Programme module's description

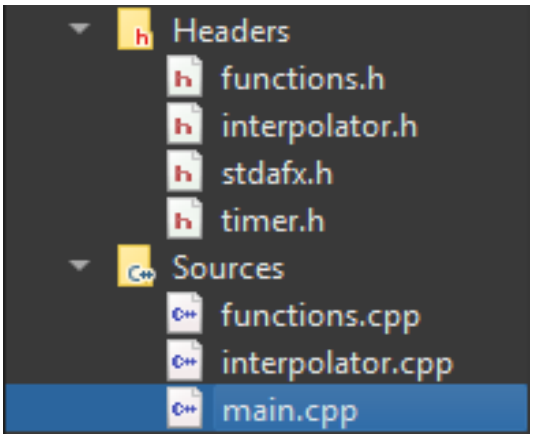
Nick Zadubrovsky

Contents:

Structure

Programme's source code is represented by 3 (**.cpp*) files each having its own corresponding *header* file (**.h*):

- *functions.cpp*
- i*nterpolator.cpp*
- *main.cpp*



Header files:

- *functions.h*
- *interpolator.h*
- *stdafx.h*
- *timer.h*

functions.cpp

Contains realisation of methods declared in **functions.h**
In turn, this .cpp is visually split into 6 major groups of functions (just on purpose of an easier navigation through it):

- .TXT FILE PROCESSING FUNCTIONS,
- .INP FILE PROCESSING FUNCTIONS,
- .CSV FILE PROCESSING FUNCTIONS,
- T FIELD FILE PROCESSING FUNCTIONS,

are responsible for processing data files with an appropriate extensions.

Each of the above sections is represented by a *parser function* (***txtParse***, ***inpParse***, ***csvParse***) and *mesh-getter function* (***GetMeshTXT***, ***GetMeshINP***, ***GetMeshCSV***). Besides parser and *mesh-getter*, the CSV section has ***GetTempCSV*** member function, which purpose is to retrieve T values from a csv table (those are the headers of columns).

- GRID PROCESSING FUNCTIONS includes ***interMesh*** function, which is responsible for interpolation of meshes passed to it.
- MATH FUNCTIONS AND BOOLEAN FILTERS represent mathematical method of integration (***Trapezoidal***, once overloaded), physical functions (***Planck***), and binary predicates (**isinX**, **isinT**).

interpolator.cpp

Along with its declaration header (*interpolator.h*) represent class which implements linear interpolation (linearly weighted) to mesh passed into its class' object as parameter. Interpolation by temperatures' grid is conducted implicitly within the procedure of getting mesh out of the data file.

timer.h

Auxiliary single-filed class used within *main.cpp* to estimate physical time of programme's execution. To do the latter, just an instance of the class needs to be declared within the scope of interest (timer starts and ends within the realm enshrouds the instance).

stdafx.h

Contains preprocessor directives with libs to be used further in the programme.

main.cpp

The main file. Consists of only *main()* function.

```
 6 int main()
 7 {
 8     Timer t; //time counter
 9
10     std::string path_csv = "..\\input\\data_sample.csv";
11
12     std::string path_inp = "..\\input\\data_sample.inp";
13
14     std::string path_txt = "..\\input\\flow_sample.txt";
15
16     std::string path_tmp = "..\\input\\T";
17
18     std::vector<std::vector<double>> kabs;
19
20     std::vector<std::pair<double, double>> mesh2;
21
22     std::vector<double> Temperatures = GetTempField(path_tmp);
```

The programme's execution goes as follows:

1. **On lines 8 – 22** here are t (object of class *Timer* declared in *timer.h*; calculates physical time of code execution within the scope it encased in), paths to the input files, and some data containers (*kabs*, *mesh2*, *Temperatures*).

```
24    // Choose the input file format for K_ABS: "inp" or "csv"
25    std::string mode = "csv";
26
27    if (mode == "inp") {
28
29        thread t1([&path_inp, &kabs]() {kabs = inpParse(path_inp);});
30
31        thread t2([&path_txt, &mesh2]() {mesh2 = GetMeshTXT(txtParse(path_txt));});
32
33        t1.join();
34
35        t2.join();
36    }
37
38    if (mode == "csv") {
39
40        thread t1([&path_csv, &kabs]() {kabs = csvParse(path_csv);});
41
42        thread t2([&path_txt, &mesh2]() {mesh2 = GetMeshTXT(txtParse(path_txt));});
43
44        t1.join();
45
46        t2.join();
47    }
```

2. **On line 25** here is *std::string mode = "csv";* – mode switcher. Switches programme's data input format for absorption coefficients.
   Default: "csv"
   Alternative: "inp"

   Note: If you change the input format, ensure you have put a file with appropriate extension to *"..\calculation\input"* with respect to the file names suggested in lines 10 or/and 12.

3. **On lines 27 – 36 and 38 – 47** here is main programme's threading branches. With respect the mode chosen, the execution is divided into 2 sub-processes of *parsing file with absorption coefficients* and *retrieving mesh from txt-file data (intensities).*

   GENERAL MESH STRUCTURE
   All the mesh (***mesh1, mesh2, mesh1new, mesh2new, func***) from now on are constructed as

   std::vector<std::pair<double,double>>

   where the *first double* number stands for the *wavelength,*
   and the *second double* – specific mesh-related value.

```
49 /*=================CREATING=AN=OUTPUT=FILE====================================*/
50
51     std::ofstream data;
52
53     data.open("..\\output\\output", std::ios_base::out);
54
55     data << Temperatures.size() << "\n(\n";
56
57 /*==============ITERATING=OVER=TEMPERATURE=RANGE==============================*/
58
59     for (size_t i = 0; i < Temperatures.size(); ++i){
60
61     double T = Temperatures[i];
62
63     vector<pair<double, double>> mesh1, mesh1new, mesh2new;
64
65 /*=============GETTING=MESH=FROM=KABS=========================================*/
66
67     if (mode == "inp") {mesh1 = GetMeshINP(kabs, T);}
68
69     if (mode == "csv") {mesh1 = GetMeshCSV(kabs, T);}
```

4. Snippet called **CREATING AN OUTPUT FILE** is responsible for the output file creation.
   On line 53, to change output destination and/or output file name, change the first, – *std::string-type* – parameter, *“..\calculation\output”*

5. Next, **ITERATING OVER TEMPERATURE RANGE** snippet stands for repeating following procedures due to different temperatures' values previously retrieved from *T field file.*

6. **GETTING MESH FROM KABS** section, with respect to the aforechosen mode (see p.2), is responsible for obtaining mesh from inp/csv parsed data, stored in ***kabs***.

```
71 /*=============INTERPOLATION=============================================================*/
72
73     mesh1new = interMesh(mesh1, mesh2);
74
75     mesh2new = interMesh(mesh2, mesh1);
76
77 /*====================ERASING=REDUNDANT=POINTS==============================*/
78
79     mesh1new.erase(std::unique(mesh1new.begin(), mesh1new.end(),
80              [](const std::pair<double, double>& a, const std::pair<double, double>& b)
81              {return std::abs(a.first - b.first) < 1e-11;}), mesh1new.end());
82
83     mesh2new.erase(std::unique(mesh2new.begin(), mesh2new.end(),
84              [](const std::pair<double, double>& a, const std::pair<double, double>& b)
85              {return std::abs(a.first - b.first) < 1e-11;}), mesh2new.end());
```

7. **INTERPOLATION** section implements *interMesh* function to the meshes retrieved from txt-file with intensities and inp/csv-file with absorption coefficients.

```
87 /*=========================SORTING=CHECK=====================================*/
88
89     if (!std::is_sorted(std::begin(mesh1new), std::end(mesh1new),
90                  [](std::pair<double, double>& a, std::pair<double, double>& b)
91                  {return a.first < b.first;}))
92              {std::sort(std::begin(mesh1new), std::end(mesh1new),
93              [](std::pair<double, double>& a, std::pair<double, double>& b)
94              {return a.first < b.first;});}
95
96     if (!std::is_sorted(std::begin(mesh2new), std::end(mesh2new),
97                         [](std::pair<double, double>& a, std::pair<double, double>& b)
98                         {return a.first < b.first;}))
99              {std::sort(std::begin(mesh2new), std::end(mesh2new),
100             [](std::pair<double, double>& a, std::pair<double, double>& b)
101             {return a.first < b.first;});}
```

8. **ERASING REDUNDANT POINTS** piece takes a double-check on whether the interpolated meshes contain equal points by sieving latter with the accuracy given (*1e-11*, for wavelength).

9. **SORTING CHECK** section is made up to ensure the mesh being processed (*mesh1new*, *mesh2new*) is sorted in ascending order by the first argument of a point (*wavelength*).

```
103 /*=========================CALCULATION=======================================*/
104
105     std::vector<std::pair<double, double>> func/*, func_eq*/;
106
107     for (size_t i = 0; i < mesh1.size(); ++i) {
108
109         double mlt = mesh1[i].second * mesh2new[i].second *
110                      mesh1[i].first * 5.0307e33;
111         if (mlt > 1e-10 && !std::isnan(mlt)) {func.push_back(
112                         std::make_pair(mesh1[i].first, mlt));}
113     }
114
115     func.pop_back();
```

10. CALCULATION section is responsible for creating a container *func* and filling it with points of under-integral function of:

$$W = 4\pi z \int_{\lambda 1^*}^{\lambda 2^*} \frac{F(\lambda) * 10^9 k_{abs}(\lambda)\,\mathrm{d}\lambda}{h\frac{c}{\lambda}}$$

but without *4πz* coefficient.

```
117 /*====================ERASING=REDUNDANT=POINTS==============================*/
118
119     func.erase(std::unique(func.begin(), func.end(),
120             [](const std::pair<double, double>& a, const std::pair<double, double>& b)
121             {return std::abs(a.first - b.first) < 1e-11;}), func.end());
122
123 /*=====================INTEGRATION=========================================*/
124
125     double result = 4 * M_PI * integrateTrapezoidal(func);
126
127     data << result << '\n';     //writing into output file
128
129 /*==============OUTPUTTING=RESULTS=========================================*/
130
131 //    std::cout << T <<  " K\tRate:\t " << result  << '\n';
132
133 /*=========================================================================*/
134
135     }
136
137     data << ")\n;\n";
138
139     data.close();   //closing output file
```

11. **ERASING REDUNDANT POINTS** snippet is on the same purpose as described in p.8. Although, this time it operates with the wavelength-vs-under-integral-function points' set.

12. **INTEGRATION** section implements *integrateTrapezoidal* function (declared in *functions.cpp)* over *func* data range as parameter.

13. **OUTPUTTING RESULTS** is an optional section which, if uncommented, would show the integral's value in console.

14. The last, **on lines 137 and 139**, bunch of commands manages to end the insertion of the results into the output file and closes the latter, respectively.

Input/Output

All input files should by default be put in *"..\calculation\input".*
NOTE: It is possible to use one of the two – csv or inp – file formats.
Input files:

data.csv – file with absorption coefficients [*1/cm*], contains header line, each column separated either by a comma or by a semicolon.
The first column is wavelength-range [*nm*].
The rest of columns are absorption coefficients' ranges corresponding to a *particular temperature.*

data.inp – file with absorption coefficients [*1/m*], contains lines with those coefficients, but the first two values of a line are the *line's index* and *temperature* respectively.
The first line contains two numbers separated by *space*: *total line number*, and *the number of values within each one* (excluding first two numbers within).
The last line contains number of wavelength-values followed by a heading and then by wavelength-values themselves.

flow.txt – file containing 4 columns of numbers, wherein the *first column* contains wavelength in [*m*], the *second* – intensity [*W/m2/sr/nm*].

T – temperatures' field. See example attached.

Output:

output – file with the integral values with respect to the temperatures given in T file. Structured akin to the latter. See example.

Build

Built by means of *Desktop Qt 5.14.2 MinGW 32-bit*

F.A.Q.