Dokumentacja

Funkcje kalendarzowe

Mateusz Jabłoński Agnieszka Kaszkowiak

Spis treści

1. Oryginalny opis zadania

Construction of the calendar. Financial transactions are settled for working days. Working days are specific for every market, i.e. currency of denomination and geographical location – PLN denominated instruments are traded in Warsaw, but also in London, and USD denominated instruments are traded in New York, London, Tokyo etc. and all these are different markets. To calculate the price of an instrument which contains many cash flows in different time moments we have to know exact moments of these cash flows. Usually instruments are defined by some general rules, say the payment days for a bond coupons come every 6 months from the date of issue. Knowing the issue date of a bond, we can calculate dates for coupon payments. These days have to be working days. When a certain date from this list is not a working day we have to use a rule specific for a given market saying which working day to use as the coupon payment date. But then come the settlement date or value date, the date in which the actual cash flow takes place. This also has to be a working day and again a specific rule says how to calculate the value date knowing the payment date. Implementing all these rules even for a single instrument on a given market is quite complicated. We have to implement the following functions (to simplify the project for a single market only – the Polish market is a first choice):

a) Construction of a standard calendar function (or functions). This function has to perform following operations:

   i) given a fixed date, calculate which day of the week is this date,

   ii) how many days are from a fixed date to any other specific date,

   iii) given a fixed date supply the date distant by n days.

b) Construction of a ”working days” function. This function has to be specific for a given market. It has to recognize holidays: Saturdays and Sundays, but also state holidays (specific for a country) and days which are financial market non-working days (”Bank holidays”), say Christmas Eve, Good Friday (in countries in which it is not state holiday) or May, 2 in Poland, i.e. days in which a local stock exchange is not working.

c) Implementation of the day-count-system – the rule which says how many days a year has and how a month is defined. This rule is specific for a market and for an instrument, i.e. on the same market two instruments can be constructed with different day-count-systems.

d) Implementation of the rule which defines how the payment date is modified if it is not a working day.

e) Implementation of the rule defining how many days from the payment date is the value date.

a. Opis zadania a zakres projektu:

Oryginalne zadania	Funkcje w programie	Opis funkcji
a) Construction of a standard calendar function (or functions). This function has to perform following operations:		
i) given a fixed date, calculate which day of the week is this date,	**weekday**(date)	Wbudowana funkcja Octave'a służąca do identyfikacji dnia tygodnia
ii) how many days are from a fixed date to any other specific date,	**day_diff**(date1,date2,basis)	Podstawowa funkcja licząca różnicę (w dniach) między dwoma datami. Nie są dokonywane żadne operacje przesunięcia do dni roboczych
iii) given a fixed date supply the date distant by n days.	**add_days**(date1, n)	Funkcja przesuwająca datę o n dni. Nie są dokonywane żadne operacje przesunięcia do dni roboczych
	add_months(date1, mwect,x)	Funkcja tworząca ciąg dat na podstawie daty początkowej i wektora długości okresów w miesiącach rozumianych nominalnie
b) Construction of a ”working days” function. This function has to be specific for a given market. It has to recognize holidays: Saturdays and Sundays, but also	**easter**(year,stock)	Funkcja pomocnicza wykorzystywana przy zadaniach związanych z dniami wolnymi

state holidays (specific for a country) and days which are financial market non-working days (”Bank holidays”), say Christmas Eve, Good Friday (in countries in which it is not state holiday) or May, 2 in Poland, i.e. days in which a local stock exchange is not working.		
	is_business_day (date,stock)	Główna funkcja sprawdzająca, czy podana data jest dniem aktywności wybranej giełdy
c) Implementation of the rule which defines how the payment date is modified if it is not a working day.	**day_shift**(date1, stock,x)	Funkcja pomocnicza przydatna w bardziej złożonych procedurach, szczególnie przy zagadnieniach związanych z konwencjami dni i płatności
	day_shift2(date 1,stock,n)	Funkcja pomocnicza przydatna w bardziej złożonych procedurach, szczególnie przy zagadnieniach związanych z value date. Przesuwa podaną datę o n dni roboczych do przodu
	mod_day_shift(date1,stock,x)	Funkcja pomocnicza wykorzystywana w date_rolling(). Dla podanej daty odnajduje najbliższy następny/poprzedni (x = 1 / x = -1) dzień roboczy w sensie konwencji Modified Following/Previous Business Day
d) Implementation of the rule defining how many days	**date_rolling**(dat	Podstawowa funkcja odnajdująca rzeczywisty dzień płatności (bez przesunięcia związanego z value date)

from the payment date is the value date.	e1,stock,conv)	
e) Implementation of the day-count-system – the rule which says how many days a year has and how a month is defined. This rule is specific for a market and for an instrument, i.e. on the same market two instruments can be constructed with different day-count-systems.	**act_act_frac**(date1,date2)	Funkcja pomocnicza wykorzystywana w year_frac(). Zwraca frakcję roku dla konwencji ACT/ACT.
	year_frac(date1, date2, basis)	Główna funkcja wyznaczająca frakcję roku.
	payments(date1,nwect,stock,basis,conv,valuedays)	Funkcja ułatwiająca korzystanie z poprzednich procedur w przypadku ciągu płatności, w tej wersji tworząca szereg dat w następujący sposób: • do date1 dodaje nwect(1) dni i otrzymuje date2, na którą działa funkcją date_rolling(), a następnie przesuwa o valuedays dni roboczych do przodu (tzn. odnajduje rzeczywisty dzień płatności) • do date1 dodaje nwect(1)+nwect(2) i otrzymuje date3, na którą działa funkcją date_rolling(), a następnie przesuwa o valuedays dni roboczych do przodu, ...itd.,

		• w efekcie uzyskuje ciąg length(nwect)+1 dat.
	payments2(dates,stock,basis,conv,valuedays)	Funkcja ułatwiająca korzystanie z poprzednich procedur w przypadku ciągu płatności, w tej wersji pobierająca explicite listę z datami, działająca na nie date_rolling(), na końcu przesuwająca o valuedays dni roboczych do przodu.
	payments3(date1,n,d,stock,basis,conv,valuedays)	Modyfikacja payments(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora dni pobiera liczbę okresów oraz (taką samą) ilość dni w każdym z nich
	payments4(date1,mwect,stock,basis,conv,valuedays,x)	Modyfikacja payments(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora dni pobiera wektor miesięcy. Przesunięcie następuje w rozumieniu zasady działania add_months().
	payments5(date1,n,m,stock,basis,conv,valuedays,x)	Modyfikacja payments4(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora miesięcy pobiera liczbę okresów oraz (taką samą) ilość miesięcy w każdym z nich. Przesunięcie następuje w rozumieniu zasady działania add_months().

2. Zagadnienia wprowadzające

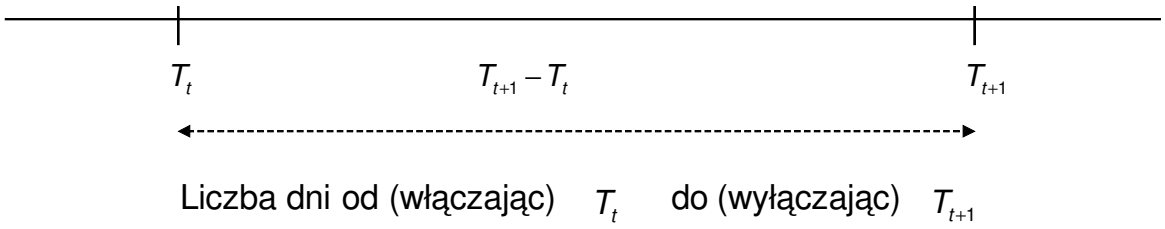
a. Baza odsetkowa

Baza odsetkowa określa umowną liczbę dni w roku dla celów obliczeniowych. Zależy od określonej konwencji dni

Konwencja dni	Liczba dni w bazie
30/360	360
30E/360	360
ACT/360	360
ACT/ACT	366 dla roku przestępnego, w przeciwnym przypadku 365
ACT/365	365

b. Konwencja dni

Konwencja dni jest to metoda, dzięki której można policzyć frakcję roku pomiędzy dwiema danymi datami.



Liczba dni od (włączając) T_t do (wyłączając) T_{t+1}

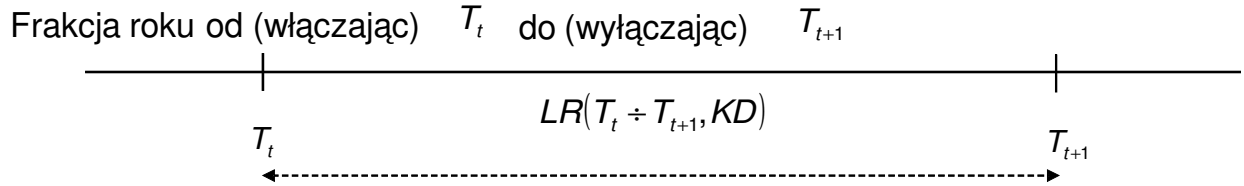
Niech: D_t M_t R_t oznacza dzień, miesiąc i rok dla daty T_t

Niech: D_{t+1} M_{t+1} R_{t+1} oznacza dzień, miesiąc i rok dla daty T_{t+1}

Niech: KR_i oznacza ostatni dzień kalendarzowy roku R_i

Na rynku międzybankowym można wyróżnić pięć podstawowych konwencji dni:

- Act/365 (rzeczywista liczba dni w okresie podzielona przez bazę 365 dni w roku)
- Act/360 (rzeczywista liczba dni w okresie podzielona przez bazę 360 dni w roku)
- Act/Act (rzeczywista liczba dni w okresie podzielona przez bazę 365 lub 366 dni w roku)
- 30/360 (liczba dni w okresie przy założeniu, że każdy miesiąc zawiera 30 dni, podzielona przez bazę 360 dni w roku
- 30E/360 (zmodyfikowana konwencja 30/360)



Definicje poszczególnych konwencji dni zobrazowane będą następującym przykładem obliczenia frakcji roku pomiędzy:

30 listopada 2002 - 29 lutego 2004

Oznaczenia przybiorą wówczas postać:

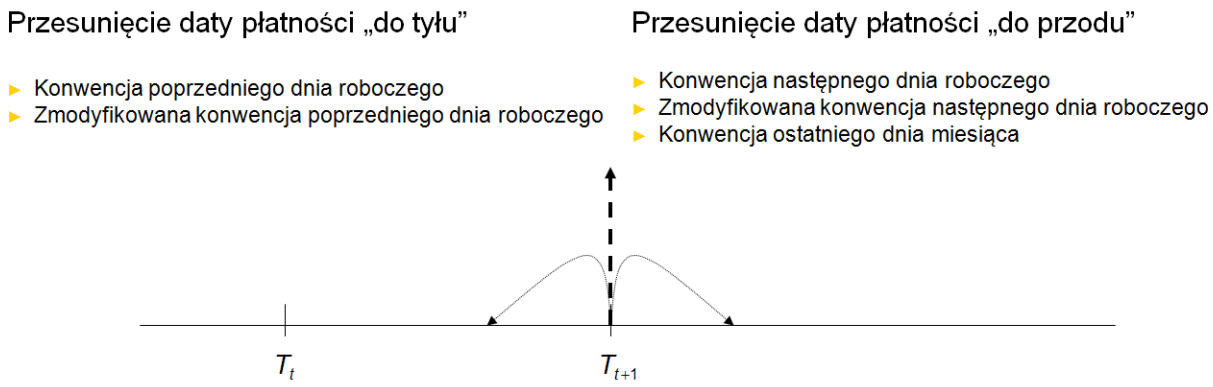
T_t	30 listopada 2002, roku nieprzestępnego
D_t	30, dzień dla daty: 30 listopada 2002
M_t	11, miesiąc dla daty: 30 listopada 2002
R_t	2002, rok dla daty: 30 listopada 2002
T_{t+1}	29 lutego 2004, roku przestępnego
D_{t+1}	29, dzień dla daty: 29 lutego 2004
M_{t+1}	02, miesiąc dla daty: 29 lutego 2004
R_{t+1}	2004, rok dla daty: 29 lutego 2004
KR_{2002}	31 grudnia 2002, ostatni dzień kalendarzowy roku 2002
KR_{2003}	31 grudnia 2003, ostatni dzień kalendarzowy roku 2003

Konwencja	**Przykład obliczeń**
Act/365	Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} $LR(T_t \div T_{t+1}, Act/365) = \frac{(T_{t+1}) - (T_t)}{365}$ $LR(T_t \div T_{t+1}, Act/365) = \frac{(29\ lut\ 2004) - (30\ lis\ 2002)}{365}$ $LR(T_t \div T_{t+1}, Act/365) = 1{,}249315$
Act/360	Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} $LR(T_t \div T_{t+1}, Act/360) = \frac{(T_{t+1}) - (T_t)}{360}$ $LR(T_t \div T_{t+1}, Act/360) = \frac{(29\ lut\ 2004) - (30\ lis\ 2002)}{360}$ $LR(T_t \div T_{t+1}, Act/360) = 1{,}266667$
Act/Act	Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} $LR(T_t \div T_{t+1}, Act/Act) =$ $LR(T_t \div KR_{2002}, Act/365) + R_{t+1} - (R_t + 1) + LR(KR_{2003} \div T_{t+1}, Act/366)$ $LR(T_t \div T_{t+1}, Act/Act) =$ $\frac{(31\ gru\ 2002) - (30\ lis\ 2002)}{365} + 2004 - (2002 + 1) +$ $\frac{(29\ lut\ 2004) - (31\ gru\ 2003)}{366}$ $LR(T_t \div T_{t+1}, Act/Act) = 1{,}248866$ 

	Jeżeli okres $T_t \div T_{t+1}$ *mieści się w ramach jednego roku kalendarzowego, to obliczenia będą wyglądać następująco:* $LR(T_t \div T_{t+1}, Act/Act) = \begin{cases} \frac{T_{t+1} - T_t}{365} & \text{dla roku nieprzestępnego} \\ \frac{T_{t+1} - T_t}{366} & \text{dla roku przestępnego} \end{cases}$
30/360	Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} Jeżeli T_{t+1} jest ostatnim dniem lutego, to należy zmienić D_{t+1} na 30, tylko j T_t jest również ostatnim dniem lutego. Jeżeli T_t jest ostatnim dniem lutego, to należy zmienić D_t na 30. Jeżeli D_{t+1} wynosi 31, to należy zmienić D_{t+1} na 30, tylko jeżeli D_t wynosi 30 lub 31, (uwzględniając możliwą zmianę według poprzedniej formu Jeżeli D_t wynosi 31, to należy zmienić D_t na 30. Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} $LR(T_t \div T_{t+1}, 30/360) = \frac{(R_{t+1} - R_t) \cdot 360 + (M_{t+1} - M_t) \cdot 30 + (D_{t+1} - D_t)}{360}$ $LR(T_t \div T_{t+1}, 30/360) = \frac{(2004 - 2002) \cdot 360 + (02 - 11) \cdot 30 + (29 - 30)}{360}$ $LR(T_t \div T_{t+1}, 30/360) = 1{,}247222$
30E/360	Frakcja roku od (włączając) T_t do (wyłączając) T_{t+1} Jeżeli D_{t+1} wynosi 31, to należy zmienić D_{t+1} na 30 Jeżeli D_t wynosi 31, to należy zmienić D_t na 30. $LR(T_t \div T_{t+1}, 30E/360) = \frac{(R_{t+1} - R_t) \cdot 360 + (M_{t+1} - M_t) \cdot 30 + (D_{t+1} - D_t)}{360}$ $LR(T_t \div T_{t+1}, 30E/360) = \frac{(2004 - 2002) \cdot 360 + (02 - 11) \cdot 30 + (29 - 30)}{360}$ $LR(T_t \div T_{t+1}, 30E/360) = 1{,}247222$

c. Konwencja płatności

Konwencja płatności określa, kiedy nastąpi przepływ wynikający z danego instrumentu finansowego, wówczas gdy data tego przepływu (z warunków transakcji) przypada na dzień wolny od pracy.



T_{t+1} Teoretyczna data płatności, która jest dniem wolnym od pracy

Istnieje pięć podstawowych typów konwencji płatności:

1. **Konwencja następnego dnia roboczego (**Standard Following Business Day) **-** Data płatności przesuwana jest na pierwszy dzień roboczy po dniu płatności.

2. **Zmodyfikowana konwencja następnego dnia roboczego** (Modified Following Business Day) **-** Data płatności przesuwana jest na pierwszy dzień roboczy po dniu płatności, jeżeli ów dzień znajduje się w tym samym miesiącu kalendarzowym co data płatności. W przeciwnym przypadku, data płatności przesuwana jest na pierwszy dzień roboczy przed dniem płatności.

3. **Konwencja poprzedniego dnia roboczego**(Standard Previous Business Day) **-** Data płatności przesuwana jest na pierwszy dzień roboczy przed dniem płatności.

4. **Zmodyfikowana konwencja poprzedniego dnia roboczego** (Modified Previous Business Day) **-** Data płatności przesuwana jest na pierwszy dzień roboczy przed dniu płatności., jeżeli ów dzień znajduje się w tym samym miesiącu kalendarzowym co data płatności . W przeciwnym przypadku, data płatności przesuwana jest na pierwszy dzień roboczy po dniu płatności.

5. **Konwencja końca miesiąca** (Standard End of Month) **-** Data płatności ustalana jest w ostatnim dniu roboczym miesiąca kalendarzowego

3. Rynek euro obligacji[1]

Konwencje naliczania odsetek – na rynku euroobligacji są stosowane różne konwencje naliczania odsetek, a najpopularniejsze z nich to:

- ACT/360 – faktyczna ilość dni/360,
- ACT/365 – faktyczna ilość dni/365,
- ACT/ACT oznacza konwencję, w której za długość okresu, przyjmuje się faktycznie przypadającą liczbę dni, natomiast długość roku jest obliczana jako wielokrotność długości okresu naliczania odsetek. Jest to konwencja najczęściej stosowana w USA, gdzie odsetki dla obligacji są naliczane dwa razy w roku, co 181, 182, 183, 184 dni i dlatego długość roku może wynieść odpowiednio 362,364, 366 oraz 368 dni,
- 30/360 przy stosowaniu której rok ma 360 dni, a długość okresu między datami D1/M1/Y1 oraz D2/M2/Y2 oblicza się według wzoru: (Y2-Y1)* 360+(M2-M1)*30 + (D2-D1)
  ale w przypadku, gdy D1=31 lub D2=31 wówczas przyjmuje się 30.

[1] Źródło: BRE Bank

4. Zakres projektu

Przygotowana dokumentacja dotyczy części Dużego Projektu 2010, przygotowanego w Octave-3.2.3.

Zawiera 17 funkcji kalendarzowych do wykorzystania w pozostałych częściach Dużego Projektu, bądź niezależnie.

Wykorzystane oznaczenia w dalszym opisie projektu:

Oznaczenie	Opis
*	dopuszczalny zakres 1 Stycznia 1970 - 19 stycznia 2038
**	Identyfikatory płatności - w obecnym buildzie dostępne: o sfbd = Standard Following Business Day o mfbd = Modified Following Business Day o spbd = Standard Previous Business Day o mpbd = Modified Previous Business Day o eom = Standard End of Month o actu = Actual (formalny parametr oznaczający brak konwencji tzn. datę faktyczną i brak przesunięcia) Opis konwencji w rozdziale 2c. Konwencje Płatności.
***	Konwencje bazy odsetkowej - w obecnym buildzie dostępne: o ACT/365, o ACT/360, o ACT/ACT, o 30/360, o 30E/360 Opis w rozdziale 2. Zagadnienia wprowadzające.
****	Rynki/giełdy - w obecnym buildzie dostępne: warsaw, london, zurich, frankfurt

*****	Identyfikatory dotyczące sposobu liczenia liczby dni miedzy dwiema datami - w obecnym buildzie dostępne: o ACT dla konwencji ACT/365, ACT/360, ACT/ACT o 30 dla konwencji 30/360 o 30E dla konwencji 30E/360

a. weekday(date)

Opis funkcji	Wbudowana funkcja Octave'a służąca do identyfikacji dnia tygodnia.
Dane wejściowe	date - string interpretowany jako data formatu "dd-mmm-yyyy" *
Dane wyjściowe	integer z zakresu [1,7] (od: 1=niedziela do: 7=sobota)
Przykład wywołania	weekday("01-Jan-2001")

b. day_diff(date1,date2,basis)

Opis funkcji	Podstawowa funkcja licząca różnicę (w dniach) między dwoma datami. Nie są dokonywane żadne operacje przesunięcia do dni roboczych.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ date2 - string interpretowany jako data formatu "dd-mmm-yyyy" (data końcowa)*, date1<date2, ▪ basis - string, identyfikator konwencji liczenia dni*****
Dane wyjściowe	integer - liczba dni pomiędzy datami w sensie danej konwencji
Przykład wywołania	day_diff("01-Jan-2001","01-Jan-2002","ACT")

c. add_days(date1,n)

Opis funkcji	Funkcja przesuwająca datę o n dni. Nie są dokonywane żadne operacje przesunięcia do dni roboczych.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ n - liczba całkowita, liczba dni (0 = data bez zmian)
Dane wyjściowe	string interpretowany jako data formatu "dd-mmm-yyyy"* (date1 plus n dni)
Przykład wywołania	add_days("01-Jan-2001",5)

d. add_months(date1, mwect,x)

Opis funkcji	Funkcja tworząca ciąg dat na podstawie daty początkowej i wektora długości okresów w miesiącach rozumianych nominalnie np. 1 stycznia 2000 + 1M = 1 lutego 2000, 1 stycznia 2000 + 1M + 3M = 1 maja 2000. W przypadku odwołania do nieistniejącego dnia np. 31 lutego, parametr x koduje cofnięcie do ostatniego dnia miesiąca (x = -1) lub przesunięcie do pierwszego dnia miesiąca następnego (x = 1). Poza tym nie są dokonywane inne przesunięcia dat.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ nwect - wektor liczb naturalnych, liczba miesięcy w poszczególnych okresach ▪ x - integer o wartości 1 lub -1
Dane wyjściowe	lista length(mwect)+1 stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*) wyznaczone z danych wejściowych
Przykład wywołania	dat_list = add_months("01-Jan-2001",[3,6,3,12],-1)

e. easter(year,stock)

Opis funkcji	Funkcja pomocnicza wykorzystywana przy zadaniach związanych z dniami wolnymi. Za pomocą algorytmu Meeusa/Jonesa/Butchera odnajduje datę Wielkanocy i oblicza powiązane z tym świętem daty świąt ruchomych na danej giełdzie. Pomijane są święta nie mające wpływu na aktywność finansową np. wypadające zawsze w niedzielę na rynku, gdzie jest ona i tak dniem wolnym.
Dane wejściowe	• date - string interpretowany jako data formatu "dd-mmm-yyyy"* • stock - string, identyfikator centrum finansowego****
Dane wyjściowe	lista ze stringami interpretowanymi jako daty (formatu "dd-mmm-yyyy"*) bank holidays wyznaczonych przez Wielkanoc na danej giełdzie
Kod	easter_hol = easter(2001,"warsaw")

i. Święta nieruchome

Dla każdego rynku charakterystyczne są święta nieruchome. Oczywistymi są soboty i niedziele. Zestawienie innych świąt nieruchomych znajduje się w poniższej tabeli. Funkcja dotyczy następujących rynków[2]:

Rynek	Święta nieruchome
Warszawa	Nowy Rok - 1 stycznia Święto Państwowe - 1 maja Święto Konstytucji 3 maja - 3 maja Wniebowzięcie NMP - 15 sierpnia Wszytkich Świętych - 1 listopada Święto Niepodległości - 11 listopada Wigila - 24 grudnia Boże Narodzenie - 25 grudnia Pierwszy Dzień Świąt - 26 grudnia
Londyn	New Year's Day - 1 stycznia Christmas Day - 25 grudnia (możliwe przesunięcie na poniedziałek lub wtorek) Boxing Day - 26 grudnia (możliwe przesunięcie na poniedziałek lub wtorek)
Zurich	New Year's Day - 1 stycznia Berchtoldstag - 2 stycznia Labour Day - 1 maja National Day - 1 sierpnia Christmas - 25 grudnia St. Stephen's Day - 26 grudnia
Frankfurt	New Year's Day - 1 stycznia Labour Day - 1 maja National Day - 3 października Christmas Eve - 24 grudnia Christmas - 25 grudnia Boxing Day - 26 grudnia New Year's Eve - 31 grudnia

[2] na podstawie przekazanego kodu w C++ od Prof. Palczewskiego oraz źródeł internetowych

ii. Święta ruchome

Funkcja zawiera implementację algorytmu Meeusa/Jonesa/Butchera do obliczania ruchomego święta Wielkanocy.

Metoda Meeusa/Jonesa/Butchera dla kalendarza gregoriańskiego[3]

Ten sposób został przedstawiony przez Jeana Meeusa w jego książce *Astronomical Algorithms* w 1991 roku. Może być uznany za lepszy od algorytmu Gaussa, ponieważ nie wymaga żadnych cyfr dla określonego zakresu czasu i nie ma od niego wyjątków. Wystarczy podać dowolny rok.

Algorytm:

1. Dzielimy liczbę roku na 19 i wyznaczamy resztę *a*.
2. Dzielimy liczbę roku przez 100, wynik zaokrąglamy w dół (odcinamy część ułamkową) i otrzymujemy liczbę *b*.
3. Dzielimy liczbę roku przez 100 i otrzymujemy resztę *c*.
4. Liczymy: *b* : 4 i wynik zaokrąglamy w dół i otrzymujemy liczbę *d*.
5. Liczymy: *b* : 4 i wyznaczamy resztę *e*.
6. Liczymy: (*b* + 8) : 25 i wynik zaokrąglamy w dół i otrzymujemy liczbę *f*.
7. Liczymy: (*b* - *f* + 1) : 3 i wynik zaokrąglamy w dół i otrzymujemy liczbę *g*.
8. Liczymy: (19 x *a* + *b* - *d* - *g* + 15) : 30 i wyznaczamy resztę *h*.
9. Liczymy: *c* : 4 i wynik zaokrąglamy w dół i otrzymujemy cyfrę *i*.
10. Liczymy: *c* : 4 i wyznaczamy resztę *k*.
11. Liczymy: (32 + 2 x *e* + 2 x *i* - *h* - *k*) : 7 i otrzymujemy resztę *l*.
12. Liczymy: (*a* + 11 x *h* + 22 x *l*) : 451 i wynik zaokrąglamy w dół i otrzymujemy liczbę *m*.
13. Liczymy: (*h* + *l* - 7 x *m* + 114) : 31 i otrzymujemy resztę *p*.
14. Dzień Wielkanocy = *p* + 1.
15. Miesiąc = Zaokrąglenie w dół dzielenia (*h* + *l* - 7 x *m* + 114) przez 31.

[3] Źródło: Wikipedia

Poza dniem Wielkanocy, dla każdego rynku charakterystyczne są również powiązane z nim święta ruchome. Zestawienie tych świąt znajduje się w poniższej tabeli[4]:

Rynek	Święta ruchome
Warszawa	Wielki Piątek - 2 dni przed Niedzielą Wielkanocną Poniedziałek Wielkanocny - 1 dzień po Wielkanocy Boże Ciało - 60 dni po wielkanocy
Londyn	Early May Bank Holiday - pierwszy poniedziałek maja Spring Bank Holiday - ostatni poniedziałek maja Summer Bank Holiday - ostatni poniedziałek sierpnia Good Friday - 2 dni przed Niedzielą Wielkanocną Easter Monday - 1 dzień po Wielkanocy
Zurich	Good Friday - 2 dni przed Niedzielą Wielkanocną Easter Monday - 1 dzień po Wielkanocy Ascension Day - 39 dni po Wielkanocy Whit Monday - 50 dni po Wielkanocy
Frankfurt	Good Friday - 2 dni przed Niedzielą Wielkanocną Easter Monday - 1 dzień po Wielkanocy Ascension Thursday - 39 dni po Wielkanocy Whit Monday - 50 dni po Wielkanocy Corpus Christi - 60 dni po Wielkanocy

[4] na podstawie przekazanego kodu w C++ od Prof. Palczewskiego oraz źródeł internetowych

f. is_business_day(date,stock)

Opis funkcji	Główna funkcja sprawdzająca, czy podana data jest dniem aktywności wybranej giełdy.
Dane wejściowe	▪ date - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ stock - string, identyfikator centrum finansowego****
Dane wyjściowe	logiczny 0 lub 1 (dany dzień nie jest/jest dniem roboczym)
Przykład wykonania	is_business_day("01-Jan-2001","warsaw")

Funkcja zawiera sprawdzenie, czy dany dzień nie jest sobotą albo niedzielą, a następnie sprawdza, czy nie jest jednym ze świąt charakterystycznym dla danego rynku[5]:

Rynek	Przykłady Świąt
Warszawa	• Nowy Rok • Św. Państwowe • Św. Konst. 3 Maja • Wnieb. NPM • Wsz. Świętych • Św. Niepodległości • Wigilia • Boże Narodzenie I • Boże Narodzenie II • Wlk. Piątek, Pon. Wielk. i Boże Ciało
Londyn	• New Year's Day • Early May Bank Holiday • Spring Bank Holiday • Summer Bank Holiday • Christmas Day • Boxing Day • Good Friday, Easter Monday

[5] na podstawie przekazanego kodu w C++ od Prof. Palczewskiego oraz źródeł internetowych

Zurich	• New Year's Day • Berchtoldstag • Labour Day • National Day • Christmas Eve • Christmas • St. Stephen's Day • New Year's Eve • Good Friday, Easter Monday, Ascension Day, Whit Monday
Frankfurt	• New Year's Day • Labour Day • National Day • Christmas Eve • Christmas • Boxing Day • New Year's Eve • Good Friday, Easter Monday, Ascension Thursday, Whit Monday, Corpus Christi

g. day_shift(date1,stock,x)

Opis funkcji	Funkcja pomocnicza przydatna w bardziej złożonych procedurach, szczególnie przy zagadnieniach związanych z konwencjami dni i płatności. Dla podanej daty odnajduje najbliższy następny (x = 1) lub najbliższy poprzedni (x = -1) dzień roboczy na danej giełdzie.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ stock - string, identyfikator centrum finansowego**** ▪ x - integer o wartości 1 lub -1
Dane wyjściowe	string interpretowany jako data formatu "dd-mmm-yyyy"* (x = 1 nabliższy następny, x = -1 najbliższy poprzedni dzień roboczy)
Przykład wywołania	day_shift("01-Jan-2001","warsaw",1)

h. day_shift2(date1,stock,n)

Opis funkcji	Funkcja pomocnicza przydatna w bardziej złożonych procedurach, szczególnie przy zagadnieniach związanych z value date. Przesuwa podaną datę o n dni roboczych do przodu/do tyłu.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ stock - string, identyfikator centrum finansowego**** ▪ n - integer, liczba dni roboczych
Dane wyjściowe	string interpretowany jako data formatu "dd-mmm-yyyy"*, data oddalona o n dni roboczych od daty wejściowej
Przykład wywołania	day_shift2("01-Jan-2001","warsaw",-5)

i. mod_day_shift(date1,stock,x)

Opis funkcji	Funkcja pomocnicza wykorzystywana w date_rolling(). Dla podanej daty odnajduje najbliższy następny/poprzedni (x = 1 / x = -1) dzień roboczy w sensie konwencji Modified Following/Previous Business Day.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ stock - string, identyfikator centrum finansowego**** ▪ x - integer o wartości 1 lub -1
Dane wyjściowe	string interpretowany jako data formatu "dd-mmm-yyyy"* (x = 1 nabliższy następny, x = -1 najbliższy poprzedni dzień roboczy w sensie konwencji Modified Following/Previous Business Day)
Przykład wywyłania	mod_day_shift("01-Jan-2001","warsaw",1)

j. date_rolling(date1,stock,conv)

Opis funkcji	Podstawowa funkcja odnajdująca rzeczywisty dzień płatności (bez przesunięcia związanego z value date).
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy"* ▪ stock - string, identyfikator centrum finansowego**** ▪ conv - string, identyfikator konwencji płatności**
Dane wyjściowe	string interpretowany jako data formatu "dd-mmm-yyyy"* (dzień płatności odpowiadający date1 wyznaczony przez konwencje)
Przykład wywołania	date_rolling("01-Jan-2001","warsaw","sfbd")

k. act_act_frac(date1,date2)

Opis funkcji	Funkcja pomocnicza wykorzystywana w year_frac(). Zwraca frakcję roku dla konwencji ACT/ACT.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ date2 - string interpretowany jako data formatu "dd-mmm-yyyy" (data końcowa)*
Dane wyjściowe	liczba rzeczywista, frakcja roku (w konwencji ACT/ACT) wyznaczona przez daty date1 i date2
Przykład wywołania	act_act_frac("01-Jan-2001","01-Jan-2002")

l. year_frac(date1, date2, basis)

Opis funkcji	Główna funkcja wyznaczająca frakcję roku.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ date2 - string interpretowany jako data formatu "dd-mmm-yyyy" (data końcowa)* ▪ basis - string, identyfikator konwencji liczenia dni***
Dane wyjściowe	liczba rzeczywista interpretowana jako frakcja roku (w zadanej konwencji) wyznaczona przez daty date1 i date2
Przykład wywołania	year_frac("01-Jan-2001","01-Jan-2002","ACT/360")

m. payments(date1,nwect,stock,basis,conv,valuedays)

Opis funkcji	Funkcja ułatwiająca korzystanie z poprzednich procedur w przypadku ciągu płatności, w tej wersji tworząca szereg dat w następujący sposób: • do date1 dodaje nwect(1) dni i otrzymuje date2, na którą działa funkcją date_rolling(), a następnie przesuwa o valuedays dni roboczych do przodu (tzn. odnajduje rzeczywisty dzień płatności) • do date1 dodaje nwect(1)+nwect(2) i otrzymuje date3, na którą działa funkcją date_rolling(), a następnie przesuwa o valuedays dni roboczych do przodu, ..., itd • w efekcie uzyskuje ciąg length(nwect)+1 dat.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ nwect - wektor liczb naturalnych, liczba dni w poszczególnych okresach ▪ stock - string, identyfikator centrum finansowego**** ▪ basis - string, identyfikator konwencji liczenia dni*** ▪ conv - string, identyfikator konwencji płatności** ▪ valuedays - integer nieujemny, liczba dni roboczych od contract date do value date
Dane wyjściowe	▪ lista length(nwect)+1 stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*, dni robocze) wyznaczone z danych wejściowych ▪ lista length(nwect) liczb rzeczywistych, frakcji roku (w zadanej konwencji) pomiędzy sąsiednimi datami z poprzedniej listy
Przykład wywołania	[data_list,frac_list] = payments("01-Jan-2001", [30,30,30,365],"warsaw","ACT/360","spbd",2)

n. payments2(dates,stock,basis,conv,valuedays)

Opis funkcji	Funkcja ułatwiająca korzystanie z poprzednich procedur w przypadku ciągu płatności, w tej wersji pobierająca explicite listę z datami, działająca na nie date_rolling(), na końcu przesuwająca o valuedays dni roboczych do przodu.
Dane wejściowe	▪ dates - lista stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*) ▪ stock - string, identyfikator centrum finansowego**** ▪ basis - string, identyfikator konwencji liczenia dni*** ▪ conv - string, identyfikator konwencji płatności** ▪ valuedays - integer nieujemny, liczba dni roboczych od contract date do value date
Dane wyjściowe	▪ lista stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*, dni robocze) wyznaczonych z listy wejściowej ▪ lista liczb rzeczywistych, frakcji roku (w zadanej konwencji) pomiędzy sąsiednimi datami z poprzedniej listy
Kod	[data_list,frac_list] = payments2({"01-Jan-2001","01-Jan-2002","01-Jan-2003"},"warsaw","ACT/360","sfbd",2)

o. payments3(date1,n,d,stock,basis,conv,valuedays)

Opis funkcji	Modyfikacja payments(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora dni pobiera liczbę okresów oraz (taką samą) ilość dni w każdym z nich.
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ n - liczba naturalna, liczba okresów ▪ d - liczba naturalna, liczba dni w pojedynczym okresie ▪ stock - string, identyfikator centrum finansowego**** ▪ basis - string, identyfikator konwencji liczenia dni*** ▪ conv - string, identyfikator konwencji płatności** ▪ valuedays - integer nieujemny, liczba dni roboczych od contract date do value date
Dane wyjściowe	▪ lista n+1 stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*, dni robocze) wyznaczone z danych wejściowych ▪ lista n liczb rzeczywistych, frakcji roku (w zadanej konwencji) pomiędzy sąsiednimi datami z poprzedniej listy
Kod	[data_list,frac_list] = payments3("01-Jan-2001",5,30,"warsaw","ACT/360","sfbd",2)

p. payments4(date1,mwect,stock,basis,conv,valuedays,x)

Opis funkcji	Modyfikacja payments(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora dni pobiera wektor miesięcy. Przesunięcie następuje w rozumieniu zasady działania add_months().
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ mwect – wektor liczb naturalnych, ilość miesięcy w poszczególnych okresach ▪ stock - string, identyfikator centrum finansowego**** ▪ basis - string, identyfikator konwencji liczenia dni*** ▪ conv - string, identyfikator konwencji płatności** ▪ valuedays - integer nieujemny, liczba dni roboczych od contract date do value date ▪ x - parametr +/-1 ("w przód"/"w tył") do add_months()
Dane wyjściowe	▪ lista length(mwect)+1 stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*, dni robocze) wyznaczone z danych wejściowych ▪ lista length(mwect) liczb rzeczywistych, frakcji roku (w zadanej konwencji) pomiędzy sąsiednimi datami z poprzedniej listy
Kod	[data_list,frac_list] = payments4("01-Jan-2001", [3,3,6,6],"warsaw","ACT/360","sfbd",2,-1)

q. payments5(date1,n,m,stock,basis,conv,valuedays,x)

Opis funkcji	Modyfikacja payments4(), różniąca się tylko sposobem przyjmowania parametrów - zamiast wektora miesięcy pobiera liczbę okresów oraz (taką samą) ilość miesięcy w każdym z nich. Przesunięcie następuje w rozumieniu zasady działania add_months().
Dane wejściowe	▪ date1 - string interpretowany jako data formatu "dd-mmm-yyyy" (data początkowa)* ▪ n - liczba naturalna, liczba okresów ▪ m - liczba naturalna, liczba miesięcy w pojedynczym okresie ▪ stock - string, identyfikator centrum finansowego**** ▪ basis - string, identyfikator konwencji liczenia dni*** ▪ conv - string, identyfikator konwencji płatności** ▪ valuedays - integer nieujemny, liczba dni roboczych od contract date do value date ▪ x - parametr +/-1 ("w przód"/"w tył") do add_months()
Dane wyjściowe	▪ lista n+1 stringów interpretowanych jako daty (formatu "dd-mmm-yyyy"*, dni robocze) wyznaczone z danych wejściowych ▪ lista n liczb rzeczywistych, frakcji roku (w zadanej konwencji) pomiędzy sąsiednimi datami z poprzedniej listy
Kod	[data_list,frac_list] = payments5("01-Jan-2001",5,6,"warsaw","ACT/360","sfbd",2,-1)