Principal Protected Notes

Ranit Bagchi

- **Tree Nodes:** Given a Willow-Tree structure on $[0, T]$, where $t_0 = 0$, $t_1 = \Delta t$, $t_2 = 2\Delta t$, $\ldots$, $t_N = N\Delta t = T$. The tree nodes are represented by S_i^n, where i represents the asset price at the i-th index at the n-th time step for $n = 1, 2, \ldots, N$, $i = 1, 2, \ldots, m$.

- **Transition Matrices:** The matrices $P^n = [P_{ij}^n]_{m\times m}$ represent the probabilities of transitions between different time steps, where $n = 1, 2, \ldots, N-1$. Additionally, $q^0 = [q_i^0]_{1\times m}$ represents the transitions between S_0 and the first time step Δt.

- **r:** Risk-free rate.

- **Payoff:** The payoff function describes the valuation of the note **V** at the asset price nodes described in the **Tree Nodes** variable.

0.1 Path-independent note, where pricing is determined based on the discounted payoff derived from the return of the underlying index

- **Payoff Function:** The Index Return is given by:

$$\text{Index Return} = \frac{\text{Final Index} - \text{Initial Index}}{\text{Initial Index}}$$

  The total payoff is calculated as:

$$\text{Total Payoff} = \max\left(\text{Index Return} \times S_0 \times \text{Participation Rate}, S_0\right)$$

  where S_0 represents the initial equity price/level, and the Participation Rate is the percentage of the performance of the underlying asset used to compute the return.

- **Similar Notes - Index Decline:** Index Decline notes return the principal amount if the index returns are positive. Otherwise, they use the index decline as a measure of the total payoff and return:

$$\text{Total Payoff} = \text{Index Decline} \times S_0 \times \text{Participation Rate}$$

Algorithm 1 Payoff Calculation

```
function PAYOFF($S, S_0, p_r$)   ▷ Variables: $S$ = [m × 1] vector (asset prices), $S_0$ = initial asset price, $p_r$ = participation rate
    $r_{\text{index}} \leftarrow \frac{S-S_0}{S_0}$   ▷ Calculate return index
    if $r_{\text{index}} > 0$ then
        return $r_{\text{index}} \times S_0 \times p_r$   ▷ Return boosted payoff
    else
        return $S_0$   ▷ Return initial asset price if return is non-positive
    end if
end function
```

0.2 Main Loop for Backward Induction

Algorithm 2 Back Induction Algorithm

```
function BACKINDUCTION($S, S_0, P, r, T, N, p_r, q^0$)
    Initialize the willow tree structure for asset prices $S_{N\times m}$ and transition matrices $P^{(N-1)}_{m\times m}$
    $\Delta_t = \frac{T}{N}$
    $V^N = \text{payoff}(S^N, S_0, p_r)$
    for $n = N-1$ to $1$ do
        $V^n = P^{(n-1)} \cdot V^{(n+1)} \cdot e^{-r\Delta_t}$
    end for
    $V^0 = (q^0)^T \cdot V^1 \cdot e^{-r\Delta_t}$
    return $V^0$
end function
```

0.3 Autocallable Notes

Autocallable notes activate when the return at a price node exceeds the autocall level. The payoff depends on the fixed return.

- **Total Index Return i_r:**

$$i_r = \frac{\text{Final Index} - \text{Initial Index}}{\text{Initial Index}}$$

  Additional variables:

  - **Fixed Return f_r:** Used to calculate the variable return on a given valuation date.
  - **Excess Return** $e_r = \max((i_r - f_r) \times \text{ex}_r, 0)$ (where the excess return variable ex_r is usually 5%).

Autocall_Payoff: This function calculates the variable return of a stock price S at time t . S_0 represents the initial stock price. f_r and ex_r represent the fixed return and excess return variable respectively.

```
function AUTOCALL_PAYOFF($S, S_0, f_r, ex_r$)
    $r_{\text{index}} \leftarrow \frac{S-S_0}{S_0}$
    $e_r \leftarrow \max(0, (r_{\text{index}} - f_r) \times ex_r)$
    $V_r \leftarrow S_0 \times (1 + f_r + e_r)$
    return $V_r$
end function
```

0.4 Backward Induction for Autocallable Notes

The backward induction for autocallable notes differs slightly. At each time step, if the return at a price node exceeds the autocall level, the node is valued at the current autocall price instead of the expected value of the nodes above. Otherwise, we use the future expected value, calculated using backward induction.

Algorithm 3 Autocallable Backward Induction Algorithm

```
function BACKINDUCTION_AUTOCALL($S, S_0, P, ac\_lvl, r, q^0, p_r, T, N, m$)
    Initialize $V[N][m]$                                   ▷ Create the matrix
    $\Delta_T = \frac{T}{N}$
    for $i = 1$ to $m$ do
        if $S_i^N > S_0 \times \text{ac\_lvl}$ then
            $V_i^N = \text{payoff}(S_i^N, S_0, p_r)$
        else
            $V_i^N = S_0$                  ▷ If below autocall level, return principal
        end if
    end for
    for $n = N - 1$ to $1$ do
        $E_p^n = P^{(n-1)} \cdot V^{(n+1)} \cdot e^{-r\Delta_T}$
        for $i = 1$ to $m$ do
            if $S_i^n > S_0 \times \text{ac\_lvl}$ then
                $V_i^n = \text{payoff}(S_i^n, S_0, p_r)$
            else
                $V_i^n = [E_p]_i^n$
            end if
        end for
    end for
    $V^0 = (q^0) \cdot V^1 \cdot e^{-r\Delta_T}$
    return $V^0$
end function
```

1 Principal at Risk Notes

Unlike Principal Protected Notes, Principal at Risk Notes do not offer downside protection. While they provide a higher potential for profit, they come with significantly higher risk. Investors may lose part or all of their principal depending on the performance of the underlying asset.

1.1 Coupon Payments and Downside Barrier Protection

Principal at Risk Notes feature path independent notes which depend on the payoff on the final day , regular coupon payment notes , boosted notes , barrier protected notes and autocall options. Key terms include:

- **Payment Threshold (p_t):** A coupon is payable at the coupon rate c_r if the index return on the valuation date t (where $t = 1, 2, \ldots$) is greater than or equal to the Payment Threshold p_t.
- **Barrier Level (b_lvl):** A predefined threshold, usually set as a percentage of the initial index level, which provides conditional principal protection. If the final index level falls below the Barrier Level and the note has not been automatically called, the investor may receive less than the full principal at maturity.

Examples of barrier protected payoffs include the following payoff functions

1. **Standard Barrier**
   - **Index Return Calculation:**
   $$\text{Index Return} = \frac{\text{Final Index Level} - \text{Opening Index Level}}{\text{Opening Index Level}}$$
   - **Maturity Redemption Payment:**
     - If the closing index level on the final valuation date is greater than or equal to the barrier level:
     $$\text{Maturity Redemption Payment} = \text{Principal Amount}$$
     - If the closing index level on the final valuation date is less than the barrier level:
     $$\text{Maturity Redemption Payment} = \text{Principal Amount} \times (1 + \text{Index Return})$$
   - Additionally, there are regular coupon payments.
2. **Barrier with Participation Rate**
   - **Index Return Calculation:**
   $$\text{Index Return} = \frac{\text{Final Index Level} - \text{Opening Index Level}}{\text{Opening Index Level}}$$

- **Maturity Redemption Payment:**
  - If the index return is positive:

    Maturity Redemption Payment = Principal Amount×[1 + (Participation Rate × Index Ret

  - If the index return is less than or equal to zero and the closing index level is greater than or equal to the barrier level times the opening index:

    Maturity Redemption Payment = Principal Amount

  - If the closing index level is less than the barrier level times the opening index:

    Maturity Redemption Payment = Principal Amount×(1+Index Return)

We look at the backward induction process for regular coupon payments.

Algorithm 4 Backward Induction with Coupons

```
function BACKWARD_INDUCTION_WITH_COUPONS($P, [p]_{m\times m}^{(N-1)}, S, S_0, b\_lvl, c_r, p_t, T$)
    $\triangle t \leftarrow \frac{T}{N}$
    Initialize $V[N][m]$
    $V_N \leftarrow$ Final_Day_Payoff($P, S[N], S_0, b\_lvl, a\_lvl, c_r, p_t, N, N$)
    for $n = N - 1$ to 1 do
        payoff $\leftarrow$ calculate_coupon($P, S_n, S_0, b\_lvl, c_r, T, n$)
        product $\leftarrow [p]_{m\times m}^{(n-1)} \cdot V^{(n+1)} \cdot e^{-r\triangle t}$        ▷ Apply discounting factor
        $V_n \leftarrow$ product + payoff
    end for
    $V^0 \leftarrow q^0 \cdot V^1 \cdot e^{-r\triangle t}$        ▷ Calculate option value at time 0
    return $V^0$
end function
```

Algorithm 5 Autocall Payoff with Coupon Payments and Barrier Protection

```
function AUTOCALL_PAYOFF_PAR($P, S, S_0, b\_lvl, a\_lvl, c\_r, p\_t, T, t$)
    Initialize $V[N][m]$
    returns ← $\frac{S-S_0}{S_0}$
    count ← 0
    for $x = 1$ to length(S) do
        if $S[x] \geq S_0 \times a\_lvl$ then
            $V$[count] ← $P \times (1 + c\_r)$                ▷ Autocall condition met
        else if $S_0 \times b\_lvl \leq S[x] < S_0 \times a\_lvl$ then
            if $t == T$ then                                  ▷ Maturity date condition
                if returns[$x$] $\geq p\_t$ then
                    $V[x] \leftarrow P \times (1 + c\_r)$
                else
                    $V[x] \leftarrow P$                       ▷ Full principal return
                end if
            else                                              ▷ Interim valuation date
                if returns[$x$] $\geq p\_t$ then
                    $V[x] \leftarrow P \times c\_r$           ▷ Coupon payment
                else
                    $V[x] \leftarrow 0$
                end if
            end if
        else                                                  ▷ Barrier breached
            if $t == T$ then
                $V[x]$ ← $P \times (1 +$ returns[$x$]$)$       ▷ Pay based on returns
            else
                $V[x] \leftarrow 0$
            end if
        end if
        count ← count + 1
    end for
    return $V$
end function
```

Algorithm 6 Backward Induction with Coupons

```
function BACKWARD_INDUCTION_WITH_COUPONS($P, [p]_{m\times m}^{(N-1)}, S, S_0, b\_lvl, a\_lvl, c_r, p_t, T$)
    $\triangle t \leftarrow \frac{T}{N}$
    Initialize $V[N][m]$;
    $V^N \leftarrow$ Autocall_Payoff_Par($P, S[N], S_0, b\_lvl, a\_lvl, c_r, p_t, N, N$)
    for $n = N - 1 : 1 : -1$ do
        $payoff \leftarrow$ calculate_autocall_coupon($P, S_n, S_0, b\_lvl, a\_lvl, c_r, p_t, T, n$)
        $product \leftarrow [p]_{m\times m}^{(n-1)} \cdot V^{(n+1)} \cdot e^{-r\triangle t}$        ▷ Apply discounting factor
        if $payoff == P \times (1 + c_r)$ then
            $V_n \leftarrow payoff$        ▷ Update option value tree at time for autocall
        else if $payoff == P \times c_r$ then
            $V_n \leftarrow payoff + product$
        else
            $V_n \leftarrow product$
        end if
    end for
    $V^0 \leftarrow q^0 \cdot V^1 \cdot e^{-r\triangle t}$        ▷ Calculate option value at time 0
    return $V^0$,
end function
```

1.2 Adding Booster to Final Payouts: Boosted Payments

- If the Index Return is between the Boost Level b_l and the Boosted Return b_r:
  - The investor receives a payment equal to a predetermined Boosted Return:

$$\text{Investor Return} = \text{Principal Amount} \times (1 + \text{Boosted Return})$$

- If the Index Return is above the Boosted Return b_r:
  - The investor receives a payment equal to the Boosted Return plus the amount by which the Index Return exceeds the Boosted Return, multiplied by a Participation Rate:

$$\text{Investor Return} = \text{Principal Amount} \times [1 + \text{Boosted Return} + (\text{Participation Rate} \times \text{Net Basket}$$

    where Net Basket = Basket Return − Boosted Return.

Algorithm 7 Payoff Function for Boosted Returns

```
function PAYOFF_BOOSTED($P, S, S_0, b_r, b_l, p_r$)    ▷ Where $S$ is an $m \times 1$ vector
    payoff ← []
    for price in $S$ do
        basket_return ← $\frac{\text{price} - S_0}{S_0}$
        net_basket_return ← basket_return - $b_r$
        if basket_return $\geq b_r$ then
            payoff.append($P \times (1 + b_r + (p_r \times net\_basket\_return))$)
        else if $b_l \leq$ basket_return $\leq b_r$ then
            payoff.append($P \times (1 + b_r)$)
        else
            payoff.append($P \times (1 + basket\_return)$)
        end if
    end for
    return payoff
end function
```

The payoff boosted function is used on the final time step in our stock price tree and then we use Algorithm 2 to find the valuation at time 0 since this note is path-independent and only depends on the final day's valuations.