

Міністерство освіти і науки України  
НТУУ «КПІ ім. Ігоря Сікорського»  
Навчально-науковий інститут атомної та теплової енергетики  
Кафедра цифрових технологій в енергетиці

Лабораторна робота №2  
з дисципліни  
«Моделювання систем в енергетиці»  
Тема «Розробка імітаційної моделі діючої сонячної  
електростанції»  
Варіант №18

Студента 4-го курсу НН ІАТЕ гр. ТР-12  
Ковальова Олександра  
Перевірила: ст. вик., Висоцька Олена Іванівна

КИЇВ 2024

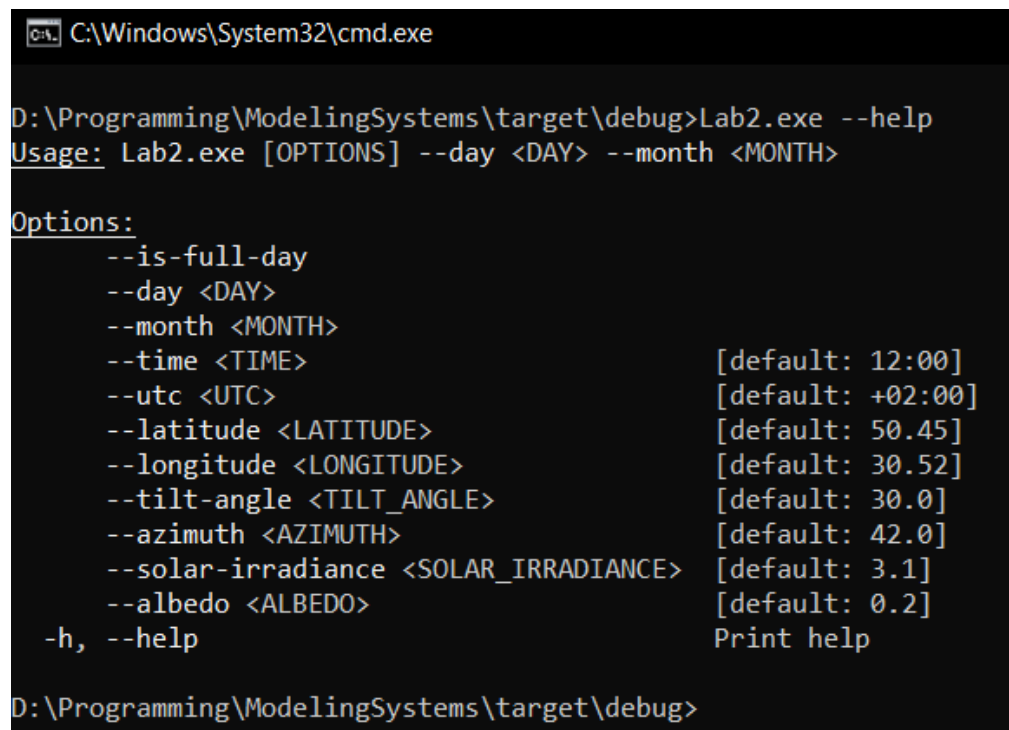
**Мета роботи.** Ознайомлення з принципами роботи сонячної електростанції та методами імітаційного моделювання. Розробка та дослідження імітаційної моделі функціонування сонячної електростанції для оцінки її продуктивності в залежності від різних параметрів, таких як інтенсивність сонячного випромінювання, кут нахилу панелей, температурних умов та інших факторів.

**Завдання:** Створення математичного апарату для визначення сумарної сонячної радіації як складової частини моделі сонячної електростанції.

Результатом роботи є визначення сумарної сонячної радіації в кожен момент часу з подальшим використанням програми в моделі прогнозування експлуатаційних характеристик сонячної електростанції.

### Хід роботи

Програма є консольним застосунком, написаним використовуючи мову програмування Rust. Інтерфейс не є інтерактивним, взаємодія відбувається за допомогою введення ключів-аргументів:



```
C:\Windows\System32\cmd.exe

D:\Programming\ModelingSystems\target\debug>Lab2.exe --help
Usage: Lab2.exe [OPTIONS] --day <DAY> --month <MONTH>

Options:
    --is-full-day
    --day <DAY>
    --month <MONTH>
    --time <TIME> [default: 12:00]
    --utc <UTC> [default: +02:00]
    --latitude <LATITUDE> [default: 50.45]
    --longitude <LONGITUDE> [default: 30.52]
    --tilt-angle <TILT_ANGLE> [default: 30.0]
    --azimuth <AZIMUTH> [default: 42.0]
    --solar-irradiance <SOLAR_IRRADIANCE> [default: 3.1]
    --albedo <ALBEDO> [default: 0.2]
    -h, --help Print help

D:\Programming\ModelingSystems\target\debug>
```

Обов'язковими аргументами на вхід є лише день та місяць. Інші змінні можна не вводити, стандартні значення наведені на скріншоті, вони відповідають типовим відносно місцезоположення міста Києва.

### Лістинг:

*main.rs* – Точка входу в програму.

```
use crate::args::Args;
use crate::context::Context;
use crate::math::angle::Angle;
use crate::model::summary_radiation;
use clap::Parser;
```

```

pub mod math {
    pub mod angle;
}

pub mod args;
pub mod context;
pub mod errors;
pub mod model;

fn main() {
    let args = Args::parse();

    let mut context = Context {
        is_full_day: args.is_full_day,
        latitude: Angle::from_degree(args.latitude.replace("'",
""").parse().unwrap()),
        longitude: Angle::from_degree(args.longitude.replace("'",
""").parse().unwrap()),
        tilt_angle: Angle::from_degree(args.tilt_angle.replace("'",
""").parse().unwrap()),
        azimuth: Angle::from_degree(args.azimuth.replace("'",
""").parse().unwrap()),
        solar_irradiance: args.solar_irradiance.replace("'",
""").parse().unwrap(),
        ..Default::default()
    };
    if context.is_full_day {
        let result = context.parse_day_month(&args.day, &args.month);
        if let Err(err) = result {
            panic!("{}", err.to_string())
        }
    } else {
        let result = context.parse_datetime(&args.day, &args.month, &args.time,
&args.utc);
        if let Err(err) = result {
            panic!("{}", err.to_string())
        }
    }

    let summary_solar_radiation = summary_radiation(context);
    println!("{}", summary_solar_radiation)
}

```

***errors/parse.rs*** – Модуль для опису «користувацьких» помилок.

```

use thiserror::Error;

#[derive(Error, Debug)]
pub enum ParseError {
    #[error("Error occurred while parsing date. Check formats in --help.")]
    DateParse(String),
}

```

***math/angle.rs*** – Структура «Кут», створена для полегшення обрахунків.

```

#[derive(Default)]
pub struct Angle {
    degree: f64,
    radian: f64,
}

impl Angle {

```

```

pub fn from_degree(degree: f64) -> Self {
    Self {
        degree,
        radian: degree * std::f64::consts::PI / 180.0,
    }
}

pub fn from_radian(radian: f64) -> Self {
    Self {
        degree: radian * 180.0 / std::f64::consts::PI,
        radian,
    }
}

pub fn degree(&self) -> f64 {
    self.degree
}

pub fn radian(&self) -> f64 {
    self.radian
}
}

```

## *args.rs* – Структура для зчитування аргументів.

```

use clap::Parser;

#[derive(Parser)]
pub struct Args {
    #[arg(long)]
    pub is_full_day: bool,

    #[arg(long)]
    pub day: String,

    #[arg(long)]
    pub month: String,

    #[arg(long, default_value = "12:00")]
    pub time: String,

    #[arg(long, default_value = "+02:00")]
    pub utc: String,

    #[arg(long, default_value = "50.45")]
    pub latitude: String,

    #[arg(long, default_value = "30.52")]
    pub longitude: String,

    #[arg(long, default_value = "30.0")]
    pub tilt_angle: String,

    #[arg(long, default_value = "42.0")]
    pub azimuth: String,

    #[arg(long, default_value = "3.1")]
    pub solar_irradiance: String,

    #[arg(long, default_value = "0.2")]
    pub albedo: String,
}

```

## *context.rs* – Контекст програми.

```
use crate::errors::parse::ParseError;
use crate::math::angle::Angle;
use chrono::{DateTime, FixedOffset};

pub const DEFAULT_YEAR: u16 = 2024;

#[derive(Default)]
pub struct Context {
    pub is_full_day: bool,
    pub albedo: f64,
    pub datetime: DateTime<FixedOffset>,
    pub latitude: Angle,
    pub longitude: Angle,
    pub tilt_angle: Angle,
    pub azimuth: Angle,
    pub solar_irradiance: f64,
}

impl Context {
    pub fn parse_day_month(&mut self, day: &str, month: &str) -> Result<(),
ParseError> {
        let datetime_str = format!("{DEFAULT_YEAR}-{month}-{
day}T00:00:00+02:00");

        self.datetime = datetime_str
            .parse()
            .map_err(|_| ParseError::DateParse(String::default()))?;

        Ok(())
    }

    pub fn parse_datetime(
        &mut self, day: &str, month: &str, time: &str, utc: &str,
    ) -> Result<(), ParseError> {
        let datetime_str = format!("{DEFAULT_YEAR}-{month}-{
day}T{time}:00{utc}");

        self.datetime = datetime_str
            .parse()
            .map_err(|_| ParseError::DateParse(String::default()))?;

        Ok(())
    }
}
```

## *model.rs* – Математична модель.

```
use crate::context::Context;
use crate::math::angle::Angle;
use chrono::{Datelike, Timelike};

pub const SOLAR_CONST: f64 = 1367.0;

pub fn summary_radiation(context: Context) -> f64 {
    let a = f64::sin(context.latitude.radian()) *
f64::cos(context.tilt_angle.radian());
    let b = f64::cos(context.latitude.radian())
        * f64::sin(context.tilt_angle.radian())
        * f64::cos(context.azimuth.radian());
```

```

        let c = f64::sin(context.tilt_angle.radian()) *
f64::sin(context.azimuth.radian());
        let d = f64::cos(context.latitude.radian()) *
f64::cos(context.tilt_angle.radian());
        let e = f64::sin(context.latitude.radian())
            * f64::sin(context.tilt_angle.radian())
            * f64::cos(context.azimuth.radian());

        let b0_coef: f64 = 360.0 / 365.0;

        let b_coef = b0_coef * (context.datetime.ordinal() as f64 - 81.0);

        let time_correction = 1.0 / 60.0
            * (9.87 * f64::sin(2.0 * b_coef) - 7.53 * f64::cos(b_coef) - 1.5 *
f64::sin(b_coef));

        let hour_angle = Angle::from_degree(
            15.0 * (context.datetime.hour() as f64
                - 12.0
                - time_correction
                - context.datetime.offset().local_minus_utc() as f64 / 60.0 / 60.0
                + context.longitude.degree()),
        );

        let solar_declination =
            Angle::from_degree(23.45 * f64::sin(b0_coef *
(context.datetime.ordinal() as f64 + 284.0)));

        let incident_angle = (a - b) * f64::sin(solar_declination.radian())
            + (c * f64::sin(hour_angle.radian()) + (d * e) *
f64::cos(hour_angle.radian()))
            * f64::cos(solar_declination.radian());
        let incident_angle = Angle::from_radian(f64::acos(incident_angle));

        let zenithal = Angle::from_radian(f64::acos(
            f64::sin(context.latitude.radian()) *
f64::sin(solar_declination.radian())
            + f64::cos(context.latitude.radian())
            * f64::cos(solar_declination.radian())
            * f64::cos(hour_angle.radian()),
        ));

        let post_atmospheric_rad = SOLAR_CONST
            * (1.0 + 0.033 * f64::cos(b0_coef * context.datetime.ordinal() as f64))
            * f64::cos(zenithal.radian());

        let i_total_horizontal = context.solar_irradiance;
        let kt_index = i_total_horizontal / post_atmospheric_rad;

        let i_diffuse_horizontal = i_total_horizontal / (1.0 + f64::exp(-5.0 + 8.6 *
kt_index));
        let i_reflected_horizontal = context.albedo * i_total_horizontal;
        let i_direct_horizontal = i_total_horizontal - i_diffuse_horizontal -
i_reflected_horizontal;

        i_direct_horizontal * (f64::cos(incident_angle.radian()) /
f64::cos(zenithal.radian()))
            + i_diffuse_horizontal * (1.0 + f64::cos(context.tilt_angle.radian())) /
2.0
            + i_reflected_horizontal * (1.0 - f64::cos(context.tilt_angle.radian()))
/ 2.0
    }

```

Для отримання статистики використовувалась мова програмування Python та бібліотека *matplotlib*. Підхід з викликом консольних застосунків з Python є частовживаним, саме через це інтерфейс не є інтерактивним.

### Лістинг:

*Months.py* – Статистика з 0:00 по 24:00 в перший день квітня, грудня, вересня та липня.

```
import subprocess
import matplotlib.pyplot as plt

time_values = []
for hour in range(24):
    time_str = f"{hour:02}:00"
    time_values.append(time_str)

all_results = []
day = 1
months = [12, 4, 7, 9]
solar_irradiance = [0.86, 3.96, 5.22, 3.12]

for i in range(len(months)):
    month = months[i]
    irradiance = solar_irradiance[i]
    results = []

    for time in time_values:
        command = ["Lab2.exe", "--day", f"{day:02}", "--month", f"{month:02}",
"--time", time,
                    "--solar-irradiance", f"{irradiance}"]
        result = subprocess.run(command, capture_output=True, text=True)
        output_value = float(result.stdout.strip())
        results.append(output_value)

    all_results.append(results)

plt.figure(figsize=(10, 6))
colors = ['b', 'g', 'r', 'c'] # Different colors for each month
for idx, month_results in enumerate(all_results):
    plt.plot(time_values, month_results, marker='o', color=colors[idx],
label=f'Month {months[idx]}')

# Adding titles and labels
plt.title(f'Summary Radiation for {day:02}st day of the month.')
plt.xlabel('Time (HH:MM)')
plt.ylabel('kWh / m^2')
plt.xticks(rotation=45)
plt.grid()
plt.legend(title='Months')
plt.tight_layout()

# Save the plot
plt.savefig('months.png')

# Show the plot (optional)
plt.show()
```

## *Azimuth\_Tilt.py* – Статистика відносно зміни азимуту та нахилу панелі.

```
import subprocess
import matplotlib.pyplot as plt

day = 1
month = 7
time = "12:00"
solar_irradiance = 5.22

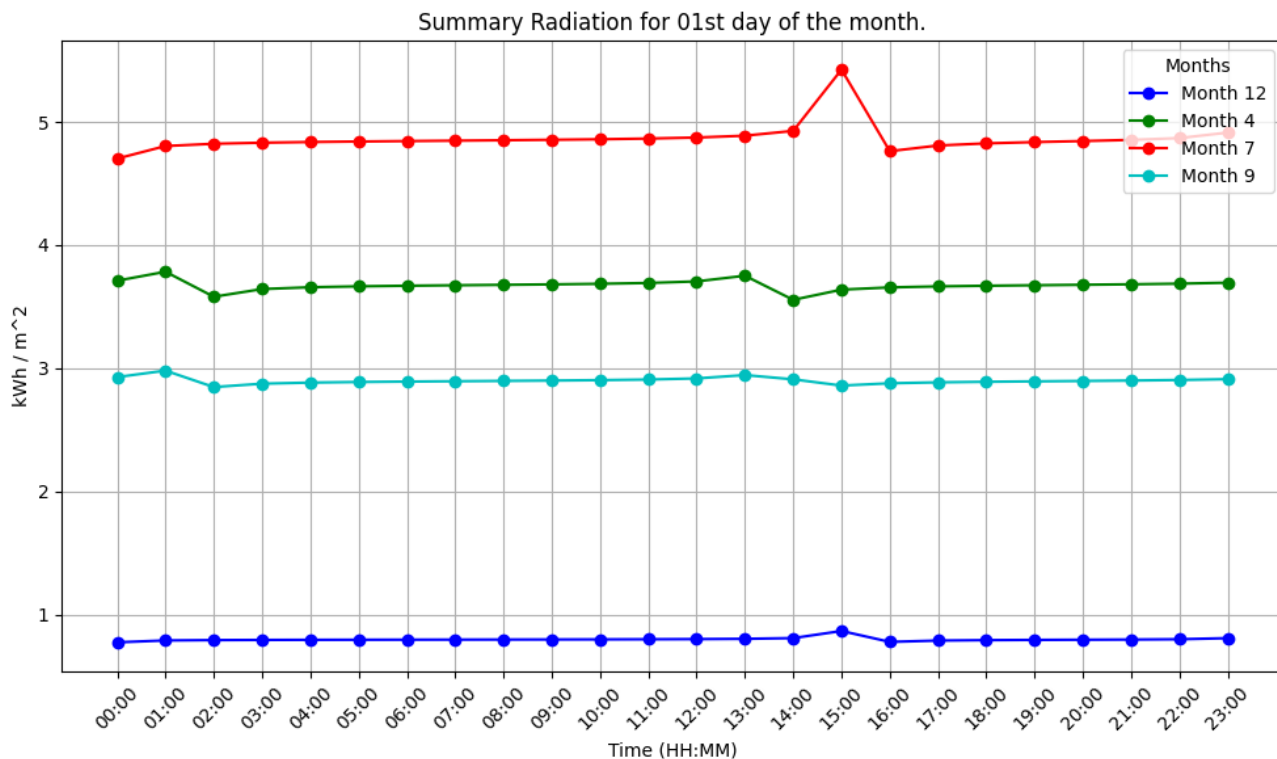
azimuths = []
for value in range(-179, 181, 10):
    string = f"{value}"
    azimuths.append(string)
results = []
for value in azimuths:
    command = ["Lab2.exe", "--day", f"{day:02}", "--month", f"{month:02}", "--
time", time,
                "--solar-irradiance", f"{solar_irradiance}", "--azimuth",
f"{value}"]
    result = subprocess.run(command, capture_output=True, text=True)
    output_value = float(result.stdout.strip())
    results.append(output_value)
plt.figure(figsize=(10, 6))
plt.plot(azimuths, results, marker='o')
plt.title(f'Radiation by changing azimuth. Date & Time: {day:02}.{month:02}
{time}.')
plt.xlabel('Azimuth')
plt.ylabel('kWh / m^2')
plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.savefig('azimuth.png')
plt.show()

tilts = []
for value in range(0, 91, 5):
    string = f"{value}"
    tilts.append(string)
results = []
for value in tilts:
    command = ["Lab2.exe", "--day", f"{day:02}", "--month", f"{month:02}", "--
time", time,
                "--solar-irradiance", f"{solar_irradiance}", "--tilt-angle",
f"{value}"]
    result = subprocess.run(command, capture_output=True, text=True)
    output_value = float(result.stdout.strip())
    results.append(output_value)
plt.figure(figsize=(10, 6))
plt.plot(tilts, results, marker='o')
plt.title(f'Radiation by changing tilt. Date & Time: {day:02}.{month:02}
{time}.')
plt.xlabel('Tilt')
plt.ylabel('kWh / m^2')
plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.savefig('tilt.png')
plt.show()
```

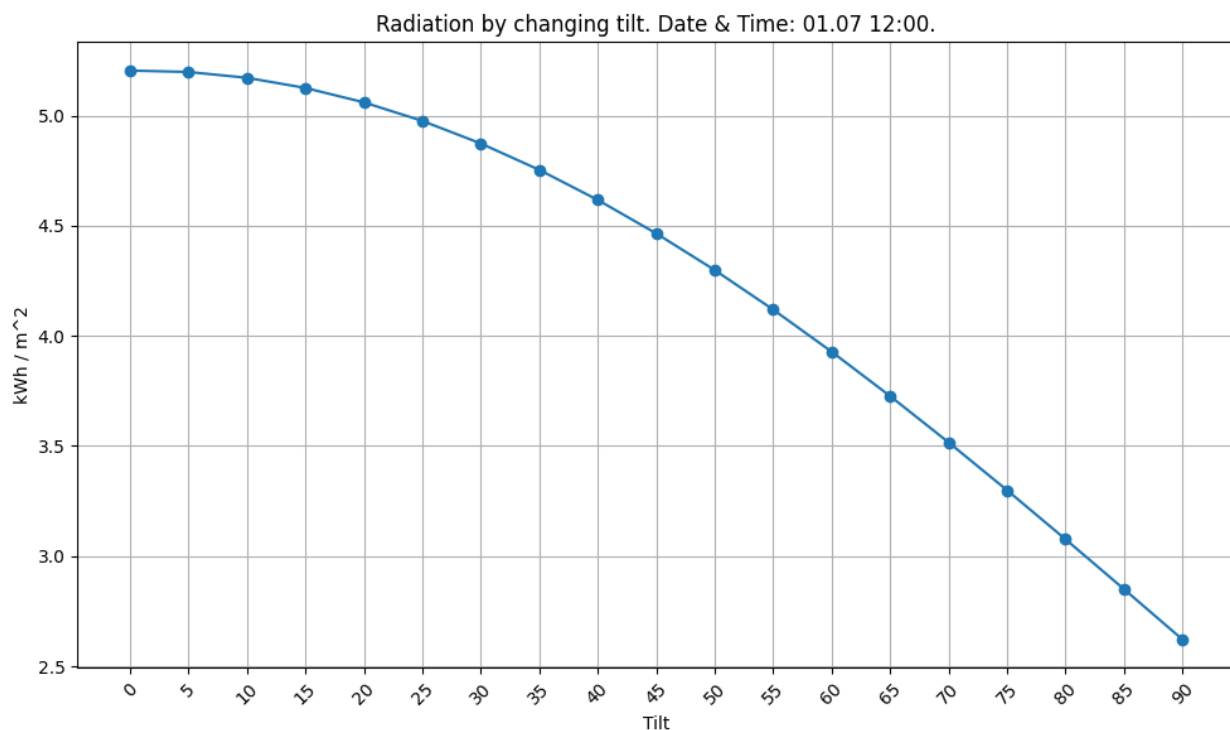


## Графіки:

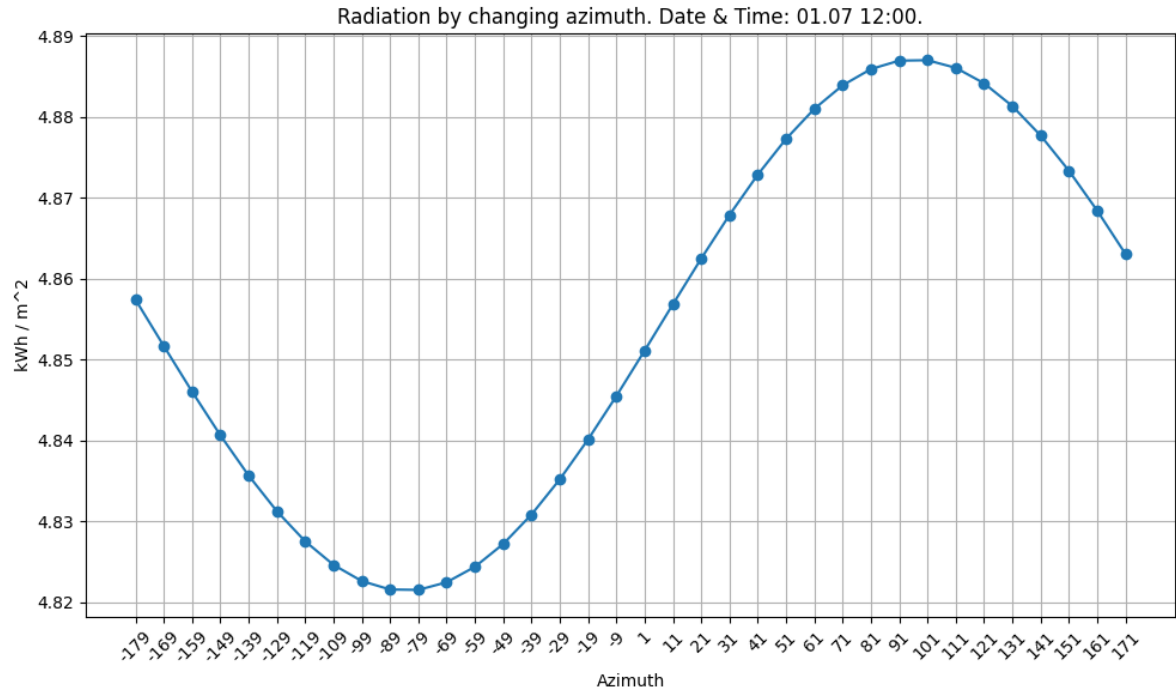
Зміни величини сонячної інсоляції протягом доби один день у грудні, квітні, липні, вересні:



Залежність сумарної радіації, падаючої на поверхню сонячної панелі при зміні кута нахилу:



Залежність сумарної радіації, падаючої на поверхню сонячної панелі при зміні напрямку розміщення панелі:



**Висновок:** У ході виконання лабораторної роботи була розроблена імітаційна модель функціонування сонячної електростанції, що дозволяє оцінювати її продуктивність залежно від таких факторів, як інтенсивність сонячного випромінювання, кут нахилу панелей, температурні умови та інші параметри. Було успішно визначено сумарну сонячну радіацію в кожен момент часу, що є важливим компонентом для прогнозування експлуатаційних характеристик сонячної електростанції. Отримані результати допоможуть оптимізувати розміщення та налаштування сонячних панелей для підвищення ефективності їх роботи.