Fixed Income Derivatives: Market Practices

Patrick S. Hagan
Mathematics Institute

pathagan1954@yahoo.com

Money

- Money exists only as an electronic entry in the banking system
– paper money (scrip) isn't money → *doesn't earn interest*

- Dollars exist *only* in US banking system
– pounds exist only in British banking system
– Euros only exist in European banking system

- Every bank accepting US dollars has a *Nostro* account in its *correspondent* bank in the US
– often a US subsidiary
– dollars on deposit actually reside in correspondent bank

- Transfer 1 US$ in London from Sanwa to Barclays
– Sanwa instructs its correspondent bank to send dollar to Barclays' correspondent bank
– dollar never leaves the US

- *payments can only be made when ccy's banking system is up*
– good USD business days: Federal wire system running
– good Euro business days: Target settlement system running

Date arithmetic

- Dates are integers, usually following Excel
  – May 8, 2006 is 38845; May 9, 2006 is 38846, ...

- Add function

$$t_{th} = Add(t, n, unit, EOMFlag, \ldots)$$

– adds n units (days, months, years, or business_days) to t
– n can be positive, negative, or zero

- Month end rules:

(1) if adding months or years ends up beyond the end of month, replace with last day of the month

May 31 + 1m is Jun 30
Mar 30 – 1m is Feb 28 (or 29)

(2) if initial date t is last day of month then

if *EOMFlag* is *true*: adding months or years always yields last day of month

Feb 29 + 1m is Mar 31
Apr 30 – 1m is Mar 31

if *EOMFlag* is *false*: adding months or years always yields the same day of month if it exists

Feb 29 + 1m is Mar 29
Apr 30 – 1m is Mar 30
May 31 + 1m is Jun 30

– *EOMFlag* is defaults to *false* for swaps

Business day rule

- Full add functionality:

$$t_{act} = Add(t, n, unit, EOMFlag, BDR, hol1, hol2, \dots)$$

– first computes

$$t_{th} = Add(t, n, unit, EOMFlag, \dots)$$

– if t_{th} is a bad bus. day, applies the business day rule *bdr*
– t_{th} is a bad day if it is a bad day in any of the holiday centers

- Five business day rules:

none: t_{act} is the theoretical date t_{th}
not appropriate for payments

following (succeeding): t_{act} is the first good day on or after t_{th}

preceding: t_{act} is the last good day on or before t_{th}
translation of legal phrase on or before

modified following (mf): t_{act} is the first good day on or after t_{th}, provided it is in the same calendar month. Otherwise return the first good day that's before t_{th}
standard bus day rule for nearly all payments

modified preceding (mp): t_{act} is the last good day on or before t_{th}, provided it is in the same calendar month. Otherwise return the first good day that's after t_{th}
never used

Leg generation

$$t_{act} = Add(t, n, unit, EOMFlag, BDR, hol1, hol2, \dots)$$

- Leg dates are generated backwards from theoretical end date
  - for a M month leg, the theoretical end date is

$$t_n^{th} = Add(t,\ \mathrm{M}, \mathrm{month}, EOMFlag, \mathrm{none}, \dots)$$

  - with m months per period, the theoretical and actual dates are

$$t_j^{th} = Add(t_n^{th}, -m(n-j), \mathrm{month}, EOMFlag, \mathrm{none}, hol1, hol2, \dots)$$

$$t_j^{act} = Add(t_n^{th}, -m(n-j), \mathrm{month}, EOMFlag, BDR, hol1, hol2, \dots)$$

  - if odd period is needed, default is *short first period*
    also called *stub up front*
  - other possibilities are *long first, short last, long last*
    last two require generating dates forward from t_0

- Holiday centers ($hol1, hol2, \dots$)
  - must include holiday center for payment ccy
    USD = NYB, GBP = LDB, ...
  - floating legs *also* have holiday center for fixing country
    LDB for Libor, TKB for Tibor, SPD for Sibor, ...

Interest payments

• Interest accrues from t_{j-1} to t_j and is paid on t_j

– if swap leg is *adjusted*, interest accrues from t_{j-1}^{act} to t_j^{act}
default for swap markets
– if swap leg is *unadjusted*, interest accrues from t_{j-1}^{th} to t_j^{th}

– payments are

$$\alpha_j \cdot R \cdot N \text{ paid on } t_j^{act} \text{ for } j = 1, 2, \ldots, n$$

Here N = notional, R = rate, and α_j = day count fraction,

$$\alpha_j = DCFrac(t_{j-1}, t_j, \text{ basis})$$

• Day count basis is a rule assigning an official fraction of a year for any two dates

– *Act360* (aka *money market basis*): $\alpha = (t_2 - t_1)/360$
used for most floating legs, credit cards, ...

– *Act365*: $\alpha = (t_2 - t_1)/365$
used for both legs in Britain and its former colonies

– *30360* (aka *bond basis*): $\alpha = numerator/360$, where

numerator = 30 * number of complete months + leftover days

used for most fixed legs

– over 80 day count bases listed in Bloomberg

Day count algorithms

```
d1 = wcGetDay(t1),          d2 = wcGetDay(t2);
m1 = wcGetMonth(t1),        m2 = wcGetMonth(t2);
y1 = wcGetYear(t1),         y2 = wcGetYear(t2);
isLeap1 = wcGetDay(t1),     isLeap2 = wcGetDay(t2);

case wc30360:                          30360 ISDA (standard)
    if (d1==31) d1=30;
    if (d2==31 && d1==30) d2 = 30;
        daycount = (d2 - d1) + 30*(m2 - m1) + 360*(y2 - y1);
        return  daycount / 360;

case wc30360PSA:                       30360 Pub Sec Admin
    if (d1 ==31 ||
        (m1 == 2 && d1 == DaysInMonth[IsLeap][2])
        d1 = 30;
    if (d2==31 && d1==30) d2 = 30;
        daycount = (d2 - d1) + 30*(m2 - m1) + 360*(y2 - y1);
        return  daycount / 360;

case wc30360E:                         30360 for most of Europe
    if (d1==31) d1=30;
    if (d2==31) d2 = 30;
        daycount = (d2 - d1) + 30*(m2 - m1) + 360*(y2 - y1);
        return  daycount / 360;

case wc30360F:                         30360 for France
    et cetera
```