ISSN 1172-496X (Print)
ISSN 1172-4234 (Online)

TECHNICAL REPORTS FROM THE ELECTRONICS GROUP
AT THE UNIVERSITY OF OTAGO

NEC2++: An NEC-2 compatible Numerical Electromagnetics Code

by

Timothy C.A. Molteno

ELECTRONICS TECHNICAL
REPORT No. 2014-3

UNIVERSITY OF OTAGO
DUNEDIN, NEW ZEALAND

Online version has
URL: `http://www.physics.otago.ac.nz/reports/electronics/ETR2014-3.pdf`
The author has homepage: `http://www.physics.otago.ac.nz/people/molteno`
E-mail: `tim@physics.otago.ac.nz`
Address: Physics Department, University of Otago, P.O. Box 56, Dunedin, New Zealand

Citing this Report

This report should be cited as:

> Timothy C.A. Molteno, “NEC2++: An NEC-2 compatible Numerical Electromagnetics Code”, *Electronics Technical Reports* No. 2014-3, ISSN 1172-496X, October 2014.

Alternatively, using BibTeX, as:

```
@techreport{etr2014-3,
title={{NEC2++:  An NEC-2 compatible Numerical Electromagnetics Code}},
author={Timothy C.A. Molteno},
series={Technical Reports from the Electronics Group at the University
of Otago},
institution={{University of Otago}},
issn={1172-496X (Print) 1172-4234 (Online)},
url={http://www.physics.otago.ac.nz/reports/electronics/ETR2014-3.pdf}
number={2014-3},
date={October 2014} }
```

NEC2++: An NEC-2 compatible Numerical Electromagnetics Code

Timothy C.A. Molteno

Abstract

We present NEC2++ – an open-source numerical electromagnetics code, compatible with NEC-2. This code is written in C++ and provides a readable, modular foundation for investigations into numerical electromagnetism and antenna simulation. Dynamic memory allocation increases performance on small structures and removes the compiled-in upper limits on structure complexity present in the original code. Linking against the LAPACK libraries is a compile-time option that can provide significant performance enhancement. The NEC2++ code has a C API and can be compiled into a library for inclusion into other software, for example automatic antenna design optimisation systems, plugins for other languages, and graphical front-ends.

Contents

Chapter 1

Introduction

Numerical Electromagnetics Code (NEC-2) is a widely used standard method-of-moments code for the analysis of antennas [2]. The original software was written in FORTRAN and there have been many modifications of the original FORTRAN released, for example versions have been optimised for parallel processor architectures [3, 8, 11]. These re-implementations have not addressed some difficulties associated with the NEC-2 source code, including lack of modularity, the unreadability of the original FORTRAN source and the interweaving of analysis code with output code. These difficulties combine to present a formidable hurdle for investigators wishing to add new features or understand how the code operates.

Translations of NEC-2 to an object oriented paradigm have also been described [9], however this work is not in the public domain, and is not available as a resource for investigations into numerical electromagnetism.

The NEC2++ code we describe in this article addresses these difficulties. The code [7] released under the GNU General Public License (GPL) [4] is written in C++ and mirrors the functionality of the original FORTRAN code, with some input from the nec2c [6] C translation. Extensive changes have been made to the code including:

- All vectors and matrices are represented as C++ objects.
- Number precision has been abstracted by use of a standard template complex number class. The code can be altered to use single precision, double precision or long double precision by changing a single line.
- Elimination of all global variables.
- Replacement of the more obscure variable names by more explicit names.
- File output code has been largely separated from the analysis code. This allows the incorporation of the NEC2++ code as a library within other applications.

To facilitate further development, a testharness (nec2diff) has been written that compares NEC-2 output files. This testharness allows rapid development of NEC-compatible code because the results of a suite of test geometries and analysis can be compared to the original FORTRAN results and any differences are immediately reported. This testharness is not described here, however source-code and details are available from the NEC2++ homepage [7].

This report describes the aims of the NEC2++ project, the current architecture of the codebase and shows a simple example that incorporates an NEC-2 simulation into a C program. NEC2++ is under constant development and we conclude with plans for future development of the codebase.

1.0.1 Advantages of Nec2++

Writing geometery description files, and parsing results files are both a barrier to new users of antenna simulation software, and significantly increase simulation time for small structures. Nec2++ provides direct access to the nec2 simulation engine using the Python, Ruby and C languages. For automatic antenna design or optimisation of small structures, removal of the need to use intermediate files, and launch sub-processes, provides several orders of magnitude increase in simulation capability.

Chapter 2

Using Nec2++ from other languages

An important use of electromagnetics software is in automated design. This chapter describes how nec2++ can be used as a library from other languages. Nec2++ has been used in this way for genetic optimization of antenna performance [13].

2.1 Use of NEC2++ from Ruby

A module for nec2++ allows antenna simulation from the Ruby high-level object-oriented language. The code for the Ruby module is located in the subdirectory 'ruby' of the necpp source code distribution. To install this module, simply execute the 'build.sh' script in this directory after installation of nec2++.

A simple example of an antenna simulation is shown below.

```
require 'necpp'
require 'complex'

nec = Necpp.nec_create
Necpp.nec_wire(nec, 1, 17, 0, 0, 2, 0, 0, 11, 0.1, 1, 1);
Necpp.nec_geometry_complete(nec, 1, 0);
Necpp.nec_gn_card(nec, 1, 0, 0, 0, 0, 0, 0, 0);
Necpp.nec_fr_card(nec, 0, 1, 30, 0);
Necpp.nec_ex_card(nec, 0, 0, 5, 0, 1.0, 0, 0, 0, 0, 0);
Necpp.nec_rp_card(nec, 0, 90, 1, 0,5,0,0, 0, 90, 1, 0, 0, 0);

result_index = 0
z = Complex(Necpp.nec_impedance_real(nec,result_index),
    Necpp.nec_impedance_imag(nec,result_index))
print "Antenna Impedance:#{z} Ohms\n"

print "Gain:#{Necpp.nec_gain_max(nec,result_index)} dB \n"
print "RHCP:#{Necpp.nec_gain_rhcp_max(nec,result_index)} dB\n"

Necpp.nec_delete(nec)
```

2.2 Use of NEC2++ from Python

Python is a high-level language that is widely used for scientific computing~citepython. Two bindings for the python language exist for the nec2++ library. PyNec, contributed by Remi Sassolas exposes the Nec2++ C++ class structure as python classes. This binding provides the most power to the user, at the expense of a little complexity. Description of how to use PyNec is beyond the scope of this document.

A simpler method exposes the C-library interface to the python language. This approach is conceptually simpler, but still provides a significant speed increase over the traditional approach to antenna modelling (see Section 1.0.1 where intermediate files are required for structure description and simulation results, and the simulation is performed by a separate process.

An example is shown below

```
import 'necpp'
Fill me in here.
```

2.3 Use of NEC2++ as a C-library

The re-factoring of the NEC2++ code to eliminate the interweaving of output and analysis code allows NEC2++ to be efficiently used without any input or output files.

The following listing shows an NEC input file that requests a radiation pattern from a vertical antenna excited at 30 MHz.

```
    GW 0 9 0. 0. 2. 0. 0. 7 .1
    GE 1
    FR 0 1 0 0 30.
    EX 0 0 5 0 1.
    GN 1
    RP 0 90 1 0500 0 90 1 0
n   EN
```

Using the C API of libnecpp, the following code performs the same simulation and returns the maximum gain without requiring any analysis of the NEC-2 output files.

```
#include <libnecpp.h>
int main(int argc, char **argv)
{
    nec_context* nec;
    nec = nec_create();

    nec_wire(nec, 0, 9, 0, 0, 2, 0, 0, 7, 0.1, 1, 1);
    nec_geometry_complete(nec, 1, 0);
    nec_fr_card(nec, 0, 1, 30, 0);
    nec_ex_card(nec, 0, 0, 5, 0, 1, 0, 0, 0, 0, 0);
    nec_gn_card(nec, 1, 0, 0, 0, 0, 0, 0, 0);
    nec_rp_card(nec, 0, 90, 1, 0500, 0, 90, 1, 0, 0, 0);
    printf("%f",nec_get_maximum_gain(nec));

    nec_delete(nec);
    return 0;
}
```

Chapter 3

Using nec2++ as a standalone tool

The nec2++ code can be used as a command-line programme that accepts NEC input files. To simulate an antenna in the file 'example2.nec' you would issue the command

```
    nec2++ -i example2.nec -o example2.out
```

This will create a file called example2.out that contains the result of the simulation.

Full usage of the command-line programme is shown below:

```
usage: nec2++ [-i<input-file-name>] [-o<output-file-name>]
       -g: print maximum gain to stdout.
       -b: Perform NEC++ Benchmark.
       -s: print results to standard output.
       -c: print results in comma-separated-value (CSV) format,
           this options is used in conjunction with (-s) above.
       -h: print this usage information and exit.
       -v: print nec2++ version number and exit.
```

3.1 Input file format

Nec2++ can read standard NEC-2 input files. The existing command-line nec2++ tool, uses a hand-coded parser. This will be replaced by a grammar for NEC geometery files using the antlr [10] parser tools. This will simplify the addition of new structure and simulation commands. Code for this new parser resides in the 'antlr' directory in the nec2++ source code distribution.

3.2 Output file format

Nec2++ can generate output files in two different formats. There are three output formatting engines. The first is compatable with the original FORTRAN NEC executable. The second output format is Comma Seperated Value (CSV) format, suitable for direct input into plotting programmes. Under development is an extgensible markup language (XML) output formatter suited to automatic parsing of results files by other tools.

3.3 Example: Simulating Patch Antennas

This section shows how nec2++ can be used to simulate patch antennas. The antenna geometry is written to a file, and nec2++ is run as a seperate executable, generating from this geometry file, and results file. We then show how the results file can be parsed to produce the simulation results.

3.3.1 Generating Input Files

As wires are described in the '.nec' input file one line at a time, the input of anything more complex than a simple dipole antenna quickly becomes a tedious process. Although nec2++ does define data cards to input geometries other than wires, such as the square patch card, these can often behave unexpectedly. For reliable results we therefore input shapes as a mesh of single wire segments. For example the following simple Ruby script prints out a list of data cards defining a rectangular surface with width 'size_x', length 'size_y', number of grid segments 'segments' and z-coordinate 'height', parallel to the x-y plane.

```
# Set global variable $tag to give each data card a unique identifier
$tag = 0

# Method to print data cards describing rectangular patch
def rec_patch(size_x, size_y, segments, height)
  seg_length_x = Float(size_x)/segments
  seg_length_y = Float(size_y)/segments
  (segments+1).times { |j|
    (segments).times { |i|
      print "GW #{$tag} 1 #{i*seg_length_x-size_x/2} #{j*seg_length_y-
          size_y/2} #{height} #{(i+1)*seg_length_x-size_x/2} #{j*
          seg_length_y-size_y/2} #{height} 0.5\n"
      $tag = $tag + 1
      }
  }
  (segments+1).times { |j|
    (segments).times { |i|
      print "GW #{$tag} 1 #{j*seg_length_x-size_x/2} #{i*seg_length_y-
          size_y/2} #{height} #{j*seg_length_x-size_x/2} #{(i+1)*
          seg_length_y-size_y/2} #{height} 0.5\n"
      $tag = $tag + 1
    }
  }
end
```

The code below calls the rec_patch method to create a 10 by 8 (cm) rectangular patch in the x-y plane with 8 segments along each axis, then prints the remaining necessary data cards to provide the simulation parameters. Combined with the above code this produces an acceptable '.nec' file ready for simulation.

```
# Call the rec_patch method
rec_patch(10, 8, 8, 0)
```

```
# Print last few data cards, setting the excitation wire, scaling
    factor, frequency data and radiation pattern
print "GW #{$tag} 3 0 0 -1 0 0 0 0.05
GS  0  0  0.1
GE
EX  0 #{$tag} 1  0  0  0  0  0  0
FR  0  1  0  0  200  100  0  0  0  0
RP  0  31  61  1000  0  0  6  6  0  0
EN
"
```

3.3.2 Parsing Output Files

The output of an nec2++ simulation lists an antenna's input parameters, currents at each location and radiation data at each value of theta and phi requested by the input. Though software exists to analyze this data and produce graphical radiation patterns, one can easily parse the output file to extract only the information required. The following Ruby code loads an nec2++ output file, locates the data for the antennas input impedance and calculates a voltage standing wave ratio assuming the antenna is coupled to an ideal 50ohm coax cable.

```
# Calculate the VSWR of two impedances
def vswr(r1, x1, r2, x2)
  top_r = r1 - r2
  top_x = x1 - x2
  bottom_r = r1 + r2
  bottom_x = x1 + x2
  ratio = Math.sqrt((top_r*bottom_r + top_x*bottom_x)/(bottom_r**2 +
     bottom_x**2))**2+((bottom_r*top_x-top_r*bottom_x)/(bottom_r**2 +
      b_bottom_x**2))**2)
  return ratio
end

# Search NEC2++ output file for impedance information
File.open("nec_output.data", "r") do |file|
  count = 0
  while line = file.gets
    if line =~ /IMPEDANCE (OHMS)/ then count=3 end
    if count > 0 then
      if count == 1 then
    resistance = Float(line.split[6])
    reactance = Float(line.split[7])
    print vswr(resistance, reactance, 0, 50)
    end
      count = count - 1
    end
  end
end
```

Chapter 4

Code structure

The NEC2++ codebase is written using the C++ language. This language has powerful object-oriented features that we invoke to simplify and clarify the original FORTRAN. An example is the use of objects to represent vectors and matrices. This dramatically streamlines the allocation and handling of memory. As a consequence, all memory is allocated dynamically – there are no limits imposed in the source code to the size of the structures that can be simulated.

The central object in the NEC2++ simulation code is the `nec_context` object, this object contains information about the state of the simulation as well as storing output information. A new `nec_context` object is created for each simulation.

After creation, a geometry is associated with the `nec_context` through a `c_geometry` object. This object contains details of the physical structure to be simulated. Analyses are then be triggered by calling methods on the `nec_context` object. These methods correspond to the analysis cards of a traditional NEC-2 input file.

4.1 Modularisation

The separation of the input file parsing, the computation and the output into separate code modules is one of the primary aims of the NEC2++ project. Aside from improvements in code readability, modularisation provides some important benefits. Some of these are described below.

The separation of input file parsing into a separate module allows the addition of new front-ends to the NEC code. For example an XML geometry description could be used, or a binary interface added to allow efficient execution of multiple simulations – as might be used in an automatic antenna design optimisation system (see for example Jones et.al [5]).

The separation of the structure simulation code from the output code allows simulation without the generation of an output file. The parsing of this output file to access appropriate information is often inefficient. In addition, moving simulation and output into separate modules improves the readability and understandability of both. The initial separation of output code and analysis code was accomplished by creating a proxy output function that could be turned off globally. Currently work is underway to design objects that represent each kind of simulation output, e.g. radiation patterns. These objects are created by the simulation code, and can be queried for specific output information, for example the maximum gain. This feature is already available for some kind outputs, for example, the following excerpt from the code shown in Section 2.3;

```
...
nec_rp_card(nec, 0, 1, ...);
gain = nec_get_maximum_gain(nec);
```

will trigger a radiation pattern simulation – just like an RP card in NEC-2 but with no output. The `nec_get_maximum_gain()` function return the maximum gain in the radiation pattern. The modularisation of the output code is under active development.

4.1.1 Error handling

Output from NEC-2 simulations consists of error information and results. In NEC2++ error information is handled using C++ exceptions. This allows a client program using the libnecpp library from C++ to trap and manage errors that occur during geometry construction and during simulation.

All the methods on the `nec_context` object now throw exceptions under error conditions however the C-style API does not allow exceptions. Work is underway to wrap each C API function with exception handling codes and provide a mechanism for C programs to access any relevant error information.

Additional work is under way to generate warnings under known situations where simulations become inaccurate, for example when the ratio of the segment length to the wire radius is less than 4.0 and the extended thin wire kernel is not employed.

4.1.2 Geometry parsing

A class `c_geometry` is defined that contains the geometry of the structure to be simulated. This geometry object can be constructed in two ways, either from a standard NEC-2 description file, or directly from a C or C++ API.

The geometry object is then passed to an `nec_context` object, that manages the simulation of the antenna. The following code snippet shows the construction of a geometry object in C++:

```
nec_context nec;
nec.initialise();

c_geometry& geo = nec.get_geometry();
geo.wire(0, 8, 0, 0, -0.25,
    0, 0, 0.25,
    0.00001,
    1.0, 1.0);
nec.geometry_complete(0,0);
```

This geometry consists of a single wire with eight segments, and is equivalent to an NEC-2 ‘GW’ card.

Chapter 5

Summary

The NEC2++ codebase provides a platform for the development of systems for analysing and optimising the electromagnetic response of structures. The source-code is freely available, and binary files are available for Microsoft Windows, Unix (Linux) and Mac OS X.

The first stage of this project – the features described in this article – are complete, and active development of new features is underway. In particular we plan to:

- Add an XML specification for antenna geometry and analysis. This will further simplify integration of NEC2++ libraries with existing applications.
- Add a physical-sensitivity analysis front-end that estimates the sensitivity of electromagnetic properties to changes in structure geometry. This is intended to provide a measure of how practical an antenna design is in a non-ideal environment, or to determine appropriate manufacturing tolerances.
- Explore performance enhancements through optimised libraries such as ATLAS [12], or distributed memory parallel codes such as the Scalable Linear Algebra Package (ScaLAPACK) [1]. Both these libraries provide dramatically faster LU decomposition routines that would accelerate one of the critical steps in the NEC-2 simulation process.

This list is only indicative, and investigators are invited to download the source, use it, modify it and add features[1].

[1]The code is released under the GNU General Public License [4] – this license allows unlimited modification and redistribution, provided the code remains under this license.

References

[1] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, A. Petitet G. Henry, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.

[2] G. Burke and A. Poggio. Numerical electromagnetics code – method of moments. Technical Report UCID-18834, Lawrence Livermore National Laboratory, 1981.

[3] P.S. Excell, G.J. Porter, Y.K. Tang, and K.W. Yip. Re-working of two standard moment-method codes for execution on parallel processors. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields.*, 8, 1995.

[4] Free Software Foundation, Inc. GNU General Public License. Version 2, 1991.

[5] E.A. Jones and W.T. Joines. Design of Yagi-Uda antennas using genetic algorithms. *IEEE Transactions on Antennas and Propagation*, 45:1386–1392, 1997.

[6] Neoklis Kyriazis. nec2c a translation of NEC-2 to C. `http://www.si-list.org/NEC_Archives/nec2c.tar.gz`, 2004-5.

[7] T.C.A. Molteno. NEC2++ homepage. `http://www.physics.otago.ac.nz/research/electronics/nec`, 2004-6.

[8] D.C. Nitch and A.P.C. Fourie. Parallel implementation of NEC. *Applied Computational Electromagnetics Society Journal*, 9(1):51–57, 1994.

[9] D.C. Nitch and A.P.C. Fourie. A redesign of NEC2 using the object-oriented paradigm. In *Antennas and Propagation Society International Symposium*, volume 2, pages 1150–1153, 1994.

[10] Terence Parr. *The definitive ANTLR reference: building domain-specific languages*. Pragmatic Bookshelf, 2007.

[11] A. Rubinstein, F. Rachidi, M. Rubinstein, and B. Reusser. A parallel implementation of NEC for the analysis of large structures. *IEEE Transactions on Electromagnetic Compatibility*, 45:177–188, 2003.

[12] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[13] PJ Williams and TCA Molteno. A comparison of genetic programming with genetic algorithms for wire antenna design. *International Journal of Antennas and Propagation*, 2008, 2008.

Electronics Group
Department of Physics
University of Otago
`elec.otago.ac.nz`