Silo Interest Rate Model

Anton Ponomarev, Alexey Egorov

September 2025

Table of Contents

Static Kink Model

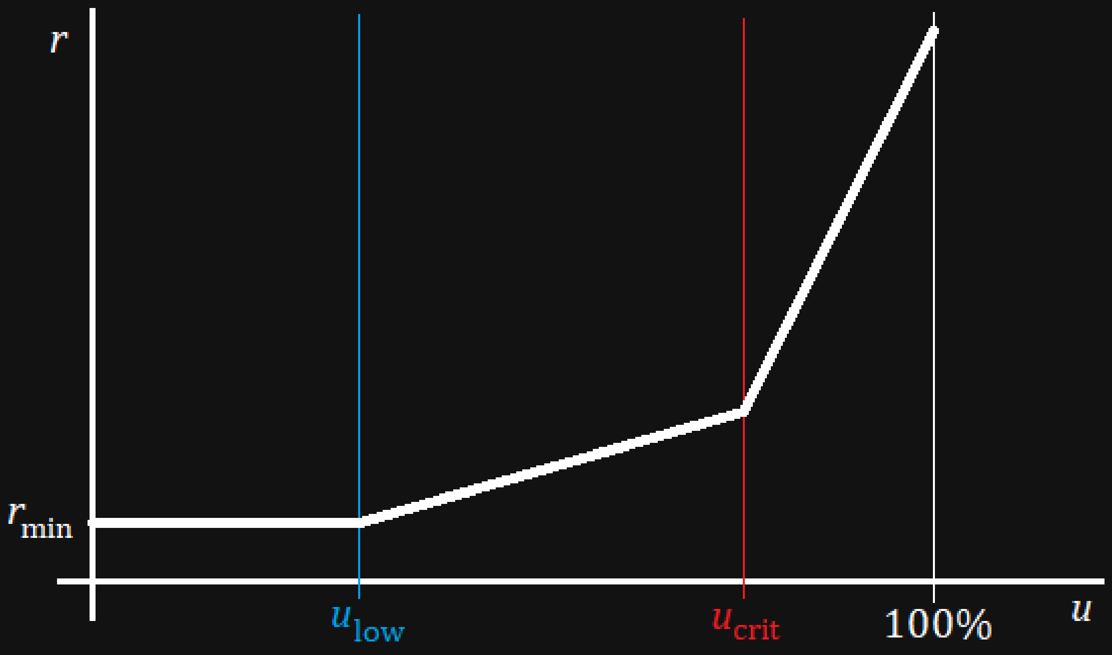
The static kink model defines a static relation between the utilization and APR. We consider 3 utilization intervals:

1. If utilization is below the low level u_{low}, APR is at a constant low level r_{min}.
2. If utilization is between u_{low} and the critically high level u_{crit}, APR is a linear function of u. The slope is k.
3. If utilization is above u_{crit}, the slope changes from k to αk, where $\alpha \geqslant 0$ is a high factor.

The formal description of the static kink model is

$$r = \begin{cases} r_{\text{min}}, & u < u_{\text{low}}, \\ r_{\text{min}} + k(u - u_{\text{low}}), & u_{\text{low}} \leq u < u_{\text{crit}}, \\ r_{\text{min}} + k(u - u_{\text{low}}) + \alpha k(u - u_{\text{crit}}), & u_{\text{crit}} \leq u. \end{cases} \tag{1}$$

The kink function is illustrated by the following diagram.



The traditional kink model (AAVE/Compound) is obtained from this one by setting $u_{\text{low}} = 0$. Then our critical value u_{crit} is called “optimal” in AAVE’s terminology.

Dynamic Kink Model

The dynamic kink model is suggested as a compromise between the static kink and the need for a dynamic model which would adapt to the changing market conditions. The PI model which is currently used may be too unconventional for the DeFi users. Thus, we propose a dynamic variation of the classical kink.

Basically, we use the same formula (1) as for the static kink model. The idea is to change the slope k depending on whether utilization is above or below an optimal interval $[u_1, u_2]$. In order to make the model less aggressive, we would like k to grow slowly and drop quickly. This is achieved by the asymmetrical integrator

$$\frac{\mathrm{d}k}{\mathrm{d}t} = \begin{cases} -c_1 - c_-(u_1 - u), & u < u_1, \\ 0, & u \in [u_1, u_2], \\ \min\{c_2 + c_+(u - u_2), d_{\max}\}, & u > u_2, \end{cases} \tag{2}$$

where $c_1, c_2, c_+, c_- \geq 0$. In addition, the following boundaries on k are imposed:

$$k_{\min} \leq k \leq k_{\max} \tag{3}$$

where $0 \leq k_{\min} \leq k_{\max}$.

Some points worth mentioning:

1. The static kink model is a special case of the dynamic kink with $c_1 = c_2 = c_+ = c_- = 0$. By replacing the current PI model with the dynamic kink we would have one model which smoothly blends the familiar static kink and its dynamic version.
2. Choosing a large c_1 it is possible to start reducing k aggressively as soon as utilization drops a hair below u_1.
3. The growth rate of k is bounded by $d_{\max}$.
4. From the borrower's point of view, at the time of transaction the APR would react predictably according to the kink curve and later would grow or fall at a linear rate.

Default Config Type

Here we describe a **default config type** – one type of model config which may be used as a default way to set up the model. Its parameters can be derived from a number of meaningful quantities describing the behavior of the model in user-friendly terms. This generates a less general config than can be created directly. For example, it does not cover the fixed APR model.

The user-friendly quantities are:

- u_{low}, u_{crit} – utilization at the kink points (see the figure),
- u_1, u_2 – boundaries of the optimal utilization interval (the model is static while utilization is in this interval),
- R_{min} – minimal APR, active below u_{low},
- $R_{\text{crit,min}}$ and $R_{\text{crit,max}}$ – minimal and maximal APR that the model can output at the critical utilization u_{crit},
- $R_{100\%,\text{max}}$ – maximal possible APR at 100% utilization,
- T_1 – time that it takes to drop from the maximal to the minimal APR at utilization u_1,
- T_2 – time that it takes to grow from the minimal to the maximal APR at utilization u_2,
- T_{low} – time that it takes to reset the model from the maximal to the minimal APR when utilization is u_{low},
- T_{crit} – time that it takes to grow from the minimal to the maximal APR at utilization u_{crit},
- T_{min} – minimal time it takes to grow from the minimal to the maximal APR at any utilization.

Parameters above should satisfy the following constraints:

- $0 \leqslant u_{\text{low}} < u_1 < u_2 < u_{\text{crit}} < 100\%$,
- $0 \leqslant R_{\text{min}} < R_{\text{crit,min}} \leqslant R_{\text{crit,max}} < R_{100\%,\text{max}}$,
- $\dfrac{R_{100\%,\text{max}} - R_{\text{min}}}{R_{\text{crit,max}} - R_{\text{min}}} \geqslant \dfrac{100\% - u_{\text{low}}}{u_{\text{crit}} - u_{\text{low}}}$,
- $0 < T_{\text{min}} \leqslant T_{\text{crit}} \leqslant T_2$,

- $0 < T_{\mathrm{low}} \leqslant T_1$.

Actual configuration parameters used by the algorithm are calculated using the following formulas where the constant $S = 365 \cdot 24 \cdot 3600$ is the number of seconds in a year.

$$r_{\mathrm{min}} = \frac{R_{\mathrm{min}}}{S} \tag{4}$$

$$k_{\mathrm{min}} = \frac{1}{S} \cdot \frac{R_{\mathrm{crit,min}} - R_{\mathrm{min}}}{u_{\mathrm{crit}} - u_{\mathrm{low}}} \tag{5}$$

$$k_{\mathrm{max}} = \frac{1}{S} \cdot \frac{R_{\mathrm{crit,max}} - R_{\mathrm{min}}}{u_{\mathrm{crit}} - u_{\mathrm{low}}} \tag{6}$$

$$\alpha = \frac{R_{100\%,\mathrm{max}} - R_{\mathrm{min}} - Sk_{\mathrm{max}}(100\% - u_{\mathrm{low}})}{Sk_{\mathrm{max}}(100\% - u_{\mathrm{crit}})} \tag{7}$$

$$c_1 = \frac{k_{\mathrm{max}} - k_{\mathrm{min}}}{T_1} \tag{8}$$

$$c_2 = \frac{k_{\mathrm{max}} - k_{\mathrm{min}}}{T_2} \tag{9}$$

$$c_- = \frac{1}{u_1 - u_{\mathrm{low}}} \left(\frac{k_{\mathrm{max}} - k_{\mathrm{min}}}{T_{\mathrm{low}}} - c_1 \right) \tag{10}$$

$$c_+ = \frac{1}{u_{\mathrm{crit}} - u_2} \left(\frac{k_{\mathrm{max}} - k_{\mathrm{min}}}{T_{\mathrm{crit}}} - c_2 \right) \tag{11}$$

$$d_{\mathrm{max}} = \frac{k_{\mathrm{max}} - k_{\mathrm{min}}}{T_{\mathrm{min}}} \tag{12}$$

Pseudocode

```
function GenerateConfig(
    uint256 T1,
    uint256 T2,
    uint256 Tlow,
    uint256 Tcrit,
    uint256 Tmin,
    uint256 u1,
    uint256 u2,
    uint256 ulow,
    uint256 ucrit,
    uint256 Rmin,
    uint256 RcritMin,
    uint256 RcritMax,
    uint256 R100
) returns (
    uint256 rmin,
    uint256 kmin,
    uint256 kmax,
    uint256 alpha,
    uint256 c1,
    uint256 c2,
    uint256 cminus,
    uint256 cplus,
    uint256 dmax
) {
  // 0 <= ulow < u1 < u2 < ucrit < DP
  require(u1.inOpenInterval(ulow, u2));
  require(ucrit.inOpenInterval(u2, DP));

  // 0 <= Rmin < RcritMin <= RcritMax < R100
  require(RMin < RcritMin);
  require(RcritMax.inOpenIntervalLowIncluded(RcritMin, R100));

  uint256 s = 365 days;

  // 0 < Tmin <= Tcrit <= T2 < 100y
  require(Tmin.inOpenIntervalTopIncluded(0, Tcrit));
  require(T2.inOpenIntervalLowIncluded(Tcrit, 100 * s));

  // 0 < Tlow <= T1 < 100y
  require(Tlow != 0);
  require(T1.inOpenIntervalLowIncluded(Tlow, 100 * s));

  // (R100 - Rmin) / (RcritMax - Rmin)
  //     >= (DP - ulow) / (ucrit - ulow)
  uint256 rCheckHi = (R100 - Rmin) * DP / (RcritMax - Rmin);
  uint256 rCheckLo = (DP - ulow) * DP / (ucrit - ulow);
  require(rCheckHi >= rCheckLo);

  rmin = Rmin / s;

  kmin = (RcritMin - Rmin) * DP / (ucrit - ulow) / s;
  kmax = (RcritMax - Rmin) * DP / (ucrit - ulow) / s;

  alpha = (rCheckHi * (ucrit - ulow) - (DP - ulow) * DP) ...
                                                  / (DP - ucrit);

  c1 = (kmax - kmin) / T1;
  c2 = (kmax - kmin) / T2;

  cMinus = ((kmax - kmin) / Tlow - c1) * DP / (u1 - ulow);
  cPlus = ((kmax - kmin) / Tcrit - c2) * DP / (ucrit - u2);

  dmax = (kmax - kmin) / Tmin;
}
```

Compound Interest

To calculate the borrowers' interest accumulated during a period of no activity (no new loans and no new deposits), we use the idea of continuous compounding.

What is continuous compounding?

In the *discrete* compounding model, interest is compounded over time intervals $\triangle t$, so a quantity x is updated according to the formula

$$x(t + \triangle t) = x(t)\big(1 + r(t)\triangle t\big),$$

where r is the instantaneous interest rate. Thus, the relation is

$$\frac{x(t + \triangle t) - x(t)}{\triangle t} = r(t)x(t).$$

To transition to continuous compounding, we take the limit $\triangle t \to 0$ which results in the differential equation

$$\dot{x}(t) = r(t)x(t),$$

whose solution over the time interval $[t_0, t_1]$ is

$$x(t_1) = x(t_0) \exp\left(\int_{t_0}^{t_1} r(t)\, dt\right).$$

Hence, the interest rate compounded over $[t_0, t_1]$ is

$$r_{\text{comp}} = \exp\left(\int_{t_0}^{t_1} r(t)\, dt\right) - 1. \tag{13}$$

How do we calculate the compound interest in practice?

To simplify the calculation of the integral in (13), let us postulate that the utilization ratio stays constant during the period of no activity:

$$u(t) = u(t_0), \quad t \in [t_0, t_1].$$

Under this assumption, we observe that the borrowing APR is changing linearly (and may possibly hit the lower bound):

$$r(t + \triangle t) = \max\left\{r(t) + \sigma \triangle t,\ r_{\text{lin}}\right\},$$

where $r_{\text{lin}} = k_{\text{lin}} u(t_0) + r_{\text{min}}$ and the slope σ is determined as

$$\sigma = k_{\text{i}}\big(u(t_0) - u_{\text{opt}}\big)$$

due to the integrator dynamics.

Given the initial value of the integrator $r_{\text{i}}(t_0)$ and of the proportional term

$$r_{\text{p}}(t_0) = \begin{cases} k_{\text{low}}\big(u(t_0) - u_{\text{low}}\big), & \text{if } u(t_0) < u_{\text{low}}, \\ 0, & \text{otherwise}, \end{cases}$$

let us introduce the values

$$\begin{aligned} r_0 &= r_{\text{i}}(t_0) + r_{\text{p}}(t_0), \\ r_1 &= r_0 + \sigma(t_1 - t_0) \end{aligned}$$

which represent the initial and final values of the current interest rate ignoring the lower bound r_{lin}. Now, in order to compute the integral in the compounding formula (13), we just have to consider four cases:

- If $r_0 \geqslant r_{\text{lin}}$ and $r_1 \geqslant r_{\text{lin}}$ then the lower bound is not activated over the time period, so $r(t) = r_0 + \sigma(t - t_0)$ is a linear function whose integral is

$$\int_{t_0}^{t_1} r(t)\, dt = \frac{r_0 + r_1}{2}(t_1 - t_0).$$

- If $r_0 < r_{\text{lin}}$ and $r_1 < r_{\text{lin}}$ then the lower bound is active the whole time, so $r(t) \equiv r_{\text{lin}}$ and

$$\int_{t_0}^{t_1} r(t)\, dt = r_{\text{lin}}(t_1 - t_0).$$

- If $r_0 \geqslant r_{\text{lin}}$ and $r_1 < r_{\text{lin}}$ then the interest rate starts above the lower bound and drops onto it at some point τ which can be found from the relation $r_0 + \sigma(\tau - t_0) = r_{\text{lin}}$. We then calculate the integral as

$$\int_{t_0}^{t_1} r(t)\,dt = \int_{t_0}^{\tau} + \int_{\tau}^{t_1} = \frac{r_0 + r_{\text{lin}}}{2}(\tau - t_0) + r_{\text{lin}}(t_1 - \tau) = r_{\text{lin}}(t_1 - t_0) - \frac{(r_0 - r_{\text{lin}})^2}{2\sigma}.$$

- If $r_0 < r_{\text{lin}}$ and $r_1 \geqslant r_{\text{lin}}$ then the interest rate starts at the lower bound and overcomes it at some point. This case is similar to the previous one. The result is

$$\int_{t_0}^{t_1} r(t)\,dt = r_{\text{lin}}(t_1 - t_0) + \frac{(r_1 - r_{\text{lin}})^2}{2\sigma}.$$

Once the integral is calculated, equation (13) yields the interest rate compounded over the time interval from t_0 till t_1. There are algorithms available to compute the exponent in (13) with sufficient precision and efficiency. One such algorithm is implemented in the library PRBMath.

Limits to Avoid Overflow

Caps

1. The current interest rate r_{cur} is capped by $1\,000\%$ APR. In the pseudocode, this is specified by the variable `RCUR_CAP = 10 * DP` which accounts for `DP` decimals.
2. The compounded interest rate r_{comp} (since the last transaction) is capped by the linear limit $1\,000\% \cdot T/(365 \cdot 24 \cdot 3600)$ where T is time in seconds since the last transaction. The rate of growth of this limit is the constant `RCOMP_CAP`.

If a cap is reached,

- the corresponding variable assumes the cap value;
- the model is reset ($k := k_{\text{min}}$).

Overflow trigger

- The exponent x in the calculation of the compounded interest rate $r_{\text{comp}} = e^x - 1$ is limited by the number 11 because $e^{11} - 1 \lesssim 2^{16}$. Compounded interest rate larger than around $2^{16} = 6\,553\,600\%$ is deemed high enough to stop compounding it. This limit is set by `X_MAX`.

Pseudocode

Algorithm parameters

Basic constants of the dynamic kink model:

- $u_{\text{low}} \in [0, 1]$ – threshold of low utilization;
- $u_1 \in [0, 1]$ – lower bound of optimal utilization range;
- $u_2 \in [u_1, 1]$ – upper bound of optimal utilization range;
- $u_{\text{crit}} \in [u_{\text{low}}, 1]$ – threshold of critical utilization;
- $r_{\text{min}} \geqslant 0$ – minimal per-second interest rate;
- $k_{\text{min}} \geqslant 0$ – minimal slope k of central segment of the kink;
- $k_{\text{max}} \geqslant k_{\text{min}}$ – maximal slope k of central segment of the kink;
- $\alpha \geqslant 0$ – factor for the slope for the critical segment of the kink;
- $c_{-} \geqslant 0$ – coefficient of decrease of the slope k;
- $c_{+} \geqslant 0$ – growth coefficient of the slope k;
- $c_1 \geqslant 0$ – minimal rate of decrease of the slope k;
- $c_2 \geqslant 0$ – minimal growth rate of the slope k;
- $d_{\text{max}} \geqslant c_2$ – maximal growth rate of the slope k.

Remark: The following code assumes that the interest rate is per-second rather than annual.

We use 18-digit precision to store floating point variables (utilization, interest rate, and model parameters). This is implemented using the additional constant:

- `DP = 1e18` – a multiplier to reduce a floating point number with 18 decimal places to an integer.

Initialization

- `k =` k_{min}

Calculation of the current interest rate

```
const RCUR_CAP = 10 * DP; // 1,000% APR
const ONE_YEAR = 365 days;

function CurrentInterestRate(
    uint t0,    // time of last transaction already written in the
                //                         blockchain (in seconds)
    uint t1,    // current time (in seconds)
    uint _k,    // state of the slope k at time t0
    uint u,     // utilization at time t0
    uint tba    // total borrow amount
) returns (
    uint rcur   // current annual interest rate shown to the user
) {
  if (tba == 0) {
    return 0;
  }

  uint T = t1 - t0;  // length of time period (in seconds)

  int k = _k;        // slope of the central segment of the kink

  if (u < u1) {
    k = max(k - (c1 + cminus * (u1 - u) / DP) * T, kmin);
  } else if (u > u2) {
    k = min(k + min(c2 + cplus * (u - u2) / DP, dmax) * T, kmax);
  }

  uint excessU = 0;        // additional interest rate
  if (u >= ulow) {
    excessU = u - ulow;
    if (u >= ucrit) {
      excessU = excessU + alpha * (u - ucrit) / DP;
    }
  }

  rcur = min(excessU * k * ONE_YEAR / DP ...
                                 + rmin * ONE_YEAR, RCUR_CAP);
}
```

Calculation of the compound interest

```
const RCOMP_CAP = RCUR_CAP / ONE_YEAR; // 1,000% APR in per-second
const X_MAX = 11 * DP;

function CompoundInterestRate(
    uint t0,    // time of last transaction already written in the
                //                         blockchain (in seconds)
    uint t1,    // current time (in seconds)
    uint _k,    // state of the slope k at time t0
    uint u,     // utilization at time t0
    uint tba    // total borrow amount
) returns (
    uint rcomp, // compounded interest rate
    uint k      // updated state of the slope
) {
  if (_tba == 0) {
    return (0, _k);
  }

  require(t0 <= t1);

  uint T = t1 - t0;        // length of time period (in seconds)

  if (T == 0) {
    return (0, _k);
  }

  int roc = 0;             // rate of change of k
  if (u < u1) {
    roc = - c1 - cminus * (u1 - u) / DP;
  } else if (u > u2) {
    roc = min(c2 + cplus * (u - u2) / DP, dmax);
  }

  int k1 = _k + roc * T;   // slope of the kink at t1 ignoring
                           //            lower and upper bounds

  uint x;                  // an integral of k
  if (k1 > kmax) {
    x = kmax * T - (kmax - _k)**2 / (2 * roc);
    k = kmax;
  } else if (k1 < kmin) {
    x = kmin * T - (_k - kmin)**2 / (2 * roc);
    k = kmin;
  } else {
    x = (_k + k1) * T / 2;
    k = k1;
  }

  uint f = 0;                 // factor for the slope in kink
  if (u >= ulow) {
    f = u - ulow;
    if (u >= ucrit) {
      f = f + alpha * (u - ucrit) / DP;
    }
  }

  x = rmin * T + f * x / DP;

  //
  // Overflow Checks
  //

  require (x <= X_MAX);

  rcomp = exp(x) - DP;

  if (rcomp > RCOMP_CAP * T) {
    rcomp = RCOMP_CAP * T;
    k = kmin;
  }
}
```