

Canadian semi-annual mortgages



MortgageMath
Precise Loan Amortization in Python

mortgagemath 0.5.2 · rendered 2026-05-02

The Interest Act §6 convention

Canadian residential mortgages quote rates as **semi-annually compounded** (j_2), per Section 6 of the federal *Interest Act*. A US “5% / 30yr” loan and a Canadian “5% / 30yr” loan have different periodic rates — and different monthly payments — even though both quote 5%.

i Note

US: monthly periodic rate is $r/12$ directly. A 5% loan uses $5/1200 = 0.004166\dots$ per month.

Canadian (j_2): the quoted j_2 is per year compounded semi-annually. The equivalent monthly periodic rate is

$$r_{\text{monthly}} = \left(1 + \frac{j_2}{200}\right)^{2/12} - 1$$

For $j_2 = 5\%$, this is $(1.025)^{1/6} - 1 \approx 0.41239\%$, **not** $5/12 \approx 0.4167\%$.

How the library models it

Pass `compounding=Compounding.SEMI_ANNUAL` and a `payment_frequency`. The same closed-form annuity formula then uses the derived periodic rate for both the closed-form payment computation and the per-row interest accrual.

```
from decimal import Decimal
from mortgagemath import (
    LoanParams, Compounding, PaymentFrequency,
    PaymentRounding, periodic_payment, amortization_schedule,
)

# Olivier's "Chans" first term: $350,100 / j_2=4.9% / 3yr term on
# 20yr amortization, monthly payments. At end of the 3-year fixed
# term, the unpaid balance is the balloon to be renewed at then-
# prevailing rates.
chans = LoanParams(
    principal=Decimal("350100"),
    annual_rate=Decimal("4.9"),
    term_months=36,
    amortization_period_months=240,
    compounding=Compounding.SEMI_ANNUAL,
    payment_frequency=PaymentFrequency.MONTHLY,
    payment_rounding=PaymentRounding.ROUND_HALF_UP,
    interest_rounding=PaymentRounding.ROUND_HALF_UP,
)

print(f"Monthly P&I:                ${periodic_payment(chans):,}")
```

```

sched = amortization_schedule(chans)
print(f"Balance after pmt 36 (balloon to renew): ${sched[36].balance:,}")

```

Monthly P&I: \$2,281.73
Balance after pmt 36 (balloon to renew): \$316,593.49

Both anchors reproduce **Olivier *Business Math* §13.4** to the cent (\$2,281.73 and \$316,593.49).

Quarterly cadence (eCampus 4.4.1)

Canadian semi-annual compounding combines naturally with non-monthly payment cadences. The eCampus Ontario *Mathematics of Finance* §4.4.1 example uses **quarterly** payments:

```

ecampus = LoanParams(
    principal=Decimal("297500"),
    annual_rate=Decimal("3.8"),
    term_months=36,
    amortization_period_months=240,
    compounding=Compounding.SEMI_ANNUAL,
    payment_frequency=PaymentFrequency.QUARTERLY,
    payment_rounding=PaymentRounding.ROUND_HALF_UP,
    interest_rounding=PaymentRounding.ROUND_HALF_UP,
)
print(f"Quarterly P&I:                      ${periodic_payment(ecampus):,}")
sched = amortization_schedule(ecampus)
print(f"Balance after 12 quarters:    ${sched[12].balance:,}")

```

Quarterly P&I: \$5,317.62
Balance after 12 quarters: \$265,830.61

Both anchors reproduce eCampus's published values (\$5,317.62 and \$265,830.61) to the cent — and the schedule walks 12 quarterly rows, not 12 monthly rows, because $\text{term_months} \times \text{payments_per_year} / 12 = 36 \times 4 / 12 = 12$ payments.

Why this matters

Without `Compounding.SEMI_ANNUAL`, a Canadian-style mortgage calculator written in stdlib Python will quote a payment **about 0.7% high** on a typical 25-year loan — a few dollars per month, but several thousand dollars over the loan life. Most Python amortization libraries silently get this wrong.

`mortgagemath` exposes the convention as a single enum value, derives the periodic rate via Decimal arithmetic to 50 digits of internal precision, and validates against multiple independent published sources (Olivier's textbook, eCampus Ontario's open-licensed online text). The convention also extends to **biweekly** and **weekly** Canadian payment cadences (`PaymentFrequency.BIWEEKLY`, `PaymentFrequency.WEEKLY` — both 26/yr and 52/yr respectively).