|  |  |  |
| --- | --- | --- |
|  |  |  |

**Semester Project - CPU Temps**

**Thomas J. Kennedy**

**Contents:**

[**1 Overview**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#overview)

[**2 Program Arguments & Execution**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#program-arguments-execution)

[**2.1 Input Format**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#input-format)

[**2.2 Import Input Libraries**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#import-input-libraries)

[**2.3 Output Format**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#output-format)

[**3 Programming Requirements & Constraints**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#programming-requirements-constraints)

[**3.1 Design & Style Requirements**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#design-style-requirements)

[**3.2 Novel Solutions**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#novel-solutions)

[**3.3 Documentation Requirements**](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#documentation-requirements)

**Additional Datasets may be added.**

**1 Overview**

As a Computer Scientist, I have a number of interests. Many of these interests overlap. While designing this project, I happened to be batch encoding some videos. I decided to write a quick script to grab CPU temperature data every 30 seconds. This resulted in three sets of data:

* [sensors-2018.12.26.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2018.12.26.txt)
* [sensors-2019.01.26.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.01.26.txt)
* [sensors-2019.02.09.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.02.09.txt)
* [sensors-2018.12.26-no-labels.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2018.12.26-no-labels.txt)
* [sensors-2019.01.26-no-labels.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.01.26-no-labels.txt)
* [sensors-2019.02.09-no-labels.txt](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.02.09-no-labels.txt)

To visualize this data I used [Gnuplot](http://www.gnuplot.info/) to generate three graphs:

* [sensors-2018.12.26.svg](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2018.12.26.svg)
* [sensors-2019.01.26.svg](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.01.26.svg)
* [sensors-2019.02.09.svg](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/Public/sensors-2019.02.09.svg)

Click on each item to view either the graph or raw text file.

Each of the encoding jobs ran for 5 to 10 hours. If you look at the data you see four temperatures for each reading. My CPU is a 4-core (8 thread) Intel i7-6700K. I found myself interested in not only the behavior of the readings, but also in the temperature differences between the 4 CPU cores.

Your task is to take the temperature readings and generate for each core:

1. A piecewise linear interpolation.
2. A global linear least squares approximation.
3. *(Optional)* A cubic spline (or other non-linear) interpolation.

**2 Program Arguments & Execution**

Your program must accept an input filename as the first command line argument. Your program **must NOT** prompt the user for a filename.

**2.1 Input Format**

Data takes the form of temperatures in a txt file. All data points are whitespace delimited. For example, if I had 5 temperature readings:

**Example 1: Sample Input with Labels**

+61.0°C +63.0°C +50.0°C +58.0°C

+80.0°C +81.0°C +68.0°C +77.0°C

+62.0°C +63.0°C +52.0°C +60.0°C

+83.0°C +82.0°C +70.0°C +79.0°C

+68.0°C +69.0°C +58.0°C +65.0°C

**Example 2: Sample Input without Labels**

61.0 63.0 50.0 58.0

80.0 81.0 68.0 77.0

62.0 63.0 52.0 60.0

83.0 82.0 70.0 79.0

68.0 69.0 58.0 65.0

would be a possible input files. Each line represents temperature readings from 4 processor cores. Process each temperature column independently. Readings are taken every 30 seconds. In this example:

* line 1 is 0 *sec*
* line 2 is 30 *sec*,
* line 3 is 60 *sec*.
* line 4 is 120 *sec*.
* line 5 is 180 *sec*.

Your first step should be to pre-process this data into a usable form. **Conceptually,** you need the data in the following format:

This table is a conceptual visualization of the data. You may select any combination data structures, e.g., ADTS (classes or structs), arrays, lists, vectors, or maps.

| **Time (sec)** | **Core 0** | **Core 1** | **Core 2** | **Core 3** |
| --- | --- | --- | --- | --- |
| 0 | 61.0 | 63.0 | 50.0 | 58.0 |
| 30 | 80.0 | 81.0 | 68.0 | 77.0 |
| 60 | 62.0 | 63.0 | 52.0 | 60.0 |
| 120 | 83.0 | 82.0 | 70.0 | 79.0 |
| 180 | 68.0 | 69.0 | 58.0 | 65.0 |

**2.2 Import Input Libraries**

Note that if you are not already logged in. You will need to [**log in first**](https://git-community.cs.odu.edu/), then click [here](https://git-community.cs.odu.edu/tkennedy/cs417-lecture-examples/tree/sum19/SemesterProject-CPU-Temps/).

You may opt to #include or import the C++, Java, or Python input libraries provided [here](https://git-community.cs.odu.edu/tkennedy/cs417-lecture-examples/tree/sum19/SemesterProject-CPU-Temps/).

**2.3 Output Format**

All output must be written to text files (one file pre core). Each line must take the form:

xk<=x<xk+1xk<=x<xk+1; yi=c0+c1xyi=c0+c1x ; *type*

where

* xkxk and xk+1xk+1 are the domain in which ykyk is applicable
* ykyk is the kthkth function
* *type* is either *least-squares* or *interpolation*

For the example data in described in [Section 2.1 (Input Format)](https://www.cs.odu.edu/~tkennedy/cs417/latest/Assts/project-cpu-temps/index.html#input-format) you would generate 4 output files.

* {basename}-core-0.{txt}
* {basename}-core-1.{txt}
* {basename}-core-2.{txt}
* {basename}-core-3.{txt}

**3 Programming Requirements & Constraints**

* Submit your solution through Blackboard by the start of lecture on the due date.
* Submit a single zip file or tarball:
  + Provide a brief ReadMe file that specifies how to compile and
  + Provide makefiles (or equivalent build files) for any compilable code.
    - For C/C++, I recommend make or Cmake
    - For Java, I recommend Gradle

**3.1 Design & Style Requirements**

* All code must follow best practices for:
  + Indentation and spacing
  + Naming conventions
  + Top-down design (i.e., no monolithic functions)
* **Deductions will be applied for requirement violations.**

**3.2 Novel Solutions**

Extra consideration will be given to **novel solutions (e.g., those written in a functional language)**.

**3.3 Documentation Requirements**

All code must be properly and fully documented using a language appropriate comment style. All functions (including parameters and return types) must be documented.

1. Doxygen can be used for *C++*, *Java*, or *JavaScript*. Consider the following Doxygen Example:

**Example 3: C++ Doxygen Documentation**

/\*\*

\* Retrieve the value stored in three selected Cells

\*

\* @param cell1Id numeric id representing the 1st desired cell

\* @param cell2Id numeric id representing the 2nd desired cell

\* @param cell3Id numeric id representing the 3rd desired cell

\*

\* @return value stored in the Cell

\*

\* @pre (cell1Id > 0 && cell1Id < 10) &&

\* (cell2Id > 0 && cell2Id < 10) &&

\* (cell3Id > 0 && cell3Id < 10)

\*/

CellTriple get3Cells(int cell1Id, int cell2Id, int cell3Id) const;

1. Javadoc can be used for Java. Consider the following Javadoc Example:

**Example 4: Javadoc Documentation**

/\*\*

\* Multi-thread Coin Flip.

\*

\* @param numTrials # flips to simulate

\* @param numThreads number of threads to use

\*

\* @return Completed FlipTasks

\*

\* @throws InterruptedException if a thread is stopped prematurely

\*/

public static FlipTask[] multiThread(long numTrials, int numThreads)

throws InterruptedException

1. Pydoc or Sphinx can be used for Python. Consider the following Pydoc Example:

**Example 5: Python 3 Pydoc Documentation**

def parse\_raw\_temps(original\_temps: TextIO,

step\_size: int=30, units: bool=True) -> Iterator[Tuple[float, List[float]] ]:

"""

Take an input file and time-step size and parse all core temps.

:param original\_temps: an input file

:param step\_size: time-step in seconds

:param units: True if the input file includes units and False if the file

includes only raw readings (no units)

:yields: A tuple containing the next time step and a List containing \_n\_

core temps as floating point values (where \_n\_ is the number of

CPU cores)

"""

or the following Sphinx Example:

**Example 6: Python 3 Sphinx Documentation**

def parse\_raw\_temps(original\_temps: TextIO,

step\_size: int=30, units: bool=True) -> Iterator[Tuple[float, List[float]] ]:

"""

Take an input file and time-step size and parse all core temps.

Args:

original\_temps: an input file

step\_size: time-step in seconds

units: True if the input file includes units and False if the file

includes only raw readings (no units)

Yields:

A tuple containing the next time step and a List containing \_n\_

core temps as floating point values (where \_n\_ is the number of

CPU cores)

"""

|  |  |  |
| --- | --- | --- |
|  |  |  |

© 2018-2019, Old Dominion Univ.