**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №2**

**з навчальної дисципліни “Моделювання систем в енергетиці”**

**Тема:**

**Розробка імітаційної моделі діючої сонячної електростанції**

**Варіант 20**

**Виконав студент групи ТР–12**

Руденко Владислав\_\_\_\_\_\_\_\_\_

Лабораторну роботу захищено

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Київ 2024**

**Мета:**

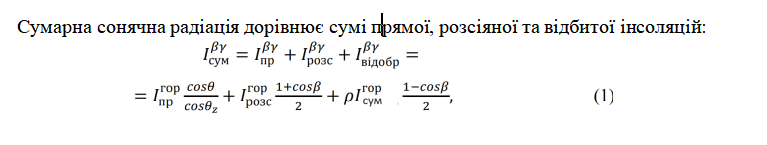
Ознайомлення з принципами роботи сонячної електростанції та методами імітаційного моделювання. Розробка та дослідження імітаційної моделі функціонування сонячної електростанції для оцінки її продуктивності в залежності від різних параметрів, таких як інтенсивність сонячного випромінювання, кут нахилу панелей, температурні умови та інші фактори**.**

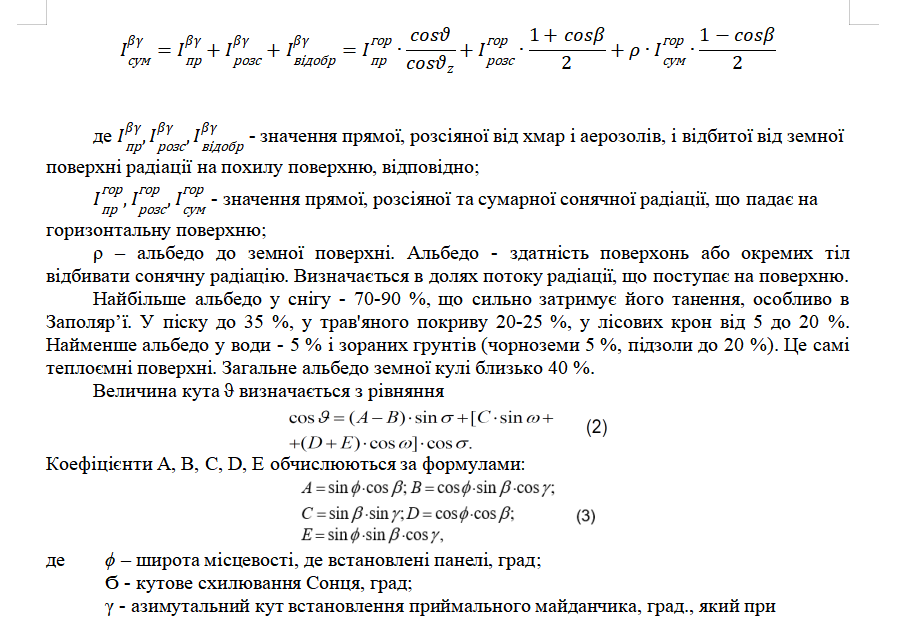
**Завдання:**

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.

**Теор.Відомості:**





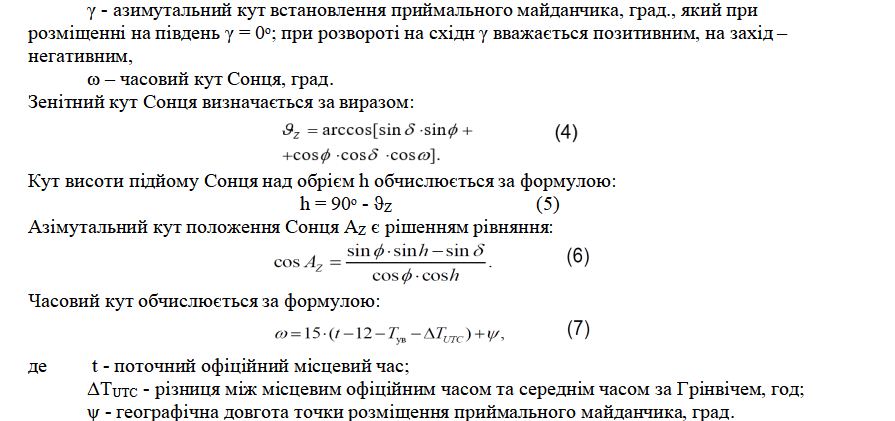




Table 1 Використані дані для прикладу

Хід Роботи

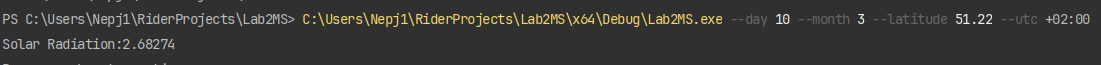


Table 2 Приклад виконання з вказаними параметрами



Table 3 Приклад виконання з параметрами за замовчуванням

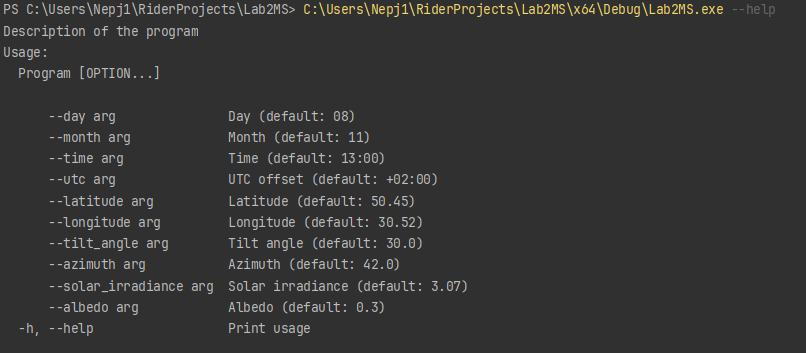


Table 4 Можливий перелік параметрів

**Лістинг:**

**Lab2MS.cpp**

#include <iostream>  
#include <math.h>  
  
#include "argsConvert.h"  
  
using namespace std;  
  
#define M\_PI 3.14159265358979323846  
#define SOLAR\_CONST 1367.0  
  
double DegreesToRadians(double degree)  
{  
 return degree \* ( M\_PI / 180);  
}  
  
double RadiansToDegrees(double degree)  
{  
 return degree \* ( 180 / M\_PI);  
}  
  
double Solution(Args args)  
{  
 double a = sin(DegreesToRadians(stod(args.latitude))) \* cos(DegreesToRadians(stod(args.tilt\_angle)));  
 double b = cos(DegreesToRadians(stod(args.latitude))) \* sin(DegreesToRadians(stod(args.tilt\_angle))) \* cos(DegreesToRadians(stod(args.azimuth)));  
 double c = sin(DegreesToRadians(stod(args.tilt\_angle))) \* sin(DegreesToRadians(stod(args.azimuth)));  
 double d = cos(DegreesToRadians(stod(args.latitude))) \* cos(DegreesToRadians(stod(args.tilt\_angle)));  
 double e = sin(DegreesToRadians(stod(args.latitude))) \* sin(DegreesToRadians(stod(args.tilt\_angle))) \* cos(DegreesToRadians(stod(args.azimuth)));  
  
 double b0\_coef = 360.0/365.0;  
  
 double b\_coedf = b0\_coef \* (args.datetime.tm\_yday - 81);  
  
 double time\_correction = ( 1.0/60 ) \* (9.87 \* sin(b\_coedf \* 2) - 7.53 \* cos(b\_coedf) - 1.5 \* sin(b\_coedf));  
  
 double hour\_angle = DegreesToRadians(15 \* ( args.datetime.tm\_hour - 12.0 - time\_correction - std::stod(args.utc)) + stod(args.longitude));  
  
 double solar\_declination = DegreesToRadians(23.45 \* std::sin(b0\_coef \* (args.datetime.tm\_yday + 284.0)));  
  
 double incident\_angle = RadiansToDegrees(acos((a-b) \* sin(DegreesToRadians(solar\_declination)) + (c \* sin(DegreesToRadians(hour\_angle))) + (d \* e) \* cos(DegreesToRadians(hour\_angle)) \* cos(solar\_declination)));  
   
 double zenithal = RadiansToDegrees(acos(  
 sin(DegreesToRadians(stod(args.latitude)))  
 \* sin(DegreesToRadians(solar\_declination))  
 + cos(DegreesToRadians(stod(args.latitude)))  
 \* cos(DegreesToRadians(solar\_declination))  
 \* cos(DegreesToRadians(hour\_angle))  
 ));  
  
 double post\_atmospheric\_rad = SOLAR\_CONST \* (1+0.033 \* cos(b0\_coef + args.datetime.tm\_yday / cos(DegreesToRadians(zenithal))));  
  
 double kt\_index = stod(args.solar\_irradiance) / post\_atmospheric\_rad;  
  
 double i\_diffuse\_horizontal = stod(args.solar\_irradiance) / (1.0 + exp(-5.0 + 8.6 \* kt\_index));  
  
 double i\_reflected\_horizontal = stod(args.albedo) \* stod(args.solar\_irradiance);  
  
 double i\_direct\_horizontal = stod(args.solar\_irradiance) - i\_diffuse\_horizontal - i\_reflected\_horizontal;  
  
 return i\_direct\_horizontal \* (cos(DegreesToRadians(incident\_angle)) / cos(DegreesToRadians(zenithal)))  
 + i\_diffuse\_horizontal \* (1.0 + cos(DegreesToRadians(stod(args.tilt\_angle)))) / 2.0  
 + i\_reflected\_horizontal \* (1.0 - cos(DegreesToRadians(stod(args.tilt\_angle)))) / 2.0 ;  
   
}  
  
int main(int argc, char\* argv[])  
{  
 argsConvert converter;  
 Args args = converter.convert(argc,argv);  
 cout << "Solar Radiation:" << Solution(args) << endl;  
 system("pause");  
 return 0;  
}

**argsConvert.cpp**

#include "argsConvert.h"  
  
#include <iostream>  
  
#include "U:\C++Lib\cxxopts-master\include\cxxopts.hpp"  
  
  
Args argsConvert::convert(int argc, char\* argv[])  
{  
 cxxopts::Options options("Program", "Description of the program");  
  
 options.add\_options()  
 ("day", "Day", cxxopts::value<std::string>()->default\_value("08"))  
 ("month", "Month", cxxopts::value<std::string>()->default\_value("11"))  
 ("time", "Time", cxxopts::value<std::string>()->default\_value("13:00"))  
 ("utc", "UTC offset", cxxopts::value<std::string>()->default\_value("+02:00"))  
 ("latitude", "Latitude", cxxopts::value<std::string>()->default\_value("50.45"))  
 ("longitude", "Longitude", cxxopts::value<std::string>()->default\_value("30.52"))  
 ("tilt\_angle", "Tilt angle", cxxopts::value<std::string>()->default\_value("30.0"))  
 ("azimuth", "Azimuth", cxxopts::value<std::string>()->default\_value("42.0"))  
 ("solar\_irradiance", "Solar irradiance", cxxopts::value<std::string>()->default\_value("3.07"))  
 ("albedo", "Albedo", cxxopts::value<std::string>()->default\_value("0.3"))  
 ("h,help", "Print usage");  
  
 auto result = options.parse(argc, argv);  
  
 if (result.count("help")) {  
 std::cout << options.help() << std::endl;  
 exit(0);  
 }  
  
 Args args;  
 args.day = result["day"].as<std::string>();  
 args.month = result["month"].as<std::string>();  
 args.time = result["time"].as<std::string>();  
 args.utc = result["utc"].as<std::string>();  
 args.latitude = result["latitude"].as<std::string>();  
 args.longitude = result["longitude"].as<std::string>();  
 args.tilt\_angle = result["tilt\_angle"].as<std::string>();  
 args.azimuth = result["azimuth"].as<std::string>();  
 args.solar\_irradiance = result["solar\_irradiance"].as<std::string>();  
 args.albedo = result["albedo"].as<std::string>();  
 args.parse\_datetime();  
  
 return args;  
}

**Struct.cpp**

#include "structs.h"  
#include <U:\C++Lib\fmt-master\include\fmt\core.h>  
#include <chrono>  
#include <iomanip>  
#include <sstream>  
  
int Args::tm\_ydayReforge(int year)  
{  
 int days\_in\_month[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
 if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {  
 days\_in\_month[2] = 29;  
 }  
 int total\_days = 0;  
 for (int m = 1; m < stoi(month); ++m) {  
 total\_days += days\_in\_month[m];  
 }  
 return total\_days + stoi(day);  
}  
  
void Args::parse\_datetime()  
{  
 const std::string DEFAULT\_YEAR = "2024";  
 std::string datetime\_str = DEFAULT\_YEAR + "-" + month + "-" + day + "T" + time + ":00" + utc;  
 std::istringstream ss(datetime\_str);  
 ss >> std::get\_time(&datetime, "%Y-%m-%dT%H:%M:%S.000%z");  
  
 datetime.tm\_yday = tm\_ydayReforge(stoi(DEFAULT\_YEAR));  
}

**Графіки:**

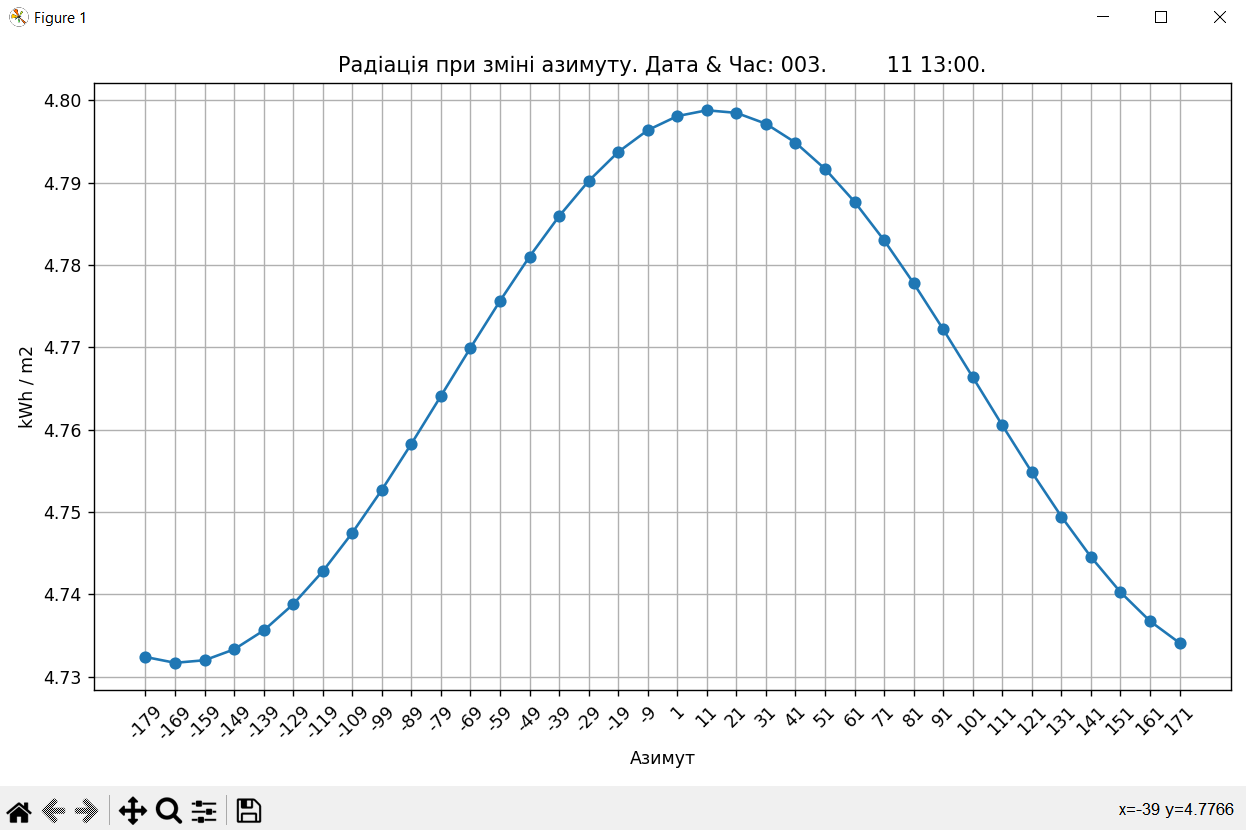


Table 5 Залежність рівня радіації від азимуту

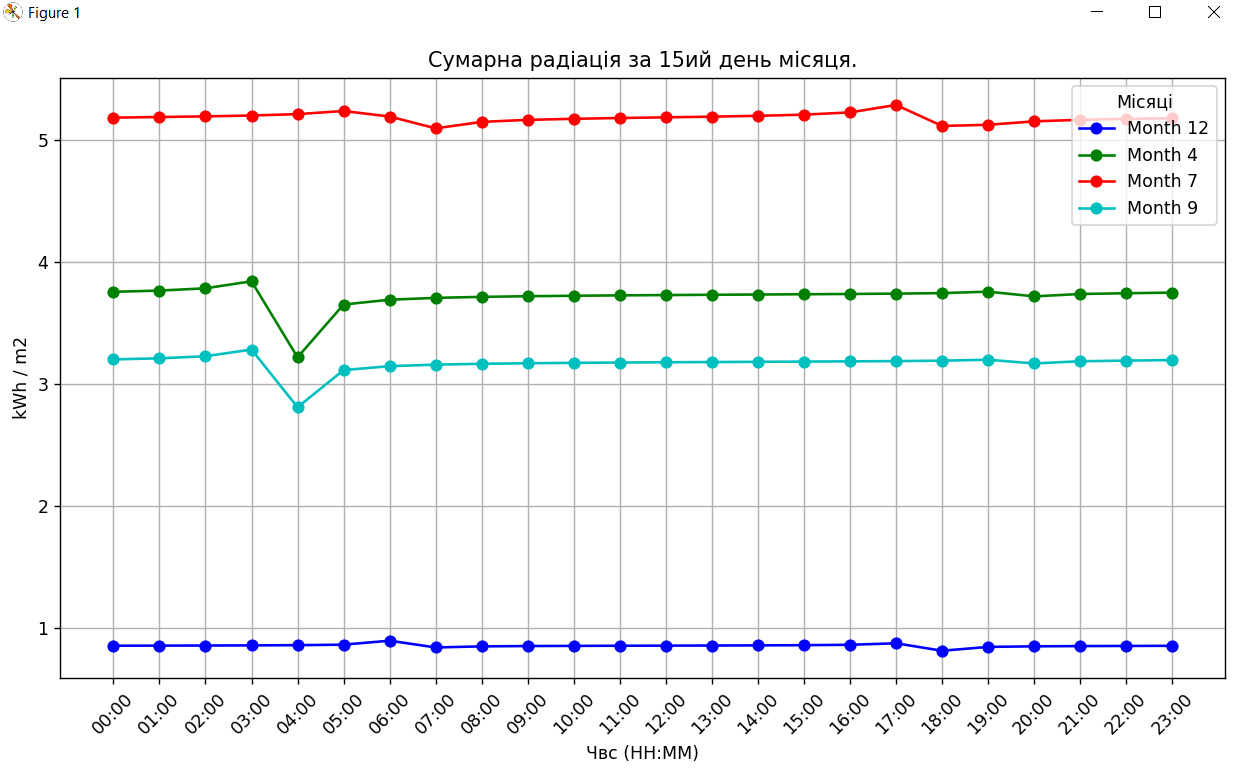


Table 6 Залежність радіації від години та місяця в місті Черкаси

**Висновок:**

В результаті виконання лабораторної роботи була створена імітаційна модель сонячної електростанції, що дозволяє визначати її продуктивність залежно від параметрів, таких як інтенсивність випромінювання та кут нахилу панелей. Отримані результати дають можливість прогнозувати ефективність роботи станції в різних умовах, що сприяє підвищенню її продуктивності.