path.hpp

```
#ifndef PATH_HPP
#define PATH_HPP

#include <string>
#include <filesystem>

namespace temp
{
    class Path
    {
    private:
        std::filesystem::path m_path;

    public:
        Path(): m_path(std::filesystem::absolute("."))
        { }
        Path(std::filesystem::path path): m_path(std::filesystem::absolute(path))
        { }
        ~Path() = default;

        friend std::ostream& operator<<(std::ostream& os, const Path& path)
        {
            os << path.m_path;

            return os;
        }

        bool is_dir() const
        {
            return std::filesystem::is_directory(this->m_path);
        }

        bool is_file() const
        {
            return std::filesystem::is_regular_file(this->m_path);
        }

        bool is_empty() const
        {
            if (!this->exists()) return true;

            return std::filesystem::is_empty(this->m_path);
        }

        bool exists() const
        {
            return std::filesystem::exists(this->m_path);
        }

        Path parent() const
        {
            return Path(this->m_path.parent_path());
        }

        std::string file_name() const
        {
            return this->m_path.filename().string();
        }

        std::string extension() const
        {
            return this->m_path.extension().string();
        }

        Path folder() const
        {
            std::filesystem::path output = this->m_path;
            return Path(output.remove_filename());
        }

        std::string to_string() const
        {
            return this->m_path.string();
        }

        uintmax_t file_size() const
        {
            return std::filesystem::file_size(this->m_path);
        }

        Path operator/(const std::string string) const
        {
            return Path(this->to_string() + "/" + string);
        }
    };
}

#endif
```

pathpy.cpp

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include <pybind11/numpy.h>
#include <regex>
#include <temp/path.hpp>
#include <temp/is_content_equal.hpp>

namespace py = pybind11;

void ex_path(py::module_ &m)
{
```

```cpp
m.def("dir", [] (const temp::Path & dir,
                 const std::string & regex)
{
    std::vector<temp::Path> output;

    if (!dir.is_dir())
    {
        throw std::invalid_argument("Path is not a valid directory");
    }

    std::regex pattern(regex);
    std::smatch m;
    std::string file;

    for (const auto& it_file:
        std::filesystem::recursive_directory_iterator(
          std::filesystem::path(dir.to_string())))
    {
        file = std::filesystem::path(it_file).string();

        if (std::regex_search(file, m, pattern))
        {
            output.push_back(temp::Path(std::filesystem::path(it_file)));
        }
    }

    return output;
}, py::arg("dir"), py::arg("pattern"));
m.def("is_content_equal", [] (const temp::Path & l_path,
                              const temp::Path & r_path)
{
    return temp::is_content_equal(l_path, r_path);
});
py::class_<temp::Path>(m, "Path")
    .def(py::init<std::string>(), py::arg("path"))
    .def("is_dir", &temp::Path::is_dir)
    .def("is_file", &temp::Path::is_file)
    .def("is_empty", &temp::Path::is_empty)
    .def("exists", &temp::Path::exists)
    .def("parent", &temp::Path::parent)
    .def("file_name", &temp::Path::file_name)
    .def("extension", &temp::Path::extension)
    .def("folder", &temp::Path::folder)
    .def("to_string", &temp::Path::to_string)
    .def("file_size", &temp::Path::file_size)
    .def("__truediv__", &temp::Path::operator/, py::arg("rhs"))
    .def("__fspath__", &temp::Path::to_string)
    .def("__repr__", [](const temp::Path & object)
    {
        std::ostringstream out;

        out << object;
```

```
            return out.str();
        })
        ;
}
```

is_content_equal.cpp

```
#ifndef IS_CONTENT_EQUAL_HPP
#define IS_CONTENT_EQUAL_HPP

#include "path.hpp"

#include <fstream>
#include <iterator>
#include <algorithm>

namespace temp
{
    bool is_content_equal(const temp::Path l_file, const temp::Path r_file)
    {
        if (!l_file.exists() || !r_file.exists())
        {
            return false;
        }

        if (l_file.file_size() != r_file.file_size())
        {
            return false; // size missmatch
        }

        std::ifstream f1(l_file.to_string(), std::ifstream::binary|std::ifstream::ate);
        std::ifstream f2(r_file.to_string(), std::ifstream::binary|std::ifstream::ate);

        if (f1.fail() | f2.fail())
        {
            return false; // file problem
        }

        f1.seekg(0, std::ifstream::beg);
        f2.seekg(0, std::ifstream::beg);

        return std::equal(std::istreambuf_iterator<char>(f1.rdbuf()),
                          std::istreambuf_iterator<char>(),
                          std::istreambuf_iterator<char>(f2.rdbuf()));
    }
}

#endif
```

recovery

```
mu = min(max(0.01, muSys - volSys * ZminusA), 0.99)
nu
temp = (1 / 9) if (mu * (1 - mu) / nu <= 1) else (mu * (1 - mu) / nu -1)
alpha = mu * temp
beta  = temp - alpha

BetaInv(randu(), alpha, beta)
```

Wavelets

```
library(ggplot2)
scale_function = Vectorize(function(x = x)
{
  if (x >= 0)
  {
    if (x <= 0.5)
    {
      return(1)
    } else if (x <= 1)
    {
      return(-1)
    }
  }

  return(0)
})

wavelet = function(x = x, j = j, k = k)
{
  return(2 ^ (j / 2) * scale_function(2 ^ j * x - k))
}

g = function(x = x)
{
  return(sin(x ^ 2 * 20) + 2)
}

haar_val = Vectorize(function(x = x, j = j, interval = interval)
{
  output = integrate(function(x) {return(ifelse(x < 0 | x > 1, 0, 1) * g(x))},
                     lower = interval[1],
                     upper = interval[2])$value

  for (it_j in 0:j)
  {
    for (it_k in 0:(2 ^ j))
    {
      output = output + (integrate(function(x) {g(x) * wavelet(x, it_j, it_k )},
                                   lower = interval[1],
```

```r
                                   upper = interval[2])$value * wavelet(x, it_j, it_k))
    }
  }

  return(output)
}, vectorize.args = "x")

ggplot() +
  geom_function(fun = g, n = 1e3) +
  geom_function(fun = haar_val, args = list(j = 4, interval = c(0, 1)),
                n = 5e2, colour = "red", size = 0.8) +
  xlim(0, 1) +
  theme_bw()
```