cpplot

Release x.y.unknown

Tom Clark

Aug 11, 2020

Contents

Attention: This library is in use in several projects, but it's still early days. Like the idea of it? Please star us on GitHub and contribute via the issues board and roadmap.

"cpplot" ~ interactive figures you can show on the web

This library allows you to create interactive graphs and charts in C++ 11 upward, which are viewable in-browser.

You can save figures to disc as `\*.json` files. Their contents are compliant with the `plotly.js` library schema so figures can be rendered in javascript.

Get going with the *quick start*, or *read about why I started this*...

CHAPTER 1

Contents

1.1 Quick Start

1.1.1 Viewing Figures

A browser based viewer is supplied at https://cpplot.herokuapp.com. Navigate to that link and you'll be able to view and export figures created with **cpplot**.

The viewer always keeps your data local (files are never uploaded to a server).

If you've not created figure files yourself yet, you can download the examples from here and then select them in the viewer. Some are simple, some are amazing ;)

1.1.2 Create Your First Figure

To create your first figure with **cpplot**, you can compile and run the program below:

```
/*
 * main.cpp - An example program to create your first figure as a Plotly-compliant .
↪json file.
 *
 * Author:              YOU
 *
 * Copyright (c) YEAR YOU. All Rights Reserved.
 *
 */
#include <stdio.h>
#include <iostream>
#include <Eigen/Dense>
#include "cpplot.h"     // TODO these may need to change to something like <cpplot>␣
↪for users
#include "eigen.h"      //  who have installed the library rather than linking␣
↪against source code
```

(continues on next page)

(continued from previous page)

```
#include "exceptions.h" //  directly

using namespace std;
using namespace Eigen;
using namespace cpplot;

int main(int argc, char* argv[]) {

    // Create a figure object
    Figure fig = Figure();

    // Create a scatter plot object and set its values
    ScatterPlot p1 = ScatterPlot();
    p1.x = Eigen::VectorXd::LinSpaced(10, 0.0, 1.0);
    p1.y = Eigen::VectorXd::LinSpaced(10, 1.0, 2.0);

    // Alter its properties
    p1.name = "my first trace";
    p.setWidth(1);
    p.setDash("dash");
    p.setColor("#e377c2");  // "Raspberry yoghurt pink", obviously the best selection

    // Add it to the figure
    fig.add(p1);

    // You can add multiple plots to a figure...
    ScatterPlot p2 = ScatterPlot();
    p2.name = "my second trace";
    fig.add(p2);

    // And whenever you want, you can write a figure to disc.
    // Note, writing is an async task so your calculations can
    // proceed whilst writing goes into the background.
    //
    // In this example we allow the write() method to automatically append a `.json`␣
↪extension
    // to the file name (if not already present), and we prevent it from printing the␣
↪json to stdout
    // (which can be a useful feature to enable if you want to use your program with␣
↪bash pipes)
    std::cout << "Writing figure to current directory: " << std::endl;
    fig.write("my_first_cpplot_figure", true, false);

}
```

1.1.3 Building Your Own Web Tools

You (or the frontend team you work with!) will probably want to display figures in your own web tools, rather than always our (fairly basic, right now) viewer utility.

The browser viewer is built in React (using `react-plotly.js`). It is also open-source, so you can use its components as a basis for your own. See https://github.com/thclark/cpplot-viewer

1.2 Installation

1.2.1 Upgrading From Previous Versions

Please see the note on version stability and backward compatibility *here*.

1.2.2 Installation With Cmake

Attention: We're working on getting cpplot added as a package in conan-central. Follow the work in progress here. Until we do that, installation is via cmake.

A cross-platform compilation file is provided using cmake, it's **not tested on windows** but might actually work, since it uses conan to handle all the tricky dependencies.

```
git clone https://github.com/thclark/cpplot
cd cpplot && mkdir -p build && cmake . -B build
```

This process will die first time around, and you'll get a message like:

```
-- Adding dependencies with conan. Make sure you've called `conan install . --install-
↪folder /Users/you/cpplot/cmake-build-debug`
CMake Error at CMakeLists.txt:16 (include):
  include could not find load file:
    /Users/you/cpplot/cmake-build-debug/conanbuildinfo.cmake
```

It'll give you the conan install command to install the third party dependencies. Enter that, run it and re-run cmake.

If you don't have conan installed, its pretty easy to get started.

1.2.3 Third Party Dependencies

The dependencies we use are as follows:

Eigen provides a linear algebra library. It isn't as consistent with MATLAB's API as armadillo (an alternative), but the API is very understandable and is used extensively in the C++ community.

glog google's asynchronous logging library, used for logging to file.

googletest for unit testing.

cpr will probably disappear soon as it's a dependency of some *old work with plotly online*.

json is a fast and popular json library for C++. Whoever nlohmann is, they rock.

boost is a library of general utilities. cpplot uses boost stacktrace to provide more helpful errors and boost filesystem utilities to ease the pain of reading, writing and handling files.

tbb is intel's threading library which allows us to write files out of the main thread, preventing us from blocking execution.

1.3 About

1.3.1 Why?

For most engineering and scientific applications that come across my desk, it's a case of "prototype in MATLAB or Python, rewrite in C++". But:

- I'm long past bothering with MATLAB (there's no credible CI/CD options, no sensible test framework, the slug size of the runtime library is 3.7Gb with a huge memory requirement making it undeployable on any sensible server, the license costs would make Bezos flinch, mex interfaces are grossly unwieldy, etc etc etc etc).
- I'm getting really really sick of writing the damn code twice.
- Most of my engineering and science apps are now getting deployed to cloud services. So displaying natively generated figures like from matplotlib in-browser is a nightmare.

What I need is a solution where I can prototype straight into a language which is fast, capable and has a pretty good ecosystem of third party libraries to help me.

Now, with:

- the great improvements to C++ in the C++11 and 14 standards (which make it much easier to handle threading and generally connect APIs of various libraries together),
- availability of libraries like Eigen, cpr, ceres-solver, Intel MKL (I could go on and on),
- great utilities like googletest and Cmake to remove all the old-skool build and test agony...

C++ is really getting there. The one thing missing is a plotting library, which allows me to:

- quickly and easily prototype new algorithms (visually diagnose what's wrong etc etc)
- display analysis results on the web
- produce the same figures on different platforms
- save publication quality figures easily

In all the other languages I use, I now use `plotly` to do those things. So now it's here for C++ too.

1.3.2 About Plotly

cpplot is based on the plot.ly json schema for figures and charts.

Figure json schema

At the core of plotly is a set of open specifications for JSON data which can be rendered into various figure types. The plotly team have hung a lot of services and offerings around it, but basically everything boils down to a json schema. **cpplot** helps generate JSON which is compliant with that schema.

Plotly Offline

`Plotly.js` is an excellent javascript library for rendering figures, which can be used offline or in any third party app. This is what's used by the browser viewer.

Plotly Online

Plot.ly have a range of online services, and anything produced by **cpplot** will be importable there, since it's compliant with their schema.

I'm not a marketing person for plotly, so how to do that is out of scope here (check their docs)!!

I have some code using the Plotly API (see online.h), which might be the very early beginning of a client SDK for C++, enabling direct manipulation of online data. However, I have very little appetite for this (I personally don't use plotly online, because I've found the support to be appalling in the past) but collaborators committed to developing and maintaining that would be welcome. File an issue if you're interested.

1.4 License

1.4.1 The boring bit

Your license here.

1.4.2 Third Party Libraries

cpplot includes or is linked against the following third party libraries:

Stuff here.

1.5 Library API

1.5.1 Class Hierarchy

1.5.2 File Hierarchy

1.5.3 Full API

Namespaces

Namespace cpplot

Contents

Classes

- *Class Axis*
- *Class BarPlot*
- *Class Figure*
- *Class Layout*
- *Class Line*
- *Class ScatterPlot*
- *Class SurfacePlot*

Enums

- *Enum AxisDirection*
- *Enum ColorScale*
- *Enum PlotType*

Functions

- *Template Function cpplot::to_json(nlohmann::json&, const Eigen::DenseBase<Derived>&)*
- *Function cpplot::to_json(nlohmann::json&, const Axis&)*
- *Function cpplot::to_json(nlohmann::json&, const Layout&)*
- *Function cpplot::to_json(nlohmann::json&, const BarPlot&)*
- *Function cpplot::to_json(nlohmann::json&, const Line&)*
- *Function cpplot::to_json(nlohmann::json&, const ScatterPlot&)*
- *Function cpplot::to_json(nlohmann::json&, const ColorScale&)*
- *Function cpplot::to_json(nlohmann::json&, const SurfacePlot&)*

Variables

- *Variable cpplot::colour_names*

Namespace google

Namespace std

Classes and Structs

Struct ErrorInPlotlyOnlineException

- Defined in *File exceptions.h*

Inheritance Relationships

Base Type

- `public exception`

Struct Documentation

`struct ErrorInPlotlyOnlineException` : **`public`** exception

Public Functions

`const` char ***`what`**`()` **`const`**

`ErrorInPlotlyOnlineException``()`

`ErrorInPlotlyOnlineException``(`**`const`** std::string *msg*`)`

Public Members

std::string **`message`** = ""

Struct InvalidAxisException

- Defined in *File exceptions.h*

Inheritance Relationships

Base Type

- `public exception`

Struct Documentation

`struct InvalidAxisException` : **`public`** exception

Public Functions

`const` char ***`what`**`()` **`const`**

Public Members

std::string **`message`** = "Axis invalid or not present"

Struct InvalidOrMissingPlotlyCredentialsException

- Defined in *File exceptions.h*

Inheritance Relationships

Base Type

- `public exception`

Struct Documentation

`struct InvalidOrMissingPlotlyCredentialsException` : **`public`** exception

Public Functions

`const` char ***`what`**`() const`

Public Members

std::string **`message`** = "Invalid or missing plotly credentials. Try setting the environment variables PLOTLY_USERNAME and

Struct NotImplementedException

- Defined in *File exceptions.h*

Inheritance Relationships

Base Type

- `public exception`

Struct Documentation

`struct NotImplementedException` : **`public`** exception

Public Functions

`const` char ***`what`**`() const`

Public Members

std::string **`message`** = "Not yet implemented"

Class Axis

- Defined in *File layout.h*

Class Documentation

class Axis

Axis class for use in layout of plots.

Serialises to something like:

```
"xaxis": {"title": "x1"}
```

Public Functions

Axis(**const** *AxisDirection dir*)

void **setDirection**(*AxisDirection dir*)

Sets the direction of the axis, as X, Y, or Z.

Parameters

- `dir`: *Axis* direction (an enum X, Y or Z)

void **setTitle**(std::string *label*)

void **setLog**()

AxisDirection **getDirection**()

std::string **getTitle**()

bool **isLog**()

Protected Attributes

AxisDirection **direction**

std::string **title**

std::string **key**

bool **is_log** = false

Friends

void **to_json**(nlohmann::json &*j*, **const** Axis &*p*)

Serialise axis into a valid json string.

Produces json like:

```
"xaxis": {"title": "x1"}
```

Class BarPlot

- Defined in *File bar.h*

Class Documentation

class BarPlot

Public Functions

BarPlot()
Construct with default basic data (for test plots)

Public Members

std::vector<std::string> **x**

Eigen::VectorXd **y**

std::string **type** = "bar"

std::string **name** = ""

Class Figure

- Defined in *File cpplot.h*

Class Documentation

class Figure

Public Functions

Figure()

template <**class** T>
void **add**(T &*plot*)
Add plot data to the figure.

Template Parameters

- `T`: type of the plot data to add

Parameters

- `plot`:

void **setLayout**(*Layout* &*lay*)
Add layout data to the figure.

void **write**(std::string *file_name*, bool *append_extension* = true, bool *print_to_stdout* = false)
Write the figure to a file.

CAUTION: Overwrites any existing file contents.

Parameters

- `file_name`: The file name to write to, including any absolute or relative path
- `append_extension`: If true, a .json extension will be appended to file_name (if not already present)
- `print_to_stdout`: Default false. If true, the json will also be printed to std::cout (useful for debugging)

Public Members

std::string **id** = ""
Optional metadata - *Figure* ID.

std::string **name** = ""
Optional metadata - *Figure* name used for display (e.g. in cpplot-viewer)

std::string **short_caption** = ""
Optional metadata - Short_caption used in report contents tables.

std::string **caption** = ""
Optional metadata - Long caption used beside the figure (e.g. in a report)

Protected Attributes

nlohmann::json **data**

nlohmann::json **layout**

nlohmann::json **meta**

Class Layout

- Defined in *File layout.h*

Class Documentation

class Layout

Public Functions

Layout(**const** std::string *title* = "", **const** bool *is_scene* = false)
Construct with default basic data (for test plots)

Example use:

```
Layout my_layout = Layout("a graph title"); // Default constructor Layout()␣
↪also works
my_layout.xLabel("ecks");
my_layout.yLabel("why";
```

Axis ***getAxis**(**const** *AxisDirection* &*dir*, bool *create* = false)
get an axis in the layout by its direction. Optionally, create if not found.

Example use:

```
Layout my_layout = Layout("a graph title"); // Default constructor Layout()␣
↪also works
axis = my_layout.getAxis(X); // raises error
axis = my_layout.getAxis(X, true); // creates the axis and adds it to the␣
↪layout
```

Return

Parameters

- `dir`:

void **xTitle**(**const** std::string *label*)
set the title of the x axis

Parameters

- `label`: the title of the axis. Can use latex within dollar signs, like "Normalised distance \$\eta\$"

void **yTitle**(**const** std::string *label*)
set the title of the y axis

Parameters

- `label`: the title of the axis. Can use latex within dollar signs, like "Normalised distance \$\eta\$"

void **zTitle**(**const** std::string *label*)
set the title of the z axis

Parameters

- `label`: the title of the axis. Can use latex within dollar signs, like "Normalised distance \$\eta\$"

void **xLog**()
change the type of the x axis from its default, 'linear', to 'log'

void **yLog**()
change the type of the y axis from its default, 'linear', to 'log'

void **zLog**()
change the type of the z axis from its default, 'linear', to 'log'

Protected Attributes

std::vector<*Axis*> **axes**

std::string **title**

bool **is_scene**

Friends

void **to_json**(nlohmann::json &*j*, **const** Layout &*p*)
Serialise layout into a valid json string.

Class Line

- Defined in *File scatter.h*

Class Documentation

class Line
Class for managing and serialising properties of a line or lines.

Public Functions

Line()
Constructor initialises a default *Line* type.

void **setColor**(**const** std::string &*value*)
set solid line colour as a string.

TODO Implement colour scaled values

See https://plot.ly/ipython-notebooks/color-scales/#6-colors for an example of different colour scales and values available

For reference, plotly's default colours are: ['#1f77b4', # muted blue '#ff7f0e', # safety orange '#2ca02c', # cooked asparagus green '#d62728', # brick red '#9467bd', # muted purple '#8c564b', # chestnut brown '#e377c2', # raspberry yogurt pink '#7f7f7f', # middle gray '#bcbd22', # curry yellow-green '#17becf' # blue-teal]

Parameters

- `value`: std::string plotly colour value, e.g. setColor("#1f77b4")

void **setDash**(**const** std::string &*value*)
set the line dash mode.

Accepts "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot" as dash_type input

Parameters

- `value`: std::string The dash type to use

void **setWidth**(**const** int *value*)
set the line width in pixels.

Parameters

- `value`: int the width in pixels for the line

bool **empty**() **const**
return boolean, false if any line properties are changed from default.

Protected Attributes

std::string **dash**

int **width**

std::string **color**

bool **is_empty**

Friends

void **to_json**(nlohmann::json &*j*, **const** Line &*p*)
Serialise line data into a valid json string.

Class ScatterPlot

- Defined in *File scatter.h*

Class Documentation

class ScatterPlot

Public Functions

ScatterPlot()
Construct with default basic data (for test plots).

void **setColor**(**const** std::string &*value*)
set solid line colour as a string.

See https://plot.ly/ipython-notebooks/color-scales/#6-colors for an example of different colour scales and values available See https://github.com/plotly/plotly.py/blob/master/plotly/colors.py#L83-L87 for plotly's master colour reference For reference, plotly's default colours are: ['#1f77b4', # muted blue '#ff7f0e', # safety orange '#2ca02c', # cooked asparagus green '#d62728', # brick red '#9467bd', # muted purple '#8c564b', # chestnut brown '#e377c2', # raspberry yogurt pink '#7f7f7f', # middle gray '#bcbd22', # curry yellow-green '#17becf' # blue-teal]

Parameters

- `value`: std::string plotly colour value, e.g. setColor("#1f77b4")

void **setDash**(**const** std::string &*value*)
set the line dash mode for this scatter plot.

This is a helper function, which directly modifies a 'line' object, allowing a clean API for the user to update scatter plots.

Accepts "solid", "dot", "dash", "longdash", "dashdot", or "longdashdot"

Parameters

- `dash_type`: std::string selected dash type

void **setWidth**(**const** int *value*)
set the line width in pixels.

Parameters

- `value`: int the width in pixels for the line

Public Members

Eigen::VectorXd **x**

Eigen::VectorXd **y**

std::string **name** = ""

Protected Attributes

Line **line**

Friends

void **to_json**(nlohmann::json &*j*, **const** ScatterPlot &*p*)
Serialise scatter plot data into a valid json string.

Produces figure data JSON similar to: {"x": [24.23, 3.4, 8.4], "y": [20.1, 14.4, 23.3], "type": "scatter"}

Class SurfacePlot

- Defined in *File surface.h*

Class Documentation

class SurfacePlot

Public Functions

SurfacePlot()
Construct with default basic data (for test plots)

Public Members

ColorScale **colorscale** = YlGnBu

Eigen::ArrayXXd **x**

Eigen::ArrayXXd **y**

Eigen::ArrayXXd **z**

std::string **type** = "surface"

std::string **name** = ""

Enums

Enum AxisDirection

- Defined in *File layout.h*

Enum Documentation

enum cpplot::**AxisDirection**

Enum of the three possible axis directions, X, Y, Z.

Values:

X

Y

Z

Enum ColorScale

- Defined in *File surface.h*

Enum Documentation

enum cpplot::**ColorScale**

Default color scales available in plotly TODO refactor these to separate file and allow fo custom colourmaps.

Values:

Blackbody

Bluered

Blues

Earth

Electric

Greens

Greys

Hot

Jet

Picnic

Portland

Rainbow

RdBu

Reds

Viridis

YlGnBu

YYlOrRd

Enum PlotType

- Defined in *File cpplot.h*

Enum Documentation

enum cpplot::**PlotType**

Values:

bar

scatter

Functions

Template Function cpplot::to_json(nlohmann::json&, const Eigen::DenseBase<Derived>&)

- Defined in *File eigen.h*

Function Documentation

template <**typename** Derived>
void cpplot::**to_json**(nlohmann::json &*j*, **const** Eigen::DenseBase<Derived> &*in*)

A serialiser to convert dense eigen arrays to json objects using nlohmann::json.

Turns eigen array into a valid json array. 1D arrays become lists: [24.23, 3.4, 8.4] 2D arrays become lists of lists: [[24.23, 3.4, 8.4],[45.23, 5.4, 9.4]]

TODO manage tensors for 3d volumetric plots

Output is written as a row major array. So (assuming eigen's default settings for orientation), doing:

Eigen::ArrayXXd arr(5,2); arr << 0.0, 0.1, 1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 4.0, 4.1; nlohmann::json j; to_json(j, arr); std::cout << "array element 0,1: " << arr(0,1) << std::endl; std::cout << "array element 1,0: " << arr(1,0) << std::endl; std::cout << j << std::endl;

Prints:

array 0,1: 0.1 array 1,0: 1 [[0,0.1],[1,1.1],[2,2.1],[3,3.1],[4,4.1]]

Template Parameters

- `Derived`: The derived array or matrix type

Parameters

- `j`: An nlohmann::json array
- `in`: The matrix or array to serialise into json array

Function cpplot::to_json(nlohmann::json&, const Axis&)

- Defined in *File layout.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *Axis* &*p*)
Serialise axis into a valid json string.

Produces json like:

```
"xaxis": {"title": "x1"}
```

Function cpplot::to_json(nlohmann::json&, const Layout&)

- Defined in *File layout.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *Layout* &*p*)
Serialise layout into a valid json string.

Function cpplot::to_json(nlohmann::json&, const BarPlot&)

- Defined in *File bar.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *BarPlot* &*p*)
Serialise bar plot data into a valid json string.

Produces figure data JSON similar to: {“x”: [“giraffes”, “orangutans”, “monkeys”], “y”: [20.1, 14.4, 23.3], “type”: “bar”}

Function cpplot::to_json(nlohmann::json&, const Line&)

- Defined in *File scatter.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *Line* &*p*)
Serialise line data into a valid json string.

Function cpplot::to_json(nlohmann::json&, const ScatterPlot&)

- Defined in *File scatter.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *ScatterPlot* &*p*)
Serialise scatter plot data into a valid json string.

Produces figure data JSON similar to: {“x”: [24.23, 3.4, 8.4], “y”: [20.1, 14.4, 23.3], “type”: “scatter”}

Function cpplot::to_json(nlohmann::json&, const ColorScale&)

- Defined in *File surface.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *ColorScale* &*c*)
Serialise color scale data into a valid json field or fields.

Produces figure data JSON similar to:

Function cpplot::to_json(nlohmann::json&, const SurfacePlot&)

- Defined in *File surface.h*

Function Documentation

void `cpplot::to_json`(nlohmann::json &*j*, **const** *SurfacePlot* &*p*)
Serialise surface plot data into a valid json string.

Produces figure data JSON similar to:

Function main

- Defined in *File main.cpp*

Function Documentation

Warning: doxygenfunction: Cannot find function “main” in doxygen xml output for project “My Project” from directory: ./doxyoutput/xml

Variables

Variable cpplot::colour_names

- Defined in *File surface.h*

Variable Documentation

“Blackbody”, “Bluered”, “Blues”, “Earth”, “Electric”, “Greens”, “Greys”, “Hot”, “Jet”, “Picnic”, “Portland”, “Rainbow”, “RdBu”, “Reds”, “Viridis”, “YlGnBu”, “YlOrRd” }]

Directories

Directory source

Directory path: `source`

Subdirectories

- *Directory types*

Files

- *File cpplot.cpp*
- *File cpplot.h*
- *File eigen.h*
- *File exceptions.h*
- *File layout.h*
- *File main.cpp*
- *File online.h*

Directory types

Parent directory (`source`)

Directory path: `source/plot_types`

Files

- *File bar.cpp*
- *File bar.h*
- *File scatter.cpp*
- *File scatter.h*
- *File surface.h*

Files

File bar.cpp

Parent directory (`source/plot_types`)

Contents

Definition (`source/plot_types/bar.cpp`)

Program Listing for File bar.cpp

Return to documentation for file (`source/plot_types/bar.cpp`)

```
/*
 * bar.cpp Implementation of BarPlot class methods and operators
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

//#include <json/single_include/nlohmann/json.hpp>
//#include "bar.h"
//#include "figures.h"
//
//
//using nlohmann::json;
```

(continues on next page)

(continued from previous page)

```
namespace cpplot {


} // end namespace
```

Namespaces

- *Namespace cpplot*

File bar.h

Parent directory (`source/plot_types`)

Contents

Definition (`source/plot_types/bar.h`)

Program Listing for File bar.h

Return to documentation for file (`source/plot_types/bar.h`)

```
/*
 * bar.h Bar chart data class
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */
```

(continues on next page)

(continued from previous page)

```
#ifndef CPPLOT_BAR_H
#define CPPLOT_BAR_H

#include <exception>
#include <math.h>
#include <stdio.h>
#include <vector>
#include <string.h>
#include <Eigen/Dense>
#include <nlohmann/json.hpp>


using nlohmann::json;


namespace cpplot {


class BarPlot {
public:

    std::vector<std::string> x;
    Eigen::VectorXd y;
    std::string type = "bar";
    std::string name = "";

    BarPlot() {
        x = {"cppgiraffes", "orangutans", "monkeys"};
        y = Eigen::VectorXd::LinSpaced(3, 2.0, 3.0);
    }

};


void to_json(nlohmann::json& j, const BarPlot& p) {

    nlohmann::json y;
    to_json(y, p.y);
    j["x"] = p.x;
    j["y"] = y;
    j["type"] = p.type;
    if (!p.name.empty()) {
        j["name"] = p.name;
    }

}


} // end namespace

#endif //CPPLOT_BAR_H
```

Includes

- Eigen/Dense

- exception (*File exceptions.h*)
- math.h
- nlohmann/json.hpp
- stdio.h
- string.h
- vector

Included By

- *File cpplot.h*

Namespaces

- *Namespace cpplot*

Classes

- *Class BarPlot*

Functions

- *Function cpplot::to_json(nlohmann::json&, const BarPlot&)*

File cpplot.cpp

Parent directory (source)

Contents

Definition (source/cpplot.cpp)

Program Listing for File cpplot.cpp

Return to documentation for file (source/cpplot.cpp)

```
/*
 * figures.cpp Implementation of Figure and associated classes for plotting with
↪Plotly from C++
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */


#include "cpplot.h"


namespace cpplot {


} // end namespace
```

Includes

- `cpplot.h` (*File cpplot.h*)

Namespaces

- *Namespace cpplot*

File cpplot.h

Parent directory (`source`)

Contents

Definition (source/cpplot.h)

Program Listing for File cpplot.h

Return to documentation for file (source/cpplot.h)

```
/*
 * figures.h Allows flexible plotting in c++ using Plotly
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

#ifndef CPPLOT_FIGURES_H
#define CPPLOT_FIGURES_H

#include <exception>
#include <math.h>
#include <stdio.h>
#include <vector>
#include <string.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <bitset>
#include <Eigen/Dense>
#include <boost/algorithm/string.hpp>
#include <boost/any.hpp>
#include <nlohmann/json.hpp>
#include "exceptions.h"
#include "eigen.h"
#include "layout.h"
#include "plot_types/bar.h"
#include "plot_types/scatter.h"
#include "plot_types/surface.h"


namespace cpplot {

    // Enumerate the different plot types that are possible with plotly
    enum PlotType { bar, scatter };

    // Base class for managing the figure creation process
    class Figure {

    protected:
```

(continues on next page)

(continued from previous page)

```
        nlohmann::json data;
        nlohmann::json layout;
        nlohmann::json meta;

    public:

        Figure(){
            data = nlohmann::json::array();
            layout = nlohmann::json::object();
            meta = nlohmann::json::object();
        }

        std::string id = "";

        std::string name = "";

        std::string short_caption = "";

        std::string caption = "";

        template <class T>
        void add(T &plot) {
            // Data is a json array of plot data
            data.push_back(plot);
        }

        void setLayout(Layout &lay) {
            layout.update(lay);
        }

        void write(std::string file_name, bool append_extension = true, bool print_to_
↪stdout = false){

            // Compile metadata into a json object. NB any other fields already added␣
↪to meta will be kept, allowing addition of arbitrary metadata to figures.
            meta["id"] = id;
            meta["name"] = name;
            meta["caption"] = caption;
            meta["short_caption"] = short_caption;

            // Convert the figure to JSON: copy each value into the JSON object
            nlohmann::json j;
            j["data"] = data;
            j["layout"] = layout;
            j["meta"] = meta;

            // Get the file name
            if (!boost::algorithm::ends_with(file_name, ".json") && append_extension)
↪{
                file_name.append(".json");
            }

            // Open the file and stream the string into it, overwriting any existing␣
↪contents
            std::ofstream file;
            file.open(file_name);
            file << j;
```

(continues on next page)

(continued from previous page)

```
            file.close();

            // Also write to stdout
            if (print_to_stdout) {
                std::cout << "Figure json:" << std::endl << j << std::endl;
            }

        }
        // TODO Represent Figure class in ostream
        /*
        ::std::ostream& operator<<(::std::ostream& os, const Figure& fig) {
            // Represent in logs or ostream
            return os << "debug statement for figure class";
        }
        */

    };

    // TODO Represent Figure class in ostream
    /*
    ::std::ostream& operator<<(::std::ostream& os, const Figure& fig) {
        // Represent in logs or ostream
        return os << "debug statement for figure class";
    }
    */


} // end namespace

#endif // CPPLOT_FIGURES_H
```

Includes

- `Eigen/Dense`
- `bitset`
- `boost/algorithm/string.hpp`
- `boost/any.hpp`
- `eigen.h` (*File eigen.h*)
- `exception` (*File exceptions.h*)
- `exceptions.h` (*File exceptions.h*)
- `fstream`
- `iostream`
- `layout.h` (*File layout.h*)
- `math.h`
- `nlohmann/json.hpp`
- `plot_types/bar.h` (*File bar.h*)
- `plot_types/scatter.h` (*File scatter.h*)

- plot_types/surface.h (*File surface.h*)
- stdio.h
- string.h
- time.h
- vector

Included By

- *File cpplot.cpp*

Namespaces

- *Namespace cpplot*

Classes

- *Class Figure*

Enums

- *Enum PlotType*

File eigen.h

Parent directory (source)

Contents

Definition (source/eigen.h)

Program Listing for File eigen.h

Return to documentation for file (source/eigen.h)

```cpp
/*
 * eigen.h Serializers for dense matrices/arrays/vectors from Eigen.
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

#ifndef CPPLOT_EIGEN_H
#define CPPLOT_EIGEN_H

#include <Eigen/Dense>


namespace cpplot {


template<typename Derived>
void to_json(nlohmann::json& j, const Eigen::DenseBase<Derived>& in) {

    // Use the eigen matrix formatting code to stream the matrix to a string.
    // This'll likely be faster than anything I can do here.
    Eigen::IOFormat vector_format(Eigen::FullPrecision, Eigen::DontAlignCols, ", ", ",
↪ ", "[", "]");
    Eigen::IOFormat matrix_format(Eigen::FullPrecision, Eigen::DontAlignCols, ", ", ",
↪ ", "[", "]", "[", "]");

    // Stream the eigen matrix, vector or array
    std::stringstream value_stream;
    if (in.cols() == 1 && in.rows() > 0) {
        value_stream << in.transpose().format(vector_format);
    } else if (in.rows() == 1) {
        value_stream << in.format(vector_format);
    } else {
        value_stream << in.format(matrix_format);
    }

    // Explicitly add the string as the json object
    // TODO - reparsing this string object is pretty inefficient. Maybe best to write␣
↪my own after all
    j = nlohmann::json::parse(value_stream.str());

}


} // end namespace

#endif //CPPLOT_EIGEN_H
```

Includes

- `Eigen/Dense`

Included By

- *File cpplot.h*
- *File scatter.h*
- *File surface.h*

Namespaces

- *Namespace cpplot*

Functions

- *Template Function cpplot::to_json(nlohmann::json&, const Eigen::DenseBase<Derived>&)*

File exceptions.h

Parent directory (`source`)

Contents

Definition (`source/exceptions.h`)

Program Listing for File exceptions.h

Return to documentation for file (`source/exceptions.h`)

```
/*
 * exceptions.h Customised exceptions for appropriate and fine grained error handling
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

#ifndef CPPLOT_EXCEPTIONS_H
```

(continues on next page)

(continued from previous page)

```
#define CPPLOT_EXCEPTIONS_H

#include <exception>


struct NotImplementedException : public std::exception {
    std::string message = "Not yet implemented";
    const char * what () const throw () {
        return message.c_str();
    }
};


struct InvalidAxisException : public std::exception {
    std::string message = "Axis invalid or not present";
    const char * what () const throw () {
        return message.c_str();
    }

};


struct InvalidOrMissingPlotlyCredentialsException : public std::exception {
    std::string message = "Invalid or missing plotly credentials. Try setting the␣
↪environment variables PLOTLY_USERNAME and PLOTLY_PASSWORD.";
    const char * what () const throw () {
        return message.c_str();
    }
};


struct ErrorInPlotlyOnlineException : public std::exception {
    std::string message = "";
    const char * what () const throw () {
        return message.c_str();
    }
    ErrorInPlotlyOnlineException() {
        message = "Error in plotly online";
    }
    ErrorInPlotlyOnlineException(const std::string msg) {
        message = msg;
    }
};

#endif // CPPLOT_EXCEPTIONS_H
```

Includes

- `exception` (*File exceptions.h*)

Included By

- *File cpplot.h*
- *File scatter.h*

- *File surface.h*

Classes

- *Struct ErrorInPlotlyOnlineException*
- *Struct InvalidAxisException*
- *Struct InvalidOrMissingPlotlyCredentialsException*
- *Struct NotImplementedException*

File layout.h

Parent directory (`source`)

Contents

Definition (`source/layout.h`)

Program Listing for File layout.h

Return to documentation for file (`source/layout.h`)

```
/*
 * layout.h
 *
 * Author:              Tom Clark  (thclark @ github)
 *
 * Copyright (c) 2019 Octue Ltd. All Rights Reserved.
 *
 */

#ifndef CPPLOT_LAYOUT_H
#define CPPLOT_LAYOUT_H

#include <exception>
#include <math.h>
#include <stdio.h>
#include <vector>
#include <string.h>
```

(continues on next page)

(continued from previous page)

```
#include <Eigen/Dense>
#include <nlohmann/json.hpp>


using nlohmann::json;


namespace cpplot {


enum AxisDirection { X, Y, Z};

class Axis {
public:
    // Allow the serialiser function to access protected members
    friend void to_json(nlohmann::json& j, const Axis& p);

    explicit Axis(const AxisDirection dir) : direction(dir){
        switch(dir){
            case X:
                key = "xaxis";
                break;
            case Y:
                key = "yaxis";
                break;
            case Z:
                key = "zaxis";
                break;
        }
    };

    void setDirection(AxisDirection dir) {
        direction = dir;
    };

    void setTitle(std::string label) {
        title = label;
    };

    void setLog() {
        is_log = true;
    };

    AxisDirection getDirection() {
        return direction;
    };

    std::string getTitle() {
        return title;
    };

    bool isLog() {
        return is_log;
    };

protected:
    AxisDirection direction;
```

(continues on next page)

(continued from previous page)

```
    std::string title;
    std::string key;
    bool is_log = false;
};


void to_json(nlohmann::json& j, const Axis& p) {
    nlohmann::json axis;
    axis["title"] = p.title;
    if (p.is_log) {
        axis["type"] = "log";
    };
    j[p.key] = axis;
}


class Layout {
public:

    // Allow the serialiser function to access protected members
    friend void to_json(nlohmann::json& j, const Layout& p);

    explicit Layout(const std::string title="", const bool is_scene=false) :␣
↪title(title), is_scene(is_scene) {};

    Axis* getAxis(const AxisDirection &dir, bool create=false) {
        for (auto &axis: axes) {
            if (axis.getDirection() == dir) {
                return &axis;
            };
        };
        if (create) {
            Axis ax(dir);
            axes.push_back(ax);
            // If a Z axis is created, turn the plot into a 3D scene
            if (dir == Z) {
                is_scene = true;
            }
            return &axes.back();
        } else {
            InvalidAxisException e;
            throw (e);
        };
    }

    void xTitle(const std::string label) {
        AxisDirection dir = X;
        getAxis(dir, true)->setTitle(label);
    }

    void yTitle(const std::string label) {
        AxisDirection dir = Y;
        getAxis(dir, true)->setTitle(label);
    }

    void zTitle(const std::string label) {
        AxisDirection dir = Z;
```

(continues on next page)

(continued from previous page)

```
        getAxis(dir, true)->setTitle(label);
    }

    void xLog() {
        AxisDirection dir = X;
        getAxis(dir, true)->setLog();
    }

    void yLog() {
        AxisDirection dir = Y;
        getAxis(dir, true)->setLog();
    }

    void zLog() {
        AxisDirection dir = Z;
        getAxis(dir, true)->setLog();
    }

protected:

    std::vector<Axis> axes;
    std::string title;
    bool is_scene;

};


void to_json(nlohmann::json& j, const Layout& p) {

    if (!p.title.empty()) {
        j["title"] = p.title;
    };
    nlohmann::json axes;
    for (auto &axis : p.axes) {
        nlohmann::json ax;
        to_json(ax, axis);
        axes.update(ax);
    };
    if (p.is_scene) {
        j["scene"] = axes;
    } else {
        j.update(axes);
    };
};


}; // end namespace cpplot

#endif // CPPLOT_LAYOUT_H
```

Includes

- `Eigen/Dense`
- `exception` (*File exceptions.h*)

- `math.h`
- `nlohmann/json.hpp`
- `stdio.h`
- `string.h`
- `vector`

Included By

- *File cpplot.h*

Namespaces

- *Namespace cpplot*

Classes

- *Class Axis*
- *Class Layout*

Enums

- *Enum AxisDirection*

Functions

- *Function cpplot::to_json(nlohmann::json&, const Layout&)*
- *Function cpplot::to_json(nlohmann::json&, const Axis&)*

File main.cpp

Parent directory (`source`)

Contents

Definition (source/main.cpp)

Program Listing for File main.cpp

Return to documentation for file (source/main.cpp)

```
/*
 * main.cpp - An example program to create and save various types of figures as␣
↪Plotly-compliant .json files.
 *
 * Author:               Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-9 T Clark. All Rights Reserved.
 *
 */
#include <stdlib.h>
#include "glog/logging.h"


using namespace std;
using namespace google;


int main(int argc, char* argv[]) {

    google::InitGoogleLogging(argv[0]);

    /*
    values   << 0.01, 0.02, 0.05, 0.08, 0.01,
                0.02, 0.05, 0.46, 0.25, 0.75,
                0.587, 0.5, 0.12, 0.99, 0.01,
                0.01, 0.02, 0.05, 0.08, 0.01,
                0.02, 0.05, 0.46, 0.25, 0.75,
                0.587, 0.5, 0.12, 0.99, 0.01;
     */

}
```

Includes

- glog/logging.h
- stdlib.h

Namespaces

- *Namespace google*
- *Namespace std*

Functions

- *Function main*

File online.h

Parent directory (`source`)

Contents

Definition (`source/online.h`)

Program Listing for File online.h

Return to documentation for file (`source/online.h`)

```
/*
 * online.h TODO - this is a collection of snippets from first working with online␣
↪plotly API.
 * They could be used to form the basis of a plotly client in C++
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

#ifndef CPPLOT_ONLINE_H
#define CPPLOT_ONLINE_H

// Encapsulate plotly config variables. Obtain credentials from the environment upon␣
↪instantiation.
/* Leaving as a comment - focussing on offline first.
struct Config {

    std::string url = "https://api.plot.ly/v2/";
    std::string user = "";
    std::string password = "";

    Config() {
        // Get configuration from environment variables. PLOTLY_URL is optional,␣
↪allowing you to run your own server
        char *c_url = getenv("PLOTLY_URL");
        char *c_user = getenv("PLOTLY_USERNAME");
        char *c_password = getenv("PLOTLY_API_KEY");
        if ((c_user == NULL) || (c_password == NULL)) {
            InvalidOrMissingPlotlyCredentialsException e;
```

(continues on next page)

(continued from previous page)

```
            throw (e);
        }
        if (c_url != NULL) {
            url = std::string(c_url);
        }
        user = std::string(c_user);
        password = std::string(c_password);
        if (url.back() == '/') {
            url.pop_back();
        }
    }
};

// Base class for managing the figure creation process
template <class ContentsType>
class Figure {

private:

    ContentsType contents;

    Config configuration;

     void setPlotlyIdFromUrl(std::string url);

protected:

    // Plotly figure id
    std::string plotly_id;

    // Plotly username
    std::string plotly_usr;

    // Refers to the python plotly library used to generate the figure
    // TODO figure out how to get the version of plotly used to render the figure␣
↪through their API; set it here
    std::string plotly_version = "unknown";

public:

    Figure(ContentsType contents);

    // Gets the plotly URL string
    std::string plotlyUrl();

    // Gets the plotly embed URL string
    std::string embedUrl();

    // Not implemented, as we don't use C++ to serve web content directly
    // html(self, **kwargs):

    // Gets the plotly PDF URL string (an expiring link to the pdf file containing␣
↪the figure)
    std::string pdfUrl();

    // Update the figure json from the plotly server
    // Not implemented - one-way so far
```

(continues on next page)

(continued from previous page)

```
    // void pull();

    // Plot the figure by POSTing its data to the plotly server
    void plot();

};


template <class ContentsType>
Figure<ContentsType>::Figure(ContentsType contents) {

    contents = contents;

    // Get plotly server access configuration and credentials from the environment
    configuration = Config();

}

template <class ContentsType>
std::string Figure<ContentsType>::plotlyUrl() {
    // Returns the URL to this figure on plotly, if the figure is rendered
    if (plotly_usr == "") {
        return "";
    }
    return configuration.url + "/~" + plotly_usr + "/" + plotly_id;
}

template <class ContentsType>
void Figure<ContentsType>::setPlotlyIdFromUrl(std::string url) {

    // Remove any trailing / in case the url is given as, e.g. https://plot.ly/~usr_
↪name/123/
    if (url.back() == '/') {
        url.pop_back();
    }

    // Split the URL into its components. Use ~ as a splitter as well as / to avoid␣
↪extra op of removing it from user
    std::vector<std::string> url_parts;
    boost::split(url_parts, url, boost::is_any_of("/~"));

    // If the url is given as, e.g. plot.ly/~usr_name/123.embed, remove the .* from␣
↪the trailing figure id
    std::vector<std::string> trailing_id_parts;
    boost::split(trailing_id_parts, url_parts.back(), boost::is_any_of("."));

    // Assign the Figure properties from the url
    plotly_id = trailing_id_parts[0];
    plotly_usr = url_parts[url_parts.size() - 2];

}

template <class ContentsType>
std::string Figure<ContentsType>::embedUrl() {
    return plotlyUrl() + ".embed";
}
```

(continues on next page)

(continued from previous page)

```
template <class ContentsType>
std::string Figure<ContentsType>::pdfUrl() {
    return plotlyUrl() + ".pdf";
}

template <class ContentsType>
void Figure<ContentsType>::plot() {
    // Plot the figure by POSTing its data to the plotly server

    // Get the json data for the figure contents
    std::string body_str = contents.toJson();

    // POST to configuration.url, with configured credentials
    auto r = cpr::Post( cpr::Url{configuration.url},
                        cpr::Body{body_str},
                        cpr::Authentication{configuration.user, configuration.
↪password},
                        cpr::Header{{"Accept", "application/json"}, {"Plotly-Client-
↪Platform", "cpp 0.0.1"}});

    // Remove any trailing / in case the url is given as, e.g. https://plot.ly/~usr_
↪name/123/
    if (r.status_code != 200) {
        std::stringstream msg;
        msg << "Error received from plotly..." <<std::endl;
        msg << "url: " << configuration.url << std::endl;
        msg << "status_code: " << r.status_code << std::endl;
        msg << "content-type: " << r.header["content-type"] << std::endl; //␣
↪application/json; charset=utf-8
        msg << "text: " << r.text << std::endl;
        ErrorInPlotlyOnlineException e(msg.str());
        throw (e);
    }

    // Expected response...
    auto json = nlohmann::json::parse(r.text);
    std::cout << "Successful response from plotly API... " << std::endl << json.
↪dump(4) << std::endl;

    NotImplementedException e;
    throw (e);

}

*/
#endif //CPPLOT_ONLINE_H
```

File scatter.cpp

Parent directory (`source/plot_types`)

Contents

- *Definition* (`source/plot_types/scatter.cpp`)

- *Namespaces*

Definition (source/plot_types/scatter.cpp)

Program Listing for File scatter.cpp

Return to documentation for file (source/plot_types/scatter.cpp)

```
/*
 * scatter.cpp Implementation of ScatterPlot methods and operators
 *
 * References:
 *
 *   [1]
 *
 * Future Improvements:
 *
 *   [1]
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-8 T Clark. All Rights Reserved.
 *
 */

//#include <json/single_include/nlohmann/json.hpp>
//#include "scatter.h"
//#include "figures.h"
//
//
//using nlohmann::json;


namespace cpplot {

} // end namespace
```

Namespaces

- *Namespace cpplot*

File scatter.h

Parent directory (source/plot_types)

Contents

- *Definition (source/plot_types/scatter.h)*
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*
- *Functions*

Definition (`source/plot_types/scatter.h`)

Program Listing for File scatter.h

Return to documentation for file (`source/plot_types/scatter.h`)

```
/*
 * scatter.h ScatterPlot data class
 *
 * Author:               Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-9 T Clark. All Rights Reserved.
 *
 */

#ifndef CPPLOT_SCATTER_H
#define CPPLOT_SCATTER_H

#include <vector>
#include <string.h>
#include <Eigen/Dense>
#include <nlohmann/json.hpp>

#include "eigen.h"
#include "exceptions.h"


namespace cpplot {


class Line {
public:
    // Allow the serialiser function to access protected members
    friend void to_json(nlohmann::json& j, const Line& p);

    Line() {
        is_empty = true;
        width = -1; // This is "empty" i.e. not set.
    }

    void setColor(const std::string &value) {
        color = value;
        is_empty = false;
    }

    void setDash(const std::string &value) {
        // TODO check the string is valid, or enumerate.
        dash = value;
```

(continues on next page)

(continued from previous page)

```
        is_empty = false;
    }

    void setWidth(const int value) {
        width = value;
        is_empty = false;
    }

    bool empty() const {
        return is_empty;
    }

protected:
    std::string dash;
    int width;
    std::string color;
    bool is_empty;

};


void to_json(nlohmann::json& j, const Line& p) {
    if (!p.dash.empty()) {
        j["dash"] = p.dash;
    };
    if (p.width != -1) {
        j["width"] = p.width;
    };
    if (!p.color.empty()) {
        j["color"] = p.color;
    }
}


class ScatterPlot {
public:
    // Allow the serialiser function to access protected members
    friend void to_json(nlohmann::json& j, const ScatterPlot& p);

    // TODO move to getter/setters
    Eigen::VectorXd x;
    Eigen::VectorXd y;
    std::string name = "";

    ScatterPlot() {
        x = Eigen::VectorXd::LinSpaced(3, 0.0, 2.0);
        y = Eigen::VectorXd::LinSpaced(3, 1.0, 3.0);
        line = Line();
    }

    void setColor(const std::string &value) {
        line.setColor(value);
    }

    void setDash(const std::string &value) {
        line.setDash(value);
    }
```

(continues on next page)

(continued from previous page)

```
    void setWidth(const int value) {
        line.setWidth(value);
    }

protected:
    Line line;

};


void to_json(nlohmann::json& j, const ScatterPlot& p) {

    nlohmann::json x;
    nlohmann::json y;
    to_json(x, p.x);
    to_json(y, p.y);
    j["x"] = x;
    j["y"] = y;
    j["type"] = "scatter";
    if (!p.name.empty()) {
        j["name"] = p.name;
    }
    if (!p.line.empty()) {
        nlohmann::json line;
        to_json(line, p.line);
        j["line"] = line;
    }
};


} // end namespace

#endif //CPPLOT_SCATTER_H
```

Includes

- Eigen/Dense
- eigen.h (*File eigen.h*)
- exceptions.h (*File exceptions.h*)
- nlohmann/json.hpp
- string.h
- vector

Included By

- *File cpplot.h*

Namespaces

- *Namespace cpplot*

Classes

- *Class Line*
- *Class ScatterPlot*

Functions

- *Function cpplot::to_json(nlohmann::json&, const ScatterPlot&)*
- *Function cpplot::to_json(nlohmann::json&, const Line&)*

File surface.h

Parent directory (`source/plot_types`)

Contents

Definition (`source/plot_types/surface.h`)

Program Listing for File surface.h

Return to documentation for file (`source/plot_types/surface.h`)

```
/*
 * scatter.h ScatterPlot data class
 *
 * Author:              Tom Clark  (thclark@github)
 *
 * Copyright (c) 2017-9 T Clark. All Rights Reserved.
 *
 */
```

(continues on next page)

(continued from previous page)

```
#ifndef CPPLOT_SURFACE_H
#define CPPLOT_SURFACE_H

#include <vector>
#include <string.h>
#include <Eigen/Dense>
#include <nlohmann/json.hpp>

#include "eigen.h"
#include "exceptions.h"


namespace cpplot {

enum ColorScale {
   Blackbody,
   Bluered,
   Blues,
   Earth,
   Electric,
   Greens,
   Greys,
   Hot,
   Jet,
   Picnic,
   Portland,
   Rainbow,
   RdBu,
   Reds,
   Viridis,
   YlGnBu,
   YYlOrRd
};
const std::vector<std::string> colour_names = {
    "Blackbody",
    "Bluered",
    "Blues",
    "Earth",
    "Electric",
    "Greens",
    "Greys",
    "Hot",
    "Jet",
    "Picnic",
    "Portland",
    "Rainbow",
    "RdBu",
    "Reds",
    "Viridis",
    "YlGnBu",
    "YlOrRd"
};

void to_json(nlohmann::json& j, const ColorScale& c) {
    j["colorscale"] = colour_names[c];
}
```

(continues on next page)

(continued from previous page)

```
class SurfacePlot {

public:

    ColorScale colorscale = YlGnBu;
    Eigen::ArrayXXd x;
    Eigen::ArrayXXd y;
    Eigen::ArrayXXd z;
    std::string type = "surface";
    std::string name = "";

    SurfacePlot() {
        x = Eigen::RowVectorXd::LinSpaced(3, -1.5, 0.6).replicate(3,1).array();
        y = Eigen::VectorXd::LinSpaced(3, -1.26, 1.26).replicate(1,3).array();
        z = x.pow(2) + y;
    };

};


void to_json(nlohmann::json& j, const SurfacePlot& p) {

    nlohmann::json x;
    nlohmann::json y;
    nlohmann::json z;
    to_json(x, p.x);
    to_json(y, p.y);
    to_json(z, p.z);
    j["x"] = x;
    j["y"] = y;
    j["z"] = z;
    to_json(j, p.colorscale);
    j["type"] = p.type;
    if (!p.name.empty()) {
        j["name"] = p.name;
    }
}


} // end namespace

#endif //CPPLOT_SURFACE_H
```

Includes

- `Eigen/Dense`
- `eigen.h` (*File eigen.h*)
- `exceptions.h` (*File exceptions.h*)
- `nlohmann/json.hpp`
- `string.h`
- `vector`

Included By

- *File cpplot.h*

Namespaces

- *Namespace cpplot*

Classes

- *Class SurfacePlot*

Enums

- *Enum ColorScale*

Functions

- *Function cpplot::to_json(nlohmann::json&, const SurfacePlot&)*
- *Function cpplot::to_json(nlohmann::json&, const ColorScale&)*

Variables

- *Variable cpplot::colour_names*

1.6 Version History

1.6.1 Origins

This library was hacked together / extracted from other repositories such as the es-flow library due to the need for a plotting library enabling me to deliver results over the web.

As more stars are being added on GitHub (14 at the time of writing, woot woot!) it's now worth properly versioning and maintaining the effort, since it seems like it's useful to more people than just me.

1.6.2 Stability

The library uses semver versioning, like *version x.y.z*.

Theoretically, only *x* increments should break backward compatibility. But, it's still early days as noted in issue 10 and I don't want to end up on version 200.4.5 in the next five minutes...

Let's be pragmatic: whilst still on version *0.0.z* please consider the API as unstable version-to-version (z increments). This is because I (or we - please help!) will be still architecting the library sensibly.

When I break it, I'll at least tell you what's breaking! Check back here as you update.

For *versions* >= *0.x.y* expect *y* increments not to break, breaking changes might occur on *x* increments.

For *versions* >= *x.y.z* where *x* >= *1* expect *x* increments to break backward compatibility.

1.6.3 0.0.1

Initial version applied to the library

New Features

1. Documentation in RestructuredText and hooked for public viewing on readthedocs.
2. Pre-commit hooks added to ensure docs always build (whatever commit you're at) and to apply consistent file formats.
3. Implemented semver versioning system which will be done with GitHub releases.

Backward Incompatible API Changes

1. n/a (Initial release)

Bug Fixes & Minor Changes

1. n/a (Initial Release)

1.6.4 0.0.2

Updated build system to use conan for third party dependencies

New Features

1. Documentation updated with a correct installation method
2. Conan package manager built into cmake for easier dependency management
3. Huge amount of custom cmake code (there to find and cope with third party deps removed)

Backward Incompatible API Changes

1. No API changes
2. Build systems updated; now requires conan.

Bug Fixes & Minor Changes

1. n/a (Initial Release)

1.6.5 0.0.3

Minor fixes to build system and docs

New Features

1. n/a

Backward Incompatible API Changes

1. n/a

Bug Fixes & Minor Changes

1. Fix for building against glog on windows
2. Corrected build instructions in documentation

1.6.6 0.0.4

Removed unused cpr dependency from build system

New Features

1. n/a

Backward Incompatible API Changes

1. n/a

Bug Fixes & Minor Changes

1. Removed unused cpr dependency from build system
2. Updated docs to reflect the removed dep and to close #17
3. Added a branch naming rule to the git pre-commit

Index

C

E

I

N

T