Strata

Release 1.0

Shashwat Sharma, Piero Triverio

Nov 08, 2021

CONTENTS

Strata is a C++ library for computing the multilayer Green's function (MGF) for computational electromagnetics codes based on the boundary element method. The source code can be found on GitHub.

If you are new to Strata, we recommend that you start from the *Basic Usage* section. A series of examples is provided in `strata/test/` for a hands-on introduction to various aspects of Strata, and we strongly recommend using these examples as a starting point in your application.

Installation instructions can be found in the *Building Strata* section.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Building Strata

1.1.1 Prerequisites

- CMake version 3.10.2 or later

1.1.2 Installation

To install Strata, clone the Strata repository and navigate to the root directory of the cloned repository in a terminal. Then run the following commands:

```
mkdir build
cd build
cmake ..
make
```

Important notes

- An internet connection is required during the installation process because some third-party libraries (OpenBLAS and yaml) will be downloaded automatically and installated locally in the directory `build/external/builds`.
- If you already have OpenBLAS installed on your machine, you can save some time by reusing it. In that case, instead of `cmake ..`, run

  ```
  cmake -DWITH_OPENBLAS_INC=location/of/openblas/headers -DWITH_OPENBLAS_LIB=location/
  ↪of/openblas/lib ..
  ```

  where `location/of/openblas/headers` is the path to the directory containing all OpenBLAS-related headers (e.g., `cblas.h` and `lapacke.h`), and `location/of/openblas/lib` is the path to the directory containing the OpenBLAS shared library, usually `libopenblas.so`.
- On macOS, the `cmake` command requires an additional option:

  ```
  cmake -DCMAKE_INSTALL_RPATH_USE_LINK_PATH="ON" ..
  ```

Advanced users may want to consult this for out-of-source builds.

To use Strata within your own application, include the directory `inc`, which contains all the headers, and link to `build/libstrata.so` while building your project.

Strata is designed for UNIX-based operating systems, and has been tested on

- Ubuntu 20.04
- Ubuntu 18.04
- CentOS 7
- macOS Catalina 10.15.6

Note: the documentation is generated automatically during the build process.

1.1.3 Run your first test case

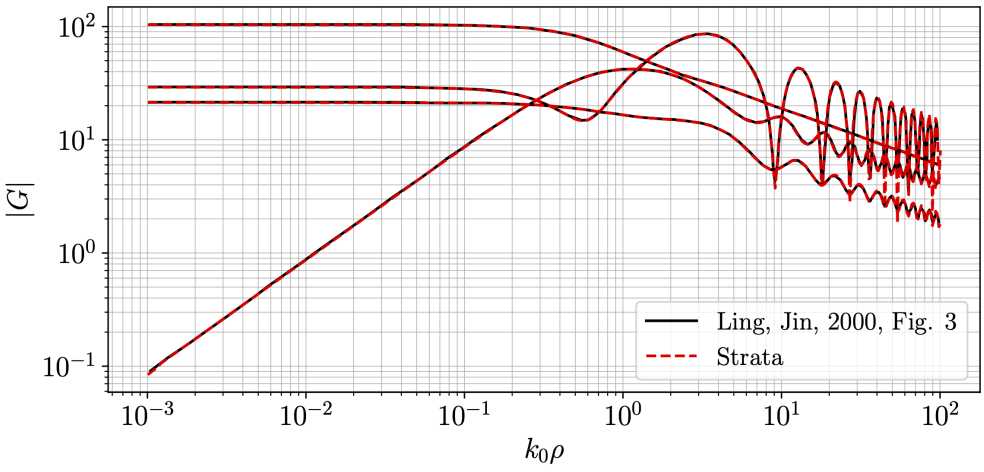
A set of example layer definition files and test cases is provided in the `test` directory. The most basic test case is in `test/testMGF.cpp`, which takes as input the layer definition file (see `doc/strata.html` or `doc/strata.pdf` for details) and a name for the output file. From within the `build` folder, run

```
./testMGF ../test/examples/ling_jin_2000/layers.yaml ../test/examples/ling_jin_2000/
↪MGFdata.txt
```

This will compute the MGF for a simple example and store the reuslts in the file `test/examples/ling_jin_2000/MGFdata.txt`. Now, assuming you have Python 3 installed, you can plot the computed MGF:

```
cd ../test/examples/ling_jin_2000
python3 makeplots.py
```

You should see the following plot:



1.2 Basic Usage

Below is an overview of the three basic steps: defining the layer stackup, initializing the MGF engine, and computing the MGF.

1.2.1 Stackup Definition

The geometric extents and material parameters of each layer are stored in, and managed by, objects of the class `LayerManager`. This object is provided with stackup information through a layer definition text file in the `.yaml` format. The formatting of the layer definition file is discussed here: *Example Technology File*. Once the layer definition file is created, the `LayerManager` object can be initialized as follows:

```
LayerManager lm;
lm.ProcessTechFile(name_of_tech_file);
```

The argument `name_of_tech_file` is of type `std::string` and contains the path to the layer definition file, including the `.yaml` extension.

1.2.2 Initialization

Next, the desired settings are provided to the MGF engine and various precomputations are performed internally. The settings are supplied via the `MGF_settings` data structure. A list of basic settings is provided here: *List of MGF Settings*. The operating frequency `f`, the `LayerManager` object, and the `MGF_settings` object are provided as arguments to the `MGF` class via the `MGF::Initialize` function:

```
double f = 30.0e9;

MGF_settings s;
s.method = MGF_INTEGRATE;

MGF mgf;
mgf.Initialize(f, lm, s);
```

The object `mgf` is now responsible for the actual computation of the MGF. Various MGF computation methods are available; `MGF_INTEGRATE` corresponds to direct Sommerfeld integration. See the examples in `testMGF.cpp`, `testInterp.cpp`, and `testDCIM.cpp` for details on other methods. The initialization step must be performed at every frequency, so it is recommended to keep the `mgf` object local to the frequency loop, and call `MGF::Initialize` at each frequency point.

1.2.3 Computation

The MGF can now be computed for given source and test coordinates, (`x_obs`, `y_obs`, `z_obs`) and (`x_src`, `y_src`, `z_src`), respectively:

```
double x_diff = x_obs - x_src;
double y_diff = y_obs - y_src;

std::array<std::complex<double>, 9> G_dyadic;
std::complex<double> G_phi;
mgf.ComputeMGF(x_diff, y_diff, z_obs, z_src, G_dyadic, G_phi);
```

Strata uses the MGF formulation-C proposed by Michalski and Zheng[1], which involves a dyadic with 9 components (stored in row-major order in `G_dyadic`), and one scalar component (stored in `G_phi`). Some entries of `G_dyadic` are always 0 in formulation-C, but all 9 components are returned for compatibility with other MGF formulations which one may want to implement in the future. Note that the MGF computed above **does include** the cos and sin pre-factors[1].

[1] K. A. Michalski and D. Zheng, "Electromagnetic scattering and radiation by surfaces of arbitrary shape in layered media. I. Theory," in *IEEE Trans. Antennas Propag.*, vol. 38, no. 3, pp. 335-344, March 1990.

1.2.4 References

1.3 List of MGF Settings

1.3.1 Basic Settings

method [{MGF_INTEGRATE, MGF_INTERPOLATE, MGF_DCIM, MGF_QUASISTATIC}] The MGF computation technique[1],[2],[3]. Default: MGF_INTERPOLATE.

extract_quasistatic [{true, false}] Choose whether to extract quasistatic contributions in the spectral domain, and then add them back analytically in the spatial domain[3]. Default: false.

extract_homogeneous [{true, false}] Choose whether to extract the homogeneous material Green's function in the spectral domain, and then add it back in the spatial domain. Default: false.

extract_singularities [{true, false}] Only relevant when either `extract_quasistatic` or `extract_homogeneous` is `true`. Choose whether to extract weakly and strongly singular terms, and return the multiplicative coefficients necessary to later add back the singular terms. See the example `testSingularity.cpp`. Default: *false*.

verbose [{true, false}] Allow messages to be written to `std::cout`. Default: *true*.

1.3.2 References

1.4 Example Technology File

Below is an annotated example of a layer definition file.

```
unit: um # applies to all spatial coordinates, and can be 'm', 'mm', 'um', or 'nm'

dielectric_layers:

    # The list of dielectric layers should include the dielectric fill corresponding to␣
↪metal / via / ground layers.

    # Layer properties can either be specified in this format...
    L1:
        zmin: 0
        h: 50
        epsr: 2.1
        mur: 1
        sigma: 0 # S/m

    # ...or in this format
    L2:
        {zmin: -50, h: 50, epsr: 12.5, mur: 1, sigma: 0}

# The following entries specify the half-spaces above ("top") and below ("bottom") the␣
↪stack-up.
```

(continues on next page)

[1] K. A. Michalski and D. Zheng, "Electromagnetic scattering and radiation by surfaces of arbitrary shape in layered media. I. Theory," in *IEEE Trans. Antennas Propag.*, vol. 38, no. 3, pp. 335-344, March 1990.

[2] M. I. Aksun, "A robust approach for the derivation of closed-form Green's functions," in *IEEE Trans. Microw. Theory Tech.*, vol. 44, no. 5, pp. 651-658, May 1996.

[3] E. Simsek, Q. H. Liu and B. Wei, "Singularity subtraction for evaluation of Green's functions for multilayer media," in *IEEE Trans. Microw. Theory Tech.*, vol. 54, no. 1, pp. 216-225, Jan. 2006.

(continued from previous page)

```
# If an infinite PEC ground plane is desired, set the conductivity "sigma" to a negative␣
↪number, e.g. -1. Usually, this would apply to the bottom halfspace.

top_halfspace:
    {epsr: 1, mur: 1, sigma: 0}

bottom_halfspace:
    {epsr: 1, mur: 1, sigma: -1}
```

1.5 People

Strata originated as part of a PhD project at the Modelics Lab at the University of Toronto, and was created by:

- Shashwat Sharma (Architect & Developer)
- Professor Piero Triverio (Principal Investigator)

1.5.1 Other contributors

CHAPTER

TWO

CITING STRATA

We request that you acknowledge the authors of Strata by citing the following:

```
@INPROCEEDINGS{strata,
        author={S. {Sharma} and P. {Triverio}},
        booktitle={2021 {IEEE} International Symposium on Antennas and Propagation and
↪{USNC-URSI} Radio Science Meeting},
        title={Strata: An Open-Source {C++} Library for Computing {Green's} Functions␣
↪for Layered Media},
        year={2021},
        month={Dec.},
        address = {Singapore}}
```

CHAPTER

THREE

ACKNOWLEDGMENT

- Strata ships with files from the libAmosBessel library by Homer Reid, which provides C++ wrappers for computing Bessel functions with the original fortran routines by T. E. Amos.
- Also included are selected files from the Boost C++ library for numerical integration.