Progetto 6

Corporate bond liquidity

P. Manzoni, S. Milanesi, E. Sordo

Financial Engineering Course, AA. 2018-19

14/06/2019

Multicurve-bootstrap L'obiettivo da raggiungere in questo punto è il multicurve-bootstrap della curva dei tassi d'interesse per i mercati EUR (OIS, Euribor 6m) in data 10 settembre 2015 fino a 12 anni. Utilizzeremo un'estensione multicurve a tre parametri del classico modello Hull&White (1990) a due parametri, come descritto in [1] [1].
Attenendoci al paper [2] [2], andremo a considerare due curve:

1. una *discounting curve*: la curva EONIA è una tipica scelta;
2. una *pseudo-discounting curve*, ovvero una curva che in generale è differente per ogni Euribor tenor. Nel nostro caso, utilizzeremo la curva Euribor6m.

Discounting Curve Come già accennato, la curva scelta come discounting curve è la curva EONIA e si usa una tecnica standard per il bootstrap.
Se l'OIS dura meno di un anno con data di inizio $t_s = t_0$ e data di fine t_e, le due controparti scambiano ad un'unica data (t_e) la differenza tra il tasso $R^{OIS}(t_0, t_e)$ stabilito in t_0 e un tasso floating stabilito in t_e:

$$R(t_s, t_e) = \frac{\prod_{k=t_s}^{t_e-1}[1 + \delta(t_k^E, t_{k+1}^E)E(t_k^E)] - 1}{\delta(t_s, t_e)} \tag{1}$$

dove $E(t_k^E)$ è il tasso EONIA il k-esimo giorno. Usando questa definizione, la discounting curve fino a un anno viene calcolata nel seguente modo:

$$B(t_0, t_e) = \frac{1}{1 + \delta(t_0, t_e)R^{OIS}(t_0, t_e)}. \tag{2}$$

Se l'OIS ha maturity maggiore di un anno, le due controparti scambiano alla fine di ogni periodo (della durata di un anno nel nostro caso) la differenza tra l'OIS $R^{OIS}(t_0, t_i)$ e il tasso OIS floating $R^{OIS}(t_{k-1}, t_k)$ con k=1,...i. In questo caso otteniamo:

$$B(t_0, t_i) = \frac{1 - R^{OIS}(t_0, t_i)\sum_{k=1}^{i-1}\delta_k B(t_0, t_k)}{1 + \delta_i R^{OIS}(t_0, t_i)} \tag{3}$$

dove δ_k è la frazione temporale corrispondente all'intervallo (t_{k-1}, t_k).

[1] [1] R.Baviera (2019), Back-of-the-envelope swaptions in a very parsimonious multicurve interest rate model , Mimeo

[2] [2] R.Baviera & A.Cassaro (2014), A note on Dual-Curve Construction: Mr. Crab's Bootstrap, Applied Mathematical Finance

Euribor Curve In questo nuovo contesto, ogni pseudo-discounting curve utilizza contratti definiti unicamente sullo stesso tenor (nel nostro caso sei mesi). Indicando con $F(t_0; t_s, t_e)$ l'Euribor6m forward, possiamo definire come segue lo pseudo-discount tra t_s e t_e:

$$\tilde{B}(t_0; t_s, t_e) = \frac{1}{1 + \delta(t_s, t_e) F(t_0; t_s, t_e)}. \tag{4}$$

Euribor Curve: Mr. Crab's Bootstrap Per la curva dell'Euribor6m, i contratti di riferimento sono i depositi a 6 mesi, i FRA 6m (1x7, 2x8, 3x9, 4x10, 5x11) e gli swaps vs Euribor6m.
Descriviamo ora brevemente la metodologia per la costruzione del bootstrap per lo *pseudo-discount*, il cui schema è rappresentato in Figura 1:

1. consideriamo l'Euribor6m per i depositi a 6 mesi e calcoliamo $\tilde{B}(t_0, t_{6m})$ tramite
$$\tilde{B}(t_0, t_{6m}) = \frac{1}{1 + \delta(t_0, t_{6m}) L(t_0, t_{6m})}; \tag{5}$$

2. utilizziamo il FRA 6x12 e calcoliamo il discount a un anno $\tilde{B}(t_0, t_{1y})$ andando avanti da t_{6m} con
$$\tilde{B}(t_0, t_{1y}) = \tilde{B}(t_0, t_{6m}) \tilde{B}(t_0; t_{6m}, t_{1y}); \tag{6}$$

3. consideriamo i FRA 1x7, 2x8, 3x9, 4x10, 5x11 per muoverci all'indietro e ottenere gli pseudo-discount mensili fino a cinque mesi ricorrendo a
$$\tilde{B}(t_0, t_i) = \frac{\tilde{B}(t_0, t_{i+6m})}{\tilde{B}(t_0; t_i, t_{i+6m})}; \tag{7}$$

4. i forward rates del secondo anno sono tipicamente calcolati usando lo swap rate a 18 mesi;

5. consideriamo i valori quotati degli swap rate dal terzo anno e calcoliamo gli pseudo-discount tramite:
$$\tilde{B}(t_0; t_{i-1}, t_i) = \prod_{k \in Y(i)} \frac{1}{1 + \delta(t_{k-1}, t_k) F(t_0; t_{k-1}, t_k)} \tag{8}$$
dove $k \in Y(i)$ indica i forward rates a sei mesi nell'i-esimo anno.

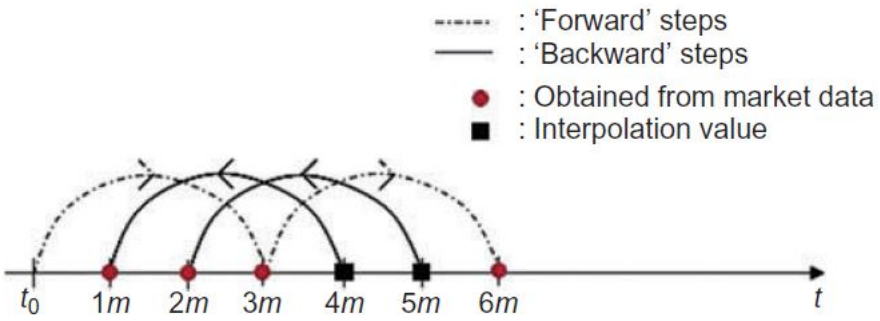


Figure 1: 'Forward-Backward' bootstrap: come ottenere la curva dell'Euribor6m a 1m, 2m, 3m, 4m, 5m dopo la settlement utilizzando i FRA 1x7, 2x8, 3x9, 4x10, 5x11

In `runFinalProject_Group6_1_2` importiamo tramite `readExcelData` i dati che ci sono stati forniti nel file `curves20150910_project.xlsx`, creando le tre seguenti struct: `datesSet_dirty`, `ratesSet` e `normal_vols`.

`datesSet_dirty` ha la seguente struttura:

- `settlement`: settlement date, ovvero 14 settembre 2015 (due giorni lavorativi dopo la fixing date, che è il 10 settembre);
- `OIS`: vettore con le date dei tassi EONIA (foglio 'OIS' del file Excel);
- `fra`: matrice con le date di inizio e di fine dei FRA (foglio 'FRA' del file Excel);
- `euribor`: data corrispondente all'Euribor a 6 mesi (foglio 'LIBORvs6M' del file Excel);
- `swaps`: vettore con le date degli swaps (foglio 'LIBORvs6M' del file Excel).

`ratesSet_dirty` è simile alla precedente (sottolineiamo che i valori percentuali sono stati trasformati in numeri decimali):

- `OIS`: vettore contenente i tassi EONIA corrispondenti alle date presenti nella precedente struct;

- `fra`: vettore con i tassi dei FRA;
- `Euribor6m`: Euribor a 6 mesi;
- `swaps`: vettore con i tassi degli swaps.

Infine `normal_vols` ha la seguente struttura (foglio 'Vols' del file Excel):

- `values`: valore della volatilità (in bps);
- `expiry`: indica l'expiry della swaption;
- `tenor`: indica la durata in anni della swap (sottostante della swaption).

Tipicamente tale operazione è computazionalmente impegnativa per Matlab e richiede infatti un tempo non trascurabile. Ciò detto, ci siamo accorti che alcune date riportate in `curves20150910_project` cadono di sabato o di domenica. Per ovviare a questo problema, abbiamo creato la funzione `clean_date` che, prese in input delle date, controlla che siano effettivamente business dates; in caso negativo, queste vengono modificate utilizzando la Modified Following day convention.

Per ottenere la *discounting curve*, eseguiamo il primo bootstrap chiamando `bootstrap_OIS` (riga 28 del run file). Nella (2) avremo bisogno della lunghezza dell'intervallo temporale tra la settlement e le date relative all'OIS, ovvero $\delta(t_0, t_i)$, che calcoliamo con `yearfrac` usando come day count Act/360 e salviamo in `deltas_settlement_to_date` (riga 38). Dobbiamo poi distinguere i due casi:

- l'OIS dura meno di un anno: troviamo via `dateMoveVec` la data che dista un anno dalla settlement date (`date_1y_after_settlement`) e la confrontiamo con le date dell'OIS fornite; per le date precedenti a `date_1y_after_settlement` ricaviamo i discount factors applicando la formula (2);
- nel caso in cui invece la maturity dell'OIS sia maggiore di un anno, creiamo un nuovo vettore di date `delta_k` contenente i Δt delle date precedenti l'anno, della data più prossima all'anno e di tutte le date successive. Calcoliamo infine i discount usando la (3).

Abbiamo aggiunto come primo elemento la settlement in `dates_EONIA` e 1 (i.e. $B(t_0, t_0)$) in `discounts_EONIA`.

Per quanto riguarda la *pseudo-discounting curve*, abbiamo implementato `bootstrap_pseudo` che ripercorre gli step teorici descritti in precedenza.
Ricaviamo dall'Euribor6m lo pseudo-discount a sei mesi (riga 51) applicando (4). Ricaviamo poi quello a un anno (riga 60) sfruttando il FRA 6x12 e le relazioni (6) e (4).
Passiamo ora alla novità introdotta da questo metodo: il bootstrap di Mr. Crab. Ricaviamo gli pseudo-discount mensili da 7m a 11m interpolando sugli zero-rates tramite `interp_discount` (riga 66). Ora (riga 75) è sufficiente applicare la (7), dove il denominatore è espresso come nella formula (4).
Finora conosciamo la *pseudo-discounting curve* fino al primo anno. Per proseguire fino al dodicesimo anno dobbiamo usare gli swaps-vs-Euribor6m. Troviamo le date corrispondenti alla floating leg (`floating_dates` riga 81) muovendoci in avanti di sei mesi in sei mesi dalla settlement alla fine dei 12 anni tramite `tenordates` (che sfrutta semplicemente `dateMoveVec`) e selezioniamo quelle annuali come date della fixed leg.
Interpoliamo linearmente sugli zero-rates i tassi EONIA per avere i discount corrispondenti alle date della floating leg (`discounts_EONIA_semiannual` riga 85). É importante sottolineare che nel modello multicurve la relazione della somma telescopica per la floating leg della swap non vale più; pertanto calcoliamo separatamente gli NPV della floating e della fixed leg. Possiamo ricorrere a

$$\sum_{k=1}^{2i} B(t_0, t_k)\delta_k F_k(t_0) = \mathcal{I}(i) \qquad i \geq 2 \tag{9}$$

dove

$$\mathcal{I}(i) = S(t_0, t_i)\sum_{k=1}^{i} \delta_k B(t_0, t_k) \qquad i \geq 2 \tag{10}$$

è calcolato alla riga 99. `forward_euribor6m_semiannual` sono i $\delta_k F_k(t_0)$ ottenuti tramite (4).
Nella sommatoria della (9) conosciamo tutti i termini (i `known_term` della riga 108) tranne gli ultimi due, che saranno il risultato del sistema di equazioni implementato in `two_eqns` risolto con `fsolve`. Nello specifico, contrariamente a quanto indicato nel paper [2], utilizziamo questo metodo già a partire dal secondo anno, dal momento che non abbiamo a disposizione lo swap rate a 18m.
Nel sistema `two_eqns`, la prima equazione è semplicemente la (9) in cui la sommatoria è separata in due parti (termini noti e incogniti) mentre la seconda è un'interpolazione lineare classica sugli zero-rates eseguita con `interp_discounts`. Mostriamo nella Figura 2 la *discount* e la *pseudo-discount curve*.

Remark. Benché il "Mr.Crab bootstrap" sia rilevante per ottenere i discount nella finestra temporale (0, 6m) [valori che altrimenti non potremmo avere, a causa della mancanza di dati dovuta al fatto che lavoriamo con un tenor pari a sei mesi], non riscontriamo di fatto alcun impatto sui discount successivi all'anno, in quanto si prosegue con la tecnica forward.

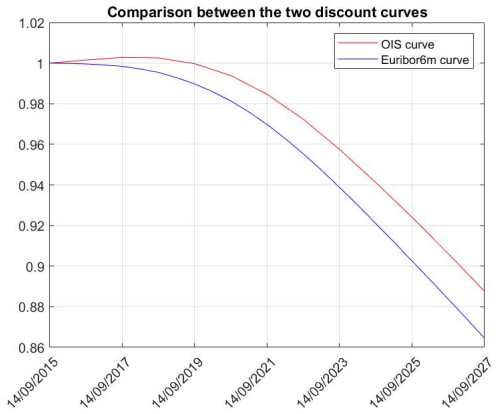


Figure 2: Curva degli OIS *discount* (in rosso) e curva degli *pseudo-discount* per l'Euribor6m (in blu) in data 10 settembre 2015, partendo dalla settlement e con un orizzonte temporale di 12 anni.

Possiamo osservare in Figura 3 gli zero-rates sia con i discount EONIA sia con gli pseudo-discounts.

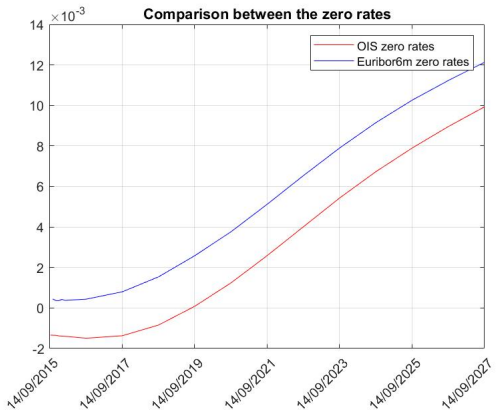


Figure 3: Curva degli zero-rates dei *discount* (in rosso) e curva degli zero-rates degli *pseudo-discount* per l'Euribor6m (in blu) in data 10 settembre 2015, partendo dalla settlement e con un orizzonte temporale di 12 anni.

Calibrazione dei parametri di volatilità Passiamo ora alla calibrazione dei parametri di volatilità come descritto in [1] sulle ATM CS receiver "diagonal" swaptions sempre in data 10 settembre 2015.
Il primo step consiste nell'eseguire il bootstrap delle curve dei discount e degli pseudo-discount, come abbiamo già fatto al punto precedente. Dovremo poi calibrare i tre parametri $\mathbf{p} := (a, \sigma, \gamma)$ del modello Multicurve Hull White usando le volatilità normali delle ATM CS swaptions vs Euribor 6m (valori diagonali fino a 10 anni, forniti nel foglio 'Vols' del file `curves20150910_project`).

I prezzi di mercato delle CS receiver swaptions sono ottenuti tramite la Normal-Black market formula (chiamata anche formula di Bachelier):

$$\mathfrak{R}_{\alpha\omega}^{C,MKT}(t_0) = B(t_0, t_\alpha) C_{\alpha\omega}(S(t_0))\{[K - S_{\alpha\omega}(t_0)]N(-d) + \sigma_{\alpha\omega}\sqrt{t_\alpha - t_0}\phi(d)\}. \tag{11}$$

dove $d = 0$ nel nostro caso, dato che la swaption è at the money.
Nel modello MHW invece le ATM CS receiver swaptions sono prezzate come segue:

$$\mathfrak{R}_{\alpha\omega}^{C,MHW}(t_0) = B(t_0, t_\alpha) \int_{-\infty}^{x^*} dx \frac{e^{-x^2/2}}{\sqrt{2\pi}} C_{\alpha\omega}(S(x))[K - S(x)]^+ \tag{12}$$

con

$$S(x) = \frac{\sum_{\iota=\alpha'}^{\omega'-1} B_{\alpha'\iota}(t_0)\beta_\iota(t_0)\exp\{-\nu_{\alpha'\iota}x - \nu_{\alpha'\iota}^2/2\}}{\sum_{j=\alpha+1}^{\omega} \delta_{j-1} B_{\alpha j}(t_0)\exp\{-\varsigma_{\alpha j}x - \varsigma_{\alpha j}^2/2\}} + \\ + \frac{\sum_{\iota=\alpha'+1}^{\omega'} B_{\alpha'\iota}(t_0)\exp\{-\varsigma_{\alpha'\iota}x - \varsigma_{\alpha'\iota}^2/2\}}{\sum_{j=\alpha+1}^{\omega} \delta_{j-1} B_{\alpha j}(t_0)\exp\{-\varsigma_{\alpha j}x - \varsigma_{\alpha j}^2/2\}}.$$

Invece x^* è ottenuto come zero della seguente funzione:

$$f(x) := \sum_{j=\alpha+1}^{\omega} c_j B_{\alpha j}(t_0) e^{-\varsigma_{\alpha j}x - \varsigma_{\alpha j}^2/2} + \\ + \sum_{\iota=\alpha'+1}^{\omega'-1} B_{\alpha'\iota}(t_0) e^{-\varsigma_{\alpha'\iota}x - \varsigma_{\alpha'\iota}^2/2} + \\ - \sum_{\iota=\alpha'}^{\omega'-1} \beta_\iota(t_0) B_{\alpha'\iota}(t_0) e^{-\nu_{\alpha'\iota}x - \nu_{\alpha'\iota}^2/2}$$

mentre $C_{\alpha\omega}(S)$ è definito come segue:

$$C_{\alpha\omega}(S) := \sum_{i=1}^{\omega-\alpha} \frac{1/m}{(1 + S/m)^i} = \frac{1}{S}\left(1 - \frac{1}{(1 + S/m)^{\omega-\alpha}}\right) \quad S > -m \tag{13}$$

dove S è lo swap-rate del sottostante a maturità della swaption, m è il numero di pagamenti effettuati ogni anno nella fixed leg dello swap sottostante ($m = 1$ nel nostro caso), t_α è l'expiry e $\omega - \alpha$ è il tenor.
Nelle precedenti relazioni, compaiono i seguenti termini:

$$\beta(t; T, T+\Delta) := \frac{B(t; T, T+\Delta)}{\tilde{B}(t; T, T+\Delta)} \tag{14}$$

è lo spread moltiplicativo;

$$\zeta_\alpha^2 := \begin{cases} \sigma^2 \dfrac{1 - e^{-2a(t_\alpha - t_0)}}{2a} & a \in \Re^+ \setminus \{0\} \\ \sigma^2 (t_\alpha - t_0) & a = 0. \end{cases} \tag{15}$$

dove

$$\sigma(t, T) = (1 - \gamma)\tilde{\sigma}(t, T) \tag{16}$$

$$\varsigma_{\alpha'\iota} := (1 - \gamma) v_{\alpha'\iota}; \tag{17}$$

$$\nu_{\alpha'\iota} := v_{\alpha'\iota} - \gamma v_{\alpha'\iota+1} \tag{18}$$

con

$$v_{\alpha'\iota} := \zeta_\alpha \frac{1 - e^{-a(t'_\iota - t'_{\iota'})}}{a} \qquad \iota = \alpha', ..., \omega'. \tag{19}$$

e con

$$\tilde{\sigma}(t, T) := \begin{cases} \sigma \dfrac{1 - e^{-a(T-t)}}{a} & a \in \Re^+ \setminus \{0\} \\ \sigma(T - t) & a = 0. \end{cases} \tag{20}$$

Remark. Tutti i termini in cui compare l'apice sono riferiti alle date della floating leg.
Si noti che in (15) si ha una discontinuità eliminabile per $a = 0$: tuttavia dopo alcune prove abbiamo deciso di non tenere conto di questo aspetto a livello implementativo. Di fatto la variabile è ben definita fintantoché $a > 10^{-16}$, mentre per valori di a inferiori ζ_α^2 diventa numericamente uguale a zero. Tuttavia abbiamo verificato sperimentalmente che nel nostro caso ciò non inficia la bontà della minimizzazione.

Avendo ora tutti gli strumenti a disposizione, per calibrare i parametri del modello è sufficiente minimizzare il quadrato della distanza tra i prezzi di mercato e quelli del modello delle CS ATM receiver swaptions:

$$\mathtt{Err}^2(p) = \sum_{i=1}^{M} [\Re_i^{C,MHW}(\mathbf{p}; t_0) - \Re_i^{C,MKT}(t_0)]^2. \tag{21}$$

Nel main `runFinalProject_Group6_1_2` tutto questo procedimento è implementato nella funzione `calibrate_model`.
Partiamo definendo tutte le date necessarie, ovvero la data un anno dopo la settlement (expiry della prima swaption) e le date dal primo anno fino al decimo ogni sei mesi (`dates_semiannual` riga 46). In seguito usiamo `interp_discount` per interpolare sugli zero-rates i discount e gli pseudo-discount corrispondenti alle date appena ricavate a partire dalle curve ottenute al primo punto.
Abbiamo bisogno anche dello spread β definito in (14), per cui dobbiamo calcolare i discount e gli pseudo-discount forward (righe 66,70).
I prezzi di mercato delle ATM CS receiver swaptions sono calcolati nella funzione `normal_vols_swaption_MHJM`.

`normal_vols_swaption_MHJM` Nella formula di Bachelier, in particolare in $C_{\alpha\omega}$, dobbiamo calcolare lo swap rate forward start come

$$S_{\alpha\omega}(t) = \frac{\mathcal{N}_{\alpha\omega}(t)}{BPV_{\alpha\omega}(t)} \tag{22}$$

con il forward basis point value (riga 64)

$$BPV_{\alpha\omega}(t) := \sum_{j=\alpha+1}^{\omega} \delta_{j-1} B_{\alpha j}(t) \tag{23}$$

e con il 'forward' NVP della floating leg (riga 68)

$$\mathcal{N}_{\alpha\omega}(t) := 1 - B_{\alpha\omega}(t) + \sum_{\iota=\alpha'}^{\omega'-1} B_{\alpha'\iota}(t)[\beta_\iota(t) - 1] \tag{24}$$

Una volta calcolati questi termini, li sostituiamo nella (13) alla riga 84 e infine, alla riga 87, applichiamo la formula di Bachelier (11).

Ora che abbiamo calcolato i prezzi di mercato, abbiamo bisogno di quelli del modello prima di passare alla minimizzazione. A tal fine, abbiamo implementato `compute_model_swaption_MHJM`.

`compute_model_swaption_MHJM` Dopo aver calcolato tutti i termini presenti nell'espressione di $f(x)$ (yearfrac, ι, ς, ν, v, ...), definiamo tale funzione come function handle di x (riga 101) e ne troviamo lo zero (ovvero x^*) tramite `fsolve`.
Osservando attentamente le espressioni di $f(x)$ e di $S(x)$ ci accorgiamo che sono molto simili: $S(x)$ corrisponde effettivamente allo swap rate tale per cui il payoff $f(x)$ è nullo. Infatti

$$\sum_{j=\alpha+1}^{\omega} c_j B_{\alpha j}(t_0) e^{-\varsigma_{\alpha j} x - \varsigma_{\alpha j}^2/2} + \sum_{\iota=\alpha'+1}^{\omega'-1} B_{\alpha'\iota}(t_0) e^{-\varsigma_{\alpha'\iota} x - \varsigma_{\alpha'\iota}^2/2} =$$

$$= \sum_{j=\alpha+1}^{\omega} \delta_j B_{\alpha j}(t_0) e^{-\varsigma_{\alpha j} x - \varsigma_{\alpha j}^2/2} + \sum_{\iota=\alpha'+1}^{\omega'} B_{\alpha'\iota}(t_0) e^{-\varsigma_{\alpha'\iota} x - \varsigma_{\alpha'\iota}^2/2}$$

poiché l'elemento $B_{\alpha\omega}(t_0) e^{-\varsigma_{\alpha\omega} x - \varsigma_{\alpha\omega}^2/2}$ può essere agilmente spostato da una sommatoria all'altra, sapendo che $t_\omega \equiv t_{\omega'}$: pertanto nel codice possiamo utilizzare le stesse componenti definite per $f(x)$ per calcolare anche $S(x)$.
Ora ci rimane solamente da calcolare $C_{\alpha\omega}$ inserendo i termini trovati nella (13) e calcolare il prezzo della ATM CS receiver swaption, come definito nella (12). Il calcolo dell'integrale è stato infine eseguito utilizzando il metodo di quadratura implementato in `quadgk`.

Otteniamo quindi i prezzi del modello come function handle dei tre parametri (a, σ, γ) che dobbiamo calibrare. Nella versione definitiva, per non appesantire eccessivamente la ricerca del minimo vincolato della funzione distanza, abbiamo utilizzato un algoritmo di ricerca locale (`fminsearch`), partendo da un punto `x_0` appositamente scelto per giungere agevolmente ai valori del paper [1]: infatti i valori che otteniamo sono $a = 12.94\%$, $\sigma = 1.26\%$ e $\gamma = 0.06\%$. La minimizzazione è in realtà stata eseguita per mezzo di un algoritmo di ricerca globale (`MultiStart`, dal pacchetto ottimizzazione di Matlab). Tuttavia ciò richiede un grande sforzo computazionale e tipicamente vengono trovati minimi "peggiori" o lievemente migliori di quello indicato.

In Figura 4 mostriamo la differenza tra i prezzi di mercato e quelli ottenuti tramite modello delle ATM CS swaptions.

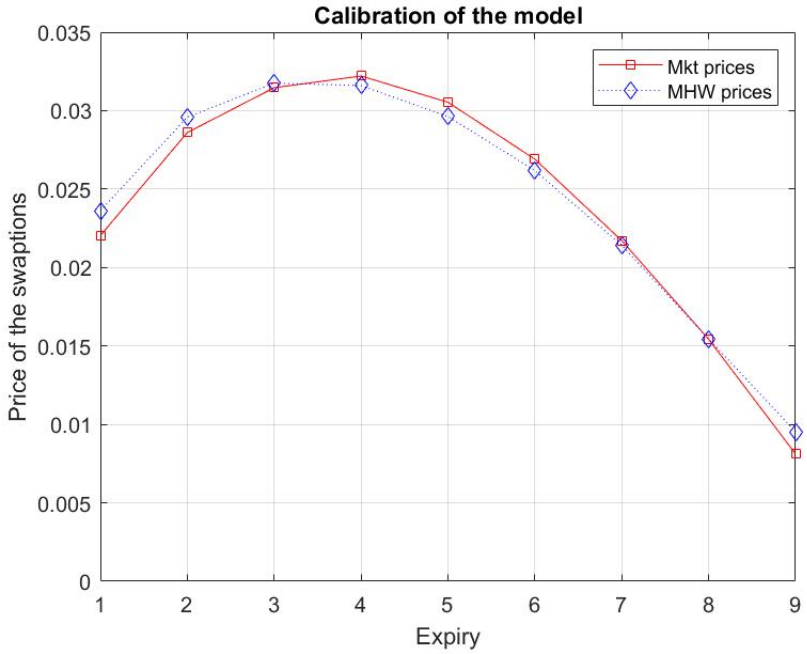


Figure 4: Prezzi di mercato delle ATM diagonal CS swaption che terminano a 10 anni (linea continua rossa e quadrati) e i prezzi corrispondenti ottenuti tramite la calibrazione del modello MHW (linea puntinata blu e diamanti).

Upper e lower bounds del liquidity Spread Passiamo ora al calcolo della differenza tra upper e lower bound del liquidity Spread sia per BNPP che per Santander nei due casi di time-to-liquidate τ pari a 2 settimane e a 2 mesi.
I dati che abbiamo a disposizione per BNPP e per Santander sono i seguenti:

- maturity di ogni bond;
- valore (in percentuale) del coupon pagato da ciascun bond (i coupon sono annuali con convezione Act/Act per il day-count);
- clean price, ovvero il prezzo quotato sul mercato (si tratta di un mid-price riferito alla fine del 10 settembre 2015);

Fortunatamente esiste una formula chiusa per gli illiquid corporate coupon bond, che ci permette anche di ricavare l'illiquidity price.
Prima però calcoliamo upper e lower bound, sempre tramite formule chiuse, come enunuciato nel teorema 3.2 del paper [3] [3] :

$$\sum_{i=1}^{N} c_i \overline{B}(t_0,t_i)(\pi_i^L(\tau)-\mathcal{P}(t_0,\tau)) \leq \Delta_\tau \leq \sum_{i=1}^{N} c_i \overline{B}(t_0,t_i)(\pi_i^U(\tau)-\mathcal{P}(t_0,\tau)) \tag{25}$$

dove $\overline{B}(t_0,t_i) = \mathbb{E}[D(t_0,T)\mathbb{1}_{t_d>T}|\mathcal{F}_0]$ è il prezzo di un *risky ZC bond* con zero-recovery, la sommatoria è limitata alle date di pagamento $t_i > \tau$ e

$$\pi_i^U(\tau) := \frac{4+\Sigma_i^2(\tau)}{2}\Phi\left(\frac{\Sigma_i(\tau)}{2}\right) + \frac{\Sigma_i(\tau)}{\sqrt{2\pi}} exp\left(-\frac{\Sigma_i^2(\tau)}{8}\right) \tag{26}$$

$$\begin{aligned}\pi_i^L(\tau) := &\int_0^1 d\eta \frac{e^{-\frac{1}{8}\Sigma_N^2}}{\pi\sqrt{1-\eta}\sqrt{\eta}} e^{-\frac{\eta}{2}\Sigma_i(\tau)(\Sigma_i(\tau)-\Sigma_N(\tau))} \\ &\left\{1+\sqrt{\frac{\pi(1-\eta)}{2}}\Sigma_N(\tau)e^{\frac{1-\eta}{8}}\Phi\left[\frac{\sqrt{1-\eta}}{2}\Sigma_N(\tau)\right]\right\} \\ &\left\{1+\sqrt{\frac{\pi\eta}{2}}(2\Sigma_i(\tau)-\Sigma_N(\tau))e^{\frac{\eta}{8}(2\Sigma_i(\tau)-\Sigma_N(\tau))^2}\Phi\left[\frac{\sqrt{\eta}}{2}(2\Sigma_i(\tau)-\Sigma_N(\tau))\right]\right\}.\end{aligned} \tag{27}$$

La volatilità cumulata è $\Sigma_i^2(\tau) := \zeta_i^2 \frac{1-e^{-2\hat{a}(\tau-t_0)}}{2\hat{a}}$ con $\zeta_i := \frac{\hat{\sigma}}{\hat{a}}(1-e^{-\hat{a}(t_i-\tau)})$. Infine $\mathcal{P}(t_0,\tau)$ è la probabilità di sopravvivenza fino al time-to-liquidate

[3] [3] R.Baviera & A.Nassigh & E.Nastasi (2019), A closed formula for illiquid corporate bonds and an application to the European market, Mimeo

dell'issuer, calcolata come $1 - default\_prob$.

Questa formula chiusa racchiude, oltre alle caratteristiche del bond, la zero-rate curve, la credit spread term-structure dell'issuer che stiamo considerando e la volatilità dei bond.
Ciò di cui abbiamo bisogno può essere ottenuto dai dati di mercato tramite le tecniche standard che spiegheremo in seguito nel commento del codice.

In `runFinalProject_Group6_3_4` dobbiamo innanzitutto estrarre i dati necessari: usiamo `readExcelData` per importare dati dal file Excel fornitoci `curves20150910_project.xlsx`, mentre le informazioni su BNPP e Santander (da paper [3]) sono state riportate in `buildStruct` (dove, avendo assunto che il face value fosse pari a 1, i prezzi forniti sono stati riscalati consistentemente).

Come accadeva nel precedente esercizio, alcune delle date importate cadono di sabato o di domenica, e chiamiamo perciò `clean_date` che le modifica secondo la Modified Following day convention.
Inoltre, i prezzi forniti sono clean prices, mentre noi dovremo utilizzare in seguito solo i dirty prices, come da standard nella modellazione degli strumenti *fixed-income*. La funzione `dirty_from_clean` ha proprio questo scopo:

> `dirty_from_clean` Iniziamo calcolando la massima distanza tra la settlement date (14 settembre 2015 in questo caso) e la maturity di ciascun liquid bond, per poi prenderne il `ceil` dato che siamo interessati all'ultima data di pagamento del coupon prima della settlement. Poichè l'istante in cui noi acquistiamo il bond si trova tra due date di pagamento di coupon, per calcolare il rateo, troviamo esattamente queste due date (chiamate rispettivamente `previous_payment_date` e `following_payment_date`) utilizzando all'indietro la funzione `dateMoveVec`. Infine calcoliamo l'accrual come la frazione di coupon che dobbiamo restituire e lo sommiamo al clean price, ottenendo così il dirty price.

Siccome la curva risk-free considerata è la OIS curve, grazie alla funzione `bootstrap_OIS` già utilizzata per il primo punto, otteniamo la curva dei $B(t_0, t)$.

Dopo tutte queste necessarie operazioni preliminari, possiamo passare alla funzione `liquidity_spread_bounds` per il calcolo di upper e lower bounds.

`liquidity_spread_bounds` Come possiamo facilmente evincere dal teorema 3.2 precendentemente menzionato, avremo bisogno delle date di pagamento dei coupon per ciascun bond che ricaviamo tramite la funzione `create_vector_payment_dates`.

`create_vector_payment_dates` In modo analogo a quanto fatto in `dirty_from_clean`, calcoliamo il numero di pagamenti futuri di coupon `num_coupon_payments` guardando alla differenza temporale tra la settlement e le maturity di ciascun bond, considerando questa volta il `floor`, dovendo rimanere sempre dopo il 14 settembre 2015.
Trattandosi di coupon annuali, calcoliamo tramite `dateMoveVec` le rispettive date di pagamento, che vengono salvate in `coupon_payment_dates_cell` definito come cell array.
Tuttavia, per non dover utilizzare anche nelle funzioni future i cell array, strutture dinamiche che richiedono l'uso di molti cicli `for` e la cui gestione della memoria risulta significativamente più lenta, decidiamo di salvare tutti le date di pagamento dei coupon in un unico vettore (variabile statica), che potremo gestire agevolmente grazie ai tre indici restituiti: `idx1`, `idx1_cs` e `idx2`.

- `idx1` indica quanti pagamenti di coupon prevede ciascun bond;
- `idx1_cs` è la somma cumulata di `idx1`. Nel vettore finale dunque le date di interesse per l'i-esimo bond saranno `coupon_payment_dates`(`idx1_cs`(`i`) + 1 : `idx1_cs`(`i` + 1));
- `idx2` contiene il numero di coupon che vengono pagati in una data intermedia tra la settlement e il time-to-liquidate e che quindi dovremo escludere. Nel nostro caso, poiché le cedole sono annuali, si tratterà sostanzialmente di una variabile binaria.

Per motivi pratici (utilizzo nei cicli), il primo elemento di ciascun vettore di indici è posto pari a zero.
Infine trasformiamo il cell array in un unico vettore grazie alla funzione di Matlab `cell2mat`.

Per capire meglio dal punto di vista pratico cosa rappresentano questi indici, consideriamo i primi tre liquid bond di BNPP con time-to-liquidate pari a due settimane e riportiamo nella seguente tabella le `coupon_payment_dates` e i corrispondenti indici:

Bond 1	Bond 2	Bond 3
27-Nov-2015	14-Mar-2016	23-Nov-2015
28-Nov-2016	14-Mar-2017	21-Nov-2016
27-Nov-2017	12-Mar-2018	21-Nov-2017
-	-	21-Nov-2018
idx1=3	idx1=3	idx1=4
idx1_cs=3	idx1_cs=6	idx1_cs=10
idx2=0	idx2=0	idx2=0

Tornando a `liquidity_spread_bounds`, per avere vettori della stessa lunghezza, `coupon_on_dates` è ottenuto replicando `idx1` volte il valore del coupon pagato dal corrispondente bond e aggiungendo il face value (che assumiamo essere pari a 1) nell'ultima data di pagamento, ovvero la maturity.

Passiamo poi al calcolo di $\pi_i^U(\tau)$ e $\pi_i^L(\tau)$ che eseguiamo separatamente in `compute_pi_ul` applicando le equazioni (26) e (27).

Per ricavare i $\overline{B}(t_0, T)$, dobbiamo fare il bootstrap della Zeta-spread curve, dove

$$Z(T) = -\frac{1}{T - t_0} \ln \frac{\overline{B}(t_0, T)}{B(t_0, T)} \tag{28}$$

In `bootstrap_Z_curve` assumiamo un valore costante per la Zeta-spread curve fino alla maturity del bond con maturity inferiore (nel nostro caso corrisponde sempre alla maturity del primo bond) e poi usiamo l'interpolazione lineare sugli Zeta-spread.

`bootstrap_Z_curve` Per eseguire il bootstrap, avremo bisogno di conoscere i $B(t_0, t_i)$ per tutte le date di pagamento dei coupon, pertanto iniziamo con la loro interpolazione sugli zero-rates implementata in `interp_discounts`.
Per quanto riguarda il primo bond, i valori di Zeta-spread saranno tutti uguali ad una costante, ottenuta risolvendo tramite un solver convesso (che permette di avere una convergenza rapida) una semplice equazione ottenuta dal seguente sistema:

$$\begin{cases} Z(t_1) = -\frac{1}{t_1-t_0} ln \frac{\overline{B}(t_0,t_1)}{B(t_0,t_1)} \\ Z(t_2) = Z(t_1) = -\frac{1}{t_2-t_0} ln \frac{\overline{B}(t_0,t_2)}{B(t_0,t_2)} \\ \vdots \\ Z(t_N) = Z(t_1) = -\frac{1}{t_N-t_0} ln \frac{\overline{B}(t_0,t_N)}{B(t_0,t_N)} \\ \overline{P}(t_0, T = t_N; \mathbf{c,t}) = \sum_{i=1}^{N} c_i \overline{B}(t_0, t_i) = \sum_{i=1}^{N} c_i B(t_0, t_i) e^{-Z(t_i)(t_i-t_0)} \end{cases}$$

Ricavati gli Zeta-spread, è immediato ricavare i $\overline{B}(t_0, t_i)$ tramite:

$$\overline{B}(t_0, t_i) = B(t_0, t_i) \exp\{-(t_i - t_0) Z(t_i)\}.$$

A questo punto la curva è nota fino alla maturity del primo bond e si può passare quindi passare al bond successivo.
Nel tentativo di dare sufficiente generalità al codice, andremo a distinguere tra i vari casi che si possono presentare: dovremo infatti risolvere problemi lievemente diversi in base al numero di `payment_dates` successive all'ultimo valore noto. Ciò definirà infatti il numero di incognite presenti in ogni iterazione del ciclo, che viene salvato di volta in volta nella variabile `flag` (riga 66). L'equazione fondamentale è la stessa già utilizzata:

$$\overline{P}(t_0, T = t_N; \mathbf{c,t}) = \sum_{i=1}^{N} c_i \overline{B}(t_0, t_i)$$

tuttavia, nel caso in cui si abbia più di un'incognita, diviene necessario fare un'assunzione sulla struttura della curva: nello specifico, abbiamo usato un'interpolazione lineare per ottenere un numero congruo di equazioni ed incognite. Ciò detto, qualsiasi valore assuma `flag`, dobbiamo calcolare gli Zeta-spread precedenti all'ultima data disponibile tramite `interp1` (riga 76) e poi calcolare i corrispondenti $\overline{B}(t_0, t_i)$ (riga 79). Come anticipato, il caso più semplice da trattare è quello in cui `flag` = `1`, ovvero tutte le date di pagamento del coupon tranne l'ultima (la maturity stessa) cadono nell'intervallo dove lo Zeta-spread e i $\overline{B}(t_0, t_i)$ sono noti, vale a dire prima della maturity del bond precedentemente utilizzato per il bootstrap. [Ipotesi base: bonds ordinati per maturity crescente]. Pertanto è sufficiente interpolare linearmente gli Zeta-spread in corrispondenza delle prime $N - 1$ date, mentre l'ultimo $\overline{B}(t_0, t_i)$ può essere ricavato dalla formula precedente (riga 85).

Usando la definizione di Zeta-spread, ricaviamo anche l'ultimo valore finora incognito (riga 89). É importante ricordare che la curva dei fattori di sconto, bootstrappata dagli OIS, è già nota, quindi l'unica incognita effettiva nell'equazione precedente risulta essere $\bar{B}(t_0, t_N)$. Invece, nel caso in cui più di un coupon cada al di fuori dell'ultimo valore noto della Zeta-spread curve (ovvero ci sono due o più incognite) il sistema che ne risulta è indeterminato. Tramite `build_system_of_eqn`, creiamo tale sistema di equazioni che poi risolveremo grazie a `fsolve`.

`build_system_of_eqn` Il sistema qui implementato è scritto in funzione dello Zeta-spread: la prima equazione è un'immediata estensione della formula precedentemente utilizzata per $\overline{P}(t_0, T = t_N; \mathbf{c,t})$ in cui abbiamo separato la sommatoria in due parti (termini noti e termini incogniti) e scritto $\overline{B}(t_0, t_i)$ in funzione dello Zeta-spread, ottenendo

$$\sum_{j=N-flag+1}^{N} c_j B(t_0, t_j) e^{-(t_j - t_0) Z(t_j)} = \overline{P}(t_0, t_N; \mathbf{c,t}) - \sum_{i=1}^{N-flag} c_i B(t_0, t_i) e^{-(t_i - t_0) Z(t_i)}$$

Le altre $\texttt{flag} - \texttt{1}$ equazioni rappresentano la condizione di interpolazione lineare: in pratica in ciascuna di esse imponiamo che le pendenze dei segmenti che stiamo creando siano uguali. Per esempio, nel caso di 3 incognite, avremmo:

$$\begin{cases} Z(t_{N-2}) = Z(t_{N-3}) + \frac{Z(t_N) - Z(t_{N-3})}{t_N - t_{N-3}} (t_{N-2} - t_{N-3}) \\ Z(t_{N-1}) = Z(t_{N-3}) + \frac{Z(t_N) - Z(t_{N-3})}{t_N - t_{N-3}} (t_{N-1} - t_{N-3}) \end{cases}$$

dove $\big(t_{N-3}, Z(t_{N-3})\big)$ corrisponde all'ultimo dato boostrappato.

Dagli Zeta-spread siamo in grado di calcolare immediatamente anche i $\overline{B}(t_0, t_i)$ di cui abbiamo bisogno, terminando così il bootstrap della Zeta-spread curve.

Remark. Si noti che, nel caso di Santander, vi sono `payment_dates` coincidenti per i vari bond. Ciò crea dei problemi nella fase di interpolazione, in quanto `interp1`(X, V, Xq) richiede valori X in input che non si ripetano. La scelta più semplice sarebbe quella di applicare la function `unique` ad ogni iterazione del ciclo. Tuttavia, per avere una maggiore efficienza computazionale, abbiamo deciso di chiamare le funzione `unique` al di fuori dal ciclo, (con l'opzione *stable* per evitare che gli elementi vengano riordinati), salvando nella variabile `filter1` gli indici degli elementi non ripetuti.
A questo punto abbiamo utilizzato un'escamotage tecnico: supponiamo di ciclare sull'i-esimo bond. Esso pagherà i coupon salvati in `coupon_on_dates`(`idx1_cs`$(i)+1$: `idx1_cs`$(i+1)$): per semplicità di notazione definiremo `fd` = `idx1_cs`$(i)+1$ e `ld` = `idx1_cs`$(i+1)$. Potremo perciò interpolare sugli Zeta-spread noti fino all'iterazione precedente (compiendo un "troncamento"), ma che dobbiamo pure "filtrare", al fine di avere l'unicità del valore richiesta da `interp1`. Tuttavia le due operazioni sono a priori incompatibili: se prima tronchiamo e poi filtriamo, il vettore `filter1` eccede certamente la dimensione dell'array troncato; viceversa, se prima filtriamo, potrebbero cambiare gli indici del vettore dei `deltas` (le X su cui interpoliamo) e potremmo non sapere più esattamente dove troncare.
La soluzione che proponiamo è di fare prima un troncamento accorto e poi applicare il filtro. Nello specifico, poiché per costruzione i `deltas` devono assumere per forza valori positivi, nella riga 69 consideriamo `deltas`$(1$: `fd` $- 1)$ e, per non variare la lunghezza del vettore aggiungiamo $-$(`fd` : `idx1_cs`(`end`))$'$, scelti appositamente come l'opposto degli indici in modo tale da essere tutti negativi (per non interpolare mai su questi punti) e diversi (per il problema di `interp1`).

Tornando in `liquidity_spread_bounds` (riga 51), avendo tutti gli elementi necessari, calcoliamo upper e lower bound di ogni bond, utilizzando l'indice `idx2` precedentemente definito per considerare solo i coupon pagati dopo il time-to-liquidate.
Ultima nota è che il codice potrebbe essere generalizzato utilizzando un input vettoriale per i `ttl` per evitare di ripetere operazioni (`bootstrap_Z_curve`) indipendenti dal `ttl`. Nel *run_file* chiamiamo `liquidity_spread_bounds` per BNPP e per Santander nei due casi di ttl richiesti, calcoliamo la differenza in valore assoluto tra upper e lower bound, che rappresentiamo in Figura 5 e Figura 6:

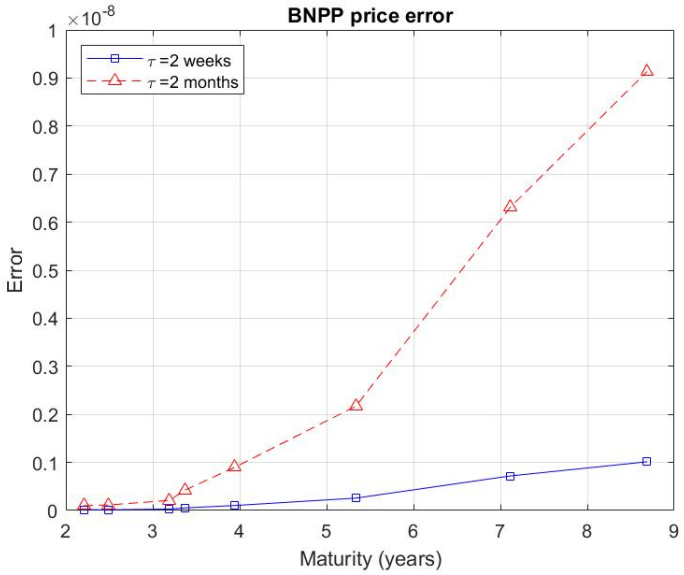


Figure 5: Differenza tra upper e lower bound per l'illiquidity price Δ_τ per i bond BNPP. Consideriamo illiquid bond con le stesse caratteristiche dei bond BNPP del paper [3] con ttl pari a 2 settimane (linea continua blu e quadrati) e a 2 mesi (linea tratteggiata rossa e triangoli). La differenza è dell'ordine di 10^{-8} del face value nel caso peggiore, quindi è trascurabile dal punto di vista pratico.

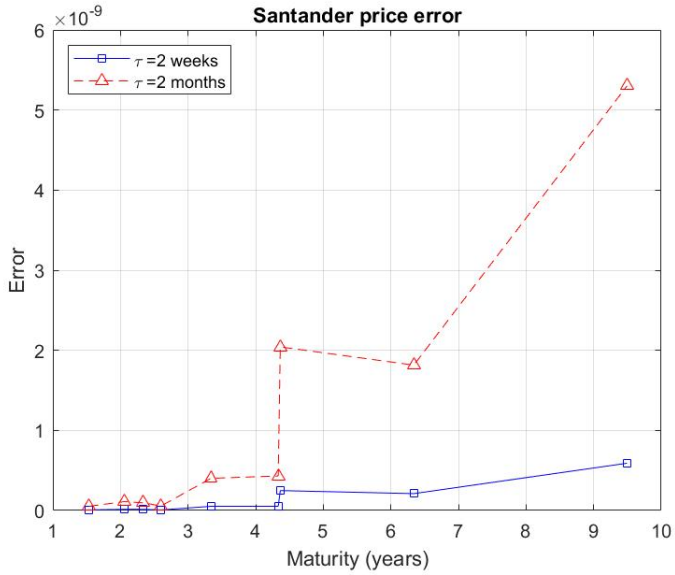


Figure 6: Differenza tra upper e lower bound per l'illiquidity price Δ_τ per i bond Santander. Consideriamo illiquid bond con le stesse caratteristiche dei bond Santander del paper [3] con ttl pari a 2 settimane (linea continua blu e quadrati) e a 2 mesi (linea tratteggiata rossa e triangoli). La differenza è dell'ordine di 10^{-9} del face value, quindi è trascurabile dal punto di vista pratico.

Bond yields Il nostro obiettivo è calcolare i bond yields di BNPP e Santander il 10 settembre 2015 per il liquid bond con time-to-liquidate pari a 2 settimane e a 2 mesi. Analogamente al punto precedente, i dati che utilizziamo sono quelli presenti nel paper [3]. Nella pratica comune, si è soliti considerare il *liquidity yield spread* come il termine che dovrebbe essere aggiunto allo yield per ottenere il prezzo dell'illiquid bond:

$$\overline{\overline{P}}(t_0, T; \mathbf{c},\mathbf{t}) =: \sum_{i=1}^{N} c_i e^{-[\mathcal{Y}(T)+\mathcal{L}_\tau(T)](t_i-t_0)} \tag{29}$$

dove $\mathcal{Y}(T)$ è lo yield del liquid bond $\overline{P}(t_0, T; \mathbf{c},\mathbf{t})$ corrispondente e $\mathcal{L}_\tau(T)$ è il liquidity yield spread per ttl pari a τ.
Implementiamo dunque la funzione `bond_yield`:

- come prima, ricaviamo le date di pagamento dei coupon di ogni bond tramite `create_vector_payment_dates` e, per avere vettori della stessa lunghezza, `coupon_on_dates` è ottenuto replicando `idx1` volte il valore del coupon pagato dal corrispondente bond e aggiungendo il face value nell'ultima data di pagamento, ossia la maturity.

- Calcoliamo poi il prezzo dell'illiquid bond come indicato nella formula (18) del paper [3]:

  $$\overline{\overline{P}}(t_0, T; \mathbf{c},\mathbf{t}) := \overline{P}(t_0, T; \mathbf{c},\mathbf{t}) - \Delta_\tau$$

  dove $\overline{P}(t_0, T; \mathbf{c},\mathbf{t})$ sono i dirty prices che avevamo già calcolato per il punto precedente e Δ_τ è approssimato dall'upper bound calcolato al punto precedente, consapevoli di commettere un errore trascurabile (dell'ordine di 10^{-8} rispetto al face value).

- Ricaviamo poi lo yield $\mathcal{Y}(T)$ del liquid bond risolvendo tramite `fsolve` la seguente equazione:

  $$\overline{P}(t_0, T; \mathbf{c},\mathbf{t}) = \sum_{i=1}^{N} c_i e^{-\mathcal{Y}(T)(t_i-t_0)}.$$

- Infine ricaviamo, sempre tramite `fsolve`, la somma dello yield $\mathcal{Y}(T)$ del liquid bond e del liquidity yield spread $\mathcal{L}_\tau(T)$, risolvendo l'equazione (29). Per trovare il liquidity yield è poi sufficiente sottrarre $\mathcal{Y}(T)$ al valore appena calcolato.

Rappresentiamo i risultati ottenuti nella Figura 7 per BNPP e nella Figura 8 per Santander.

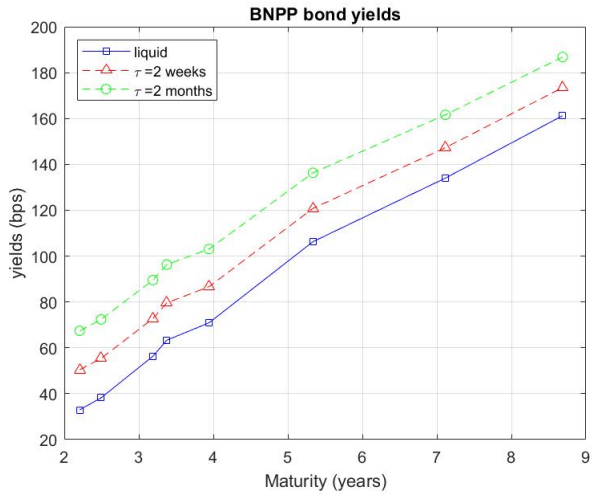


Figure 7: BNPP bond yields. Consideriamo tutte le benchmark issues con maturity inferiore a 10 anni riportate nel paper [3] e i rispettivi yields (linea continua blu e quadrati). Mostriamo anche lo yield ottenuto dagli illiquid bonds con le stesse caratteristiche con ttl pari a 2 settimane (linea tratteggiata rossa e triangoli) e a 2 mesi (linea tratteggiata verde e pallini).

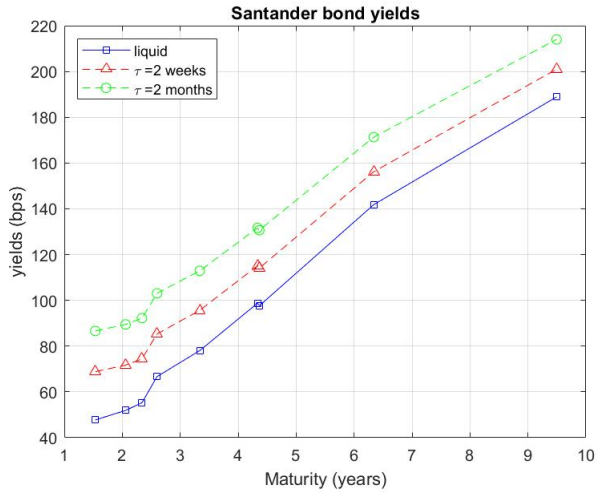
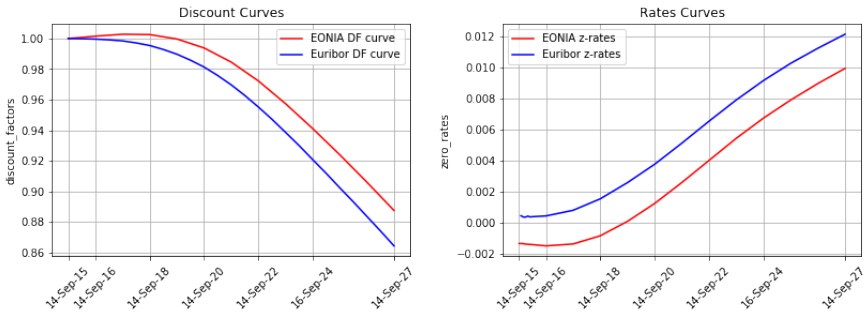


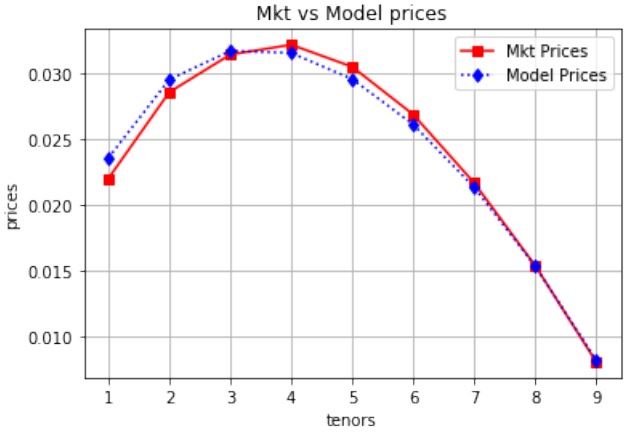
Figure 8: Santander bond yields. Consideriamo tutte le benchmark issues con maturity inferiore a 10 anni riportate nel paper [3] e i rispettivi yields (linea continua blu e quadrati). Mostriamo anche lo yield ottenuto dagli illiquid bonds con le stesse caratteristiche con ttl pari a 2 settimane (linea tratteggiata rossa e triangoli) e a 2 mesi (linea tratteggiata verde e pallini).

Python Abbiamo realizzato questo progetto anche in Python, seguendo lo stesso procedimento che abbiamo usato in Matlab. Ovviamente abbiamo dovuto implementare anche delle funzioni di base, come `yearfrac`, che sono già presenti su Matlab ma inesistenti in Python.
I risultati ottenuti coincidono con quelli precedenti; riportiamo qui solo i grafici.

Punto 1 Per quanto riguarda il multicurve-bootstrap otteniamo:



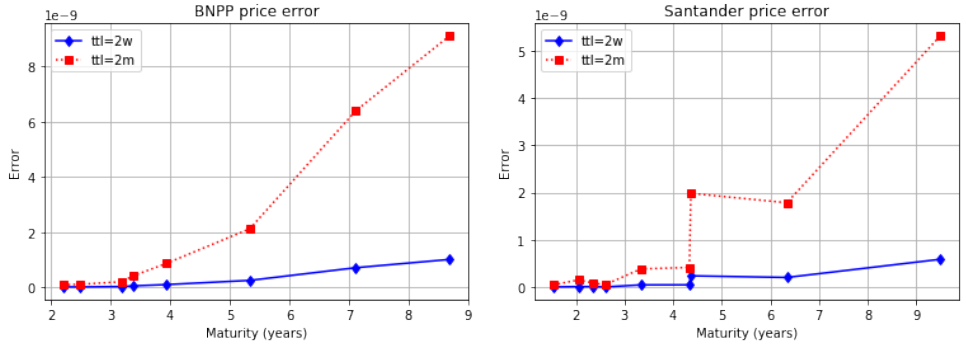
Punto 2 Il confronto dei prezzi delle ATM CS receiver swaptions è riportato nella seguente figura



Remark. In questo caso riscontriamo un risultato più attinente a quanto riportato nel paper [1], rispetto a quello ottenuto con Matlab. Dopo un'analisi approfondita siamo giunti alla conclusione che tale differenza sia dovuta ai diversi metodi di quadratura utilizzati dai due framework. In Python abbiamo usato la funzione `quad` (fornita dall'ecosistema `scipy`) che sfrutta l'algoritmo

di quadratura di *Clenshaw-Curtis*. Tale algoritmo si basa sull'approssimazione dell'integranda tramite polinomi di Chebyshev. In Matlab si è utilizzato l'algoritmo adaptive *Gauss-Kronrod quadrature* (tramite la funzione `quadgk`). Tale routine è un'estensione dell'algoritmo di quadratura gaussiana. Si noti che cambiando il metodo di integrazione in Matlab (e.g. usando la funzione `quad`) si ottengono risultati più vicini a quelli ottenuti con la routine di Python.

Punto 3 La differenza tra upper e lower bounds del liquidity spread per BNPP e Santander è



Punto 4 Infine i bond yields di BNPP e di Santander sono riportati di seguito