The Costs of Counterparty Risk in Long Term Contracts Code Guide - Section 4

Michael Duarte Gonçalves

July 4, 2025

Overview

This section establishes the **baseline model with counterparty risk (CPR)**. The analysis simulates how contract markets operate when sellers face the risk that buyers may default or fail to honor contracts, parameterized by the share of opportunistic buyers γ.

Contents

1 Modeling Markets With Counterparty Risk (CPR) - Baseline

1.1 Theoretical Methodology

Seller Profits under CPR Sellers who enter a contract at price $f \in (0, 1)$ earn profits:

$$\Pi_S(f; \gamma; C) = q_i x \left[\gamma \int_0^f p\phi(p)\, dp + f \cdot \Big(1 - \gamma\Phi(f)\Big) \right] - R_i(f, \gamma) - C(k_i). \tag{1}$$

where:

q_i: is the total production of plant i, in MWh.

$\phi(p)$, $\Phi(p)$: represent respectively the PDF and CDF of the Beta distribution.

$R_i(f, \gamma)$ displays the risk premium for each plant:

$$R_i(f, \gamma) = r_0\, \gamma\, q_i^2\, x^2 \left[Var(\tilde{p}) + (1 - \gamma) \times \Big(f - \mathbb{E}(\tilde{p})\Big)^2 \right].$$

- **Important:** $\tilde{p} = \min\{f, p\}$, where $p \sim \text{Beta}(\alpha, \beta)$, represents a truncated Beta random variable. We will see the exact computation afterwards.

$C(k_i)$: total costs of plant i with capacity k_i.

Spot Market Profits For comparison, profits from participating in the spot market are given by:

$$\Pi_S^0 = q_i x \cdot \mathbb{E}(p) - C(k_i) - r_i, \tag{2}$$

where $r_i = r_0 (q_i x)^2 \cdot Var(p)$.

Constraints

- Contract prices f_c^* need to satisfy two constraints:

  1. **Break-even constraint**: Contract profits (1) have to be non-zero. For each plant, we compute the f^c that equates (1) to zero.

  $$\Pi_S(f; \gamma; c) = 0.$$

  2. **Spot market constraint**: f_c has to such that the contract is more profitable than trading in the spot market. For each plant, we compute the f_{spot} that equates (1) and (2).

  $$\Pi_S(f; \gamma; c) = \Pi_S^0(c).$$

  3. For each plant, the chosen price is:

  $$f_{max} = \max\{f_c, f_{spot}\}.$$

  (a) For each plant, we find the maximum between xf_c and xf_{spot}, which is represented by variable f_{max}. *Note:* that all those plants with positive spot market profits have $xf_{spot} > xf_c$.

  (b) We rank plants in increasing order according to their f_{max}, and plot the cumulative curve using the plant's production (contract supply curve). For instance, suppose that we have two plants:

i. plant A has $xf_c = 10, xf_{spot} = 20$ ($f_{max} = 20$) and production 100;

ii. plant B has $xf_c = 30$, $xf_{spot} = 10$ ($f_{max} = 30$) and production 50.

Then, the curve is $p = 20$ up to quantity 100 and $p = 30$ from 100 to 150.

1.2 Numerical Methods

1.2.1 Compute the truncated random variable $\tilde{p}$

Let $p \sim \text{Beta}(\alpha, \beta)$, and define the truncated variable:

$$\tilde{p} = \min\{p, f\}, \quad \text{for } f \in [0, 1]$$

We aim to compute:

$$\mathbb{E}[\tilde{p}] = \int_0^f p \cdot \phi(p)\, dp + f \cdot \mathbb{P}(p > f),$$

$$\mathbb{E}[\tilde{p}^2] = \int_0^f p^2 \cdot \phi(p)\, dp + f^2 \cdot \mathbb{P}(p > f),$$

where $\phi(p)$ is the PDF of the Beta distribution:

$$\phi(p) = \frac{1}{\text{Beta}(\alpha, \beta)} p^{\alpha-1}(1-p)^{\beta-1}.$$

1.2.2 Use of Incomplete Beta Functions

We avoid direct integration by using known properties of the Beta distribution:

$$\int_0^f p \cdot \phi(p)\, dp = \mathbb{E}[p] \cdot \text{Beta CDF}(f; \alpha + 1, \beta)$$

$$\int_0^f p^2 \cdot \phi(p)\, dp = \mathbb{E}[p^2] \cdot \text{Beta CDF}(f; \alpha + 2, \beta)$$

where:

$$\mathbb{E}[p] = \frac{\alpha}{\alpha + \beta}, \quad \mathbb{E}[p^2] = \frac{\alpha(\alpha+1)}{(\alpha+\beta)(\alpha+\beta+1)}$$

We define in the code:

$$I_1 = \text{Beta CDF}(f; \alpha + 1, \beta), \quad I_2 = \text{Beta CDF}(f; \alpha + 2, \beta)$$

These correspond to incomplete Beta functions and represent cumulative distribution values of related Beta distributions, used to reweight the moments over the truncated domain.

1.2.3 Related R Code

1. Compute Moments of $\tilde{p}$:

```r
compute_tilde_p_stats <- function(f, alpha, beta) {

  # --- Purpose ---
  # Compute the first moment (mean) and variance of a Beta-distributed
     variable,
  # where the variable p ~ Beta(alpha, beta) is truncated at value f.
  # That is, tilde_p = min(p, f). The moments of tilde_p are:

    #   E[tilde_p]
    #   E[tilde_p^2]

  # Compute the cumulative probability up to the truncation threshold f
  # This gives the total probability mass from 0 to f

  cdf_f <- pbeta(f, alpha, beta)

  # Compute the tail probability beyond the truncation threshold f
  # This is the probability that p > f

  one_minus_cdf <- 1 - cdf_f

  # Compute incomplete Beta function values for moment calculations
  # These represent the cumulative probabilities under adjusted Beta
  # dist.,
  # which arise from integrating p * Beta(p) and p^2 * Beta(p)

  # I1 corresponds to the integral of p * Beta(p), which behaves like
     Beta(alpha + 1, beta)
  I1 <- pbeta(f, alpha + 1, beta)

  # I2 corresponds to the integral of p^2 * Beta(p), which behaves like
      Beta(alpha + 2, beta)
  I2 <- pbeta(f, alpha + 2, beta)

  # Compute the expected value (first moment) of tilde_p ---
  # Formula:
  #   E[tilde_p] = (E[p] for p in [0, f]) + f * P(p > f)
  #              = (alpha / (alpha + beta)) * I1 + f * (1 - CDF(f))


  first_moment <- (alpha / (alpha + beta)) * I1 + f * one_minus_cdf
  # Same as:

  # first_moment <- integrate(function(p) p * beta_pdf(p),
  #                           lower = 0, upper = f)$value +
  #   f * (1 - beta_cdf(f))

  # Compute the second moment of tilde_p
  # Formula:
  #   E[tilde_p^2] = (E[p^2] for p in [0, f]) + f^2 * P(p > f)
  #                = (alpha(alpha+1) / ((alpha+beta)(alpha+beta+1))) *
     I2 + f^2 * (1 - CDF(f))

  second_moment <- (alpha * (alpha + 1)) / ((alpha + beta) * (alpha +
     beta + 1)) * I2 +
```

```
    f^2 * one_minus_cdf

  # Compute the variance of tilde_p

  # Var[tilde_p] = E[tilde_p^2] - (E[tilde_p])^2
  var_tilde_p <- second_moment - first_moment^2

  # --- Return a named list containing the results ---
  return(list(E_tilde_p = first_moment, Var_tilde_p = var_tilde_p))
}
```

2. Compute $R_i(f,\gamma)$:
Given the computation of $\tilde{p}$, we can now estimate $R_i(f,\gamma)$:

$$R_i(f,\gamma) = r_0\,\gamma\,q_i^2\,x^2\left[Var(\tilde{p}) + (1-\gamma)\times\Big(f - \mathbb{E}(\tilde{p})\Big)^2\right].$$

```
# Compute R(f, \gamma) with compute_tilde_p_stats

# Function to compute R_i(f, gamma)
compute_R_value_gamma <- function(f, gamma, q_i, x, r_0, alpha, beta) {
  # Compute the tilde_p statistics
  stats <- compute_tilde_p_stats(f, alpha, beta)
  E_tilde_p <- stats$E_tilde_p
  Var_tilde_p <- stats$Var_tilde_p

  # Compute R_i(f, gamma) using the given formula
  R_value_gamma <- r_0 * gamma * (q_i^2) * (x^2) *
    (Var_tilde_p + (1 - gamma) * (f - E_tilde_p)^2)

  return(R_value_gamma)
}
```

1.2.4 Profit Functions Creation

Afterwards, we create two functions that mimic equations (1) and (2):

- `Pi_S_general()`: profit function under CPR scheme (equation (1))
- `Pi_S0()`: baseline spot market profit function (equation (2))

Each function captures revenues and costs under their respective market rules, factoring in production uncertainty via a truncated Beta distribution.

`Pi_S_general` :

```
Pi_S_general <- function(f, q_i, x, gamma, r_0, alpha, beta, total_
    costs, T_values) {
  # Compute            p    (p) dp numerically
  integral_result <- integrate(function(p) p * dbeta(p, alpha, beta),
                               lower = 0, upper = f)$value

  # Compute Phi(f)
  Phi_f <- pbeta(f, alpha, beta)

  # Compute R_i(f, gamma)
  R_value_gamma <- compute_R_value_gamma(f, gamma, q_i, x, r_0, alpha,
      beta)

  # Final profit
  profit <- q_i * x * (gamma * integral_result + f * (1 - gamma * Phi_f
      )) -
    R_value_gamma - total_costs + (T_values * q_i)

  return(profit)
}

# Vectorized version
vectorized_pi_s <- Vectorize(Pi_S_general)
```

This function computes the producer's expected profit under a CPR contract, where:

$$\Pi_S(f) = q_i x \left[\gamma \cdot \int_0^f p \cdot \phi(p)\, dp + f \cdot (1 - \gamma \cdot \Phi(f))\right] - R(f,\gamma) - C(k_i) + T \cdot q_i$$

Important: Here, we're creating a general function that can also be used on other occasions (for example in the case of public subsidies, hence the presence of $T \cdot q_i$). In the Baseline results, there's no T, so the function simply becomes:

$$\Pi_S(f) = q_i x \left[\gamma \cdot \int_0^f p \cdot \phi(p)\, dp + f \cdot (1 - \gamma \cdot \Phi(f))\right] - R(f,\gamma) - C(k_i)$$

`Pi_S0`: Below, we are creating a function that produces the profits in the spot market, Π_S^0:

```
Pi_S0 <- function(q_i, x, expected_p, total_cost, r) {
  revenue <- q_i * x * expected_p
  profit <- revenue - total_cost - r
  return(profit)
}
```

which is basically this equation:

$$\Pi_S^0 = q_i x \cdot \mathbb{E}[p] - C(k_i) - r_i$$

1.2.5 Roots Finding

```r
# --- Why we use a grid search before uniroot() ---
  # The profit function Pi_S(f) is not guaranteed to be monotonic.
  # It may have multiple roots or no root at all.
  # uniroot() only works if we supply an interval [a, b] where
  # the function changes sign - i.e., f(a) times f(b) < 0.
  # To ensure this, we evaluate the function on a grid and
  # look for intervals with a sign change before calling uniroot().

find_sign_change_index <- function(values, nth = 1) {
  signs <- sign(values)
  change_locs <- which(diff(signs) != 0)
  if (length(change_locs) < nth) return(NA_integer_)
  return(change_locs[nth])
}

# Root where Pi_S_general crosses zero
find_f_root <- function(q_i, x, gamma, r_0, alpha, beta, total_costs, T
   _values,
                        f_min = 0, f_max = 1, n = 100, tol = 1e-20) {
  f_grid <- seq(f_min, f_max, length.out = n)

  profits <- vapply(
    f_grid,
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
       costs, T_values),
    numeric(1)
  )

  if (max(profits, na.rm = TRUE) < 0) return(NA_real_)

  i <- find_sign_change_index(profits, nth = 1)
  if (is.na(i)) return(NA_real_)

  root <- uniroot(
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
       costs, T_values),
    lower = f_grid[i], upper = f_grid[i + 1], tol = tol
  )$root

  return(root)
}
```

We aim to solve for the value(s) of $f \in [0, 1]$ for which a profit function $\Pi_S(f)$ becomes zero:

$$\Pi_S(f) = 0.$$

This equation may have multiple roots, no root, or a complex shape due to the potentially non-monotonic nature of $\Pi_S(f)$. Figure 1 shows an example of a non-monotonic function that we have in our dataset.

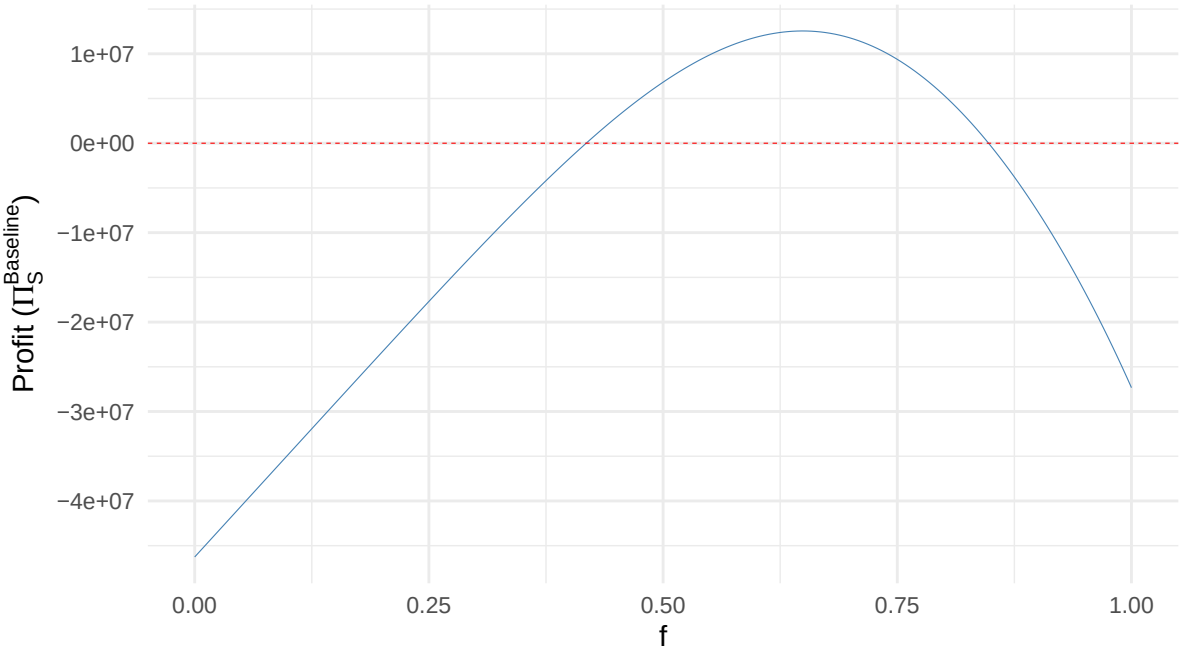


Figure 1: Francisco Pizarro Solar Farm 12, with $\gamma = 0.5$

Here is a small, but concise explanation of this code chunk:

- `find_sign_change_index()`: This helper function detects where a numeric vector (e.g., a profit function) changes sign. It uses the `sign()` function to extract signs ($-1, 0, +1$), then checks where the sign changes (`diff(signs) != 0`). It returns the index of the `nth` such sign change (that we define to be the first one). If there are no sign changes (so the function never crosses the 0 threshold), it returns `NA`.
- `find_f_root()`: This function numerically finds the root (zero-crossing point) of the profit function `Pi_S_general`. The steps are:
  1. A grid of values for f is generated using `seq()` between `f_min` and `f_max`. By definition, between 0 and 1.
  2. The profit function $\Pi_S^{general}(f)$ is evaluated over this grid using `vapply()`.
  3. If all profits are negative, it concludes no crossing occurs above zero, and returns `NA`.
  4. It then calls `find_sign_change_index()` to detect the first interval where the profit changes sign. Again, if there is no sign changes, the algorithm returns `NA`.
  5. If the first sign change is found, it calls `uniroot()` over the first sign change that interval to compute the root more precisely.
- This approach avoids direct use of `uniroot()` on unknown intervals. It ensures that the root-finding is safe and robust, particularly because the profit function is not guaranteed to be monotonic.

```r
find_f_spot_root <- function(q_i, x, gamma, r_0, alpha, beta, total_
    costs, expected_p, r, T_values, f_min = 0, f_max = 1, n = 100, tol =
     1e-20) {
  f_grid <- seq(f_min, f_max, length.out = n)

  pi_s0_val <- Pi_S0(q_i, x, expected_p, total_costs, r)

  g_vals <- vapply(
    f_grid,
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
        costs, T_values) - pi_s0_val,
    numeric(1)
  )

  i <- find_sign_change_index(g_vals, nth = 1)
  if (is.na(i)) return(NA_real_)

  root <- uniroot(
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
        costs, T_values) - pi_s0_val,
    lower = f_grid[i], upper = f_grid[i + 1], tol = tol
  )$root

  return(root)
}
```

Explanation:

- This function computes the value of f such that the general profit function $\Pi_S(f)$ equals the spot-market-based profit Π_S^0. This breakeven value of f is important for determining competitiveness under different market designs.
- As before, `f_grid` creates a sequence of f values between `f_min` and `f_max` to scan for sign changes.
- `pi_s0_val` stores the profit under the spot market, using the function previously created above.
- The function then evaluates the difference $\Pi_S(f) - \Pi_S^0$ over a grid of f values. A sign change in this difference implies that the two profit functions intersect. We are effectively solving:

$$\Pi_S(f) = \Pi_S^0$$

- The first such sign change is identified using `find_sign_change_index()`, and a precise root is computed using `uniroot()` over the interval where the sign change occurs.
- If no such sign change is found (i.e., the curves never cross), the function returns `NA`.

```r
find_upper_root <- function(q_i, x, gamma, r_0, alpha, beta, total_
    costs, T_values,
                            f_min = 0, f_max = 1, n = 100, tol = 1e-20)
                                {
  f_grid <- seq(f_min, f_max, length.out = n)

  profits <- vapply(
    f_grid,
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
        costs, T_values),
    numeric(1)
  )

  i <- find_sign_change_index(profits, nth = 2)
  if (is.na(i)) return(NA_real_)

  root <- uniroot(
    function(f) Pi_S_general(f, q_i, x, gamma, r_0, alpha, beta, total_
        costs, T_values),
    lower = f_grid[i], upper = f_grid[i + 1], tol = tol
  )$root

  return(root)
}
```

Explanation:

- This function identifies the second value of f in the interval $[0, 1]$ for which the profit function (`Pi_S_general`) crosses zero for the second time.
- A grid of f-values is first created using `seq()`, and the profit function is evaluated at each point.
- Using `find_sign_change_index(profits, nth = 2)`, the function searches for the second location where the sign of profits changes — that is, where the curve crosses zero a second time.
- If no such second sign change is found, the function returns `NA`, indicating that the profit function is either monotonic or crosses zero only once.
- If the second sign change is found, `uniroot()` is used to accurately locate the second root of $\Pi_S^{\text{general}}(f) = 0$ within the narrow interval where the sign change occurs.

Why is this important?

- The profit function $\Pi_S^{\text{general}}(f)$ is not necessarily monotonic — it can increase and then decrease, resulting in multiple zero crossings, as in Figure 1.
- The first crossing typically identifies the smallest f that leads to non-negative profit. However, a second root may exist where profitability ends again due to rising risk or costs.
- Identifying the second root provides an upper bound on the range of viable f.

1.2.6 Dataset Creation

```
wind_solar_proj_2022_long <- wind_solar_proj_2022 |>
  crossing(gamma = gamma_values) |>   # create a line for which gamma
     defined above
  rowwise() |>
  mutate(
    # f_c_cpr computation
    f_c_cpr = coalesce(
      find_f_root(q_i = q_i_mwh,
                  x = x,
                  gamma = gamma,
                  r_0 = r_0,
                  alpha = alpha,
                  beta = beta,
                  total_costs = total_cost,
                  T_values = 0),
      1),

    # f_spot_cpr: value for f for which Pi_S_general - Pi_S0 = 0
    f_spot_cpr = coalesce(
      find_f_spot_root(q_i = q_i_mwh,
                       x = x,
                       gamma = gamma,
                       r_0 = r_0,
                       alpha = alpha,
                       beta = beta,
                       total_costs = total_cost,
                       expected_p = expected_p,
                       r = r,
                       T_values = 0),
      0),

    # f_upper_cpr: second root of Pi_S_general(f) = 0, if it exists
    f_upper = find_upper_root(q_i = q_i_mwh,
                              x = x,
                              gamma = gamma,
                              r_0 = r_0,
                              alpha = alpha,
                              beta = beta,
                              total_costs = total_cost,
                              T_values = 0)
  ) |>
  ungroup() |>
  mutate(
    f_upper_message = if_else(
      !is.na(f_upper),
      if_else(f_upper > expected_p, "f_upper > E(p)", "f_upper <= E(p)"
         ),
      NA_character_
    ),
    f_max_cpr = pmax(f_c_cpr, f_spot_cpr, na.rm = TRUE),
    xf_c_cpr = x * f_c_cpr,
    xf_spot_cpr = x * f_spot_cpr,
    xf_upper_cpr = x * f_upper,
    xf_max_cpr = x * f_max_cpr
  )
```

Explanation:

- This code processes our dataset of wind and solar projects by extending it across a set of values for the parameter γ using a Cartesian product with `crossing()`.
- For each row (i.e., each project and γ pair), we compute three key values of the policy parameter f:
  1. `f_c_cpr`: The first root of the profit function $\Pi_S(f) = 0$, i.e., the smallest f for which the project becomes profitable.
  2. `f_spot_cpr`: The breakeven point where the general profit equals the spot-based benchmark $\Pi_S(f) = \Pi_S^0$.
  3. `f_upper`: The second root of the profit function, if it exists — this marks the upper limit beyond which the project becomes unprofitable again.
- The results are used to derive additional indicators:
  - `f_upper_message` notes whether the second root lies above or below the expected price $\mathbb{E}(p) = \frac{2}{3}$.
  - `f_max_cpr` selects the maximum of the two candidates `f_c_cpr` and `f_spot_cpr` values for policy calibration.
  - `xf_c_cpr`, `xf_spot_cpr`, `xf_upper_cpr`, and `xf_max_cpr` scale each f-value by project-specific capacity factor x, which is used for ranking or cumulative analysis.

1.2.7 Contract Supply Curves

```
# Create of cumulative capacity and production

wind_solar_proj_2022_long <- wind_solar_proj_2022_long |>
  arrange(gamma, xf_max_cpr) |>
  group_by(gamma) |>
  mutate(cumulative_production = cumsum(q_i_mwh),
         cumulative_capacity = cumsum(capacity),
         x_q_exp_p_total_costs = x * expected_p * q_i_mwh - total_cost
  ) |>
  ungroup()


# We compute R_f_max_cpr_gamma. If we replace in the profits function
   Pi_S_general with
# f_max_cpr_gamma and R_f_max_cpr_gamma, the function Pi_S_general
   should be 0 for each line

wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long |>
  rowwise() |>
  mutate(R_f_max_cpr_gamma = compute_R_value_gamma(f_max_cpr, gamma, q_
     i_mwh, x, r_0, alpha, beta)) |>
  relocate(R_f_max_cpr_gamma, .after = gamma)


wind_solar_proj_2022_long_baseline <- crossing(
  wind_solar_proj_2022_long_baseline,
  theta = theta_values
) |>
  arrange(theta, xf_max_cpr, gamma)
```

Explanation:

- The first block augments the project-level dataset with additional cumulative statistics:
  - `cumulative_production` is the cumulative sum of expected electricity generation (q_i) per (γ), ordered by $xf_{\text{max\_cpr}}$.
  - `cumulative_capacity` accumulates plant capacity in the same order.
  - `x_q_exp_p_total_costs` computes $x \cdot \mathbb{E}[p] \cdot q_i - C(k_i)$ for each plant.
- In the second block, we compute `R_f_max_cpr_gamma`:
  - This is the value of the revenue offset term $R(f, \gamma)$ evaluated at the critical value $f_{\text{max\_cpr}}$ for each plant and gamma.
  - This value ensures that if substituted back into $\Pi_S^{\text{general}}(f)$, the profit should be exactly zero, confirming internal consistency.
- In the third block, we extend the baseline dataset over a grid of θ-values (2500, 3500, 4500) using `crossing()`, to prepare for further policy simulations or curve aggregation. The data is then sorted by θ, $xf_{\text{max\_cpr}}$, and γ to facilitate cumulative analysis and plotting.

```r
# Compute contract supply using xf_max_cpr

contract_supply_cpr <- wind_solar_proj_2022_long_baseline |>
  mutate(gamma = as.factor(gamma)) |>  # Ensure gamma is a factor for
     grouping
  group_by(gamma, xf_max_cpr, theta) |>
  summarise(
    total_capacity = sum(capacity, na.rm = TRUE),
    total_production = sum(q_i_mwh, na.rm = TRUE)
  ) |>
  ungroup() |>
  arrange(theta, gamma, xf_max_cpr) |>
  group_by(gamma, theta) |>
  mutate(
    cumulative_capacity = cumsum(total_capacity),
    cumulative_production = cumsum(total_production),
    q_0 = q_0,
    G_expected_p_x = cumsum(if_else(xf_max_cpr <= expected_p * x, total
       _capacity, 0)),
    gamma_num = as.numeric(as.character(gamma))  # Numeric for gradient
        coloring
  ) |>
  mutate(
    G_expected_p_x = last(G_expected_p_x, order_by = xf_max_cpr)
  ) |>
  ungroup()

# Plot loop by theta
for (i in seq_along(theta_values)) {
  theta_val <- theta_values[i]
  index_label <- sprintf("%02d", i)

  # Demand curve (vertical line)
  demand_segments <- tibble(
    x_start = theta_val,
    x_end   = theta_val,
    y_start = threshold_price,
    y_end   = min(contract_supply_cpr$xf_max_cpr)
  )

  # Filter for supply curve and create plot
  supply_cpr <- ggplot(
    contract_supply_cpr |>
      filter(xf_max_cpr <= expected_p * x),
    aes(
      x = cumulative_capacity,
      y = xf_max_cpr,
      group = gamma,          # Keeps step-line structure per gamma
      color = gamma_num       # Uses gradient color
    )
  ) +
    geom_step() +
    geom_point(size = 0.5) +
    geom_segment(
      data = demand_segments,
      aes(x = x_start, xend = x_end, y = y_start, yend = y_end),
      color = "black", linetype = "solid", size = 1,
```

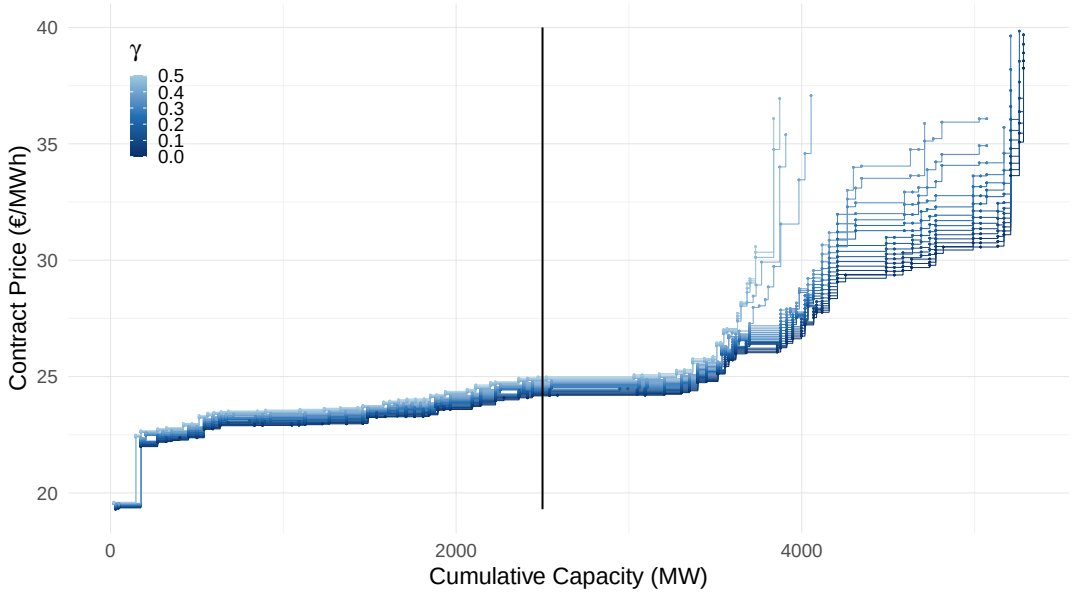
```
      inherit.aes = FALSE
    ) +
    scale_color_gradientn(
      colours = theme_palette_gamma,
      name = expression(gamma)
    ) +
    labs(
      x = "Cumulative Capacity (MW)",
      y = "Contract Price (EUR/MWh)"
    ) +
    theme_minimal(base_size = base_s) +
    theme(
      legend.position = c(0.05, 0.95),  # Top-left inside plot (x, y
         from 0 to 1)
      legend.justification = c(0, 1),  # Anchor top-left corner of the
         legend box
      legend.background = element_rect(
        fill = alpha("white", 0.2),  # Semi-transparent white
           background
        color = NA                              # No border
      ),
      legend.title = element_text(face = "bold"),
      panel.grid.major = element_line(color = "grey90", size = 0.2),
      panel.grid.minor = element_line(color = "grey95", size = 0.1)
    )

  print(supply_cpr)

  plot_filename <- paste0(index_label, "_supply_function_cpr_theta_",
     theta_val, ".pdf")
  plot_path_cpr <- file.path(baseline_fig_path, plot_filename)

  ggsave(plot_path_cpr, plot = supply_cpr, width = 16, height = 9, dpi
     = 300)

  message("Saved plot for theta = ", theta_val, " at: ", plot_path_cpr)
}
```

Explanation:

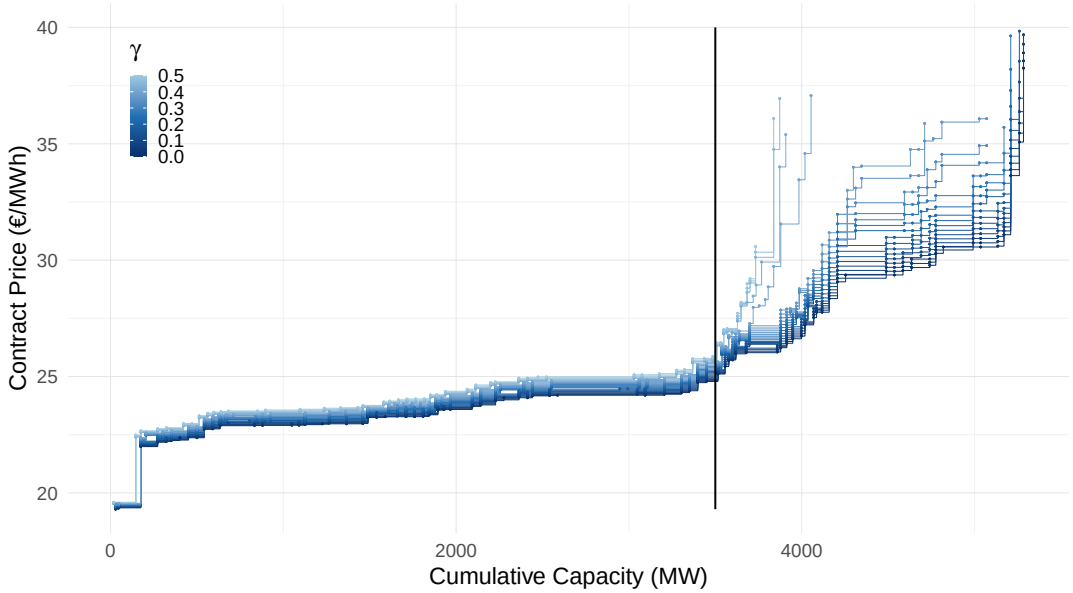
- This code block computes and visualizes the **contract supply function** for a set of renewable energy projects, under different levels of the design parameter θ.
- **Step 1: Aggregation**
  - Projects are grouped by γ, θ, and the computed contract price threshold $xf_{\text{max\_cpr}}$.
  - For each group, the total project capacity and production are summed.
  - Cumulative sums are then computed to create a stepwise supply curve ordered by price.
- **Step 2: Plotting**
  - For each θ, a supply curve is plotted showing how much cumulative capacity is available at or below each price.
  - The vertical black line represents demand at the given θ and intersects the supply curve at the equilibrium contract price.
  - Colors represent the value of γ, using a continuous color gradient.

- The result is a set of supply curves (one per θ) that illustrate how many projects would accept a given contract price under different risk-sharing schemes defined by γ.

Figure 2 represents some contract supplies generated with the above code.



(a) $\theta = 2500$



(b) $\theta = 3500$

Figure 2: Baseline Contract Supplies

1.2.8 Equilibrium Prices Dataset

```
# Equilibrium Prices Dataset Creation

# Pre-process the data once
wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline %>%
  arrange(gamma, cumulative_capacity)

# Compute equilibrium prices across theta values
equilibrium_prices <- purrr::map_dfr(theta_values, function(theta_val)
    {

  # Get the last row *before* theta (for each gamma), with price <
      threshold
  before_theta <- wind_solar_proj_2022_long_baseline %>%
    group_by(gamma) %>%
    filter(cumulative_capacity < theta_val, xf_max_cpr < threshold_
        price) %>%
    slice_tail(n = 1) %>%
    ungroup()

  # Get the first row *at or after* theta (for each gamma)
  after_theta <- wind_solar_proj_2022_long_baseline %>%
    filter(cumulative_capacity >= theta_val) %>%
    group_by(gamma) %>%
    slice_head(n = 1) %>%
    select(gamma, next_price = xf_max_cpr) %>%
    ungroup()

  # Join and determine equilibrium price logic
  before_theta %>%
    left_join(after_theta, by = "gamma") %>%
    mutate(
      equilibrium_price = if_else(
        !is.na(next_price) & next_price >= threshold_price,
        threshold_price,
        next_price
      ),
      equilibrium_quantity = cumulative_capacity,
      theta = theta_val
    ) %>%
    select(gamma, equilibrium_quantity, equilibrium_price, theta)
})

# Sort for plotting
equilibrium_prices <- equilibrium_prices %>%
  arrange(theta, gamma)
```

Explanation:

- This block computes the **contract market-clearing price (equilibrium)** for each θ and γ.
- For each value of θ, it finds the point at which cumulative supply meets the demand threshold.
- The price selected is either the first price at or above θ, or the threshold price if it is higher. The optimal quantity is the one just below crossing the vertical demand line θ.

- The result is a dataset with equilibrium prices and quantities for each combination of γ and θ.

1.2.9 Welfare Dataset

```
# Welfare Dataset Creation

wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline |>
  left_join(
    equilibrium_prices |>
      select(gamma, theta, equilibrium_price, equilibrium_quantity),
    by = c("gamma", "theta")
  )

wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline |>
  mutate(f_equilibrium = equilibrium_price / x,
         xf_equilibrium = equilibrium_price) |>
  select(-equilibrium_price)


wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline |>
  arrange(theta, gamma, xf_max_cpr) |>
  group_by(gamma) |>
  mutate(
    R_f_equilibrium_cpr = compute_R_value_gamma(
      f_equilibrium, gamma, q_i_mwh, x, r_0, alpha, beta
    )
  ) |>
  ungroup() |>
  mutate(
    x_q_exp_p_total_costs_R = x * expected_p * q_i_mwh - total_cost - R
        _f_equilibrium_cpr
  )


wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline |>
  mutate(
    profit_cpr_contracts = vectorized_pi_s(
      f = f_equilibrium,
      q_i = q_i_mwh,
      x = x,
      gamma = gamma,
      r_0 = r_0,
      alpha = alpha,
      beta = beta,
      total_costs = total_cost,
      T_values = 0
    )
  )
```

- **Step 1: Attach Equilibrium Contract Outcomes**

  Each project is joined with the market-clearing price and quantity obtained from the equilibrium market simulation:

  `equilibrium_price, equilibrium_quantity`

  These are matched by both γ and θ, reflecting risk-sharing and demand scenarios.

- **Step 2: Recover the Implied Contract Value $f_{\text{equilibrium}}$**

  $$\text{f\_equilibrium} \equiv \frac{\text{equilibrium\_price}}{x} = \frac{\cancel{x} f^*}{\cancel{x}}$$

  and define this value explicitly for use in downstream computations. The variable equilibrium_price represents xf^*. For consistency with the other naming conventions, we call this variable `xf_equilibrium`.

- **Step 3: Compute the Risk-Adjusted Transfer $R(f^*, \gamma)$**

  The function `compute_R_value_gamma()` is applied using the recovered $f_{\text{equilibrium}}$, for each plant.

- **Step 4: Compute Net Expected Revenue (Excluding Full Profit Function)**

  For each project, we compute a simplified economic surplus expression:

  $$x \cdot \mathbb{E}[p] \cdot q_i - C(k_i) - R(f^*, \gamma)$$

  N.B.: $C(k_i)$ is the variable `total_cost`.

- **Step 5: Evaluate Full Contract Profit $\Pi_S(f^*)$**

  Finally, we use a vectorized implementation of the general profit function to compute:

  $$\Pi_S(f^*, q_i, x, \gamma, r_0, \alpha, \beta, C(k_i), T = 0)$$

  The function `vectorized_pi_s` is defined earlier in the code (Subsection 1.2.4)

The resulting variables—especially `profit_cpr_contracts`—are used to determine plant-level selection under contracts and to aggregate social welfare. This step ensures that equilibrium results are not just theoretical, but tied to project-level outcomes consistent with the defined economic model.

```r
# Reorder dataframe
wind_solar_proj_2022_long_baseline <- wind_solar_proj_2022_long_
    baseline |>
  relocate(
    profits_sp_no_cpr, .after = last_col()
  ) |>
  relocate(all_of(ordered_vars), .after = profits_sp_no_cpr)

# STEP 1: Prepare list of unique gamma values
gammas <- wind_solar_proj_2022_long_baseline |>
  distinct(gamma) |>
  arrange(gamma)

# STEP 2: Compute W^0 (baseline welfare) for gamma = 0
W_0 <- wind_solar_proj_2022_long_baseline |>
  filter(gamma == 0, profits_sp_no_cpr >= 0) |>
  group_by(theta) |>
  summarise(
    W_0 = sum(x_q_exp_p_total_costs - r, na.rm = TRUE),
    .groups = "drop"
  ) |>
  ungroup()

# STEP 3: Compute W(gamma): welfare under contracts for each gamma
W_gamma <- wind_solar_proj_2022_long_baseline |>
  group_by(gamma, theta) |>
  summarise(
    welfare_gamma_eur = sum(
      if_else(xf_max_cpr <= xf_equilibrium & cumulative_capacity <
          theta,
               x_q_exp_p_total_costs_R, 0),
      na.rm = TRUE
    ),
    .groups = "drop"
  ) |>
  ungroup()

# STEP 4: Extract W(gamma) at gamma = 0 to use as reference in ratios
W_gamma_0 <- W_gamma |>
  filter(gamma == 0) |>
  select(theta, welfare_gamma_ref_eur = welfare_gamma_eur)

# STEP 5: Extract equilibrium quantity and price for gamma = 0
equilibrium_gamma_0 <- equilibrium_prices |>
  filter(gamma == 0) |>
  select(
    theta,
    eq_quantity_gamma_0 = equilibrium_quantity,
    eq_price_gamma_0 = equilibrium_price
  )

# STEP 6: Assemble final dataset
welfare_dataset_baseline <- equilibrium_prices |>
  left_join(W_gamma, by = c("gamma", "theta")) |>
  left_join(W_gamma_0, by = "theta") |>
  left_join(W_0, by = "theta") |>
  left_join(equilibrium_gamma_0, by = "theta") |>
```

```
  mutate(
    eq_price = equilibrium_price,
    eq_quantity = equilibrium_quantity,

    welfare_ratio_percent = (welfare_gamma_eur / welfare_gamma_ref_eur)
        * 100,
    welfare_gap_million_eur = (welfare_gamma_ref_eur - welfare_gamma_
       eur) / 1e6,
    welfare_gain_million_eur = (welfare_gamma_eur - W_0) / 1e6,

    eq_quantity_ratio_percent = (eq_quantity / eq_quantity_gamma_0) *
       100,
    eq_price_ratio_percent = (eq_price / eq_price_gamma_0) * 100
  ) |>
  select(
    gamma, theta,
    eq_price, eq_price_ratio_percent,
    eq_quantity, eq_quantity_ratio_percent,
    W_0, welfare_gamma_eur, welfare_ratio_percent,
    welfare_gap_million_eur, welfare_gain_million_eur
  ) |>
  arrange(theta, gamma)

welfare_dataset_baseline
```

This code block constructs a dataset to evaluate the welfare implications of different contract schemes (indexed by γ) across a range of total demand levels θ. It combines equilibrium outcomes, plant-level profitability, and cumulative project selection logic. Below is a step-by-step explanation:

- **Step 1: Compute Baseline Welfare** W^0
  - For $\gamma = 0$, the welfare is defined as the sum of spot-market profits for all projects with non-negative profits:

$$W^0 = \sum_i (x \cdot \mathbb{E}[p] \cdot q_i - C_i - r_i)$$

- **Step 5: Compute Contract Welfare** $W(\gamma)$
  - Welfare is computed only for plants that:
    1. are selected under the contract mechanism (i.e., $xf_{\max} \leq xf_{\text{equilibrium}}$), and
    2. whose cumulative capacity is below the demand level θ.
  - The formula aggregates the adjusted revenue:

$$W(\gamma) = \sum_i \left[x \cdot \mathbb{E}[p] \cdot q_i - C(k_i) - R(f^*, \gamma) \right]$$

- **Step 6: Compare to Reference and Build Output**
  - The dataset is finalized by computing:
    - **Welfare ratios** relative to $W(\gamma = 0)$
    - **Welfare gap** from the reference case (in million EUR)
    - **Welfare gain** relative to the pure spot-market baseline W^0 (in million EUR)
    - **Contract equilibrium quantities and prices** relative to those under $\gamma = 0$

1.2.10 Profits

```
# Profits by gamma and theta

# Step 1: Compute seller profits under contracts
profits_by_gamma_theta <- wind_solar_proj_2022_long_baseline |>
  filter(xf_max_cpr <= xf_equilibrium & cumulative_capacity < theta) |>
  group_by(gamma, theta) |>
  summarise(
    seller_profits_eur = sum(profit_cpr_contracts, na.rm = TRUE),
    .groups = "drop"
  ) |>
  arrange(theta, gamma)

# Step 2: Join with existing welfare dataset
welfare_with_profits_baseline <- welfare_dataset_baseline |>
  left_join(profits_by_gamma_theta, by = c("gamma", "theta"))

# Step 3: Compute profit shares for sellers and buyers
welfare_with_profits_baseline <- welfare_with_profits_baseline |>
  mutate(
    buyer_profits_eur = welfare_gamma_eur - seller_profits_eur,
    seller_profit_share_percent = round((seller_profits_eur / welfare_
       gamma_eur) * 100, 2),
    buyer_profit_share_percent  = round((buyer_profits_eur / welfare_
       gamma_eur) * 100, 2)
  ) |>
  select(
    gamma, theta,
    welfare_gamma_eur,
    seller_profits_eur,
    buyer_profits_eur,
    seller_profit_share_percent,
    buyer_profit_share_percent
  ) |>
  arrange(theta, gamma)
```

This section breaks down the total welfare under contracts into the share captured by sellers and buyers, for each policy scenario defined by γ and θ.

1. **Step 1: Seller Profits**

   We compute the total profits of sellers who were selected in the contract mechanism:

   $$\texttt{seller\_profits\_eur} = \sum_{i \in \mathcal{S}_{\gamma,\theta}} \Pi_{S,i}$$

   where $\mathcal{S}_{\gamma,\theta}$ is the set of selected projects under scenario (γ, θ).

2. **Step 2: Merge with Welfare Dataset**

   The seller profit totals are joined with the dataset containing total welfare values $W(\gamma, \theta)$.

3. **Step 3: Compute Distribution Shares**

   We define the buyer surplus as the difference between total welfare and seller profits:

   $$\texttt{buyer\_profits\_eur} = W(\gamma, \theta) - \texttt{seller\_profits\_eur}$$

and then compute relative shares:

$$\texttt{seller\_profit\_share\_percent} = 100 \cdot \frac{\texttt{seller\_profits\_eur}}{W(\gamma, \theta)}$$

$$\texttt{buyer\_profit\_share\_percent} = 100 \cdot \frac{\texttt{buyer\_profits\_eur}}{W(\gamma, \theta)}$$

1.2.11 Plotting Functions

1.2.12 Plot Functions and Output Summary

This section of the code defines reusable plotting functions to consistently visualize how the contract design parameter (γ) and market size (θ) affect various economic outcomes in the model. These plots help evaluate the distributional effects and overall efficiency of contract-based energy investment schemes.

1. `plot_profit_metric()` This is a general-purpose plotting function designed to visualize economic metrics such as profits or profit shares. It plots lines and points of a metric (e.g., profits) against a variable (e.g., γ), grouped by another factor (e.g., θ).

- Supports formatting for monetary units (e.g., millions of euros) and percentages.
- Uses custom color palettes and shape mappings.

2. `plot_line_by_gamma()` This is a specialized plotting function tailored for visualizing model outputs as a function of γ, grouped by θ. It emphasizes clarity and comparability across scenarios.

- Customizes shapes, colors, legends, and labels.
- Includes optional comma-based or million-based axis formatting.

3. Plot Outputs A series of plots are generated using the above functions. Each figure offers insight into how changes in γ influence investment and welfare outcomes under varying θ:

Plot #	Description
04	Equilibrium contract price vs. γ
05	Equilibrium quantity (MW) vs. γ
06	Total welfare (M€) vs. γ
07	Seller profits (M€) vs. γ
08	Buyer profits (M€) vs. γ
09	Seller profit share (%) vs. γ
10	Buyer profit share (%) vs. γ

Each figure uses θ to differentiate lines, allowing for comparative policy insights. Legends are positioned within the plots for readability and space efficiency.

1.2.13 Baseline Dataset Export to Excel

At the final stage of the analysis, we export key datasets to an Excel file for documentation and further use. Only selected ($\gamma \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$) values are retained to reduce file size and highlight targeted scenarios.

Procedure Overview

- A list of datasets is filtered based on `selected_gammas`.
- Each filtered dataset is written to a separate sheet in an Excel workbook.
- Sheets include:
    1. Wind Solar Projects

2. Equilibrium Prices & Quantities
3. Welfare
4. Seller/Buyer Profits

- The workbook is saved as `01_wind_solar_projects_cpr_theta.xlsx`.

1.2.14 Dataset Contents

Table 1: Wind Solar Projects Sheet (`wind_solar_proj_2022_long_baseline`)

Variable	Description
projectname	Name of the wind or solar project
capacity	Installed capacity (MW)
avgcapacityfactor	Average capacity factor
type	Technology type (Wind/Solar)
hours	Annual generation hours
power_kw	Installed capacity (kW)
q_i_kwh	Annual energy output (kWh)
q_i_mwh	Annual energy output (MWh)
v_q_i_mwh	Consumer valuation (MWh)
c_inv	Investment cost (€/kW)
c_om	O&M cost (€/MWh)
total_cost	Total project cost - $C(k_i)$ (€)
avg_cost_euro_kwh	Average cost per kWh (€)
avg_cost_euro_mwh	Average cost per MWh (€)
r_0	Baseline risk factor
r	r_i
profits_sp_no_cpr	Π_S^0 — Spot market profit without contract (€)
theta	θ — Market clearing threshold (MW) - 2500, 3500 & 4500
gamma	γ — Share of oppotunistic buyers
f_c_cpr	f_c
f_spot_cpr	f_{spot}
f_upper	f_{upper} — Upper bound of feasible contract share
f_upper_message	Message related to f_upper status
f_max_cpr	$f_{max} = \max\{f_c, f_{spot}\}$
f_equilibrium	f^* for each γ, θ
R_f_equilibrium_cpr	$R(f^*, \gamma)$
xf_c_cpr	xf_c
xf_spot_cpr	xf_{spot}
xf_upper_cpr	xf_{upper}
xf_max_cpr	xf_{max}: corresponds to the maximum between xf_c and xf_{spot}
xf_equilibrium	xf^* for each γ, θ
equilibrium_quantity	Equilibrium market quantity (MW)
x_q_exp_p_total_costs	$x \cdot q_i \cdot \mathbb{E}(p) - C(k_i)$
x_q_exp_p_total_costs_R	$x \cdot q_i \cdot \mathbb{E}(p) - C(k_i) - R_i(f^*, \gamma)$
cumulative_production	Cumulative energy production (MWh)
cumulative_capacity	Cumulative capacity (MW)
profit_cpr_contracts	Profit under CPR contracts, using f^*

Table 2: Equilibrium Prices/Quantities Sheet `equilibrium_prices`

Variable	Description
`gamma`	Risk-aversion parameter influencing contract adoption
`equilibrium_quantity`	Total investment (MW) q^*, for each γ and θ
`equilibrium_price`	Contract price xf^*, for each γ and θ
`theta`	θ — Market clearing threshold (MW) - 2500, 3500 & 4500

Table 3: Welfare Sheet `welfare_dataset_baseline`

Variable	Description
`gamma`	Share of opportunistic buyers
`theta`	θ — Market clearing threshold (MW) - 2500, 3500 & 4500
`eq_price`	Equilibrium contract price xf^* (€/MWh)
`eq_price_ratio_percent`	Ratio of equilibrium price to the baseline (xf^* when $\gamma = 0$), for each θ
`eq_quantity`	Equilibrium quantity of cumulative investment (MW), for each θ, γ
`eq_quantity_ratio_percent`	Ratio of equilibrium quantity to the baseline (when $\gamma = 0$)
`W_0`	Welfare W^0 for pure spot-market baseline, in million EUR
`welfare_gamma_eur`	Total welfare under given γ and θ, in million EUR
`welfare_ratio_percent`	Welfare compared to reference ($W(\gamma = 0)$), for each θ)
`welfare_gap_million_eur`	Welfare loss compared to reference ($W(\gamma = 0)$), in million EUR
`welfare_gain_million_eur`	Welfare gain relative to W^0, in million EUR

Table 4: Profits Sheet `welfare_with_profits_baseline`

Variable	Description
`gamma`	Share of opportunistic buyers
`theta`	θ — Market clearing threshold (MW) - 2500, 3500 & 4500
`welfare_gamma_eur`	Total welfare under given γ and θ, in EUR
`seller_profits_eur`	Total profit earned by sellers under contracts, in million EUR
`buyer_profits_eur`	Buyer surplus: welfare $W(\gamma, \theta)$ minus seller profits, in million EUR
`seller_profit_share_percent`	Seller share of total welfare, %
`buyer_profit_share_percent`	Buyer share of total welfare, %