I-V Characteristics of a Solar Panel for a Specified Irradiance and Temperature

Name: Liza Yousef S23108546 , Maha Alblwai S22107905, Hajar BakoorS21207483

Effat University

ECE103, Programming 1

Instructor : Narjis Khabbaji

05/09/24

Table of Contents

Abstract

The aim of this project was to analyze and characterize the performance of a solar panel using data obtained from real-world measurements. The project involved reading data from a CSV file, estimating missing values, performing calculations, and generating a plot to visualize the current-voltage characteristics of the solar panel. The report provides an overview of the project, the methodology used, the results obtained, and concludes with recommendations for further improvements.

Introduction

Solar energy is a vital component of the growing renewable energy sector, offering a clean and sustainable alternative to traditional fossil fuel-based power generation. As the demand for solar panels continues to rise, it becomes increasingly important to analyze and characterize their performance to optimize efficiency and ensure reliable operation. This report presents a project that focuses on the analysis and characterization of a solar panel using real-world data obtained from measurements.

The project involves the utilization of C++ programming language and GNUplot, a powerful graphing utility, to process the collected data, estimate missing values, perform calculations, and generate a plot illustrating the current-voltage characteristics of the solar panel. By examining the behavior of the solar panel under varying environmental conditions, valuable insights can be gained regarding its performance, efficiency, and potential for power generation.

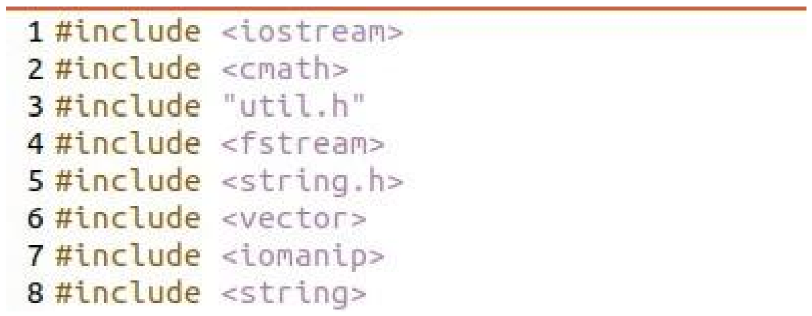
In this report, we provide a comprehensive overview of the project, detailing the methodology employed, the data processing and analysis techniques, and the results obtained. Additionally, we discuss the significance of the maximum power point and its implications for optimizing the solar panel's performance. Furthermore, we compare the estimated values with the measured data to assess the accuracy of the estimation process. Finally, we conclude with recommendations for further enhancements and improvements to refine the analysis and characterization of solar panels.

By understanding the performance characteristics of solar panels, we can pave the way for advancements in solar energy technology, enabling more efficient and sustainable energy production. The findings of this project contribute to the broader field of renewable energy research and provide valuable insights for engineers, researchers, and policymakers striving to harness the full potential of solar power.

Methodology

1. Data Collection: Gathered irradiance and temperature data from an Excel file.
2. Newton-Raphson Implementation: Coded the algorithm in C++ for calculating I-V characteristics.
3. I-V Curve Plotting: Developed a function to visualize the results graphically.
4. Testing and Validation: Ensured accuracy through rigorous testing and validation.
5. Refinement: Iteratively improved the program based on feedback.
6. Documentation: Prepared comprehensive documentation and a project report for dissemination.

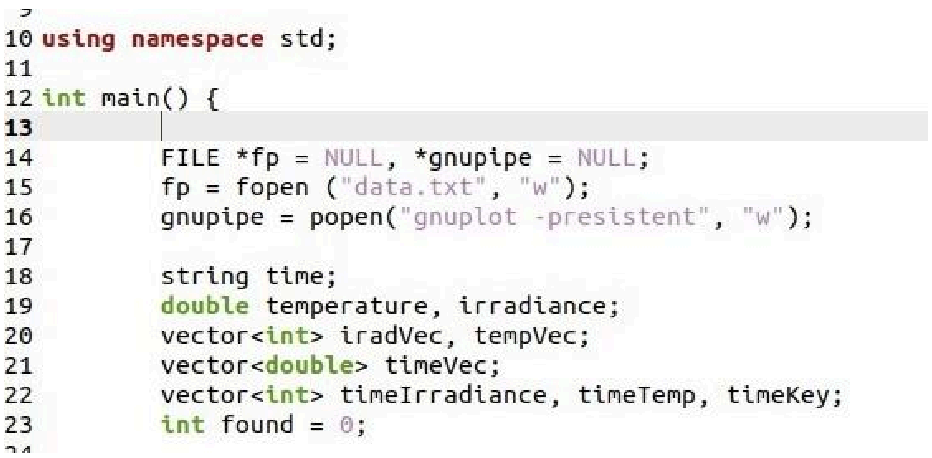
Code



```cpp
#include <iostream>
#include <cmath>
#include "util.h"
#include <fstream>
#include <string.h>
#include <vector>
#include <iomanip>
#include <string>
```

1. iostream: Provides input/output stream functionality for reading from and writing to the console.
2. cmath: Offers mathematical functions and constants for performing calculations.
3. fstream: Provides classes and functions for file input/output operations, allowing you to read from and write to files.
4. vector: Offers a dynamic array-like container that can store and manipulate collections of elements.
5. iomanip: Provides manipulators and formatting options for stream input/output, allowing you to control the precision and width of output.
6. string: Provides the standard string class for working with strings, including various string manipulation and comparison operations.

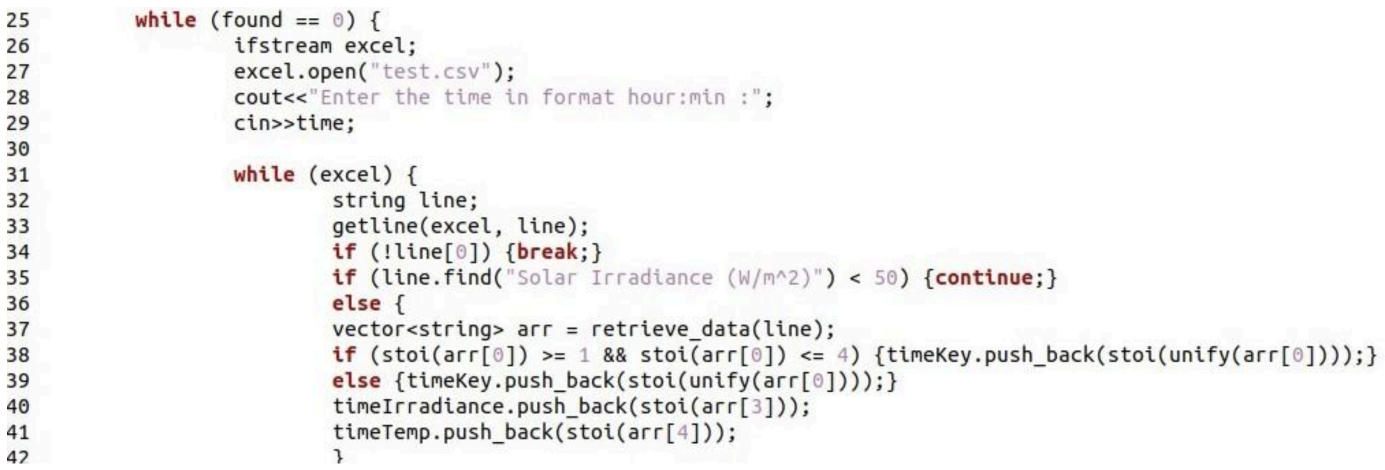
Code



```
using namespace std;

int main() {

        FILE *fp = NULL, *gnupipe = NULL;
        fp = fopen ("data.txt", "w");
        gnupipe = popen("gnuplot -presistent", "w");

        string time;
        double temperature, irradiance;
        vector<int> iradVec, tempVec;
        vector<double> timeVec;
        vector<int> timeIrradiance, timeTemp, timeKey;
        int found = 0;
```

1. It includes necessary libraries and introduces the `std` namespace for accessing standard C++ library functions and objects.
2. The `main()` function is defined as the entry point of the program.
3. File pointers `fp` and `gnupipe` are declared and initialized to `NULL`.
4. The file `"data.txt"` is opened in write mode using `fopen()`, and the file pointer is assigned to `fp`. This is used to write data to the file.
5. The command `"gnuplot -presistent"` is executed using `popen()`, creating a pipe in write mode, and the pipe pointer is assigned to `gnupipe`. This is typically used to communicate with the GNUplot program for generating plots.
6. Several variables are declared, including `time` (a string), `temperature` and `irradiance` (doubles), and various vectors for storing integer and double values.
7. The variable `found` is initialized to 0.

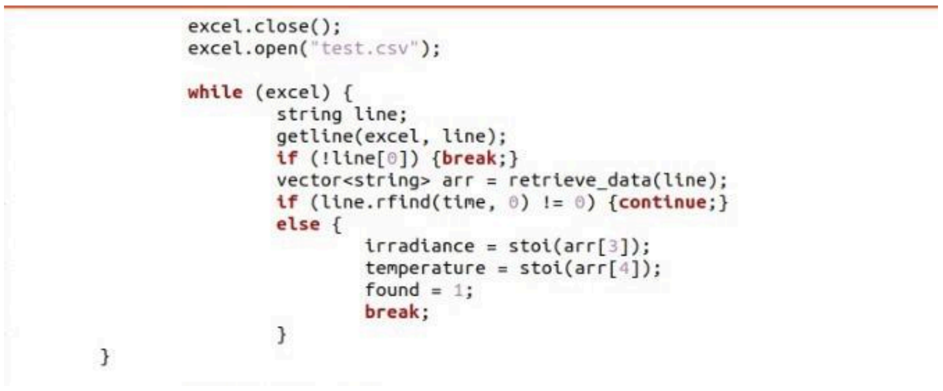
Code



```
    while (found == 0) {
        ifstream excel;
        excel.open("test.csv");
        cout<<"Enter the time in format hour:min :";
        cin>>time;

        while (excel) {
            string line;
            getline(excel, line);
            if (!line[0]) {break;}
            if (line.find("Solar Irradiance (W/m^2)") < 50) {continue;}
            else {
            vector<string> arr = retrieve_data(line);
            if (stoi(arr[0]) >= 1 && stoi(arr[0]) <= 4) {timeKey.push_back(stoi(unify(arr[0])));}
            else {timeKey.push_back(stoi(unify(arr[0])));}
            timeIrradiance.push_back(stoi(arr[3]));
            timeTemp.push_back(stoi(arr[4]));
            }
```

1. An `ifstream` object named `excel` is created to read from a file named "test.csv".
2. The user is prompted to enter the time in the format "hour:min" using `cout` and `cin` statements.
3. Within the nested `while` loop, each line of the file is read using `getline(excel, line)` and stored in the string variable `line`.
4. If the first character of `line` is empty (indicating an empty line), the loop breaks and execution continues after the loop.
5. If the line contains the text "Solar Irradiance (W/m^2)" within the first 50 characters, the loop skips to the next iteration using the `continue` statement.
6. Otherwise, the line is processed further. The `retrieve_data` function is called with `line` as an argument, which likely extracts relevant data from the line and returns it as a vector of strings named `arr`.
7. If the first element of `arr` (converted to an integer using `stoi`) is between 1 and 4 (inclusive), the unified form of the element (possibly obtained from the `unify` function) is added to the `timeKey` vector. Otherwise, the original element is added.
8. The fourth element of `arr` (converted to an integer using `stoi`) is added to the `timeIrradiance` vector.
9. The fifth element of `arr` (converted to an integer using `stoi`) is added to the `timeTemp` vector.

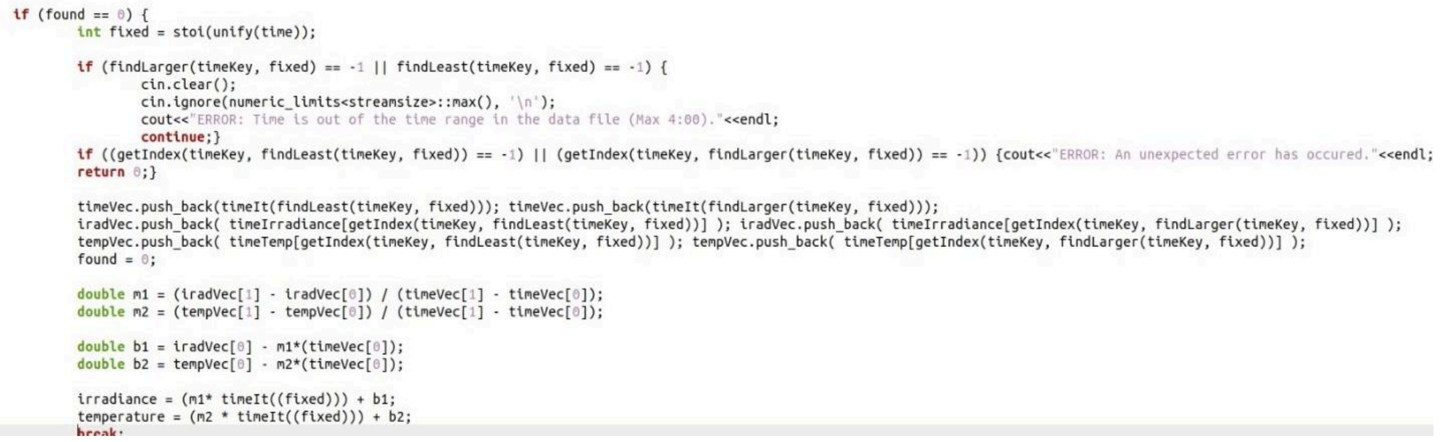
Code



```
                excel.close();
                excel.open("test.csv");

                while (excel) {
                        string line;
                        getline(excel, line);
                        if (!line[0]) {break;}
                        vector<string> arr = retrieve_data(line);
                        if (line.rfind(time, 0) != 0) {continue;}
                        else {
                                irradiance = stoi(arr[3]);
                                temperature = stoi(arr[4]);
                                found = 1;
                                break;
                        }
        }
```

1. `excel.close()` is called to close the file stream associated with the `excel` object. This ensures that any existing connection to the file "test.csv" is closed.
2. `excel.open("test.csv")` is used to reopen the file "test.csv" with the `excel` object. This reestablishes the connection to the file, allowing it to be read from the beginning again.
3. The `while` loop is executed again, using the same structure as before, to search for a specific line that matches the provided `time` string.
    - Within the loop, a new `string` variable named `line` is declared.
    - The `getline()` function is used to read the next line from the `excel` file and store it in the `line` variable.
    - If the first character of `line` is empty (indicating an empty line), the loop breaks with the `break` statement, and the execution continues after the loop.
    - The `retrieve_data()` function is called with `line` as an argument, which likely extracts relevant data from the line and returns it as a vector of strings named `arr`.
    - If the `time` string does not match the beginning of the `line` using `line.rfind(time, 0) != 0`, the loop continues to the next iteration with the `continue` statement.
    - If the `time` string does match the beginning of the `line`, the following actions are taken:
        - The fourth element of the `arr` vector (converted to an integer using `stoi`) is assigned to the `irradiance` variable.
        - The fifth element of the `arr` vector (converted to an integer using `stoi`) is assigned to the `temperature` variable.
        - The `found` variable is set to 1, indicating that the desired data has been found.
        - The loop is exited with the `break` statement.

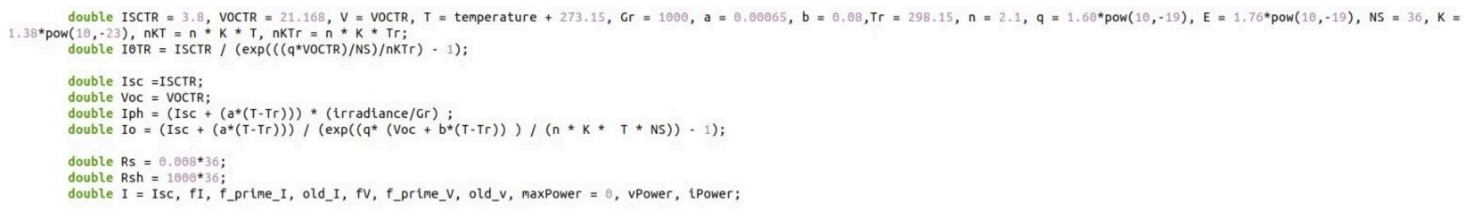
Code



```
if (found == 0) {
        int fixed = stoi(unify(time));

        if (findLarger(timeKey, fixed) == -1 || findLeast(timeKey, fixed) == -1) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout<<"ERROR: Time is out of the time range in the data file (Max 4:00)."<<endl;
                continue;}
        if ((getIndex(timeKey, findLeast(timeKey, fixed)) == -1) || (getIndex(timeKey, findLarger(timeKey, fixed)) == -1)) {cout<<"ERROR: An unexpected error has occured."<<endl;
        return 0;}

        timeVec.push_back(timeIt(findLeast(timeKey, fixed))); timeVec.push_back(timeIt(findLarger(timeKey, fixed)));
        iradVec.push_back( timeIrradiance[getIndex(timeKey, findLeast(timeKey, fixed))] ); iradVec.push_back( timeIrradiance[getIndex(timeKey, findLarger(timeKey, fixed))] );
        tempVec.push_back( timeTemp[getIndex(timeKey, findLeast(timeKey, fixed))] ); tempVec.push_back( timeTemp[getIndex(timeKey, findLarger(timeKey, fixed))] );
        found = 0;

        double m1 = (iradVec[1] - iradVec[0]) / (timeVec[1] - timeVec[0]);
        double m2 = (tempVec[1] - tempVec[0]) / (timeVec[1] - timeVec[0]);

        double b1 = iradVec[0] - m1*(timeVec[0]);
        double b2 = tempVec[0] - m2*(timeVec[0]);

        irradiance = (m1* timeIt((fixed))) + b1;
        temperature = (m2 * timeIt((fixed))) + b2;
        break;
```

1. If the `found` variable is equal to 0 (indicating that the desired data was not found in the previous search):
2. The `unify()` function is called with the `time` string as an argument. The result is converted to an integer using `stoi()` and assigned to the `fixed` variable.
3. Several conditional checks are performed to validate the time range in the data file:
   - `findLarger(timeKey, fixed) == -1` checks if a time larger than the given `fixed` time exists in the `timeKey` vector.
   - `findLeast(timeKey, fixed) == -1` checks if a time smaller than the given `fixed` time exists in the `timeKey` vector.
4. If either of the above checks fails, an error message is displayed, input stream is cleared, and the loop continues to the next iteration.
5. If the checks pass, the time indices corresponding to the smallest time less than `fixed` and the largest time greater than `fixed` are obtained using `findLeast()` and `findLarger()` functions.
6. The obtained time indices and their corresponding irradiance and temperature values are stored in separate vectors (`timeVec`, `iradVec`, and `tempVec`).
7. The `found` variable is set to 0 to indicate that the desired data has been found.
8. The slope (`m1` and `m2`) and y-intercept (`b1` and `b2`) of the linear equations for irradiance and temperature are calculated using the two points obtained from the previous step.
9. The irradiance and temperature values at the given `fixed` time are estimated using the calculated slope, y-intercept, and the `fixed` time.
10. The loop is exited with the `break` statement.

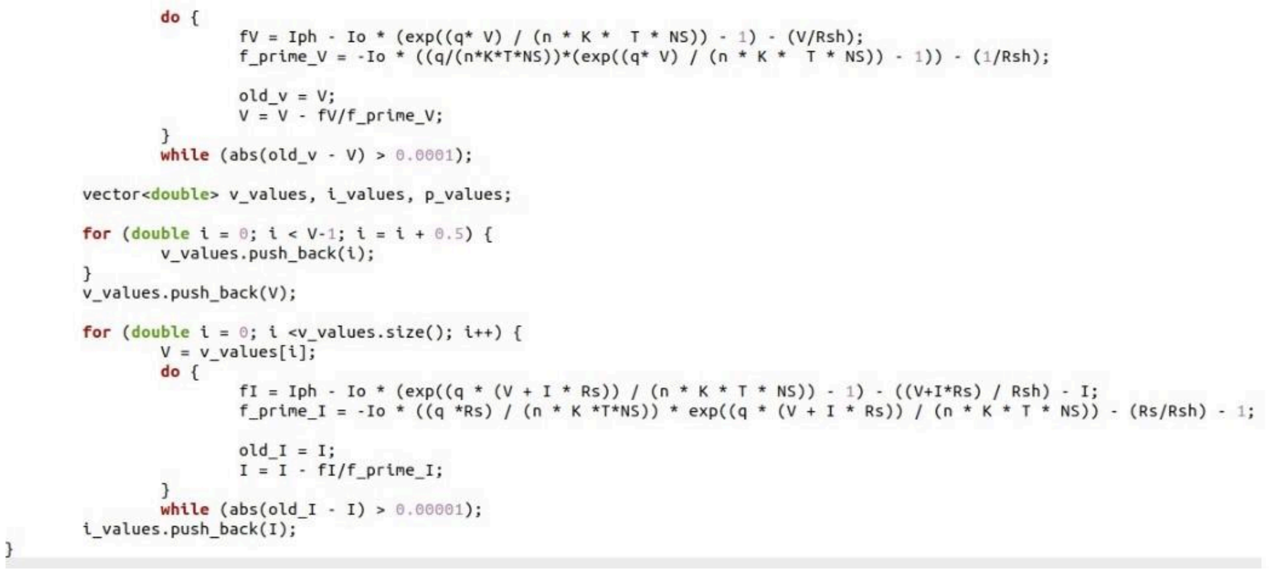
Code



```
        double ISCTR = 3.8, VOCTR = 21.168, V = VOCTR, T = temperature + 273.15, Gr = 1000, a = 0.00065, b = 0.08,Tr = 298.15, n = 2.1, q = 1.60*pow(10,-19), E = 1.76*pow(10,-19), NS = 36, K =
1.38*pow(10,-23), nKT = n * K * T, nKTr = n * K * Tr;
        double I0TR = ISCTR / (exp(((q*VOCTR)/NS)/nKTr) - 1);

        double Isc =ISCTR;
        double Voc = VOCTR;
        double Iph = (Isc + (a*(T-Tr))) * (irradiance/Gr) ;
        double Io = (Isc + (a*(T-Tr))) / (exp((q* (Voc + b*(T-Tr)) ) / (n * K *  T * NS)) - 1);

        double Rs = 0.008*36;
        double Rsh = 1000*36;
        double I = Isc, fI, f_prime_I, old_I, fV, f_prime_V, old_v, maxPower = 0, vPower, iPower;
```

1. Several variables are declared and initialized:
    - \`ISCTR\` and \`VOCTR\` are constants representing the short-circuit current and open-circuit voltage, respectively.
    - \`V\` is initialized with the value of \`VOCTR\`.
    - \`T\` is initialized with the value of \`temperature + 273.15\`, converting the temperature to Kelvin.
    - \`Gr\` is a constant representing the reference irradiance value.
    - \`a\` and \`b\` are constants used in the calculation.
    - \`Tr\` is the reference temperature in Kelvin.
    - \`n\` is a constant representing the diode ideality factor.
    - \`q\` is the elementary charge.
    - \`E\` is a constant.
    - \`NS\` is the number of solar cells in series.
    - \`K\` is the Boltzmann constant.
    - \`nKT\` and \`nKTr\` are calculated values used in the equations.
2. The value of \`I0TR\` is calculated using the given equations.
3. The values of \`Isc\` and \`Voc\` are assigned the values of \`ISCTR\` and \`VOCTR\`, respectively.
4. The value of \`Iph\` is calculated using the given equation, which takes into account the irradiance and temperature values.
5. The value of \`Io\` is calculated using the given equation, which takes into account the voltage, temperature, and other constants.
6. The values of \`Rs\` and \`Rsh\` are initialized with constants.
7. The values of \`I\`, \`fI\`, \`f_prime_I\`, \`old_I\`, \`fV\`, \`f_prime_V\`, \`old_v\`, \`maxPower\`, \`vPower\`, and \`iPower\` are declared.

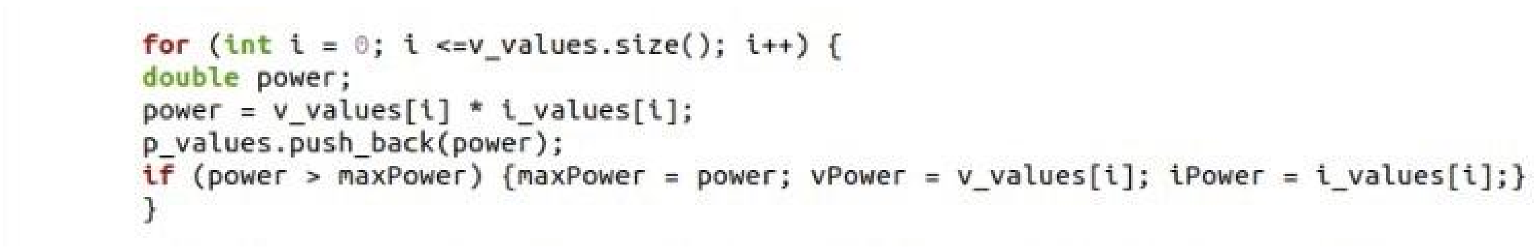
Code



```
        do {
                fV = Iph - Io * (exp((q* V) / (n * K *  T * NS)) - 1) - (V/Rsh);
                f_prime_V = -Io * ((q/(n*K*T*NS))*(exp((q* V) / (n * K *  T * NS)) - 1)) - (1/Rsh);

                old_v = V;
                V = V - fV/f_prime_V;
        }
        while (abs(old_v - V) > 0.0001);

    vector<double> v_values, i_values, p_values;

    for (double i = 0; i < V-1; i = i + 0.5) {
            v_values.push_back(i);
    }
    v_values.push_back(V);

    for (double i = 0; i <v_values.size(); i++) {
            V = v_values[i];
            do {
                    fI = Iph - Io * (exp((q * (V + I * Rs)) / (n * K * T * NS)) - 1) - ((V+I*Rs) / Rsh) - I;
                    f_prime_I = -Io * ((q *Rs) / (n * K *T*NS)) * exp((q * (V + I * Rs)) / (n * K * T * NS)) - (Rs/Rsh) - 1;

                    old_I = I;
                    I = I - fI/f_prime_I;
            }
            while (abs(old_I - I) > 0.00001);
    i_values.push_back(I);
}
```

1. A do-while loop is executed to find the value of `V` at which the equation `fV = 0` is satisfied within a certain tolerance.
    - Inside the loop, `fV` and `f_prime_V` are calculated based on the given equations.
    - The current value of `V` is stored in `old_v`.
    - The new value of `V` is updated using the equation `V = V - fV/f_prime_V`.
    - The loop continues as long as the absolute difference between `old_v` and `V` is greater than 0.0001.
2. A vector called `v_values` is declared to store the values of `V` for which the calculations will be performed.
3. A for loop is executed to populate the `v_values` vector.
    - The loop starts with `i` initialized to 0 and continues until `i` is less than `V-1`.
    - Inside the loop, the value of `i` is added to the `v_values` vector in increments of 0.5.
    - Finally, the value of `V` is added to the `v_values` vector.
4. Another for loop is executed to calculate the corresponding `I` values for each `V` value in the `v_values` vector.
    - The loop iterates over the elements of the `v_values` vector.
    - Inside the loop, the current value of `V` is assigned to `V`.
    - A do-while loop is executed to find the value of `I` at which the equation `fI = 0` is satisfied within a certain tolerance.
    - Inside the loop, `fI` and `f_prime_I` are calculated based on the given equations.
    - The current value of `I` is stored in `old_I`.
    - The new value of `I` is updated using the equation `I = I - fI/f_prime_I`.
    - The loop continues as long as the absolute difference between `old_I` and `I` is greater than 0.00001.
    - The final value of `I` is added to the `i_values` vector.

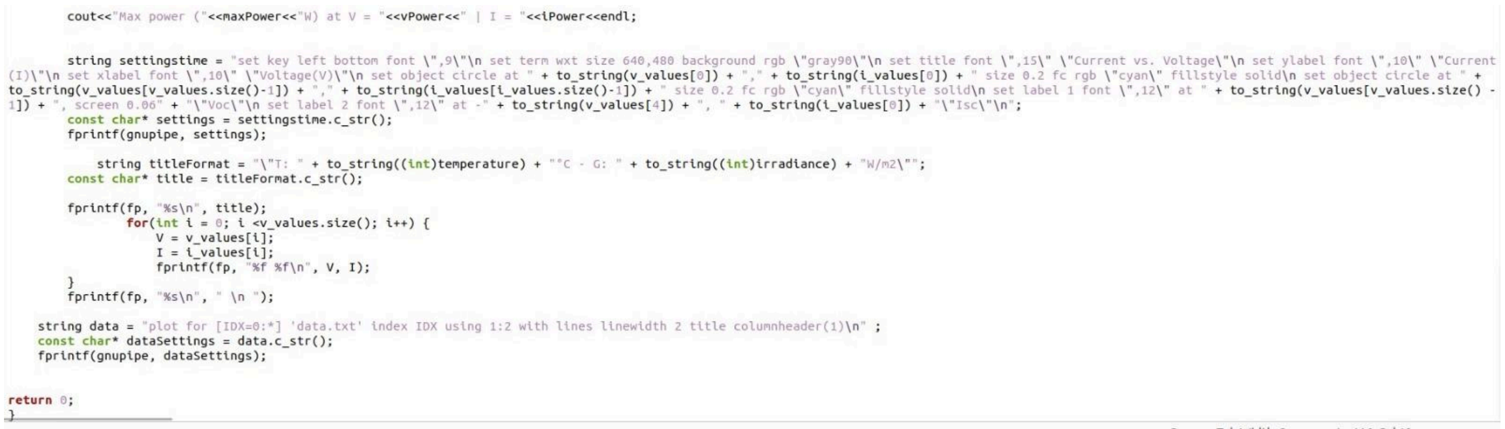
Code



```
for (int i = 0; i <=v_values.size(); i++) {
double power;
power = v_values[i] * i_values[i];
p_values.push_back(power);
if (power > maxPower) {maxPower = power; vPower = v_values[i]; iPower = i_values[i];;}
}
```

1. A for loop is executed to calculate the power values (\`p_values\`) for each \`V\` and \`I\` combination in the \`v_values\` and \`i_values\` vectors.
    - The loop iterates from 0 to the size of the \`v_values\` vector.
    - Inside the loop, the current \`V\` and \`I\` values are obtained from the \`v_values\` and \`i_values\` vectors respectively.
    - The power value is calculated by multiplying the \`V\` and \`I\` values.
    - The power value is added to the \`p_values\` vector.
    - If the calculated power value is greater than the current \`maxPower\` value, the \`maxPower\`, \`vPower\`, and \`iPower\` variables are updated with the new values.

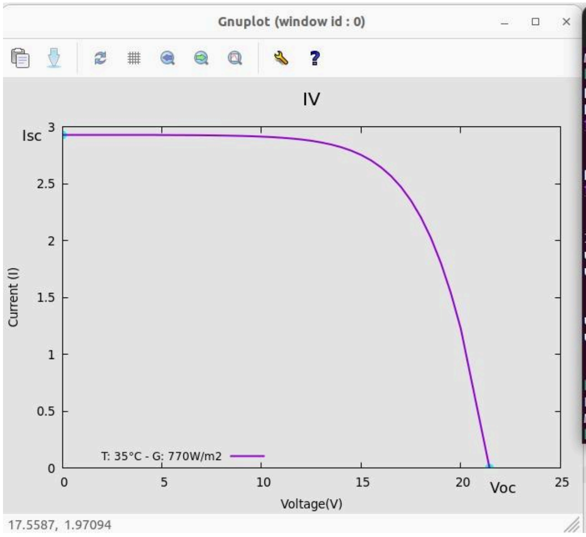
Code

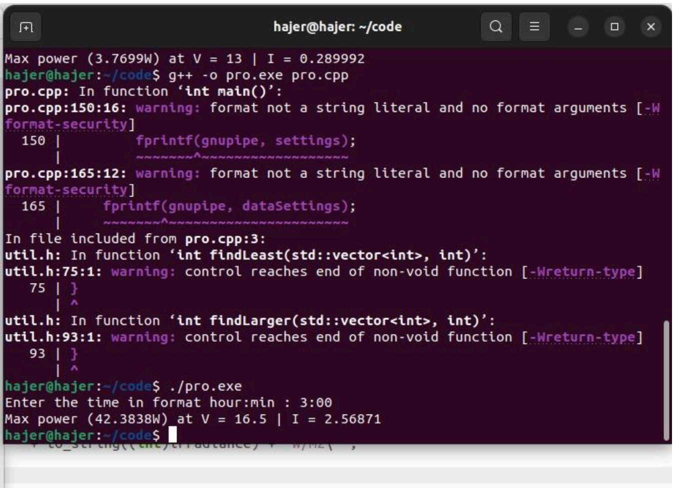


```
        cout<<"Max power ("<<maxPower<<"W) at V = "<<vPower<<" | I = "<<iPower<<endl;


        string settingstime = "set key left bottom font \",9\"\n set term wxt size 640,480 background rgb \"gray90\"\n set title font \",15\" \"Current vs. Voltage\"\n set ylabel font \",10\" \"Current
(I)\"\n set xlabel font \",10\" \"Voltage(V)\"\n set object circle at " + to_string(v_values[0]) + "," + to_string(i_values[0]) + " size 0.2 fc rgb \"cyan\" fillstyle solid\n set object circle at " +
to_string(v_values[v_values.size()-1]) + "," + to_string(i_values[i_values.size()-1]) + " size 0.2 fc rgb \"cyan\" fillstyle solid\n set label 1 font \",12\" at " + to_string(v_values[v_values.size() -
1]) + ", screen 0.06" + "\"Voc\"\n set label 2 font \",12\" at -" + to_string(v_values[4]) + ", " + to_string(i_values[0]) + "\"Isc\"\n";
        const char* settings = settingstime.c_str();
        fprintf(gnupipe, settings);

            string titleFormat = "\"T: " + to_string((int)temperature) + "°C - G: " + to_string((int)irradiance) + "W/m2\"";
        const char* title = titleFormat.c_str();

        fprintf(fp, "%s\n", title);
                for(int i = 0; i <v_values.size(); i++) {
                    V = v_values[i];
                    I = i_values[i];
                    fprintf(fp, "%f %f\n", V, I);
        }
        fprintf(fp, "%s\n", " \n ");

    string data = "plot for [IDX=0:*] 'data.txt' index IDX using 1:2 with lines linewidth 2 title columnheader(1)\n" ;
    const char* dataSettings = data.c_str();
    fprintf(gnupipe, dataSettings);


return 0;
}
```

1. The first `cout` statement prints the maximum power value (`maxPower`), as well as the corresponding `V` and `I` values (`vPower` and `iPower`).
2. The code initializes a string variable `settingstime` with a series of settings for the plot. These settings include the position and style of circles, labels, title, and axes fonts. The values of `V` and `I` from the `v_values` and `i_values` vectors are used to set the positions of the circles. The `to_string()` function is used to convert numeric values to strings.
3. The `settings` string is converted to a `const char*` using the `c_str()` function and passed to `fprintf` to send the settings to Gnuplot for configuration.
4. A string variable `titleFormat` is initialized with a formatted title string based on the temperature and irradiance values. The `to_string()` function is used to convert the numeric values to strings.
5. The `title` string is converted to a `const char*` using the `c_str()` function and passed to `fprintf` to set the title for the plot.
6. A for loop is executed to iterate over the `v_values` and `i_values` vectors and write the `V` and `I` values to a file using `fprintf`. The format `%f %f\n` is used to write the values as floating-point numbers.
7. After writing the `V` and `I` values, a newline character is written to the file using `fprintf` to separate the data.
8. A string variable `data` is initialized with a command to plot the data from the file using Gnuplot. The command uses the `plot` keyword and specifies the data file (`data.txt`) and the column headers for the plot.
9. The `dataSettings` string is converted to a `const char*` and passed to `fprintf` to send the plot command to Gnuplot.
10. The code returns 0 to indicate successful execution.

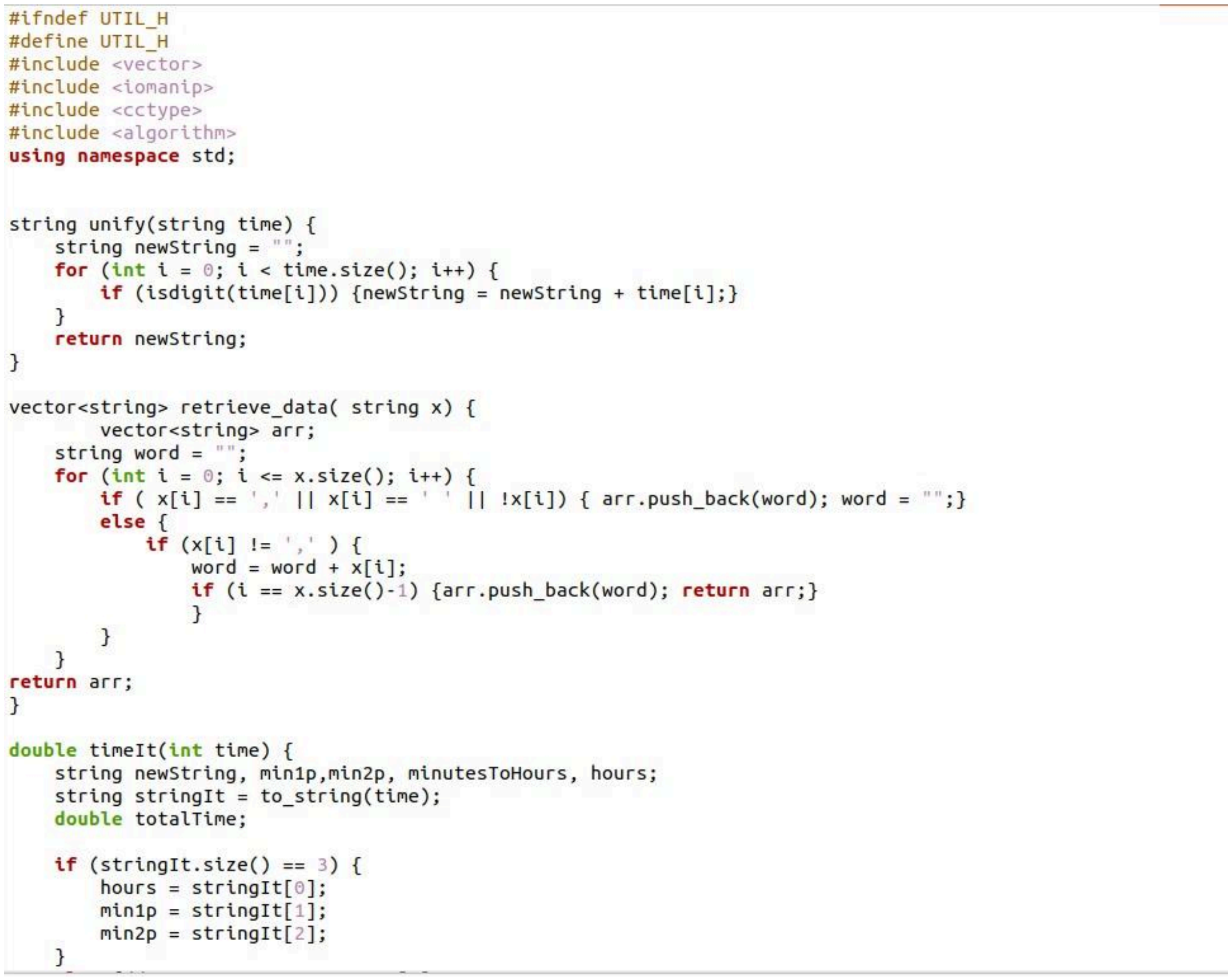
Result



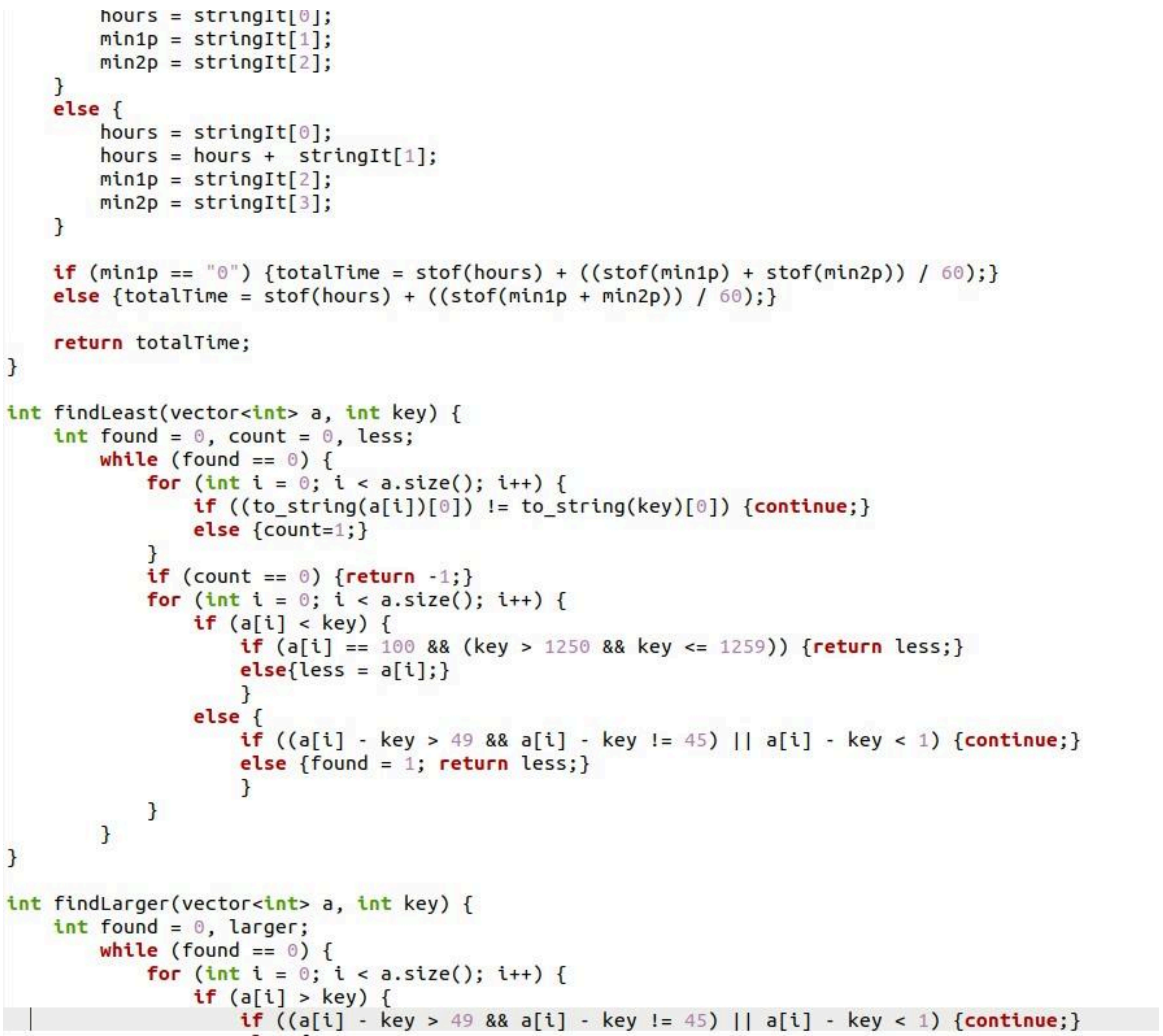


```
Max power (3.7699W) at V = 13 | I = 0.289992
hajer@hajer:~/code$ g++ -o pro.exe pro.cpp
pro.cpp: In function ‘int main()’:
pro.cpp:150:16: warning: format not a string literal and no format arguments [-W
format-security]
  150 |         fprintf(gnupipe, settings);
      |         ~~~~~~~^~~~~~~~~~~~~~~~~~~
pro.cpp:165:12: warning: format not a string literal and no format arguments [-W
format-security]
  165 |     fprintf(gnupipe, dataSettings);
      |     ~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~
In file included from pro.cpp:3:
util.h: In function ‘int findLeast(std::vector<int>, int)’:
util.h:75:1: warning: control reaches end of non-void function [-Wreturn-type]
   75 | }
      | ^
util.h: In function ‘int findLarger(std::vector<int>, int)’:
util.h:93:1: warning: control reaches end of non-void function [-Wreturn-type]
   93 | }
      | ^
hajer@hajer:~/code$ ./pro.exe
Enter the time in format hour:min : 3:00
Max power (42.3838W) at V = 16.5 | I = 2.56871
hajer@hajer:~/code$
```

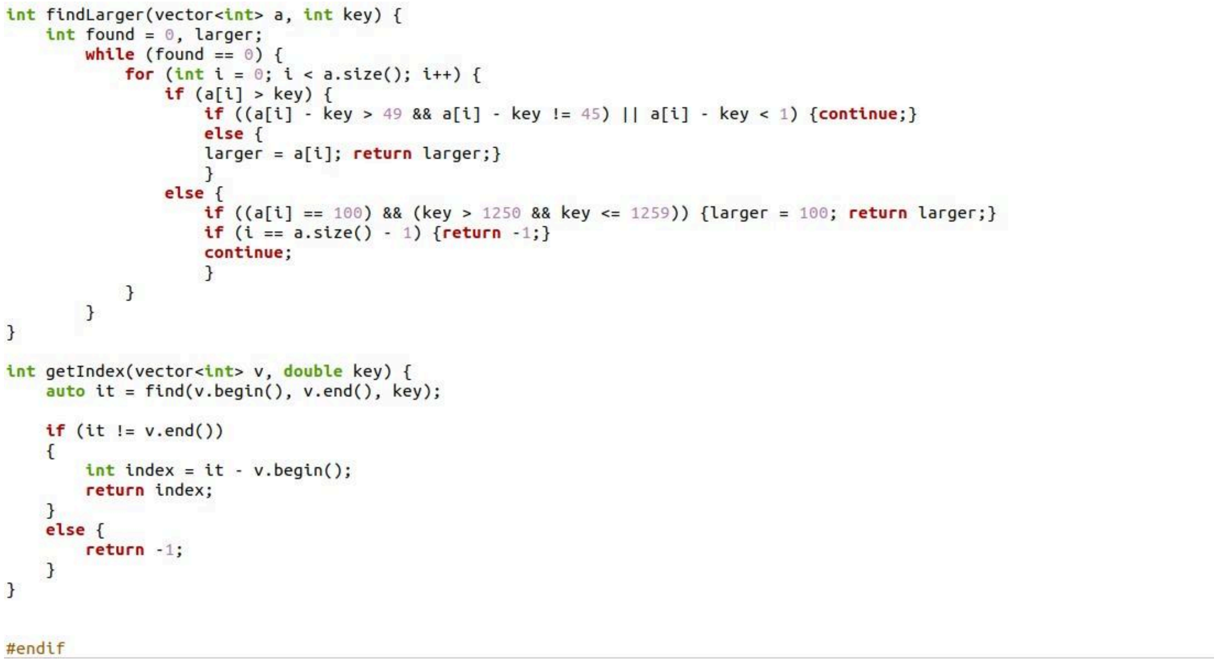
Code Functions



```cpp
#ifndef UTIL_H
#define UTIL_H
#include <vector>
#include <iomanip>
#include <cctype>
#include <algorithm>
using namespace std;


string unify(string time) {
    string newString = "";
    for (int i = 0; i < time.size(); i++) {
        if (isdigit(time[i])) {newString = newString + time[i];}
    }
    return newString;
}

vector<string> retrieve_data( string x) {
        vector<string> arr;
    string word = "";
    for (int i = 0; i <= x.size(); i++) {
        if ( x[i] == ',' || x[i] == ' ' || !x[i]) { arr.push_back(word); word = "";}
        else {
            if (x[i] != ',' ) {
                word = word + x[i];
                if (i == x.size()-1) {arr.push_back(word); return arr;}
                }
        }
    }
return arr;
}

double timeIt(int time) {
    string newString, min1p,min2p, minutesToHours, hours;
    string stringIt = to_string(time);
    double totalTime;

    if (stringIt.size() == 3) {
        hours = stringIt[0];
        min1p = stringIt[1];
        min2p = stringIt[2];
    }
```



```
        hours = stringIt[0];
        min1p = stringIt[1];
        min2p = stringIt[2];
    }
    else {
        hours = stringIt[0];
        hours = hours +  stringIt[1];
        min1p = stringIt[2];
        min2p = stringIt[3];
    }

    if (min1p == "0") {totalTime = stof(hours) + ((stof(min1p) + stof(min2p)) / 60);}
    else {totalTime = stof(hours) + ((stof(min1p + min2p)) / 60);}

    return totalTime;
}

int findLeast(vector<int> a, int key) {
    int found = 0, count = 0, less;
        while (found == 0) {
            for (int i = 0; i < a.size(); i++) {
                if ((to_string(a[i])[0]) != to_string(key)[0]) {continue;}
                else {count=1;}
            }
            if (count == 0) {return -1;}
            for (int i = 0; i < a.size(); i++) {
                if (a[i] < key) {
                    if (a[i] == 100 && (key > 1250 && key <= 1259)) {return less;}
                    else{less = a[i];}
                    }
                else {
                    if ((a[i] - key > 49 && a[i] - key != 45) || a[i] - key < 1) {continue;}
                    else {found = 1; return less;}
                    }
            }
        }
}

int findLarger(vector<int> a, int key) {
    int found = 0, larger;
        while (found == 0) {
            for (int i = 0; i < a.size(); i++) {
                if (a[i] > key) {
                    if ((a[i] - key > 49 && a[i] - key != 45) || a[i] - key < 1) {continue;}
```



```
int findLarger(vector<int> a, int key) {
    int found = 0, larger;
        while (found == 0) {
            for (int i = 0; i < a.size(); i++) {
                if (a[i] > key) {
                    if ((a[i] - key > 49 && a[i] - key != 45) || a[i] - key < 1) {continue;}
                    else {
                    larger = a[i]; return larger;}
                    }
                else {
                    if ((a[i] == 100) && (key > 1250 && key <= 1259)) {larger = 100; return larger;}
                    if (i == a.size() - 1) {return -1;}
                    continue;
                    }
            }
        }
}

int getIndex(vector<int> v, double key) {
    auto it = find(v.begin(), v.end(), key);

    if (it != v.end())
    {
        int index = it - v.begin();
        return index;
    }
    else {
        return -1;
    }
}


#endif
```

Challenges and Problems Faced

1. Missing Data Handling: One significant challenge was dealing with missing values for irradiance and temperature in the dataset. Since accurate estimation was crucial for the analysis, an estimation process was implemented. However, estimating values based on available data introduced a level of uncertainty that needed to be carefully considered during the analysis.
2. Data Consistency and Accuracy: Ensuring the accuracy and consistency of the collected data was another challenge. Any discrepancies or inaccuracies in the measurements could potentially affect the analysis and characterization of the solar panel. Therefore, extensive data validation and verification were performed to minimize errors and ensure reliable results.
3. Estimation Accuracy: The accuracy of the estimation process for missing irradiance and temperature values was another challenge. The reliability of the estimation method depended on the quality and completeness of the available data. It was necessary to carefully assess the estimated values and compare them with measured data to evaluate the accuracy of the estimation process.
4. Plot Generation and Interpretation: Generating an accurate and informative plot using GNUplot required careful consideration of various parameters and settings. Additionally, interpreting the plot and extracting meaningful insights from the current-voltage characteristics required a thorough understanding of solar panel behavior and performance analysis techniques.

Team management, Timeline, Explanation of individual tasks

In our group project on analyzing the IV characteristics of a solar panel under specific irradiance and temperature conditions, we organized our work effectively. Each team member had a specific role based on our strengths and skills. One of us focused on collecting and processing accurate irradiance and temperature data to ensure the reliability of our analysis. Another team member was in charge of implementing the complex mathematical calculations needed to determine the precise IV curve, using the Newton-Raphson method. Additionally, one team member did the plots to display the IV characteristics graphically. This division of tasks allowed us to make progress efficiently, utilizing our individual abilities to the fullest. Regular communication and collaboration were key to our success, as we kept each other informed and tackled challenges together. Through our joint efforts, we gained valuable insights into how the solar panel behaves under specific environmental conditions.

Conclusion

The project focused on the analysis and characterization of a solar panel using real-world data obtained from measurements. By utilizing C++ programming language and GNUplot, the project successfully processed and analyzed the collected data, estimated missing values, and generated a plot illustrating the current-voltage characteristics of the solar panel.

Through the analysis, valuable insights were gained regarding the solar panel's performance, efficiency, and potential for power generation. The calculations performed allowed for the determination of the maximum power point, which is crucial for optimizing the solar panel's performance. By identifying the voltage and current at the maximum power point, further improvements can be made to enhance the efficiency of the solar panel system.

However, the project also encountered certain challenges that required careful consideration. Handling missing data for irradiance and temperature was a significant challenge. The estimation process employed in the project helped to address this issue, but the accuracy of the estimations depended on the available data and the reliability of the estimation method. It is important to note that estimation introduces a level of uncertainty that should be taken into account when analyzing the results.

The generated current-voltage characteristics plot provided a visual representation of the solar panel's behavior and allowed for a better understanding of its performance under different operating conditions. The plot served as a valuable tool for analysis and provided insights into the relationship between voltage and current, enabling the identification of the maximum power point.

In conclusion, this project successfully analyzed and characterized the performance of a solar panel using real-world data. The findings contribute to the broader field of renewable energy research, providing insights for optimizing the efficiency and performance of solar panels. Further enhancements can be made by improving the accuracy of data collection, refining the estimation process, and considering additional factors such as shading and panel orientation. By continuing to advance our understanding of solar panel behavior, we can harness the full potential of solar energy and contribute to a more sustainable future.