СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

Пояснювальна записка

до магістерської дипломної роботи

магістр

(освітньо-кваліфікаційний рівень)

на тему: Методи і засоби підвищення автономності роботи гібридних систем безперебійного живлення з резервуванням вхідних каналів

Виконав: студент 2 курсу, групи ЕЛ-24дм 171 «Електроніка»

(шифр і назва спеціальності)

Полтавський І.А.

(прізвище та ініціали)

Керівник Зінченко В.Л.

(прізвище та ініціали)

Рецензент Іванов В.Г.

(прізвище та ініціали)

Київ – 2025 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____
Кафедра _____інформаційних технологій та програмування_____
Освітньо-кваліфікаційний рівень __магістр_____
Спеціальність ____171 «Електроніка»_____
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., проф.Захожай О.І.
(підпис)
« ___ » _____2025 р.

ЗАВДАННЯ

на магістерську дипломну роботу студенту
_____Полтавський Іван Андрійович_____
(прізвище, ім'я, по батькові)

1.Тема роботи: _Методи і засоби підвищення автономності роботи гібридних систем безперебійного живлення з резервуванням вхідних каналів_____

керівник роботи __к.т.н., Зінченко Володимир Леонідович,_____
(вчене звання, науковий ступінь, прізвище, ім'я, по батькові)

затверджені наказом університету від «_28_»_11_ 2025року №_241/17.03_

2. Строк подання студентом роботи:_18 грудня 2025 р._____

3. Вихідні дані до роботи: _Матеріали науково-дослідної практики, науково-методична література, дані інтернет-мережі_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ

4.2 Аналітичний огляд

4.3 Розробка адаптивного алгоритму та моделі

4.4 Дослідження ефективності розробленого алгоритму керування за допомогою програмного моделювання

4.4 Висновки

4.5 Перелік використаних джерел

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____

_____

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 05.11. 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Загальний розгляд питань і завдань	05.11.2025	виконано
2	Аналітичний огляд	09.11.2025	виконано
3	Формування вимоги до алгоритму	13.11.2025	виконано
4	Розробка алгоритму	19.11.2025	виконано
5	Вибір методу моделювання	20.11.2025	виконано
6	Створення моделі	24.11.2025	виконано
7	Моделювання роботи алгоритму	01.12.2025	виконано
8	Аналіз результатів моделювання	09.12.2025	виконано
9	Оформлення пояснювальної записки	17.12.2025	виконано
10	Передача роботи на перевірку та рецензування	18.12.2025	виконано
11	Підготовка доповіді та презентації	21.12.2025	виконано

Студент _____ Полтавський І. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Зінченко В. Л.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 81 сторінка, 11 рисунків, 10 посилань.

Тема наукової роботи: «Методи і засоби підвищення автономності роботи гібридних систем безперебійного живлення з резервуванням вхідних каналів».

Об’єктом дослідження є гібридна система безперебійного живлення з резервуванням вхідних каналів**.**

Мета роботи - полягає у підвищенні автономності роботи систем безперебійного живлення з резервуванням вхідних каналів.

В процесі роботи були вирішені наступні завдання:

- Проведено аналіз предметної області та існуючих рішень.
- Розроблено програмну модель гібридної системи, що включає мережу, сонячні панелі, акумуляторну батарею та навантаження.
- Розроблено нові методи та засоби керування, спрямовані на підвищення автономності.
- Розроблено адаптивний алгоритм керування системою
- Проведено дослідження ефективності запропонованих рішень шляхом комп'ютерного моделювання різних сценаріїв роботи.

В результаті роботи обґрунтовано необхідність розробки адаптивного алгоритму керування**,** здатного інтегрувати зовнішню прогностичну інформацію (таку як графіки планових відключень), для забезпечення максимальної автономності та ефективності гібридної системи. Запропонований підхід дозволить системі активно підтримувати необхідний резерв заряду акумуляторної батареї.

ГІБРИДНА СИСТЕМА, АВТОНОМНІСТЬ, АЛГОРИТМ КЕРУВАННЯ, БЕЗПЕРЕБІЙНЕ ЖИВЛЕННЯ, АКУМУЛЯТОРНА БАТАРЕЯ, МОДЕЛЮВАННЯ

ЗМІСТ

ВСТУП 6
1 АНАЛІТИЧНИЙ ОГЛЯД…….8
1.1 Аналіз існуючих рішень…….....….8
1.2 Аналіз режимів роботи та принципів керування гібридними системами..11
1.3 Вимоги до адаптивного алгоритму керування……..…….....13
1.4 Висновки за розділом.....14

2 РОЗРОБКА АДАПТИВНОГО АЛГОРИТМУ ТА МОДЕЛІ.....16
2.1 Створення алгоритму та його концепція……...16
2.2 Обґрунтування вибору методу моделювання для дослідження ефективності алгоритму……..….18
2.3 Опис математичної моделі.....20
2.4 Вибір засобів для створення програмної моделі.....22
2.5 Опис розробленого програмного комплексу для аналізу автономності....23
2.6 Висновки за розділом.....28

3 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО АЛГОРИТМУ КЕРУВАННЯ ЗА ДОПОМОГОЮ ПРОГРАМНОГО МОДЕЛЮВАННЯ.....30
3.1 Опис програмної реалізації та інтерфейсу користувача……..29
3.2 Цінність програмної реалізації для розробників енергетичних систем….33
3.3 Обґрунтування вихідних даних та сценаріїв моделювання……34
3.4 Вибір сценаріїв відключень та тривалості симуляції.....35
3.5 Аналіз результатів моделювання та оцінка ефективності.....36
3.6 Висновки до розділу.....47
ВИСНОВКИ.....49
ПЕРЕЛІК ПОСИЛАНЬ 51
Додаток А.....53
Додаток Б.....54

ВСТУП

Сучасне суспільство на всіх рівнях – від побутового до промислового та інфраструктурного – демонструє критичну залежність від стабільного та безперебійного електропостачання. Функціонування центрів обробки даних, телекомунікаційних мереж, медичного обладнання та інших критично важливих систем залежить від наявності електроенергії [1]. Будь-яке порушення живлення, навіть короткочасне, може призвести до значних фінансових втрат, втрати даних, зупинки технологічних процесів та виникнення загроз для здоров'я і життя людей.

Особливої гостроти ця проблема набуває в умовах тривалих та непередбачуваних відключень електроенергії. Причини таких відключень можуть бути різноманітними: техногенні аварії на лініях електропередач, перевантаження енергосистем, стихійні лиха та бойові дії.

Існуючі підходи для забезпечення безперебійного живлення, зокрема гібридні системи, не завжди ефективно вирішують задачу тривалої автономної роботи під час блекаутів, оскільки їхні стандартні алгоритми керування не є оптимальними [2]. Однак в даній роботі планується дослідити існуючі рішення та предметну область, та розробити ефективний алгоритм управління такими системами.

Запропонований метод заснований на оптимальному розподілі енергетичних потоків між джерелами, що може допомогти максимізувати час автономної роботи, а також передбачає використання графіків відключення для ще більш ефективного забезпечення автономності системи.

Розробка таких алгоритмів може стати інноваційним кроком до створення по-справжньому надійних та автономних систем безперебійного живлення, і подальші дослідження в цій галузі можуть призвести до більш ефективних рішень для захисту критичної інфраструктури.

Таким чином, розробка методів та засобів, що підвищують автономність роботи в кризових ситуаціях, шляхом інтелектуального керування

резервованими вхідними каналами, є надзвичайно актуальною науково-технічною задачею.

Тому можна сформулювати, що:

Метою роботи є підвищення автономності роботи систем безперебійного живлення з резервуванням вхідних каналів.

Об’єктом дослідження є гібридна система безперебійного живлення з резервуванням вхідних каналів.

Предметом дослідження є режими роботи та принципи керування гібридною системою безперебійного живлення з резервуванням вхідних каналів

Для досягнення поставленої мети необхідно вирішити наступні **завдання:**

1. Провести аналіз предметної області та існуючих рішень.
2. Розробити математичну модель гібридної системи у вигляді програми, що включає мережу, сонячні панелі, акумуляторну батарею та навантаження.
3. Розробити нові методи та алгоритм керування, спрямовані на підвищення автономності.
4. Провести дослідження ефективності запропонованих рішень шляхом комп'ютерного моделювання різних сценаріїв роботи.

АНАЛІТИЧНИЙ ОГЛЯД

1.1 **Аналіз існуючих рішень**

Надійність електропостачання забезпечується різними типами систем безперебійного живлення, вибір яких критично залежить від вимог до тривалості автономної роботи. Сучасні рішення можна класифікувати за їхньою здатністю інтегрувати відновлювані джерела енергії та керувати енергетичними потоками.

1. **Традиційні ДБЖ** (класів off-line, line-interactive та on-line) призначені для захисту навантаження від короткочасних перебоїв та імпульсних перешкод. Вони використовують акумуляторні батареї як єдине резервне джерело живлення.

- Переваги: Висока надійність, миттєва реакція на зникнення мережі.
- Недоліки: Дуже низька автономність. Їхня архітектура принципово розрахована на короткочаснее живлення (5-30 хвилин) для безпечного завершення роботи обладнання, а не на тривалу автономну роботу [3]. При тривалих блекаутах заряд АКБ вичерпується, і система припиняє роботу, оскільки не має жодних механізмів поповнення енергії з альтернативних джерел під час відсутності мережі.

Рисунок 1 – Класичні ДБЖ

2. **Мережеві інвертори.** Ці системи призначені для генерації електроенергії від сонячних панелей та її подачі безпосередньо в мережу.

- Переваги: Максимальне використання сонячної енергії.
- Недоліки: Повна залежність від мережі. При зникненні зовнішнього живлення мережеві інвертори зобов'язані негайно припинити роботу та відключитися. Це вимагається міжнародними стандартами безпеки (наприклад, IEEE 1547-2018) для запобігання ураженню струмом ремонтних бригад, які працюють на лінії [4]. Таким чином, вони не мають можливості резервування і не здатні забезпечувати автономність під час блекаутів.

Рисунок 2 – Мережевий інвертор

3. Гібридні системи безперебійного живлення (Hybrid UPS/Inverters)

Гібридні інвертори поєднують функції класичного ДБЖ та мережевого інвертора. Вони здатні одночасно працювати з кількома джерелами: зовнішньою мережею, сонячними панелями (СЕС) та АКБ.

- Переваги: Висока гнучкість, можливість автономної роботи, інтеграція вторинних джерел електроживлення .
- Недоліки: Висока вартість впровадження. Ефективність автономної роботи критично залежить від внутрішнього алгоритму керування. Як показує аналіз, неоптимальні алгоритми (наприклад, ті, що просто пріоретезують економію) можуть призвести до надмірного розряду-заряду АКБ у денний час [5]. У результаті, при настанні раптового вечірнього блекауту, система матиме недостатньо запасеної енергії, що зводить її автономність до мінімуму. Таким чином, потенціал системи не реалізується.

Рисунок 3 – Гібридна система безперебійного живлення

Висновки щодо існуючих рішень: Для досягнення тривалої автономності гібридні системи є єдиним задовільним рішенням. Однак, їхній потенціал часто не реалізується через недосконаліть алгоритмів керування.

1.2. Аналіз режимів роботи та принципів керування гібридними системами

Ключовим елементом, що визначає рівень автономності та економічної ефективності гібридної системи, є стратегія керування інвертором. Вона регулює потоки енергії між мережею, сонячною електростанцією (СЕС), акумуляторною батареєю (АКБ) та навантаженням.

Сучасні гібридні інвертори зазвичай підтримують чотири основні режими роботи, кожен з яких орієнтований на вирішення конкретних завдань [6]:

1) **Режим власного споживання (Self-Consumption Mode)**

- *Логіка роботи:* Пріоритет віддається використанню власної сонячної енергії. Енергія від СЕС спочатку йде на навантаження, надлишок спрямовується на зарядку АКБ, і лише після повного заряду — продається в мережу. Коли генерації СЕС недостатньо, використовується енергія з АКБ, і лише в останню чергу — мережа.
- *Мета:* Максимізація економії та енергетичної самодостатності.
- *Недолік:* Акумулятор розряджається ввечері/вночі для економії. У випадку раптового нічного відключення мережі, будинок може залишитися з розрядженою батареєю.

2) **Режим безперебійного живлення (UPS Mode / Back-up)**

- *Логіка роботи:* Система забезпечує резервне живлення. АКБ постійно підтримується у зарядженому стані за рахунок СЕС або мережі. Розряд батареї відбувається виключно у момент зникнення напруги в зовнішній мережі. Час перемикання зазвичай становить менше 10 мс.

- *Мета:* Гарантія надійності живлення критично важливих споживачів.
- *Недолік:* Низька економічна ефективність. Сонячна енергія може втрачатися, якщо АКБ вже заряджена, а власне споживання будинку низьке (інвертор обмежує генерацію замість того, щоб використовувати АКБ для циклічної економії).

3) **Режим згладжування піків (Peak Shaving Mode)**

- *Логіка роботи:* Використовується для оптимізації витрат за наявності тарифів «день/ніч» або пікових тарифів. Інвертор налаштовується на зарядку АКБ в години низького тарифу (вночі) та розрядку в години пікового тарифу (ввечері), зменшуючи навантаження на мережу та рахунки за електроенергію.
- *Мета:* Економічна оптимізація витрат на електроенергію.
- *Недолік:* Алгоритм діє за жорстким розкладом тарифів і не враховує ймовірність аварійних відключень у «пікові» години, коли він примусово розряджає АКБ.

4) **Автономний режим (Off-Grid Mode)**

- *Логіка роботи:* Повна ізоляція від мережі. Інвертор працює як єдине джерело напруги, балансуючи генерацію СЕС та заряд АКБ відповідно до навантаження.
- *Мета:* Електрифікація об'єктів без доступу до загальної мережі.

Проведений аналіз стандартних режимів роботи (Self-Consumption, UPS, Peak Shaving) дозволяє виявити їх спільний недолік в умовах нестабільної мережі: вони є статичними**.**

- Режим *Self-Consumption* економить гроші, але ризикує залишити користувача без світла під час блекауту.
- Режим *UPS* гарантує світло, але ігнорує економію від сонячної генерації.
- Режим *Peak Shaving* працює за фіксованим часом і не знає про графіки планових відключень.

Жоден із існуючих режимів не здатен адаптуватися до графіків планових відключень, які є реальністю сьогодення. Тому виникає критична необхідність у розробці адаптивного режиму.

Таким чином у роботі пропонується створення алгоритму, який:

1. Отримує зовнішні дані (графіки відключень).
2. Динамічно перемикається між стратегіями: економить кошти, коли мережа стабільна, і примусово заряджає АКБ (навіть від мережі) за 2-3 години до прогнозованого відключення.

Таким чином, розробка адаптивного алгоритму дозволить поєднати економічність режиму Self-Consumption з надійністю режиму UPS, усуваючи недоліки стандартних рішень.

1.3. Вимоги до адаптивного алгоритму керування

Основним завданням алгоритму є забезпечення ефективної роботи гібридної енергосистеми з максимальною автономністю та оптимальним використанням ресурсів. На відміну від стандартних режимів, алгоритм повинен адаптуватися до зовнішніх умов.

Головним пріоритетом алгоритма повинна бути автономність. Алгоритм повинен забезпечувати енергозабезпечення будинку під час відключень мережі. Система має не лише реагувати на збій, але й активно підтримувати готовність до нього. Для цього в будь-який момент часу, особливо в періоди високої ймовірності відключень, в акумуляторі зберігається заданий резерв заряду. Алгоритм буде інтегрувати відомі графіки планових відключень, щоб заздалегідь забезпечити повний заряд акумулятора.

Також важливою має бути і оптимізація роботи АКБ. Акумуляторна батарея є критичним компонентом системи. Алгоритм керування повинен подовжити її ресурс використання. Це досягається шляхом запобігання глибоких розрядів та визначення безпечних меж роботи. Алгоритм підтримує

рівень заряду, достатній для покриття критичних потреб, уникаючи режимів експлуатації, що призводять до передчасної деградації.

Ключова функція алгоритму —розподіл енергії між сонячними панелями, акумулятором, навантаженням та мережею. На відміну від систем з фіксованими пріоритетами, алгоритм безперервно визначає оптимальні пріоритети на основі поточних умов.

Для досягання максимальної автономності алгоритм враховує зовнішній фактор - графіки відключень*:* При отриманні інформації про майбутнє відключення система переходить у режим підвищеної готовності, забезпечуючи близький до повному заряд акумулятора.

Вхідні данні для функціонування системи:

- Енергетичний баланс: поточне споживання об’єкту, потужність генерації сонячних панелей і рівень заряду акумулятора.
- Статус мережі: наявність напруги в центральній мережі.
- Зовнішня інформація: графік планових відключень.

Очікуваний результат: Алгоритм забезпечить оптимальний режим роботи системи, що поєднує максимально можливу автономність з дотриманням вимог до збереження акумулятора. Система автоматично заряджає акумулятор за кілька годин до відключень для покриття дефіциту мережі, при цьому мінімізуючи витрати на енергоносії, дотримуючись оптимальних режимів заряду-розряду АКБ для продовження його експлуатаційного терміну. Ефективність роботи системи забезпечуватиметься за рахунок перерозподілу енергетичних потоків між доступними джерелами живлення та навантаженням, з урахуванням поточних параметрів системи та прогнозованих умов експлуатації.

1.4 Висновки за розділом

Гібридні системи безперебійного живлення з резервуванням вхідних каналів є ключовим рішенням для забезпечення стабільного

електропостачання критичних об'єктів у умовах тривалих та непередбачуваних відключень. Проведений аналіз існуючих рішень показав, що класичні ДБЖ мають обмежену автономність через залежність від ємності АКБ, мережеві інвертори не здатні функціонувати при відсутності мережі через вимоги безпеки, тоді як гібридні системи поєднують переваги обох підходів, але їхня ефективність суттєво залежить від алгоритмів керування.

Дослідження стандартних стратегій керування виявило їхню основну слабкість — системи не можуть адаптуватися до змінних умов експлуатації. Відсутність планування призводить до того, що в момент відключення мережі акумуляторна батарея часто виявляється недостатньо зарядженою, що різко знижує автономність системи.

Це обґрунтовує необхідність розробки адаптивного алгоритму керування, здатного інтегрувати зовнішню прогностичну інформацію, таку як графіки планових відключень. Запропонований підхід дозволить перейти до більш ефективного управління енергетичними потоками, забезпечуючи підготовку системи до прогнозованих збоїв живлення.

Наступним етапом роботи є перевірка ефективності запропонованого алгоритму шляхом математичного моделювання та порівняння його характеристик із базовою стратегією керування. Таке дослідження дозволить кількісно оцінити вплив алгоритму на тривалість автономної роботи та ефективність використання енергоресурсів.

РОЗРОБКА АДАПТИВНОГО АЛГОРИТМУ ТА МОДЕЛІ

2.1 Створення алгоритму та його концепція

В основу розробки покладено гіпотезу про те, що інтеграція графіка планових відключень у логіку роботи гібридного інвертора здатна значно збільшити час автономної роботи системи без необхідності збільшення фізичної ємності акумуляторних батарей.

Стандартні алгоритми управління BESS (Battery Energy Storage Systems) працюють реактивно: вони реагують на зникнення напруги в мережі лише в момент самого відключення. Це створює критичний ризик: якщо відключення відбувається ввечері або після похмурого дня, рівень заряду (SOC) може перебувати на мінімальному рівні, чого недостатньо для проходження тривалого блекауту.

Суть запропонованого методу полягає у превентивному реагуванні. Використовуючи вікно прогнозування, система визначає відсутність мережі заздалегідь. Якщо у найближчі години прогнозується відключення, алгоритм примусово змінює пріоритети: він припиняє економію та активує форсовану зарядку від мережі, щоб підійти до моменту відключення з показником SOC, до 99%, оскільки повний заряд до 100% може негативно впливати на ресурс акумуляторної батареї**.**

Для верифікації ефективності нової методики як еталон для порівняння було обрано алгоритм Self-Consumption**.** Цей алгоритм також послужив основою для розробки адаптивного методу**.** На сьогодні це найбільш поширений та економічно ефективний сценарій роботи гібридних інверторів у світі. Його логіка спрямована на максимальну економічну ефективність:

1. Сонячна енергія в першу чергу живить будинок.
2. Надлишки направляються в АКБ.
3. При нестачі сонячної енергії АКБ розряджається, заміщуючи платну мережу.

4. Зарядка АКБ від мережі заборонена.

Саме з цим «класичним» підходом, який прийнятий за базовий буде порівняно результати моделювання, щоб довести переваги адаптивного методу в умовах нестабільної мережі.

З метою демонстрації перспективності методу випереджального заряду було розроблено спеціалізований алгоритм управління потоками енергії. Його логічна структура представлена на блок-схемі наведеній у додатку А**.**

Алгоритм працює циклічно з кроком в 1 годину та приймає рішення на основі даних (поточний заряд, генерація PV, навантаження) та зовнішніх даних (графік відключень).

Ключові відмінності адаптивного алгоритму від базового (Self-Consumption):

1. Модуль прогнозування: На відміну від базового алгоритму, Smart-версія аналізує графік відключень на 3 години вперед.
2. Динамічна зміна пріоритетів:

- *У базовому алгоритмі* пріоритетом завжди є економія (розряд АКБ у мережу при будь-якій можливості).
- *В удосконаленому алгоритмі* при виявленні загрози вмикається «Режим підготовки»: розряд АКБ блокується, а навантаження будинку переводиться на живлення від мережі, щоб зберегти накопичений заряд.

3. Форсована зарядка: У базовий алгоритм АКБ заряджається виключно від сонячної енергії. Адаптивний алгоритм при загрозі блекауту агрегує потужності: він використовує і сонце, і мережу одночасно, розраховуючи необхідний струм заряду так, щоб досягти 99% ємності за 1 годину до прогнозованого відключення.
4. Управління глибиною розряду: Алгоритм адаптивно змінює ліміти: у режимі економії він тримає недоторканний резерв 20%, але в режимі автономної роботи (блекауту) дозволяє глибокий розряд до 10%, максимізуючи час роботи системи. При досягненні рівня заряду АКБ

10%, система вимикається і настає повний блекаут для споживача, щоб уникнути критичного розряду та не завдати шкоди акумуляторній батареї.

2.2 Обґрунтування вибору методу моделювання для дослідження ефективності алгоритму

При розробці методів і засобів підвищення автономності гібридних систем безперебійного живлення з резервуванням вхідних каналів критично важливим етапом є вибір адекватного інструментарію для перевірки запропонованих алгоритмів керування.

Для вирішення поставлених задач було обрано метод математичного моделювання з програмною реалізацією. Такий підхід забезпечує низку переваг:

1. Універсальність та незалежність від конкретного обладнання. Імітаційне моделювання, як правило, вимагає деталізації на рівні фізичних компонентів (ключів, транзисторів, специфічних контролерів), що прив’язує результати дослідження до конкретної апаратної реалізації. Натомість, головною метою роботи є розробка універсальних алгоритмів керування, які мають працювати в широкому спектрі технічних реалізацій. Математична модель дозволяє оперувати узагальненими параметрами системи (ємність АКБ, номінальна потужність СЕС, ККД перетворення), що забезпечує інваріантність розроблених рішень та можливість їх масштабування на системи різної конфігурації без необхідності перебудови складної схемотехнічної моделі.

2. Швидкість розрахунків для тривалих періодів часу. Оцінка автономності системи вимагає аналізу її поведінки на тривалих проміжках часу. Імітаційні моделі, що працюють у масштабі реального часу або з малим кроком дискретизації (для розрахунку перехідних процесів у напівпровідниках), потребують значних обчислювальних ресурсів, що

унеможливлює проведення серії експериментів за тривалий період. Математична модель, побудована на основі рівнянь енергетичного балансу з дискретністю, достатньою для оцінки заряду/розряду накопичувачів, дозволяє миттєво моделювати роботу системи протягом значного часу, що є критичним для оцінки ефективності алгоритмів підвищення автономності.

3. Інтеграція стохастичних факторів та зовнішніх впливів. Специфіка роботи гібридних систем передбачає сильну залежність від випадкових величин, які важко формалізувати в класичних імітаційних пакетах. Математична модель дозволяє гнучко імплементувати:

- Метеорологічні дані: інтеграцію реальних масивів даних інсоляції або спрощені коефіцієнти, для коректного розрахунку генерації відновлюваних джерел енергії.
- Графіки відключень мережі: моделювання сценаріїв планових відключень різної тривалості та частоти, що є ключовим фактором для оцінки надійності системи в умовах нестабільного енергопостачання.

Таким чином, математична модель у вигляді програми є найбільш адекватним інструментом для вирішення задач, оскільки дозволяє зосередитися на логіці перерозподілу енергії, оптимізації режимів роботи та оцінці автономності, ігноруючи несуттєві для даного рівня абстракції перехідні процеси.

У рамках підрозділу було проведено порівняльний аналіз можливостей імітаційного (схемотехнічного) та математичного моделювання. На основі аналізу для досягнення поставленої мети було обрано математичне моделювання з реалізацією у вигляді програми**.**

Даний вибір обумовлений специфікою об'єкта дослідження та необхідністю вирішення оптимізаційних задач на тривалих проміжках часу.

2.3 Опис математичної моделі

Для кількісної оцінки автономності та дослідження режимів роботи гібридної системи розроблено математичну модель у дискретному часі з кроком $\Delta t = 1$ год. Стан системи у кожен момент часу t визначається вектором параметрів енергетичного балансу та рівнем заряду накопичувача.

1. Модель фотоелектричної генерації

Генерація активної потужності сонячної електростанції (СЕС) $P_{PV}(t)$ моделюється як функція від встановленої потужності панелей P^{nom}_{PV}, погодних умов та часу доби h(t):

$$P_{PV}(t) = \begin{cases} P_{PV}^{nom} \cdot K_{weather} \cdot \left(1 - \frac{|h(t)-12|}{6}\right), & \text{якщо } 6 \leq h(t) < 18 \\ 0, & \text{якщо } h(t) < 6 \vee h(t) \geq 18 \end{cases} \quad (1)$$

де $K_{weather} \in \{0.5;\ 1.0\}$ — коефіцієнт інтенсивності інсоляції залежно від хмарності.

2. Енергетичний баланс та динаміка SOC

Закон збереження енергії для струму у системі описується рівнянням балансу потужностей:

$$P_{load}(t) = P_{PV \to load}(t) + P_{grid \to load}(t) + P_{batt \to load}(t) \quad (2)$$

де P_{load} — споживання навантаження, $P_{PV \to load}$ — пряме споживання від СЕС, $P_{grid \to load}$ — споживання з мережі, $P_{batt \to load}$ — потужність розряду АКБ. Зміна стану заряду акумулятора описується наступним різницевим рівнянням:

$$SOC(t+1) = SOC(t) + \frac{\Delta t}{E_{nom}} \cdot \left(P_{ch}(t) \cdot \eta_{ch} - \frac{P_{dis}(t)}{\eta_{inv}}\right) \quad (3)$$

де:

- E_{nom} — номінальна енергоємність батареї, кВт·год;

- $P_{ch(t)}$ — потужність заряду, кВт;

- $P_{dis(t)}$ — потужність розряду, кВт;

- η_{ch}, η_{inv} — коефіцієнти корисної дії процесів заряду та інвертування відповідно.

3. Модель обмежень заряду

Для врахування нелінійності заряду літій-іонних накопичувачів максимальна потужність заряду P^{max}_{ch} обмежується відповідно до поточного рівня SOC. Модель імітує дві фази заряду (Constant Current / Constant Voltage):

$$P_{ch}^{max}(t) = \begin{cases} C_{rate} \cdot E_{nom}, & \text{якщо } SOC(t) \leq 0.8 \\ C_{rate} \cdot E_{nom} \cdot \frac{1-SOC(t)}{1-0.8}, & \text{якщо } SOC(t) > 0.8 \end{cases} \quad (4)$$

де C_{rate} — допустима швидкість заряду (прийнято 0.5C), а 0.8 — поріг початку фази CV.

4. Алгоритм адаптивного керування

Керуючий вплив адаптивного алгоритму базується на прогнозному векторі стану мережі G_{future} на горизонті планування k = 3 години. Якщо прогнозується відключення ($\exists S_{grid(t+k)} = 0$), система активує режим форсованого заряду. Необхідна потужність заряду від мережі визначається як:

$$P_{grid \to batt}(t) = \min\left(P_{ch}^{max}(t), \frac{(SOC_{target} - SOC(t)) \cdot E_{nom}}{\Delta t_{rem}}\right) \quad (5)$$

де SOC_{target} = 0.99, а Δt_{rem} — час, що залишився до прогнозованого початку відключення.

2.4 Вибір засобів для створення програмної моделі

Для практичної реалізації математичної моделі створено програму, реалізовану мовою програмування Python (версія 3.12.10). Вибір даного інструментарію обумовлений наступними факторами:

1) Високий рівень абстракції та швидкість розробки: Синтаксис Python дозволяє фокусуватися на реалізації логіки алгоритмів керування (енергоменеджменту) без необхідності низькорівневого управління пам’яттю, що притаманно компільованим мовам (C/C++).
2) Модульність та масштабованість: Об’єктно-орієнтований підхід Python дозволив структурувати модель у вигляді окремих функціональних блоків (модуль симуляції, модуль інтерфейсу, модуль обробки даних), що спрощує подальшу модернізацію системи.
3) Кросплатформеність: Інтерпретована природа мови забезпечує роботу програмного комплексу на різних операційних системах (Windows, Linux, macOS) без необхідності перекомпіляції.

Для реалізації функціоналу використано наступні бібліотеки:

- Standard Library (sys, os, math): Використання стандартних бібліотек Python дозволило реалізувати ядро математичної моделі, що базується на ітеративному розрахунку балансу потужностей. Відмова від використання важковагових бібліотек для простих арифметичних обчислень забезпечила високу швидкість виконання симуляції навіть для тривалих часових проміжків.
- PyQt5: Для розробки графічного інтерфейсу користувача (GUI) використано бібліотеку PyQt5. Це дозволило створити зручний інструмент для дослідження.

Реалізація моделі у вигляді десктопного додатку дозволяє проводити численні експерименти з різними конфігураціями обладнання та алгоритмами керування в реальному часі, отримуючи миттєвий зворотний зв'язок щодо ефективності запропонованих рішень.

2.5 Опис розробленого програмного комплексу для аналізу автономності

Повний текст програмного коду наведено у додатку Б. Код представлено у повному обсязі з метою створення відкритої базової платформи, що дозволить розробникам та дослідникам гібридних систем використовувати цей інструментарій як основу для подальшого розширення та вдосконалення моделі. Цей підрозділ містить опис основних закладених принципів та алгоритмічної логіки розробленої моделі.

Розроблений програмний модуль є детермінованою імітаційною моделлю дискретного часу, що описує функціонування гібридної системи з резервуванням каналів. Система включає в себе СЕС, АКБ Li-ion та зовнішню електричну мережу з графіком постачання енергії.

Основним завданням моделі є симуляція потоків потужності для кількісної оцінки автономності при різних стратегіях управління. Модель призначена для порівняльного аналізу ефективності базового (Self consumption) та розробленого адаптивного алгоритмів управління в умовах відключень мережі.

Основні Компоненти:

Архітектура додатка побудована на принципах модульності з чітким розділенням обчислювального ядра та рівня представлення.

- Обчислювальне ядро: Інкапсулює фізику процесів та логіку контролера. Реалізовано у вигляді чистої функції без побічних ефектів, що забезпечує відтворюваність експериментів:

"def simulate(sim_time_h=24, batt_capacity_ah=200, batt_voltage_v=48, pv_power_kw=3.0, load_kw=0.5, weather="Сонячно", outage_schedule=None, soc_init=0.5, eta_inv=0.95, algo="base"):"

- Графічний інтерфейс (GUI): Реалізований на базі фреймворку PyQt5. Клас MainWindow керує параметризацією експерименту, а клас SimpleOutageCalendar надає інструмент для матричного задання графіка відключень (дні/години):

"class MainWindow(QMainWindow):" "class SimpleOutageCalendar(QWidget):"

Математичне Моделювання Підсистем:

Модель використовує квазістаціонарний підхід, де перехідні процеси в межах години усереднюються.

Модель Фотоелектричної Генерації

Генерація СЕС моделюється детермінованою функцією, що залежить від часу доби та погодного коефіцієнта. Використовується лінійна апроксимація інсоляції («трикутний профіль»), що досягає піку о 12:00 і дорівнює нулю поза інтервалом 06:00–18:00.

- Врахування погодних умов: Для симуляції впливу хмарності на генерацію використовується словник коефіцієнтів, що визначають частку від номінальної потужності масиву. Коефіцієнт pv_coeff обирається на основі вхідного параметра погоди:

  "weather_coeff = {"Сонячно": 1.0, "Хмарно": 0.5}" "pv_coeff = weather_coeff.get(weather, 1.0)"

- Розрахунок інтенсивності сонячного випромінювання: Підсумкова генерація розраховується як добуток встановленої потужності, погодного коефіцієнта та геометричного фактора часу доби:

"intensity = 1 - abs(hour - peak_hour) / 6" "return pv_power_kw * pv_coeff * max(0.0, intensity)"

Електрохімічна Модель Акумулятора:

Модель акумулятора враховує нелінійність процесу заряду, характерну для Li-ion хімії, зокрема профіль CC/CV (Constant Current / Constant Voltage).

Константи процесу:

- Нормативна швидкість заряду (0.5C): "LI_ION_C_RATE = 0.5"
- Кулонівська ефективність (95%): "LI_ION_CHARGE_EFF = 0.95"
- Поріг початку фази постійної напруги (80% SOC): "CV_START_SOC = 0.80"

Обмеження потужності заряду (CV-фаза): При досягненні рівня заряду вище 80% (CV_START_SOC), допустима потужність заряду лінійно знижується, імітуючи падіння струму при стабілізації напруги. Це критично важливий аспект для реалістичної оцінки часу, необхідного для повної зарядки перед відключенням:

"if soc > CV_START_SOC:" "cv_factor = (1.0 - soc) / (1.0 - CV_START_SOC)" "current_max_charge *= max(0.0, cv_factor)"

Розрахунок фактичного енергообміну: Фактично записана в акумулятор енергія розраховується з урахуванням обмежень за потужністю джерела, допустимою потужністю прийому (C-рейтинг/CV) та доступною вільною ємністю:

"energy_input_limit = space_kwh / LI_ION_CHARGE_EFF" "actual_input_kw = min(power_limit, energy_input_limit)"

Алгоритмічна Логіка та Стратегії Управління

Симуляція виконується ітеративно з кроком $\Delta t = 1$ год . На кожному кроці вирішується задача балансу потужності.

Керуючі Структури

- Дискретизація часу: Основний цикл проходить по всьому горизонту моделювання, обчислюючи поточний день і годину для зіставлення з графіком відключень:
  "for t in range(int(sim_time_h)):" "current_day = (total_hours_from_start // 24) % 28"
- Визначення стану мережі: Перевірка доступності зовнішньої мережі здійснюється через пошук поточного часового слота у множині відключень outage_schedule:
  "if current_day in outage_schedule:" "if current_hour in outage_schedule[current_day]:" "grid_ok = False"

Стратегія: Базовий Алгоритм

Реалізує стратегію мінімізації використання мережі та збереження ресурсу акумулятора («пік-шейвінг» та самоспоживання).

1. Пріоритети:
   - Навантаження покривається в першу чергу від СЕС.
   - Нестача покривається мережею (якщо доступна) або АКБ.
   - Надлишок СЕС спрямовується на заряд АКБ.
2. Захист глибини розряду: За наявності мережі розряд акумулятора блокується, якщо рівень заряду падає нижче порогу збереження (20%), що резервує ємність на випадок раптової аварії:

   "if rest_load > 0 and soc > SOC_MIN_BASE:"

Стратегія: Адаптивний Алгоритм

Реалізує проактивне управління з горизонтом планування.

1. Прогнозування (Look-ahead): Алгоритм сканує графік відключень на 3 години вперед для виявлення найближчих відключень:
   "for h_offset in range(1, lookahead_hours + 1):" "if d in outage_schedule and h in outage_schedule[d]: upcoming_outages.append(h_offset)"
2. Адаптація уставки заряду: Якщо виявлено майбутнє відключення, цільовий рівень заряду (target_soc) підвищується до 99%, і активується

примусовий заряд від мережі. Алгоритм розраховує необхідну швидкість заряду (needed_charge_rate), враховуючи час, що залишився, та уповільнення заряду в CV-фазі:

"min_time_cv_phase = k_factor * (math.log(1.0 - CV_START_SOC) - math.log(1.0 - target_soc))" "needed_charge_rate = energy_to_cv_start / time_for_bulk"

Це дозволяє заряджати акумулятор рівно настільки швидко, наскільки це необхідно до моменту відключення, мінімізуючи пікові навантаження на мережу, але гарантуючи готовність.

3. Аварійний режим роботи: Під час відключення мережі алгоритм дозволяє глибший розряд акумулятора (до 10% замість 20%), жертвуючи ресурсом АКБ заради забезпечення живлення критичного навантаження:

   "if rest_load > 0 and soc > SOC_HARD_LIMIT:"

Вихідні Дані та Результат

Функція повертає набір із трьох елементів, що забезпечують повну спостережуваність процесу симуляції:

1. Словник метрик (results): Агреговані показники для швидкого аналізу.
   - hours_no_power: Час дефіциту енергії.
   - hours_autonomy: Час повного покриття навантаження у моменти відключень.
2. Лог стану заряду (soc_log): Вектор значень SOC для побудови часових діаграм:

   "soc_log.append(soc * 100)"
3. Історія потоків (history): Деталізовані часові ряди для кожного компонента балансу (мережа, СЕС, АКБ, навантаження, зарядна потужність):

   "history = { "grid": hist_grid, "pv": hist_pv, ... }"

4.2. Механізм повернення та обробки

Дані повертаються у контекст виклику через оператор return, після чого клас MainWindow використовує їх для розрахунку ефективності (приріст автономності у %) та генерації візуальних звітів за допомогою бібліотеки matplotlib.

"return results, soc_log, history"

2.6 Висновки за розділом

У розділі розроблено та обґрунтовано підхід до підвищення автономності гібридних систем безперебійного живлення шляхом удосконалення алгоритмів керування енергопотоками. Запропонований алгоритм базується на принципі превентивного реагування та використовує інформацію про графіки планових відключень для випереджального заряду акумуляторної батареї, що принципово відрізняє його від класичних реактивних стратегій типу *Self-Consumption*.

Показано, що застосування математичного моделювання з програмною реалізацією є найбільш доцільним для дослідження ефективності алгоритмів керування автономністю, оскільки такий підхід забезпечує універсальність, високу швидкість розрахунків на тривалих часових інтервалах та можливість інтеграції стохастичних зовнішніх факторів, зокрема метеорологічних умов і графіків відключень мережі.

Обґрунтовано вибір мови програмування Python як інструменту реалізації моделі завдяки її модульності, кросплатформеності та зручності для швидкої реалізації алгоритмів енергоменеджменту. Розроблений програмний комплекс має чітко розділену архітектуру, що забезпечує відтворюваність результатів та можливість подальшого розширення.

Таким чином, у розділі сформовано теоретичну та інструментальну основу для подальшого чисельного експерименту і порівняльної оцінки базового та адаптивного алгоритмів керування, що дозволяє об’єктивно оцінити вплив запропонованих рішень на автономність гібридних систем в умовах нестабільного електропостачання.

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО АЛГОРИТМУ КЕРУВАННЯ ЗА ДОПОМОГОЮ ПРОГРАМНОГО МОДЕЛЮВАННЯ

3.1 Опис програмної реалізації та інтерфейсу користувача

Архітектура програми побудована на принципах об'єктно-орієнтованого програмування (ООП), що забезпечує чітке розділення логіки симуляції (Back-end) та графічного інтерфейсу (Front-end). Такий підхід дозволив створити гнучке середовище ("віртуальний стенд"), де користувач може в реальному часі змінювати конфігурацію обладнання та сценарії подій, миттєво отримуючи візуалізацію енергетичного балансу системи.

Інтерфейс програмної моделі реалізовано у вигляді головного вікна з вкладками для швидкого доступу до основних функцій. Вкладка «Моделювання» є робочою панеллю користувача і структурно поділена на три зони (рис. 4):

1. *Панель налаштування параметрів (ліворуч)*. Містить поля для введення вихідних даних, згруповані за логічними блоками:
    - *Час симуляції та умови:* дозволяє задати тривалість експерименту (днів/годин), обрати погодний сценарій («Сонячно», «Хмарно», «Дощ») та встановити постійне навантаження споживача (кВт).
    - *Акумулятори (Li-ion/LiFePO4):* налаштування параметрів накопичувача, включаючи номінальну ємність (А·год), напругу (В) та початковий рівень заряду (%).
    - *СЕС та Інвертор:* введення потужності СЕС та коефіцієнта корисної дії інвертора.
2. *Блок керування.* Розташований під панеллю налаштувань і містить дві ключові кнопки:
    - *«Симуляція»* — запускає розрахунок математичної моделі для введених даних.

- *«Генерація графіків»* — формує та зберігає пакет візуалізацій (графіки динаміки SOC, діаграми автономності) у окрему папку для подальшого аналізу.

3. *Таблиця результатів (праворуч).* Інтерактивна таблиця, що відображає підсумкові метрики відразу після завершення розрахунку. Вона дозволяє миттєво порівняти ефективність Базового та Адаптивного алгоритмів за п'ятьма ключовими показниками:
   - Сумарний час без електропостачання (год).
   - Час гарантованої автономної роботи (год).
   - Приріст автономності роботи (%).

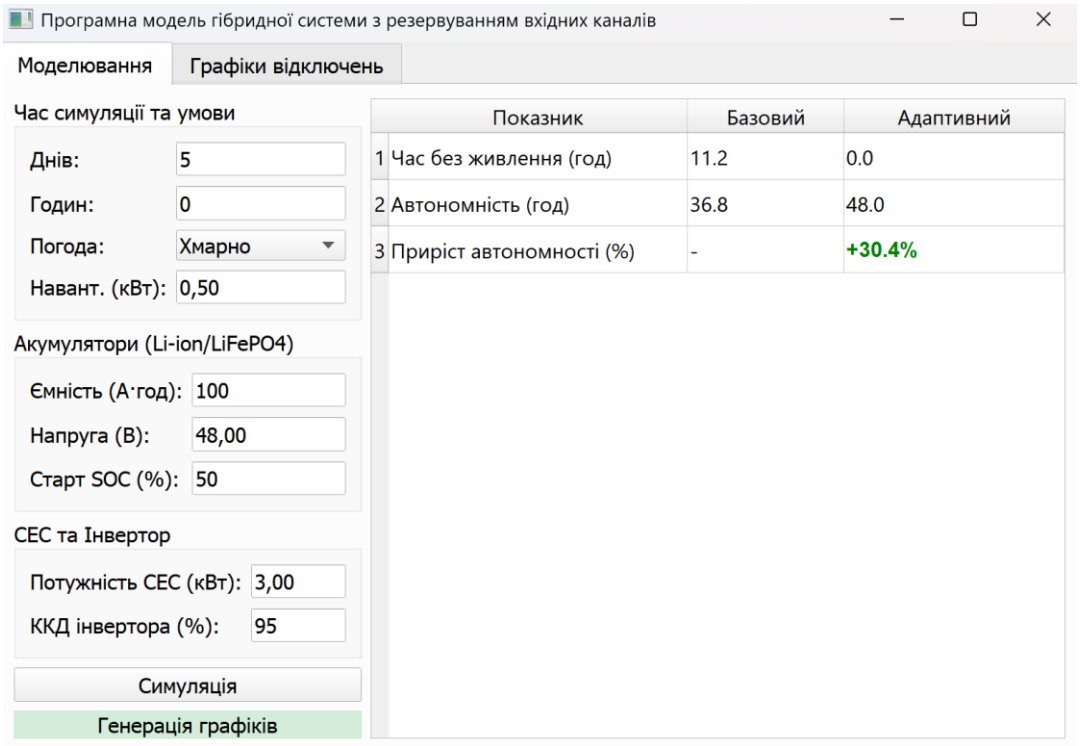


Рисунок 4 – Головне вікно програмного комплексу

Окремо винесена вкладка «Графіки відключень», яка містить інтерактивну таблицю-календар для налаштування погодинного розкладу наявності електромережі (рис. 5).

Її інтерфейс надає користувачеві наступні можливості:

1. *Керування шаблонами.* У верхній частині розміщено випадаючий список для швидкого вибору попередньо налаштованих графіків (наприклад, «4 через 4», «6 через 6», «Ніч без мережі»). Це дозволяє миттєво сформувати типовий сценарій для тестування.
2. *Інтерактивна часова матриця.* Основний простір займає таблиця-календар, що охоплює період у 4 тижні (28 днів).

- *Структура:* Рядки відповідають дням тижня, згрупованим у тижневі блоки. Стовпці відповідають годинам доби (від 00:00 до 23:00).
- *Взаємодія:* Кожна клітинка є інтерактивним чекбоксом. Встановлена позначка означає, що у цю годину електропостачання відсутнє (відключення). Користувач може вручну редагувати кожну годину, створюючи унікальні, нестандартні графіки.

3. *Очищення даних*. Кнопка «Очистити» в нижній частині вікна дозволяє скинути всі налаштування розкладу для початку нового моделювання.

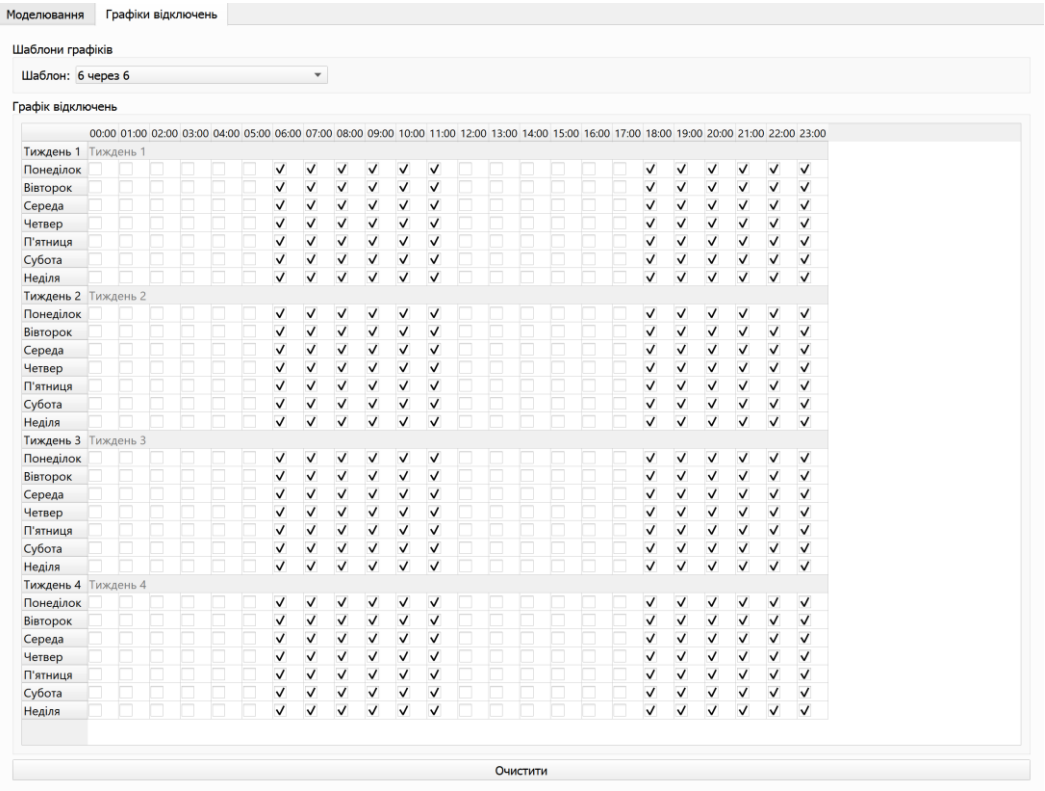


Рисунок 5 – Вкладка «Графіки відключень» програмної моделі

3.2 Цінність програмної реалізації для розробників енергетичних систем

Розроблена програмна модель має не лише наукове, а й прикладне значення для інженерів-проектувальників систем безперебійного живлення. Обрана форма реалізації (об'єктно-орієнтований підхід мовою Python з графічним інтерфейсом) надає розробникам наступні переваги:

1. Швидке прототипування та перевірка гіпотез. Використання програмної симуляції дозволяє перевірити ефективність нових алгоритмів керування (наприклад, зміну логіки порогів заряду) за лічені секунди без необхідності проведення натурних випробувань. Це значно скорочує цикл розробки (R&D), оскільки виключає ризик пошкодження дорого обладнання (інверторів та літієвих АКБ) внаслідок помилок у логіці керування на ранніх етапах.

2. Верифікація параметрів обладнання перед закупівлею. Модульна структура програми дозволяє змінювати параметри системи у широкому діапазоні. Це дає змогу розробнику точно підібрати необхідну ємність акумуляторів під конкретний профіль споживання та графік відключень ще до моменту закупівлі обладнання. Наприклад, модель дозволяє визначити, чи вистачить масиву СЕС потужністю 3 кВт для заряду АКБ ємністю 200 А·год у зимовий період при графіку відключень "4 через 4".

3. Стрес-тестування граничних сценаріїв. Інструментарій створення кастомних графіків відключень дозволяє моделювати критичні ситуації, які складно або неможливо відтворити в лабораторних умовах: робота системи при тривалому графику відключень або поведінка алгоритму при хаотичних, непередбачуваних відключеннях.

4. Адекватність фізичної моделі. На відміну від спрощених лінійних калькуляторів енергії, дана модель враховує нелінійні характеристики заряду. Це критично важливо для розробників, оскільки ігнорування цих факторів призводить до завищених очікувань щодо швидкості відновлення заряду.

Програмна модель показує реальну картину: останні 10-15% ємності заряджаються повільно, що може бути вирішальним фактором при коротких інтервалах наявності мережі.

3.3 Обґрунтування вихідних даних та сценаріїв моделювання

Для забезпечення адекватності результатів розробленої моделі реальним умовам експлуатації, вхідні параметри системи були обрані на основі технічних характеристик типового обладнання, що масово використовується для забезпечення енергонезалежності приватних об’єктів.

1. *Акумуляторна батарея (АКБ)*. В якості моделі акумуляторної батареї обрано літій-залізо-фосфатну (LiFePO4) технологію з номінальною напругою 48 В та ємністю 100 А·год. Тому що літієві батареї мають набагато вищу енергетичну щільність порівняно зі свинцево-кислотними, що дозволяє їм зберігати більше енергії у меншому об’ємі. Інтеграція літієвих батарей 100 А·год, 48 В у гібридні системи, створює ефективну та надійну систему накопичення енергії. Це є сучасним стандартом для гібридних систем[7].

2. *Гібридний інвертор*. Параметри перетворення енергії відповідають характеристикам сучасних однофазних гібридних інверторів з параметром ККД 95% Це консервативне значення, що враховує реальний КПД сучасних інверторів [8]. У розробленій математичній моделі прийнято припущення про ідеальну роботу контролера пошуку точки максимальної потужності (MPPT). Втрати на DC-DC перетворення в колі сонячних панелей вважаються несуттєвими для балансової моделі або такими, що вже враховані у інтегральному погодному коефіцієнті генерації.

3. *Фотоелектричні модулі (СЕС)*. Обрана потужність: 3.0 кВт. Обґрунтування: Це оптимальна потужність для міських умов (обмежена площа території розміщення) [9]. Така потужність забезпечується, наприклад, масивом з 7-8 панелей номіналом 400-450 Вт.

4. *Профіль навантаження (Споживання)*. Обрана потужність: 0.5 кВт (500 Вт). Вибір значення 0.5 кВт (500 Вт) зроблений для моделювання критичного базового навантаження. Це типова мінімальна потужність, необхідна для підтримки життєво важливих приладів у будинку під час відключень. Використання постійного значення спрощує аналіз, дозволяючи моделі точно оцінити, як система управління справляється саме з постійним, гарантованим попитом у складних умовах.

5. *Погодні умови*. Обране значення: хмарно. Вибір обумовлений необхідністю моделювання умов, типових для зимового або перехідного сезонів, коли ймовірність віялових відключень (блекаутів) є найвищою. Хмарність у цей період суттєво знижує генерацію енергії сонячною електростанцією, забезпечуючи наближення моделі до реальних сценаріїв найбільшого енергетичного стресу.

6. *Початковий рівень АКБ*. Обране значення: 50 %. Експеримент починається з нейтрального початкового рівня заряду, що є стандартним підходом для імітаційного моделювання. Це дозволяє уникнути зміщення результатів: система не стартує ані в ідеально підготовленому стані (100% SOC), ані в аварійному (низький SOC). Таким чином, модель об'єктивно оцінює здатність алгоритмів управління (базового та покращеного) відновлювати або підтримувати баланс енергії, починаючи з середнього, неоптимізованого стану.

3.4 Вибір сценаріїв відключень та тривалості симуляції

Для проведення порівняльного аналізу ефективності алгоритмів було відібрано три сценарію моделювання, для отримання максимально змістовних результатів, які дозволять показати принципи роботи удосконалено алгоритму:

1. Сценарій «Без відключень» (Електропостачання 24 год/добу, тривалість симуляції 5 днів)

Опис: Режим, при якому енергосистема працює в нормальних, умовах, і всі споживачі отримують електроенергію цілодобово без будь-яких примусових обмежень.

Обґрунтування: Дозволить показати роботу алгоритмів у базових умовах.

2. Сценарій «Тривалий дефіцит» (6 годин мережа є,6 немає, тривалість симуляції 5 днів).

Опис: Режим значного обмеження електропостачання, за якого електроенергія подається протягом 6 годин, після чого слідує 6 годин відключення. Загальна тривалість наявності електроенергії — 12 годин на добу.

Обґрунтування: Дозволяє оцінити ефективність системи в умовах, що відповідають середньому рівню обмежень електропостачання, які спостерігалися в Україні протягом зими 2024–2025 року [10].

3. Сценарій «Критичні обмеження» (Електропостачання день є, день немає, тривалість симуляції 5 днів)

Опис: Сценарій передбачає чергування діб з наявним електропостачанням та діб повного відключення за схемою «1 день з електроенергією / 1 день без електроенергії» протягом п’яти послідовних днів. Такий режим моделює багатодобову роботу системи в умовах періодичної повної відсутності мережі з необхідністю тривалого автономного функціонування.

Обґрунтування: Даний сценарій відповідає кризовим умовам, характерним для аварійних або прифронтових регіонів, а також ситуаціям після масованих пошкоджень енергетичної інфраструктури, коли відновлення електропостачання відбувається поетапно і з тривалими перервами. На відміну від добових графіків відключень, режим «день через день» створює підвищене навантаження на акумуляторні батареї та алгоритми керування, оскільки вимагає накопичення енергії з розрахунком на повну добу автономної роботи. Це дозволяє оцінити стійкість системи, ефективність стратегій управління.

3.5 Аналіз результатів моделювання та оцінка ефективності

В ході експерименту було проведено серію симуляцій тривалістю 5 діб (120 годин) для трьох обраних сценаріїв відключень. Порівняння проводилося між Базовим алгоритмом (Self-Consumption) та розробленим Удосконаленим алгоритмом. Для комплексної оцінки ефективності було обрано набір візуалізацій: Динаміка рівня заряду протягом періоду симуляції, діаграма часу автономної роботи системи, енергетичний баланс системи для обох алгоритмів.

Сценарій 1 (Без відключень)

Графік (рис. 6) отриманий в ході моделювання, приведено для демонстрації поведінки алгоритмів у штатному режимі, коли електромережа працює стабільно і графік відключень порожній. Головна мета — показати, що за відсутності загроз адаптивний алгоритм не виконує зайвих дій і працює максимально ефективно. На графіку видно дві лінії: зелена суцільна (Адаптивний), червона пунктирна (Базовий). Вони повністю накладаються одна на одну протягом усього періоду симуляції. Це підтверджує, що за відсутності тривоги в модулі прогнозування, адаптивний алгоритм переходить у режим очікування і діє за логікою *Self-Consumption*, повністю дублюючи базовий алгоритм. З 06:00 до 18:00 рівень заряду стрімко зростає до максимуму (~95-100%). Це відбувається виключно за рахунок надлишків сонячної генерації. Зарядка від мережі в цей час заблокована в обох алгоритмах для економії коштів. Після 18:00 сонячна генерація зникає, і обидва алгоритми перекривають потреби будинку за рахунок енергії, накопиченої в акумуляторі. Обидві криві опускаються вниз і алгоритмах на рівні 20% (як у реальних системах). Це демонструє роботу обмеження, закладеного в коді. У штатному режимі система не дозволяє розряджати батарею глибше 20%, зберігаючи цей обсяг як недоторканний аварійний запас на випадок раптового зникнення мережі.

Висновок: Графік доводить, що інтеграція покращеного функціоналу не погіршує економічну ефективність системи у звичайні дні. Поки мережа стабільна, система не витрачає мережеву електроенергію на зарядку АКБ, використовуючи лише безкоштовний ресурс сонячної енергії, ідентично до класичних гібридних інверторів.

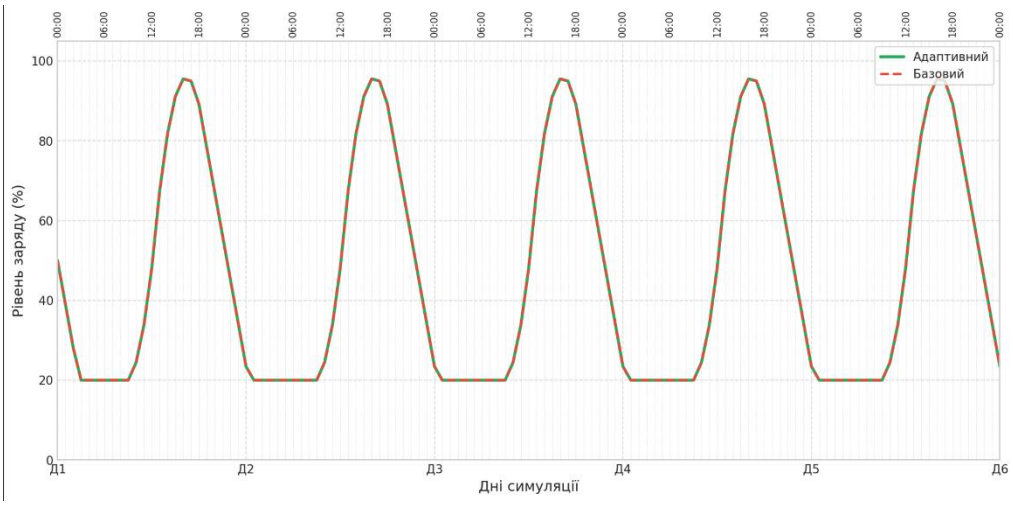


Рисунок 6 - Динаміка рівня заряду протягом періоду симуляції (Сценарій – без відключень)

Діаграму (рис. 7) згенеровано для демонстрації структури енергоспоживання у штатному режимі роботи системи. Головна мета — показати, що за відсутності активних загроз відключення обидва алгоритми (Базовий та Адаптивний) мають ідентичний профіль використання джерел енергії, максимізуючи частку безкоштовної сонячної генерації та мінімізуючи споживання з платної мережі. Сектори діаграми, що відповідають прямій генерації сонячної енергії та енергії з АКБ (яка була накопичена від Сонця), займають основну частку діаграми. Це свідчить про успішну реалізацію стратегії *Self-Consumption*: система максимізує використання безкоштовного ресурсу. Частка споживання від мережі є мінімальною і обумовлена лише моментами слабкої сонячної генерації або періодами, коли заряд АКБ досягає встановленого ліміту резерву (20%).

Висновок: Діаграми балансу для Базового та Адаптивного алгоритмів у стабільних умовах є ідентичними. Це підтверджує тезу про те, що інтеграція функцій прогнозування не призводить до додаткових витрат або неефективного використання ресурсів, коли загроза відключення відсутня. Система діє так само ощадливо, як і класичний алгоритм.

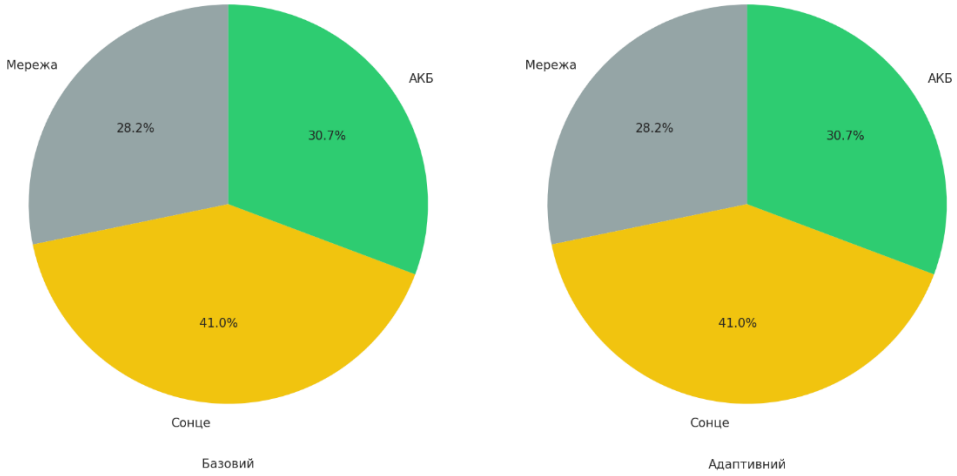


Рисунок 7 - Кругові діаграми енергетичного балансу системи обох алгоритмів в порівнянні. (Сценарій – без відключень)

Сценарій 2 (Вимкнення мережі на 6 годин кожні 6 годин)

Графік (рис.8) побудовано для демонстрації роботи системи в умовах жорсткого графіку відключень (6 годин світло є, 6 годин немає). Головна мета — візуалізувати перевагу адаптивного алгоритму, який заздалегідь готує систему до блекауту, порівняно з реактивним базовим алгоритмом. На графіку з'явилися вертикальні сірі смуги. Вони позначають періоди часу, коли зовнішня мережа фізично відсутня (графік відлючень). У ці моменти живлення будинку можливе виключно за рахунок енергії, накопиченої в акумуляторі або згенерованої сонцем у реальному часі. На відміну від попереднього сценарію, тут лінії поводяться по-різному перед початком сірої зони. Перед кожним входом у сіру зону (за 2 години) лінія яка відображає рівень заряду системи з

покращеним алгоритмом різко йде вгору і досягає 99%. Це спрацьовує модуль прогнозування: система бачить майбутнє відключення і примусово заряджає АКБ від мережі («Форсована зарядка»), ігноруючи економію заради безпеки. Базовий же алгоритм продовжує працювати таким чином як і звичайному сценарії коли є мережа. Тому він підходить до відключень не підготовленим. Якщо відключення припадає на вечір або ніч, червона лінія входить у сіру зону з низьким зарядом оскільки система не знала про наближення відключення. Система з покращеним алгоритмом починає розряджатися з повного заряду. Завдяки цьому запасу заряду вистачає на весь 6-годинний період блекауту, і лінія не опускається до критичного мінімуму. Тоді як система з базовим алгоритмом, часто починає розряд з низького рівня. На графіку видно моменти, коли червона лінія падає до аварійного порогу (низьке плато), що означає повне знеструмлення будинку (реальний блекаут для споживача).

Висновок: Графік доводить ефективність гіпотези. У той час як базовий алгоритм допускає знеструмлення будинку через «незнання» розкладу, адаптивний алгоритм гарантує автономність, жертвуючи економією заради повного заряду перед аварією.

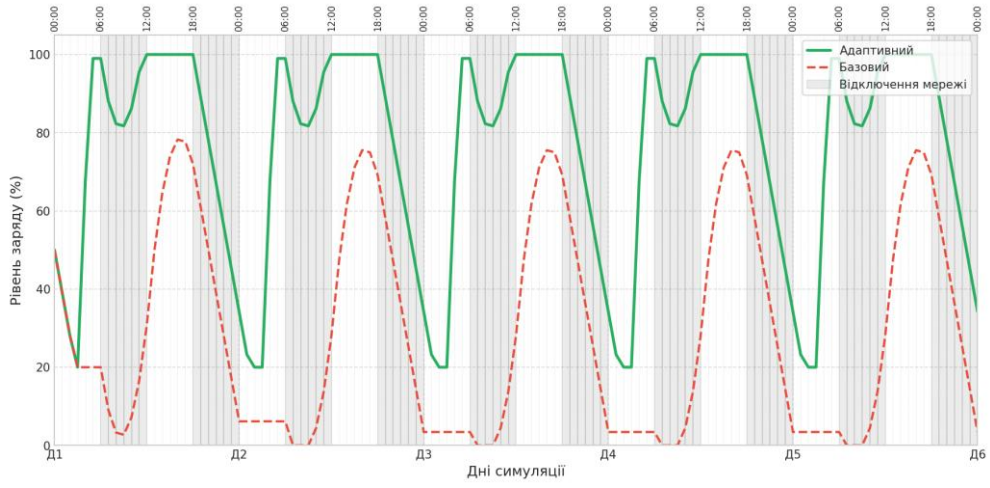


Рисунок 8 - Динаміка рівня заряду протягом періоду симуляції (Сценарій - 6 через 6)

Діаграма (рис. 9) побудована для демонстрації, як зміна пріоритетів алгоритму впливає на структуру джерел енергії в умовах частих відключень. Вона дозволяє оцінити «ціну» надійності, яку сплачує користувач за безперебійне живлення.

Порівняльний аналіз:

1. Баланс мережі:

- Базовий: 23.7%
- Адаптивний: 40.7% Це найбільш показова зміна. Адаптивний алгоритм майже вдвічі збільшує споживання платної енергії з мережі. Це прямий наслідок роботи функції «Форсованої зарядки»: система не чекає на сонце, а активно бере енергію з мережі, щоб встигнути зарядити батарею у короткі проміжки наявності світла.

2. Баланс сонячної енергії:

- Базовий: 42.8%
- Адаптивний: 26.5% Через те, що батарея часто примусово заряджається від мережі перед відключенням, у ній залишається менше вільного місця для сонячної енергії. Коли з'являється сонце, батарея може бути вже повною (або відключеною від мережі), що змушує систему ігнорувати частину потенційної генерації.

Висновок: Діаграма показує, що в кризових умовах адаптивний алгоритм свідомо жертвує економічною ефективністю (частка безкоштовної енергії падає) заради енергетичної безпеки. Це стратегічне рішення дозволяє уникнути блекаутів, які наводяться на попередньому графіку рівня зарядів.

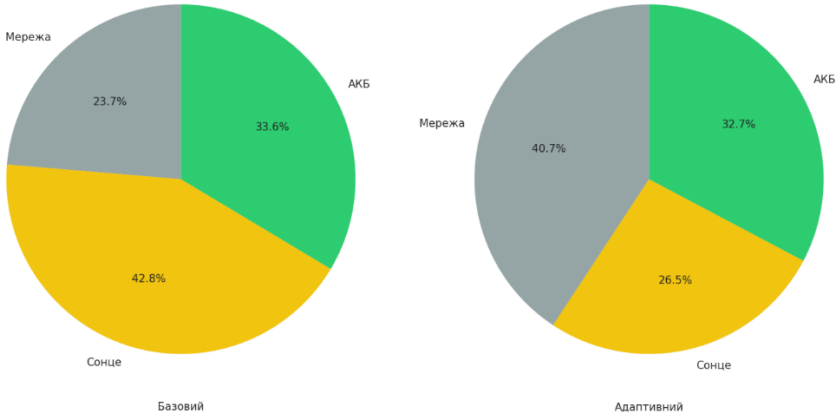


Рисунок 9 - Кругові діаграми енергетичного балансу системи обох алгоритмів в порівнянні. (Сценарій відключення - 6 через 6)

Аналіз показників автономності та надійності:

На рис.10 відображені кількісні результати моделювання.

Ключові показники:

1. *Час без живлення:*
   - Базовий: 4.8 години. Це час, коли мешканці будинку залишалися без електроенергії. Базовий алгоритм не зміг забезпечити живлення через низький рівень заряду на момент початку відключень.
   - Адаптивний: 0.0 годин. Це головне досягнення. Попри те, що графік відключень був дуже жорстким (кожні 6 годин без світла), жодного разу не допустив повного розряду батареї, забезпечивши 100% безперебійне живлення.
2. *Години автономності:*
   - Базовий: 55.2 год.
   - Адаптивний: 60.0 год. Різниця у 4.8 години — це саме той критичний час, коли базовий алгоритм "здався", а адаптивний продовжив роботу завдяки попередньо накопиченому резерву.

3. *Приріст автономності (+8.7%):* Цей відсоток демонструє реальну ефективність алгоритму. Без збільшення фізичної ємності батареї система отримала майже 9% додаткового часу роботи виключно за рахунок оптимізації логіки управління.

Висновок: Результати моделювання підтверджують, що проактивна стратегія заряду дозволяє перетворити систему з дефіцитом енергії (4.8 години) на систему повної автономності в ідентичних умовах, за умови імплементації адаптивного алгоритму. Це доводить перспективність впровадження покращеного алгоритму у побутові інвертори для регіонів з нестабільним електропостачанням.

	Показник	Базовий	Адаптивний
1	Час без живлення (год)	4.8	0.0
2	Автономність (год)	55.2	60.0
3	Приріст автономності (%)	-	**+8.7%**

Рисунок 10 - Кількісні результати моделювання (Сценарій – відключення 6 через 6) (Скріншот з програмної моделі)

Сценарій 3 (Відключення через день на 24 години)

Одним з результатів моделювання є графік (рис. 11) який ілюструє роботу системи в умовах інтенсивних відключень, де базовий алгоритм не справляється із забезпеченням потреб будинку. Мета візуалізації — показати, як стратегія попереджувального заряду дозволяє уникнути глибоких "провалів" енергопостачання.

Червона пунктирна лінія яка позначає динаміку рівня заряду демонструє численні падіння до нуля. Це моменти, коли батарея повністю розряджена, а

зовнішня мережа відсутня. Для користувача це означає повне знеструмлення (блекаут).

Зелена лінія яка відповідає за адаптивний алгоритм демонструє характерні "піки" перед зонами відключень. Система завчасно заряджає АКБ до 99% від мережі. Завдяки цьому, навіть під час тривалих відключень, зелена лінія не опускається до критичних значень, забезпечуючи безперервне живлення.

Висновок: Графік наочно демонструє, що в цьому сценарії базовий алгоритм є неспроможним: він призводить до регулярних відключень споживача. Натомість розроблений алгоритм, завдяки агресивній підготовці, утримує систему у робочому стані весь час.

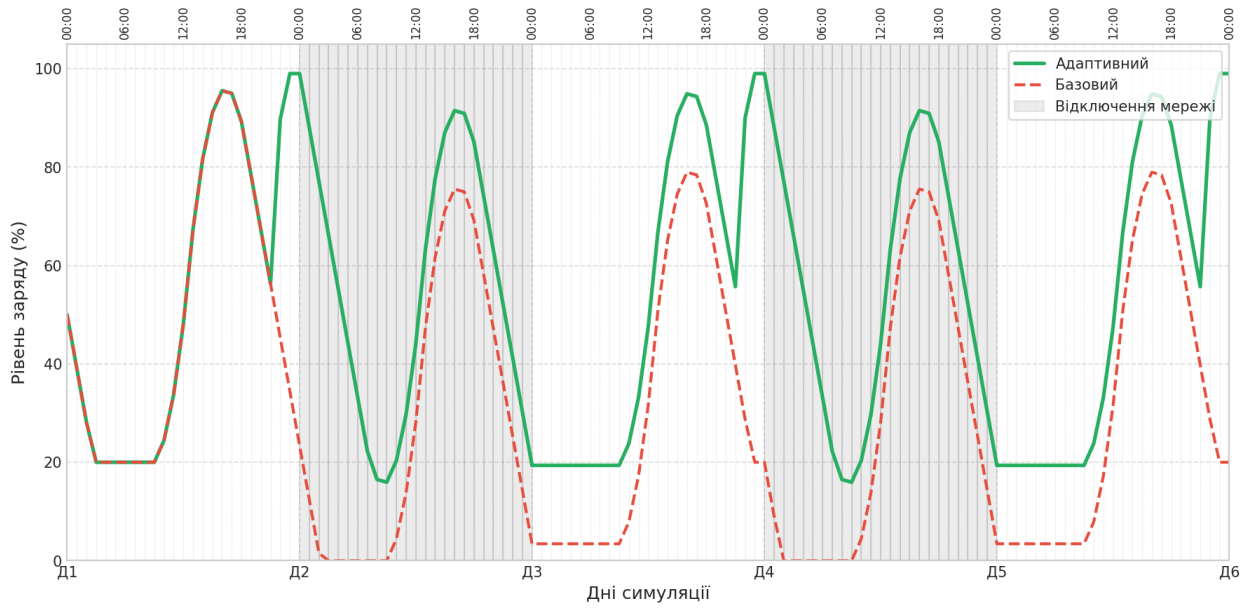


Рисунок 11 - Динаміка рівня заряду протягом періоду симуляції (Сценарій – відключення через день довжиною 24 години)

Одним з результатів моделювання є кругова діаграма (рис. 12) яка показує зміни структури джерел енергії при переході на "розумне" управління в умовах значного дефіциту мережі.

Порівняльний аналіз:

1. Частка Мережі:
    - Базовий: 20.6%
    - Адаптивний: 30.0% Алгоритм збільшив споживання платної енергії майже на 10 відсоткових пунктів. Це втрати економії за надійність: замість того, щоб знаходитися у блекауті система добрала необхідну енергію з мережі заздалегідь.
2. Частка сонячної енергії:
    - Базовий: 45.3%
    - Адаптивний: 37.9% Через пріоритетну зарядку від мережі перед відключенням, система іноді "пропускає" можливість зарядитися від сонячної енергії (бо батарея вже повна). Однак, враховуючи критичність ситуації, це виправдано.
3. Частка АКБ :
    - Базовий: 33.6%
    - Адаптивний: 32.7% Цікавим фактом є те, що частка енергії, яка проходить через акумулятор, залишається практично незмінною. Це свідчить про те, що батарея працює як стабільний буфер незалежно від джерела її наповнення. В обох випадках система використовує доступну цикловану ємність АКБ майже на максимум, але в "розумному" режимі цей ресурс гарантовано доступний саме в моменти відключень.

Висновок: Дана діаграма візуалізує компроміс, закладений в основу адаптивного алгоритму. Ми спостерігаємо помірне, але критично важливе зміщення енергетичного балансу в бік гарантованого джерела (мережі), що є менш радикальним, ніж у сценарії 6/6, проте достатнім для повної стабілізації системи. Зниження частки сонячної генерації в балансі пояснюється ефектом заміщення: щоб гарантувати заряд близький к повному на момент відключення, система змушена ігнорувати частину потенційної сонячної

генерації, якщо батарея вже була превентивно заряджена від мережі. Такий підхід трансформує систему з економічно-орієнтованої на безпеково-орієнтовану без надмірних витрат ресурсів.

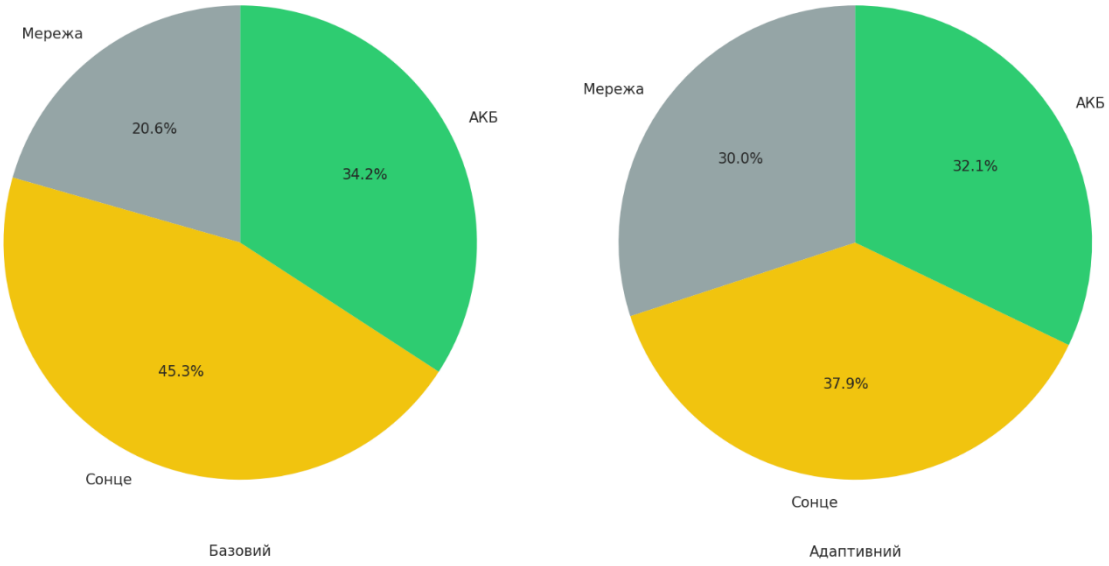


Рисунок 12 - Кругові діаграми енергетичного балансу системи обох алгоритмів в порівнянні. (Сценарій – відключення на день через день)

Аналіз показників автономності та надійності:

На рис.13 відображені кількісні результати моделювання для фіксація фінальних метрик ефективності, які доводять перевагу розробленого методу кількісних показниках.

Ключові показники:

1. Час без живлення:
    - Базовий: 11.2 години. Це критичний показник. Майже 11 годин споживач перебував без електроенергії. Це свідчить про повний провал стратегії *Self-Consumption* у даних умовах.
    - Адаптивний: 0.0 годин. Абсолютна надійність. Система жодного разу не підвела користувача.

2. Години автономності:
    - Базовий: 36.8 год.
    - Адаптивний: 48.0 год. Система з покращеним алгоритмом працювала автономно на 11.2 години довше (рівно на той час, який базовий алгоритм провів у блекауті). Фактично, це означає 100% покриття потреб протягом 2 діб (48 годин) відсутності мережі.
3. Приріст автономності (+30.4%): Це рекордний показник ефективності серед протестованих сценаріїв. Збільшення часу роботи на третину (+30.4%) без заміни обладнання — це вагомий аргумент на користь програмної модернізації інверторів.

Загальний висновок: Моделювання показало, що чим складніші умови, тим більший приріст автономності показує адаптивний алгоритм. Якщо в м'яких сценаріях приріст складав ~9%, то в критичних умовах він сягає понад 30%, перетворюючи недієздатну систему на повністю автономну.

	Показник	Базовий	Адаптивний
1	Час без живлення (год)	11.2	0.0
2	Автономність (год)	36.8	48.0
3	Приріст автономності (%)	-	**+30.4%**

Рисунок 13 - Кількісні результати моделювання сценарію №3 (Сценарій – відключення на день через день) (Скріншот з програмної моделі)

3.6 Висновки до розділу

Експериментальні дослідження підтвердили працездатність та ефективність запропонованого адаптивного алгоритму керування. Порівняльний аналіз показав, що використання прогностичних даних про графіки відключень дозволяє суттєво підвищити рівень гарантованої

автономності системи порівняно зі стандартними алгоритмами власного споживання. Програмна реалізація моделі надає зручний та гнучкий інструмент для подальшого аналізу та оптимізації параметрів гібридних систем електроживлення.

ВИСНОВКИ

У дипломній роботі вирішено актуальну науково-технічну задачу підвищення автономності гібридних систем безперебійного живлення шляхом розробки адаптивного алгоритму керування енергетичними потоками.

У ході дослідження було проведено комплексний аналіз предметної області та існуючих рішень, який дозволив встановити, що традиційні стратегії керування гібридними інверторами є статичними та неефективними в умовах частих та тривалих відключень електроенергії. Виявлено, що головним недоліком існуючих систем є неможливість прогнозування стану мережі, що може призводити до критичного розряду акумуляторних батарей безпосередньо перед початком аварійного відключення.

Ключовим результатом роботи стала розробка адаптивного алгоритму керування. Запропоновано новий підхід до формування керуючих впливів. На відміну від базових алгоритмів, розроблений метод використовує модуль прогнозування, який аналізує графік планових відключень та завчасно ініціює примусовий заряд АКБ лише у випадку прогнозованого дефіциту енергії.

Для перевірки ефективності та адекватності запропонованого алгоритму було розроблено математичну модель гібридної системи у програмній реалізації. Створена модель, реалізована мовою Python, враховує фізичні обмеження літій-іонних акумуляторів, зокрема залежність швидкості заряду. Це дозволило забезпечити високу точність прогнозування часу заряду та оцінки реальної автономності системи.

Для підтвердження ефективності запропонованого рішення проведено порівняльне моделювання роботи системи під керуванням розробленого адаптивного алгоритму та базової стратегії. Експериментальні дослідження показали суттєву перевагу запропонованого алгоритму над базовим алгоритмом. Зокрема, при моделюванні роботи системи протягом 5 днів з інтенсивним графіком відключень зафіксовано скорочення часу перебування об'єкта без електропостачання значні проміжки часу при різних сценаріях

моделювання . Приріст автономності при впроваджені адаптивного алгоритму у модель становить 8.7% та 30.4%, при виконанні другого (відключення через 6 годин на 6 годин) та третього сценарію відключень (відключення через день на день).

Доведено доцільність впровадження адаптивного алгоритму як ефективного засобу підвищення автономності. Запропоноване рішення дозволяє досягти балансу між мінімізацією споживання електроенергії з мережі в спокійний час та гарантуванням максимальної тривалості автономної роботи в кризові моменти. Алгоритм не лише забезпечує надійне живлення навантаження під час блекаутів, але й запобігає глибоким розрядам АКБ, що за оцінками може подовжити термін експлуатації накопичувачів енергії.

Таким чином, запропоновані методи та засоби дозволяють створити інтелектуальну систему керування, яка перетворює звичайну гібридну систему з резервуванням каналів на надійне джерело безперебійного живлення, адаптоване до складних умов експлуатації.

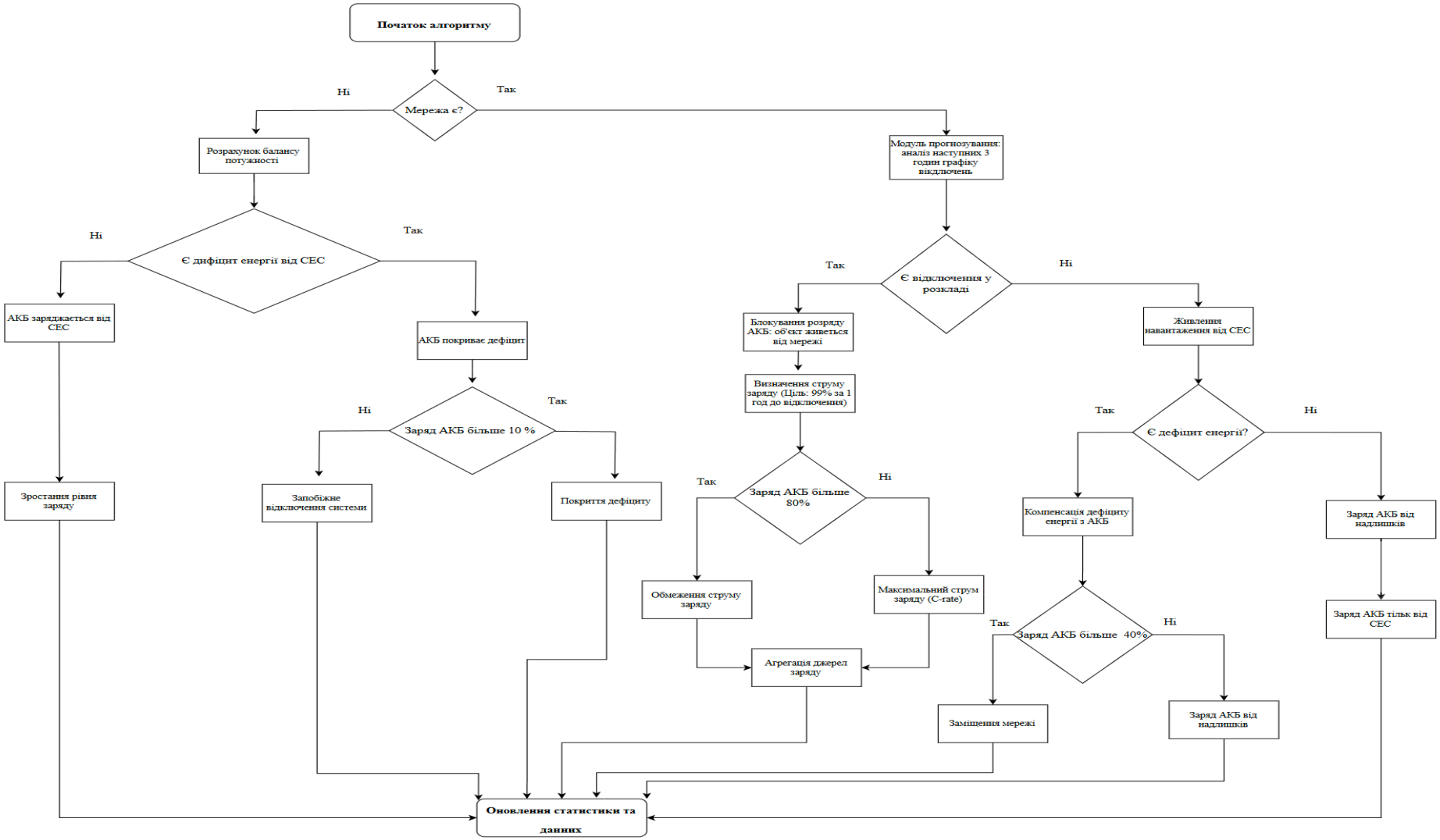
ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке блекаут: повний гід з причинами, наслідками та прикладами [Електронний ресурс] / Режим доступу: https://kokl.ua/shho-take-blekaut-povnyj-gid-z-prychynamy-naslidkamy-ta-prykladamy/ [Дата звернення 01.11.2025]
2. How do grid-tied energy storage systems operate during blackouts? [Електронний ресурс] / Режим доступу: https://nenpower.com/blog/how-do-grid-tied-energy-storage-systems-operate-during-blackouts/?utm [Дата звернення 02.11.2025]
3. Battery backup for computers: essential protection for every workspace [Електронний ресурс] / Режим доступу: https://www.lenovo.com/us/en/knowledgebase/battery-backup-for-computers-a-comprehensive-guide/ [Дата звернення 02.11.2025]
4. A primer on the unintentional islanding
Protection requirement in ieee std 1547-2018 [електронний ресурс] / Режим доступу: https://docs.nrel.gov/docs/fy22osti/77782.pdf?utm [Дата звернення 02.11.2025]
5. Research on energy management strategy of photovoltaic–battery energy storage system [електронний ресурс] / Режим доступу: https://academic.oup.com/ijlct/article/doi/10.1093/ijlct/ctac024/6540291?utm [Дата звернення 02.11.2025]
6. Каковы 4 режима работы гибридного инвертора? [електронний ресурс] / Режим доступу: https://www.bsl-battery.com/ru/news/what-are-the-4-operating-modes-of-a-hybrid-inverter/ [Дата звернення 07.11.2025]
7. Why 48V Battery Systems Are Ideal for Integrated Battery and Inverter Systems in the European Market [електронний ресурс] / Режим доступу: https://www.greenteche.com/why-48v-battery-systems-are-ideal-for-integrated-battery-and-inverter-systems-in-the-european-market.html?utm_ [Дата звернення 25.11.2025]

8. Efficiency of Inverters [електронний ресурс] / Режим доступу: https://courses.ems.psu.edu/eme812/node/738?utm_ [Дата звернення 25.11.2025]
9. 3 kW solar panel system: Is it right for your home? [електронний ресурс] / Режим доступу: https://www.theecoexperts.co.uk/solar-panels/3kw-solar-panel-system?utm_ [Дата звернення 30.11.2025]
10. Energy infrastructure attacks: updated outlook and impact during the 2024–2025 cold season [електронний ресурс] / Режим доступу: https://www.acaps.org/fileadmin/Data_Product/Main_media/20250219_ACAPS_Ukraine_-_Energy_infrastructure_attacks-_Updated_outlook_and_impact_during_the_2024-2025_cold_season_.pdf?utm_ [Дата звернення 30.11.2025]

Додаток А

Блок-схема адаптивного алгоритму



Додаток Б

Ісходний код розробленної програмної моделі

```
import sys
import os
import math
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QLabel,
    QSpinBox, QDoubleSpinBox, QPushButton, QTabWidget, QFormLayout,
QTableWidget,
    QTableWidgetItem, QComboBox, QGroupBox, QMessageBox, QTextEdit,
QHBoxLayout,
    QHeaderView, QAbstractSpinBox
)
from PyQt5.QtGui import QFont, QColor
from PyQt5.QtCore import Qt

try:
    import matplotlib.pyplot as plt
    import numpy as np
    import matplotlib.colors as mcolors
    from matplotlib.ticker import MultipleLocator
    MATPLOTLIB_AVAILABLE = True
except ImportError:
    MATPLOTLIB_AVAILABLE = False

def simulate(sim_time_h=24, batt_capacity_ah=200, batt_voltage_v=48,
             pv_power_kw=3.0, load_kw=0.5, weather="Сонячно",
             outage_schedule=None,
```

```
        soc_init=0.5, eta_inv=0.95, algo="base"):

LI_ION_C_RATE = 0.5
LI_ION_CHARGE_EFF = 0.95
CV_START_SOC = 0.80

SOC_MIN_BASE = 0.20
SOC_HARD_LIMIT = 0.10

def get_pv_generation(hour, pv_power_kw, pv_coeff):
    if 6 <= hour < 18:
        peak_hour = 12
        intensity = 1 - abs(hour - peak_hour) / 6
        return pv_power_kw * pv_coeff * max(0.0, intensity)
    return 0.0

weather_coeff = {"Сонячно": 1.0, "Хмарно": 0.5}
pv_coeff = weather_coeff.get(weather, 1.0)

batt_capacity_kwh = (batt_capacity_ah * batt_voltage_v) / 1000.0
max_batt_charge_power_kw = batt_capacity_kwh * LI_ION_C_RATE

soc = soc_init
soc_log = []

hist_grid = []
hist_pv = []
hist_batt = []
```

```
hist_load = []
hist_charge_kw = []

grid_energy = 0.0
pv_to_load_ac = 0.0
batt_to_load_ac = 0.0
pv_total_dc = 0.0
pv_to_batt_dc = 0.0

hours_no_power = 0.0
hours_autonomy = 0.0
min_soc = soc_init

if outage_schedule is None: outage_schedule = {}

for t in range(int(sim_time_h)):
    total_hours_from_start = t
    current_day = (total_hours_from_start // 24) % 28
    current_hour = total_hours_from_start % 24

    step_charge_dc = 0.0
    step_pv_used_ac = 0.0
    step_batt_used_ac = 0.0
    step_grid_used = 0.0

    grid_ok = True
    if current_day in outage_schedule:
        if current_hour in outage_schedule[current_day]:
```

```
            grid_ok = False

        pv_gen = get_pv_generation(current_hour, pv_power_kw, pv_coeff)
        pv_total_dc += pv_gen
        load_need = load_kw

        current_max_charge = max_batt_charge_power_kw

        if soc > CV_START_SOC:
            cv_factor = (1.0 - soc) / (1.0 - CV_START_SOC)
            current_max_charge *= max(0.0, cv_factor)

        def calculate_charge(source_power_available, target_limit_soc=1.0):
            power_limit = min(source_power_available, current_max_charge)
            space_kwh = max(0.0, (target_limit_soc - soc) * batt_capacity_kwh)
            energy_input_limit = space_kwh / LI_ION_CHARGE_EFF
            actual_input_kw = min(power_limit, energy_input_limit)
            return actual_input_kw

        if algo == "base":
            if grid_ok:
                step_pv_used_ac = min(load_need, pv_gen * eta_inv)
                rest_load = load_need - step_pv_used_ac

                if rest_load > 0 and soc > SOC_MIN_BASE:
                    batt_energy_available_kwh = (soc - SOC_MIN_BASE) *
batt_capacity_kwh
                    batt_power_available_ac = batt_energy_available_kwh * eta_inv
```

```
            step_batt_used_ac = min(rest_load, batt_power_available_ac)

            discharged_kwh_dc = step_batt_used_ac / eta_inv
            soc -= discharged_kwh_dc / batt_capacity_kwh
            rest_load -= step_batt_used_ac

        step_grid_used = rest_load

        pv_used_for_load_dc = step_pv_used_ac / eta_inv
        pv_excess_dc = max(0.0, pv_gen - pv_used_for_load_dc)

        if pv_excess_dc > 0 and soc < 1.0:
            step_charge_dc = calculate_charge(pv_excess_dc, 1.0)
            soc += (step_charge_dc * LI_ION_CHARGE_EFF) / batt_capacity_kwh

    else:
        step_pv_used_ac = min(load_need, pv_gen * eta_inv)
        pv_used_for_load_dc = step_pv_used_ac / eta_inv
        rest_load = load_need - step_pv_used_ac

        if rest_load > 0 and soc > 0.001:
            available_batt_energy_ac = soc * batt_capacity_kwh * eta_inv
            step_batt_used_ac = min(rest_load, available_batt_energy_ac)

            discharged_kwh_dc = step_batt_used_ac / eta_inv
            soc -= discharged_kwh_dc / batt_capacity_kwh
            rest_load -= step_batt_used_ac
```

```
            pv_excess_dc = max(0.0, pv_gen - pv_used_for_load_dc)
            if pv_excess_dc > 0 and soc < 1.0:
                charge_kw = calculate_charge(pv_excess_dc, 1.0)
                soc += (charge_kw * LI_ION_CHARGE_EFF) / batt_capacity_kwh
                step_charge_dc += charge_kw

            if load_need > 1e-6:
                fraction_deficit = rest_load / load_need
                fraction_deficit = max(0.0, min(1.0, fraction_deficit))
                hours_no_power += fraction_deficit
                hours_autonomy += (1.0 - fraction_deficit)
            else:
                hours_autonomy += 1.0

    elif algo == "improved":
        lookahead_hours = 3
        upcoming_outages = []

        for h_offset in range(1, lookahead_hours + 1):
            check = total_hours_from_start + h_offset
            d, h = (check // 24) % 28, check % 24
            if d in outage_schedule and h in outage_schedule[d]:
                upcoming_outages.append(h_offset)

        target_soc = 0.99
        needed_charge_rate = 0.0

        if upcoming_outages and grid_ok:
```

```
            next_outage_delay = min(upcoming_outages)

            time_to_charge_hours = max(1.0, next_outage_delay - 1.0)

            k_factor = (1.0 - CV_START_SOC) / LI_ION_C_RATE
            min_time_cv_phase = k_factor * (math.log(1.0 - CV_START_SOC) -
math.log(1.0 - target_soc))
            min_time_cv_phase *= 1.05

            if soc >= CV_START_SOC:
                needed_charge_rate = max_batt_charge_power_kw
            else:
                time_for_bulk = time_to_charge_hours - min_time_cv_phase

                if time_for_bulk <= 0.1:
                    needed_charge_rate = max_batt_charge_power_kw
                else:
                    energy_to_cv_start = (CV_START_SOC - soc) * batt_capacity_kwh
                    if energy_to_cv_start > 0:
                        needed_charge_rate = energy_to_cv_start / time_for_bulk
                        needed_charge_rate *= 1.1
                    else:
                        needed_charge_rate = 0

            if soc < target_soc:
                step_grid_used = load_need

                if pv_gen > 0:
```

```
                pv_charge = calculate_charge(pv_gen, target_soc)
                step_charge_dc += pv_charge
                soc += (pv_charge * LI_ION_CHARGE_EFF) / batt_capacity_kwh

            if soc < target_soc:
                actual_charge_rate = min(needed_charge_rate, current_max_charge)
                grid_charge = calculate_charge(actual_charge_rate, target_soc)
                step_charge_dc += grid_charge
                step_grid_used += grid_charge
                soc += (grid_charge * LI_ION_CHARGE_EFF) / batt_capacity_kwh

        else:
            if grid_ok:
                step_pv_used_ac = min(load_need, pv_gen * eta_inv)
                rest_load = load_need - step_pv_used_ac

                if rest_load > 0 and soc > SOC_MIN_BASE:
                    batt_energy_available_kwh = (soc - SOC_MIN_BASE) *
batt_capacity_kwh
                    batt_power_available_ac = batt_energy_available_kwh * eta_inv
                    step_batt_used_ac = min(rest_load, batt_power_available_ac)

                    discharged_kwh_dc = step_batt_used_ac / eta_inv
                    soc -= discharged_kwh_dc / batt_capacity_kwh
                    rest_load -= step_batt_used_ac

                step_grid_used = rest_load
```

```
            pv_used_for_load_dc = step_pv_used_ac / eta_inv
            pv_excess_dc = max(0.0, pv_gen - pv_used_for_load_dc)

            if pv_excess_dc > 0 and soc < 1.0:
                step_charge_dc = calculate_charge(pv_excess_dc, 1.0)
                soc += (step_charge_dc * LI_ION_CHARGE_EFF) / batt_capacity_kwh

    if not grid_ok:
        step_pv_used_ac = min(load_need, pv_gen * eta_inv)
        pv_used_for_load_dc = step_pv_used_ac / eta_inv
        rest_load = load_need - step_pv_used_ac

        if rest_load > 0 and soc > SOC_HARD_LIMIT:
            soc_avail = soc - SOC_HARD_LIMIT
            energy_avail = soc_avail * batt_capacity_kwh * eta_inv
            step_batt_used_ac = min(rest_load, energy_avail)

            discharged_kwh_dc = step_batt_used_ac / eta_inv
            soc -= discharged_kwh_dc / batt_capacity_kwh
            rest_load -= step_batt_used_ac

        pv_excess_dc = max(0.0, pv_gen - pv_used_for_load_dc)
        if pv_excess_dc > 0 and soc < 1.0:
            charge_kw = calculate_charge(pv_excess_dc, 1.0)
            soc += (charge_kw * LI_ION_CHARGE_EFF) / batt_capacity_kwh
            step_charge_dc += charge_kw

        if load_need > 1e-6:
```

```
                fraction_deficit = rest_load / load_need
                fraction_deficit = max(0.0, min(1.0, fraction_deficit))
                hours_no_power += fraction_deficit
                hours_autonomy += (1.0 - fraction_deficit)
            else:
                hours_autonomy += 1.0

        soc = max(0.0, min(1.0, soc))
        min_soc = min(min_soc, soc)
        soc_log.append(soc * 100)

        grid_energy += step_grid_used
        pv_to_load_ac += step_pv_used_ac
        batt_to_load_ac += step_batt_used_ac
        pv_to_batt_dc += step_charge_dc

        hist_grid.append(step_grid_used)
        hist_pv.append(step_pv_used_ac)
        hist_batt.append(step_batt_used_ac)
        hist_load.append(load_need)
        hist_charge_kw.append(step_charge_dc)

    avg_soc = sum(soc_log) / len(soc_log) if soc_log else 0

    results = {
        "grid_energy": grid_energy,
        "pv_to_load_ac": pv_to_load_ac,
        "batt_to_load_ac": batt_to_load_ac,
```

```
        "hours_no_power": hours_no_power,
        "hours_autonomy": hours_autonomy,
        "min_soc": min_soc * 100,
        "avg_soc": avg_soc,
    }
    history = {
        "grid": hist_grid,
        "pv": hist_pv,
        "batt": hist_batt,
        "load": hist_load,
        "charge_kw": hist_charge_kw
    }
    return results, soc_log, history

class SimpleOutageCalendar(QWidget):
    def __init__(self):
        super().__init__()
        self.outage_schedule = {}
        self.days_of_week = ["Понеділок", "Вівторок", "Середа", "Четвер", "П'ятниця",
"Субота", "Неділя"]
        self.init_ui()

    def init_ui(self):
        layout = QVBoxLayout()
        templates_group = QGroupBox("Шаблони графіків")
        templates_layout = QVBoxLayout()
        template_combo_layout = QHBoxLayout()
        template_combo_layout.addWidget(QLabel("Шаблон:"))
```

```
self.template_combo = QComboBox()
self.template_combo.addItems([
    "Власний шаблон",
    "Без відключень",
    "Ранок та Вечір (08:00–11:00, 18:00–22:00)",
    "День через день",
    "4 через 4",
    "2 через 4",
    "6 через 6",
    "Ніч без мережі (23:00–07:00)",
])
self.template_combo.currentTextChanged.connect(self.apply_template)
template_combo_layout.addWidget(self.template_combo)
template_combo_layout.addStretch()
templates_layout.addLayout(template_combo_layout)
templates_group.setLayout(templates_layout)
layout.addWidget(templates_group)

table_group = QGroupBox("Графік відключень (День 1 - Понеділок)")
table_layout = QVBoxLayout()

self.table = QTableWidget(32, 24)

row_headers = []
for week in range(4):
    row_headers.append("")
    for day in self.days_of_week:
        row_headers.append(day)
```

```
self.table.setVerticalHeaderLabels(row_headers)
hour_labels = [f"{h:02d}:00" for h in range(24)]
self.table.setHorizontalHeaderLabels(hour_labels)
self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Fixed)
self.table.verticalHeader().setSectionResizeMode(QHeaderView.Fixed)
for i in range(24): self.table.setColumnWidth(i, 60)
for i in range(32): self.table.setRowHeight(i, 30)

row_index = 0
for week in range(4):
    for hour in range(24):
        item = QTableWidgetItem()
        item.setFlags(Qt.NoItemFlags)
        item.setBackground(QColor(240, 240, 240))
        self.table.setItem(row_index, hour, item)
    self.table.setSpan(row_index, 0, 1, 24)
    self.table.item(row_index, 0).setText(f"Тиждень {week+1}")
    row_index += 1

    for day in range(7):
        for hour in range(24):
            item = QTableWidgetItem()
            item.setTextAlignment(Qt.AlignCenter)
            item.setFlags(item.flags() | Qt.ItemIsUserCheckable)
            item.setCheckState(Qt.Unchecked)
            self.table.setItem(row_index, hour, item)
        row_index += 1
```

```
        self.table.itemChanged.connect(self.on_item_changed)
        table_layout.addWidget(self.table)
        table_group.setLayout(table_layout)
        layout.addWidget(table_group)

        btns_layout = QHBoxLayout()
        self.btn_cl = QPushButton("Очистити")
        self.btn_cl.clicked.connect(self.clear_all)
        btns_layout.addWidget(self.btn_cl)
        layout.addLayout(btns_layout)
        self.setLayout(layout)

    def get_day_index(self, table_row):
        week = table_row // 8
        day_in_week = (table_row % 8) - 1
        if day_in_week < 0 or day_in_week >= 7: return -1
        return week * 7 + day_in_week

    def get_table_row(self, day_index):
        week = day_index // 7
        day_in_week = day_index % 7
        return week * 8 + 1 + day_in_week

    def on_item_changed(self, item):
        if item is None: return
        day_index = self.get_day_index(item.row())
        if day_index == -1: return
```

```
    hour = item.column()
    if day_index not in self.outage_schedule: self.outage_schedule[day_index] = set()
    if item.checkState() == Qt.Checked: self.outage_schedule[day_index].add(hour)
    else: self.outage_schedule[day_index].discard(hour)

def clear_all(self):
    self.table.blockSignals(True)
    for d in range(28):
        r = self.get_table_row(d)
        for h in range(24):
            it = self.table.item(r, h)
            if it: it.setCheckState(Qt.Unchecked)
    self.table.blockSignals(False)
    self.outage_schedule = {}

def apply_template(self, name):
    self.clear_all()
    self.table.blockSignals(True)

    if name == "Ранок та Вечір (08:00–11:00, 18:00–22:00)":
        for d in range(28):
            r = self.get_table_row(d)
            for h in range(8, 11): self.table.item(r, h).setCheckState(Qt.Checked)
            for h in range(18, 22): self.table.item(r, h).setCheckState(Qt.Checked)

    elif name == "День через день":
        for d in range(1, 28, 2):
            r = self.get_table_row(d)
```

```
            for h in range(24): self.table.item(r, h).setCheckState(Qt.Checked)

        elif name == "4 через 4":
            for d in range(28):
                r = self.get_table_row(d)
                for h in [4,5,6,7, 12,13,14,15, 20,21,22,23]:
                    self.table.item(r, h).setCheckState(Qt.Checked)

        elif name == "2 через 4":
            for d in range(28):
                r = self.get_table_row(d)
                for start_h in range(2, 24, 6):
                    for k in range(4):
                        if start_h+k < 24: self.table.item(r,
start_h+k).setCheckState(Qt.Checked)

        elif name == "6 через 6":
            for d in range(28):
                r = self.get_table_row(d)
                for h in range(6, 12): self.table.item(r, h).setCheckState(Qt.Checked)
                for h in range(18, 24): self.table.item(r, h).setCheckState(Qt.Checked)

        elif name == "Ніч без мережі (23:00–07:00)":
            for d in range(28):
                r = self.get_table_row(d)
                for h in range(24):
                    is_night_outage = (h >= 23 or h < 7)
                    if d == 0 and h < 7:
```

```
                continue
            if is_night_outage:
                self.table.item(r, h).setCheckState(Qt.Checked)

    self.table.blockSignals(False)
    self.outage_schedule = {}
    for d in range(28):
        r = self.get_table_row(d)
        for h in range(24):
            if self.table.item(r, h).checkState() == Qt.Checked:
                if d not in self.outage_schedule: self.outage_schedule[d] = set()
                self.outage_schedule[d].add(h)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.normal_font = QFont()
        self.normal_font.setPointSize(10)
        self.setWindowTitle("Програмна модель гібридної системи з резервуванням
вхідних каналів")
        self.resize(1400, 900)
        self.outage_calendar = SimpleOutageCalendar()

        tabs = QTabWidget()
        tabs.setFont(self.normal_font)
        tabs.addTab(self.build_tab_simulation(), "Моделювання")
        tabs.addTab(self.build_tab_outages(), "Графіки відключень")
        self.setCentralWidget(tabs)
```

```
def create_spinbox(self, val, min_v, max_v):
    sb = QSpinBox()
    sb.setFont(self.normal_font)
    sb.setRange(min_v, max_v)
    sb.setValue(val)
    sb.setMinimumHeight(35)
    sb.setButtonSymbols(QAbstractSpinBox.NoButtons)
    return sb

def create_dspinbox(self, val, min_v, max_v):
    sb = QDoubleSpinBox()
    sb.setFont(self.normal_font)
    sb.setDecimals(2)
    sb.setRange(min_v, max_v)
    sb.setValue(val)
    sb.setMinimumHeight(35)
    sb.setButtonSymbols(QAbstractSpinBox.NoButtons)
    return sb

def build_tab_outages(self):
    w = QWidget()
    l = QVBoxLayout()
    l.addWidget(self.outage_calendar)
    w.setLayout(l)
    return w

def build_tab_simulation(self):
```

```
widget = QWidget()
main_layout = QHBoxLayout()
left_layout = QVBoxLayout()

grp_time = QGroupBox("Час симуляції та умови")
tl = QFormLayout()
self.in_days = self.create_spinbox(7, 0, 365)
self.in_hours = self.create_spinbox(0, 0, 23)
self.in_weather = QComboBox()
self.in_weather.addItems(["Сонячно", "Хмарно"])

self.in_load_kw = self.create_dspinbox(0.5, 0.0, 50.0)

tl.addRow("Днів:", self.in_days)
tl.addRow("Годин:", self.in_hours)
tl.addRow("Погода:", self.in_weather)
tl.addRow("Навант. (кВт):", self.in_load_kw)
grp_time.setLayout(tl)

grp_batt = QGroupBox("Акумулятори (Li-ion/LiFePO4)")
bl = QFormLayout()
self.in_batt_ah = self.create_spinbox(200, 1, 10000)
self.in_batt_v = self.create_dspinbox(48.0, 12.0, 400.0)
self.in_soc_init = self.create_spinbox(50, 0, 100)
bl.addRow("Ємність (А·год):", self.in_batt_ah)
bl.addRow("Напруга (В):", self.in_batt_v)
bl.addRow("Старт SOC (%):", self.in_soc_init)
grp_batt.setLayout(bl)
```

```
        grp_pv = QGroupBox("СЕС та Інвертор")
        pl = QFormLayout()
        self.in_pv_kw = self.create_dspinbox(3.0, 0.0, 50.0)
        self.in_eta = self.create_spinbox(95, 50, 100)

        pl.addRow("Потужність СЕС (кВт):", self.in_pv_kw)
        pl.addRow("ККД інвертора (%):", self.in_eta)
        grp_pv.setLayout(pl)

        left_layout.addWidget(grp_time)
        left_layout.addWidget(grp_batt)
        left_layout.addWidget(grp_pv)

        self.btn_run = QPushButton("Симуляція")
        self.btn_run.setFont(self.normal_font)
        self.btn_run.setMinimumHeight(40)
        self.btn_run.clicked.connect(self.on_run_clicked)
        left_layout.addWidget(self.btn_run)

        self.btn_graphs = QPushButton("Генерація графіків")
        self.btn_graphs.setFont(self.normal_font)
        self.btn_graphs.setMinimumHeight(40)
        self.btn_graphs.setStyleSheet("background-color: #d4edda; border: 1px solid
#c3e6cb;")
        self.btn_graphs.clicked.connect(self.generate_all_graphs)
        left_layout.addWidget(self.btn_graphs)
```

```
    left_layout.addStretch()

    self.table = QTableWidget(0, 3)
    self.table.setHorizontalHeaderLabels(["Показник", "Базовий", "Адаптивний"])
    self.table.setColumnWidth(0, 250)
    header = self.table.horizontalHeader()
    header.setStretchLastSection(True)

    main_layout.addLayout(left_layout, 1)
    main_layout.addWidget(self.table, 2)
    widget.setLayout(main_layout)
    return widget

def get_params(self):
    sim_time = self.in_days.value() * 24 + self.in_hours.value()
    if sim_time <= 0: return None
    return dict(
        sim_time_h=sim_time,
        batt_capacity_ah=self.in_batt_ah.value(),
        batt_voltage_v=self.in_batt_v.value(),
        pv_power_kw=self.in_pv_kw.value(),
        load_kw=self.in_load_kw.value(),
        weather=self.in_weather.currentText(),
        outage_schedule=self.outage_calendar.outage_schedule,
        soc_init=self.in_soc_init.value() / 100.0,
        eta_inv=self.in_eta.value() / 100.0,
    )
```

```
def on_run_clicked(self):
    p = self.get_params()
    if not p: return
    try:
        r_base, _, _ = simulate(**p, algo="base")
        r_improved, _, _ = simulate(**p, algo="improved")

        auto_base = r_base.get("hours_autonomy", 0)
        auto_improved = r_improved.get("hours_autonomy", 0)

        if auto_base > 0:
            pct_increase = ((auto_improved - auto_base) / auto_base) * 100.0
        elif auto_improved > 0:
            pct_increase = 100.0
        else:
            pct_increase = 0.0

        keys = [
            ("Час без живлення (год)", "hours_no_power"),
            ("Автономність (год)", "hours_autonomy"),
        ]

        self.table.setRowCount(len(keys) + 1)

        for i, (name, key) in enumerate(keys):
            self.table.setItem(i, 0, QTableWidgetItem(name))
            v1 = r_base.get(key, 0)
            v2 = r_improved.get(key, 0)
```

```
            self.table.setItem(i, 1, QTableWidgetItem(f"{v1:.1f}"))
            self.table.setItem(i, 2, QTableWidgetItem(f"{v2:.1f}"))

        row_idx = len(keys)
        self.table.setItem(row_idx, 0, QTableWidgetItem("Приріст автономності
(%)"))
        self.table.setItem(row_idx, 1, QTableWidgetItem("-"))

        item_res = QTableWidgetItem(f"+{pct_increase:.1f}%")
        if pct_increase > 0:
            item_res.setForeground(QColor("green"))
            item_res.setFont(QFont("Arial", 10, QFont.Bold))

        self.table.setItem(row_idx, 2, item_res)

    except Exception as e:
        QMessageBox.critical(self, "Помилка", str(e))

def generate_all_graphs(self):
    if not MATPLOTLIB_AVAILABLE:
        QMessageBox.warning(self, "Помилка", "Не встановлено
matplotlib.\nЗапустіть: pip install matplotlib numpy")
        return

    demo_params = self.get_params()
    if not demo_params: return
```

```
    sim_h = demo_params['sim_time_h']
    days_total = math.ceil(sim_h / 24)
    real_schedule = demo_params['outage_schedule']

    output_dir = "Results"
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    try:
        res_base, soc_base, hist_base = simulate(**demo_params, algo="base")
        res_improved, soc_improved, hist_improved = simulate(**demo_params,
algo="improved")

        start_val = demo_params['soc_init'] * 100
        plot_soc_base = [start_val] + soc_base
        plot_soc_improved = [start_val] + soc_improved

        plt.style.use('seaborn-v0_8-whitegrid')
        plt.rcParams['font.family'] = 'DejaVu Sans'

        fig, ax1 = plt.subplots(figsize=(12, 6))
        t = np.arange(len(plot_soc_base))

        ax1.plot(t, plot_soc_improved, color='#27ae60', linestyle='-', linewidth=2.5,
label='Адаптивний')
        ax1.plot(t, plot_soc_base, color='#e74c3c', linestyle='--', linewidth=2,
label='Базовий')
```

```
        added_label = False
        for h in range(len(soc_base)):
            d = (h // 24) % 28
            hr = h % 24
            if d in real_schedule and hr in real_schedule[d]:
                lbl = 'Відключення мережі' if not added_label else None
                ax1.axvspan(h, h+1, color='gray', alpha=0.15, label=lbl)
                added_label = True

        ax1.set_ylabel('Рівень заряду (%)', fontsize=12)
        ax1.set_xlim(0, sim_h)
        ax1.set_ylim(0, 105)

        ticks = np.arange(0, sim_h + 1, 24)
        ax1.set_xticks(ticks)
        ax1.set_xticklabels([f"Д{i+1}" for i in range(len(ticks))])
        ax1.set_xlabel('Дні симуляції', fontsize=12)

        ax1.xaxis.set_minor_locator(MultipleLocator(1))
        ax1.grid(which='minor', axis='x', linestyle=':', linewidth=0.5, color='gray',
alpha=0.3)

        ax2 = ax1.twiny()
        ax2.set_xlim(ax1.get_xlim())
        major_ticks_top = np.arange(0, sim_h + 1, 6)
        ax2.set_xticks(major_ticks_top)
        ax2.set_xticklabels([f"{h%24:02d}:00" for h in major_ticks_top], rotation=90,
fontsize=8)
```

```
        ax2.grid(False)

        ax1.grid(True, linestyle='--', alpha=0.7)
        ax1.legend(loc='upper right', frameon=True)
        plt.tight_layout()
        plt.savefig(f"{output_dir}/1_SOC_Dynamics.png", dpi=150,
bbox_inches='tight')
        plt.close()

        plt.figure(figsize=(6, 6))
        bars = ['Базовий', 'Адаптивний']
        vals_np = [res_base['hours_no_power'], res_improved['hours_no_power']]
        plt.bar(bars, vals_np, color=['#e74c3c', '#27ae60'])
        plt.ylabel('Години')
        plt.tight_layout()
        plt.savefig(f"{output_dir}/2a_Hours_No_Power.png", dpi=150,
bbox_inches='tight')
        plt.close()

        plt.figure(figsize=(6, 6))
        vals_au = [res_base['hours_autonomy'], res_improved['hours_autonomy']]
        plt.bar(bars, vals_au, color=['#e74c3c', '#27ae60'])
        plt.ylabel('Години')
        plt.tight_layout()
        plt.savefig(f"{output_dir}/2b_Hours_Autonomy.png", dpi=150,
bbox_inches='tight')
        plt.close()
```

```
        fig_pie, (ax_p1, ax_p2) = plt.subplots(1, 2, figsize=(12, 6))

        labels = ['Мережа', 'Сонце', 'АКБ']
        colors = ['#95a5a6', '#f1c40f', '#2ecc71']

        d_base = [res_base['grid_energy'], res_base['pv_to_load_ac'],
res_base['batt_to_load_ac']]
        if sum(d_base) < 0.1: d_base = [1,0,0]

        ax_p1.pie(d_base, labels=labels, autopct='%1.1f%%', colors=colors,
startangle=90)
        ax_p1.set_xlabel("Базовий")

        d_imp = [res_improved['grid_energy'], res_improved['pv_to_load_ac'],
res_improved['batt_to_load_ac']]
        if sum(d_imp) < 0.1: d_imp = [1,0,0]

        ax_p2.pie(d_imp, labels=labels, autopct='%1.1f%%', colors=colors,
startangle=90)
        ax_p2.set_xlabel("Адаптивний")

        plt.tight_layout()
        plt.savefig(f"{output_dir}/3_Energy_Balance.png", dpi=150,
bbox_inches='tight')
        plt.close()

        QMessageBox.information(self, "Успіх", f"Графіки збережено в
папку:\n{os.path.abspath(output_dir)}")
```

```
            if sys.platform == 'win32': os.startfile(os.path.abspath(output_dir))
            elif sys.platform == 'darwin': os.system(f'open "{output_dir}"')
            else: os.system(f'xdg-open "{output_dir}"')

        except Exception as e:
            QMessageBox.critical(self, "Помилка генерації", str(e))

def main():
    app = QApplication(sys.argv)
    app.setStyle('Fusion')
    w = MainWindow()
    w.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```