
- user_preference answers "*which appliances tend to appear together on the same day?*". It's a measure of *type of day/activity pattern* and models which appliances belong to the same daily routine, acting jointly across multiple appliances. It introduces *correlated behavior*.

Example:

We want to model a rural household where some electricity uses happen every day and some uses happen only on specific types of days (these special days are considered mutually exclusive).

We identify three typical daily routines (each simulated day belongs to exactly one of these routines):

1. Baseline day. Normal weekday: lighting, phone charging, TV in the evening.

2. Cooking-intensive day. Extended cooking (e.g. market day, guests, celebrations).
3. Laundry day. Washing machine used once every few days.

User-level setting: user_preference = 3

Meaning: “This user has 3 mutually exclusive daily activity patterns”. Each day, RAMP will randomly pick one of these: rand_daily_pref ∈ {1, 2, 3}.

There are two possible approaches or patterns in appliances definition now:

Pattern 1: Mutually exclusive routines (regime switching)

Each appliance is assigned to one and only one routine.

- pref_index = 1 → used on baseline days
- pref_index = 2 → used on cooking-intensive days
- pref_index = 3 → used on laundry days

This creates clearly distinct daily profiles, useful for regime-based planning, peak stress testing, teaching demand diversity concepts. In other words, each day is a different type of day, appliances belong to that type of day and appliances do not overlap across types: each preference is a self-contained activity regime, not as an additive layer. This approach is NOT designed to say: “Some appliances are always used, others are sometimes added”. For that, you would need baseline duplication (Pattern 2).

Pattern 2: Baseline + conditional activities (layered behavior)

Often, we want something more realistic where some appliances are used every day while others are added only on specific days. This is achieved by combining pref_index = 0 with user_preference.

- pref_index = 0 → appliance is always active, regardless of routine
- pref_index > 0 → appliance is active only when the day matches

The resulting behavior is: every day (lights, phone, TV), cooking day (baseline + cooking appliances) and laundry day (baseline + washing machine), producing layered demand profiles, closer to real household behavior.

Time-allocation stochasticity

- **func_time**: Expected daily total ON time (minutes). Used in: *rand_total_time_of_use().* Statistical meaning: defines mean stopping time of the point process. Basically, RAMP decides how much total time the appliance should run today, it then places multiple switch-on events, each event consumes some minutes and once the total allocated time reaches func_time, the process stops. Total consumption stays stable, but the shape changes.
- **time_fraction_random_variability:** randomizes total daily duration $T_d \sim U\big(T, T \cdot (1+\delta)\big)$. This introduces day-to-day variance without changing expected value. This parameter controls day-to-day variability of total usage time: “some days you use the appliance a bit more, some days a bit less”. For instance, if func_time = 240 and time_fraction_random_variability = 0.1, the appliance is going to be used between 240–264 minutes.
- **func_cycle**: minimum duration of a single switch-on event (hard constraint). It ensures no switch-on shorter than func_cycle, enforcing granularity in the point process: “once this appliance turns on, it must stay on at least this long”.

Windows: temporal support of the process

- **num_windows**: number of separate (disjoint admissible) time intervals during which the appliance may be used.
- **window_1_start, window_1_end (and 2,3):** support of switch-on times meaning "The appliance is allowed to switch on only within this interval." Technically, it defines the support of the time random variable: $t_{on} \in \bigcup_k [a_k, b_k]$.
- **random_var_w:** random window deformation $a'_k = a_k + U(-\epsilon, +\epsilon)$, $b'_k = b_k + U(-\epsilon, +\epsilon)$. This prevents artificial synchronization across days. It represents how much the time window boundaries can shift, relative to the window length itself: "people don't follow the same schedule every day". Low value (0–0.2) → disciplined schedules (schools, offices), Medium value (0.2-0.4) → households, High value (>0.5) → informal, flexible routines.

Coincidence and peak behavior (population interaction)

- **fixed:** controls whether multiple identical appliances behave like one synchronized group or like many partially independent units, and RAMP increases simultaneity during peak hours using a simple calibrated probabilistic rule. Where used: *calc_coincident_switch_on()*. RAMP implements this by splitting the day into: *peak window and off-peak*. The peak window is computed globally from the "maximum theoretical profile" of all users (in UseCase.calc_peak_time_range()), then each switch-on event is checked: is it inside this peak range or not? Inside the peak window (and if fixed="no"), RAMP draws coincidence from a Gaussian and then clamps it between 1 and number (defaults are currently hard-coded in switch_on_parameters(): mu_peak = 0.5, s_peak = 0.5 meaning "during peak hours, typically about half of the units are on simultaneously, with some spread").

  - peak hours → Gaussian draw: $N_{on} \sim N(\mu N, \sigma N)\ \ where\ \ \mu = 0.5\,, \sigma = 0.5$

  Outside the peak window, RAMP reduces coincidence. It draws a random probability prob and map that probability to an integer number of appliances ON. In practice, the code implements a simple monotonic mapping rather than a carefully normalized probability law, where higher k values are less likely (especially off-peak)

  - Off-peak → Discrete uniform thinning: $P(N_{on} = k) \propto \frac{k}{N}$

  Off-peak coincidence is sampled from a uniform discrete distribution over a truncated set of possible coincidence levels, whose upper bound is reduced by op_factor which controls maximum possible number of identical appliances that can be on simultaneously outside peak hours (currently hard-coded op_factor=0.5). Technically, outside peak hours, RAMP samples the number of simultaneously operating identical appliances from a uniform discrete distribution over a truncated support, whose upper bound is reduced by the parameter op_factor. This suppresses full coincidence off-peak while preserving diversity.

Duty cycles (marked process extension)

A duty cycle is a way to represent appliances whose power is not constant while they are "on". Instead of "When it's on, it always consumes 100 W", you can represent "When it's on, it follows a pattern like 300 W for 10 minutes, then 100 W for 20 minutes" (or any other two-step pattern). Typical appliances that benefit from this approach are fridges/freezers (compressor cycles), electric cooking (high power bursts + simmer), washing machines (phases), pumps (start-up surge + steady) and battery charging / devices with phases.

In RAMP terminology, duty cycles are “marks” attached to switch-on events: once an event is placed, the load is filled using a cycle pattern rather than constant power.

- **fixed_cycle**: number of predefined duty cycle templates.
    - If fixed_cycle = 0 → No duty cycle. Switch-on events are filled with (possibly randomized) constant power.
    - fixed_cycle = 1 → use “cycle 1”
    - fixed_cycle = 2 → allow cycle 1 and cycle 2
    - fixed_cycle = 3 → allow cycle 1, 2, and 3

  Internally the appliance stores up to three cycle definitions as:

    - cycle 1: (p_11, t_11) then (p_12, t_12) with windows cw11, cw12 and randomness r_c1
    - cycle 2: (p_21, t_21) then (p_22, t_22) with windows cw21, cw22 and randomness r_c2
    - cycle 3: (p_31, t_31) then (p_32, t_32) with windows cw31, cw32 and randomness r_c3
- **p_ij, t_ij**: power and duration of each duty cycle segment (e.g. for t_11 minutes → power = p_11, then for t_12 minutes → power = p_12 etc.).
- **r_cj**: randomization factor for duty cycle durations $t'_{i,j} \sim U\left(t_{i,j}(1-\delta), t_{i,j}(1+\delta)\right)$
- **cwij_start, cwij_end**: cycle-specific admissible windows. They do not replace the main appliance window_1/2/3, instead, they decide which duty cycle is allowed given where the switch-on event happens. Technically, this is a conditional mark selection: $P(cycle = k \mid t_{on} \in cw_k)$: “If the event starts during this part of the day, use this cycle template”.
- **thermal_p_var**: applies multiplicative noise to power: $P \sim U\big(P(1-\delta), P(1+\delta)\big)$. Use it when appliance power depends on conditions like cooking intensity, heater depending on temperature, hot water usage duration/intensity.

Example A - Fridge / Freezer (compressor cycling)

Goal: represent compressor ON/OFF cycling. When the fridge runs, it doesn’t draw constant power: it alternates.

- fixed_cycle = 1
- p_11 = 80 W, t_11 = 15 min (compressor ON)
- p_12 = 5 W, t_12 = 30 min (idle / thermostat satisfied)
- r_c1 = 0.2 (cycle durations vary)
- thermal_p_var = 0.05 (ambient temperature effect)

Example B - Electric cooking (burst + simmer)

Goal: represent cooking sessions as high power followed by lower power.

- fixed_cycle = 1
- p_11 = 1500 W, t_11 = 10 min (boil / initial heating)
- p_12 = 400 W, t_12 = 20 min (simmer)
- r_c1 = 0.3 (cooking phase lengths vary)
- thermal_p_var = 0.15 (people cook differently)

Example C - Same appliance, different “modes” by time of day

Suppose a household uses the same cooker differently: breakfast with short high burst while dinner with longer cycle. Set:

- fixed_cycle = 2
- Cycle 1 (“breakfast”): p_11=1200, t_11=5, p_12=300, t_12=10, cw11/cw12 placed in the morning window (e.g., 06:00–09:00)
- Cycle 2 (“dinner”): p_21=1500, t_21=12, p_22=500, t_22=25, cw21/cw22 placed in the evening window (e.g., 17:00–21:00)

RAMP will pick which cycle to apply based on when the cooking event happens.

Building Year-Long Stochastic Demand Profiles with RAMP

What is RAMP?

RAMP is an open-source **stochastic generator of high-resolution electricity demand profiles**, designed to represent realistic user behaviour and variability in bottom-up energy system planning.

Rather than producing a single “average” load curve, RAMP generates **multiple plausible realizations of daily demand**, capturing uncertainty, diversity, and peak coincidence effects.

What does the core algorithm do?

At its core, RAMP simulates electricity demand as a hierarchical stochastic process:

- Users → statistically independent agents
- Appliances → stochastic processes owned by each user
- Time windows & cycles → constraints on when and how appliances operate
- Probabilistic rules → generate minute-level load realizations

For every simulated day, the model decides whether an appliance is used, how long it is used in total, when it is allowed to operate, and how many identical appliances operate simultaneously. Switch-on events are placed sequentially within admissible windows, without overlap, until the daily usage time is exhausted. During peak hours, coincidence between appliances is increased using a calibrated probabilistic model. Power consumption is then assigned either as constant power or via duty-cycle patterns.

Through its parameters, RAMP can represent many real-world behaviours: **user heterogeneity**, appliance ownership, **daily routines**, **occasional and sporadic uses**, **peak synchronization**, and realistic appliance operation. These behaviours are implemented using standard statistical mechanisms such as Bernoulli trials, uniform and Gaussian random variables, conditional probabilities, and Monte Carlo aggregation.

From daily simulations to a full year

Importantly, RAMP is not intended to produce a single deterministic year. Instead, it should be used as a **generator of pools of plausible daily demand profiles**. In a pooled, block-based approach, the user first defines a set of representative day types, for example by season or by weekday/weekend. RAMP is then run multiple times for each day type, generating a pool of realistic daily realizations at minute resolution. These realizations are then sampled and chained in coherent blocks, such as weeks or seasons, to reconstruct a full year that respects both calendar structure and behavioural consistency.

Recommended workflow:

1. Define representative day types (e.g. weekday vs weekend, dry vs rainy season, school term vs holidays)
2. Run RAMP multiple times per day type, generating pools of daily realizations (each run → one plausible day at 1-minute resolution)

3. Sample from these pools in coherent blocks (e.g. week-long or seasonal blocks, not single days)
4. Reconstruct a full year by chaining blocks according to: calendar structure, seasonal shares, planning assumptions.

The final outcome is a year-long demand time series that retains high temporal resolution, incorporates stochasticity in a controlled way, and is well suited for detailed energy system planning, including sizing, dispatch, and robustness analysis.