SANDIA REPORT

SAND-2012-3087
Unlimited Release
Printed April 2012

C++ Tensor Toolbox User Manual (v 1.0)

Todd D. Plantenga, Tamara G. Kolda

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>

C++ Tensor Toolbox User Manual (v 1.0)

Todd D. Plantenga
Sandia National Laboratories
Livermore, CA
Email: tplante@sandia.gov

Tamara G. Kolda
Sandia National Laboratories
Livermore, CA
Email: tgkolda@sandia.gov

Abstract

The C++ Tensor Toolbox is a software package for computing tensor decompositions. It is based on the Matlab Tensor Toolbox, and is particularly optimized for sparse data sets. This user manual briefly overviews tensor decomposition mathematics, software capabilities, and installation of the package.

Acknowledgments

This work was funded by the U.S. Department of Energy, in part through the Office of Advanced Scientific Computing Research (ASCR), as part of the Applied Mathematics Research Program (<http://www.er.doe.gov/ascr/Research/AppliedMath.html>).

Thanks to Grey Ballard, Eric Chi, and Ben Allan for developing portions of the C++ code.

Contents

1	Introduction	7
1.1	CP Tensor Decomposition	7
1.2	CP-ALS	7
1.3	CP-APR	8
1.4	Citing Use of the Software	8
2	Software	9
3	Building Tensor Toolbox	11
3.1	Unpack Tensor Toolbox Source Code	11
3.2	Download and Install CMake	11
3.3	Build an LAPACK Library	12
3.4	Build and Test Tensor Toolbox	14
3.5	CMake Tips	17
	References	18

This page intentionally left blank.

1 Introduction

Tensors (also known as multidimensional arrays or N-way arrays) are used in a variety of applications ranging from chemometrics to network analysis. The Tensor Toolbox provides classes for manipulating dense, sparse, and structured tensors in C++. The Toolbox compiles into libraries and is intended for use with custom applications written by users. Classes are based on the MATLAB Tensor Toolbox (<http://www.sandia.gov/~tgkolda/TensorToolbox/>).

The current release implements a subset of functionality from the MATLAB Tensor Toolbox. It emphasizes efficient computation of tensor decompositions, especially for sparse data sets. Algorithms are targeted for workstation machines, and do not exploit any form of parallelism. Nevertheless, it should be possible to factorize data sets with millions of nonzero elements in reasonable time (though performance is always problem-dependent).

Contact information:

Todd Plantenga, Sandia National Laboratories

Email: tplante@sandia.gov

1.1 CP Tensor Decomposition

A tensor is an N -way or multi-way array. We are interested here in tensor decomposition, specifically, CANDECOMP/PARAFAC (CP) [2, 4]. Given a real-valued tensor $\mathcal{X}$ of size $I_1 \times \cdots \times I_N$, the goal is to express it as the weighted sum of outer products of vectors, i.e.,

$$\mathcal{X} \approx \mathcal{M} \equiv \sum_{r=1}^R \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)}.$$

We say the tensor $\mathcal{M}$ is the R -component *model*. Each λ_r is a scalar for $r = 1, \dots, R$. Each $\mathbf{a}_r^{(n)}$ is a vector of length I_n for $n = 1, \dots, N$ and $r = 1, \dots, R$. The symbol $\circ$ represents an outer product so that the (i, j, k) element of $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ is defined by $a_i b_j c_k$. Each summand is a rank-one tensor, also known as a *component*. There are N *factor matrices* defined by

$$\mathbf{A}^{(n)} \equiv \begin{bmatrix} \mathbf{a}_1^{(n)} & \cdots & \mathbf{a}_R^{(n)} \end{bmatrix} \quad \text{for } n = 1, \dots, N.$$

The n th factor matrix is of size $I_n \times R$. We call $\boldsymbol{\lambda} = [\lambda_1 \ \cdots \ \lambda_R]^\top$ the *weight* vector. A full description of the CP decomposition can be found in [5]. We discuss two methods for fitting CP decomposition for *sparse* tensors in the subsections that follow.

1.2 CP-ALS

The typical approach to fitting tensor data is to use a least squares fit, which is appropriate if the data comes from a Gaussian distribution. We consider the least squares objective function, i.e.,

$$\min_{\mathbf{i}} \sum_{\mathbf{i}} (x_{\mathbf{i}} - m_{\mathbf{i}})^2 \quad \text{subject to} \quad \mathcal{M} = \sum_{r=1}^R \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)}.$$

Here $\mathbf{i}$ denotes the multi-index $(i_1, \dots, i_N)$. A standard approach is *alternating least squares* (ALS) where all factors matrices but one is fixed at each inner iteration. Each subproblem is a standard matrix least squares problem and can be solved exactly. For details, we refer the reader to Fig. 3.3 of [5]. For sparse tensors, we achieve efficiency with specialized operations as described in [1].

1.3 CP-APR

For sparse count data, it may be better described by a Poisson distribution. In this case, we consider the generalized KL-divergence objective function, i.e.,

$$\min_{\mathbf{i}} \sum_{\mathbf{i}} m_{\mathbf{i}} - x_{\mathbf{i}} \log m_{\mathbf{i}} \quad \text{subject to} \quad \mathbf{M} = \sum_{r=1}^R \lambda_r \mathbf{a}_r^{(1)} \circ \dots \circ \mathbf{a}_r^{(N)} \quad \text{and} \quad \lambda, \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)} \geq 0.$$

We once again take an alternating approach, but this time we do *alternating Poisson regression* (APR). The details of the algorithm and implementation are described in [3].

1.4 Citing Use of the Software

For CP-ALS, cite this technical report and

B. W. Bader and T. G. Kolda. Efficient MATLAB computations with sparse and factored tensors, *SIAM Journal on Scientific Computing* 30(1):205-231, December 2007. (doi:10.1137/060676489).

For CP-APR, cite this technical report and

E. C. Chi and T. G. Kolda. *On Tensors, Sparsity, and Nonnegative Factorizations*, arXiv:1112.2414 [math.NA], December 2011. (<http://arxiv.org/abs/1112.2414>).

2 Software

Tensor Toolbox provides source code for tensor classes and decomposition methods. Source code is not intended to be modified by users. The header files are provided so users can write applications that call Tensor Toolbox methods. Source files are made available to inspect internal documentation, or to modify for debugging.

The current release of Tensor Toolbox implements a subset of functionality from the MATLAB Tensor Toolbox. The C++ code is completely independent of MATLAB code, but follows the same object-oriented design of classes, and provides some I/O methods for interoperability with MATLAB. Future releases will fill in missing functionality, and may add features not available in the MATLAB Toolbox.

The primary emphasis of this release is to enable tensor decomposition of sparse data sets for C++ applications. Methods are provided to read data from files into tensors, manipulate tensor objects, and compute decompositions. An application might use the decomposition to analyze patterns in the data, project to a low dimensional subspace, or estimate missing data elements.

Source code is organized under the directories:

```
TensorToolboxCPP-1.0-src
  TTB
  TTB_MathLibs
  TTB_PerfTests
  TTB_Test
```

Files under these directories contain source code for the Tensor Toolbox libraries (TTB), unit tests (TTB_Test), performance tests (TTB_PerfTests), and some basic math routines (TTB_MathLibs). Examples of using the library can be found by examining the performance tests and unit tests.

User application code needs to compile and link with the Tensor Toolbox libraries. Tensor Toolbox is built with the CMake system (Section 3) and a local C++ compiler. User applications will reference the header files and build with the same compiler, but do not have to integrate with CMake.

For example, assume an application file on Linux is named `MyTensorApp.cpp`. It includes one or more headers in the TTB directory, and needs to link with the Tensor Toolbox libraries. Using the g++ compiler, this can be accomplished with:

```
> g++ -c MyTensorApp.cpp -I$TTBSRC /TTB
> g++ -o MyExeName MyTensorApp.o -L$TTBBLD -lttb -lttb_mathlibs
```

where `$TTBSRC` is set to the source code parent directory `TensorToolboxCPP-1.0-src`, and `$TTBBLD` is set to the build directory (in Section 3.4 it is named `TTB_build`).

This page intentionally left blank.

3 Building Tensor Toolbox

Tensor Toolbox uses the CMake build system (<http://cmake.org/>) to support compilation on multiple platforms, including Linux, Windows, and Mac OSX. This section describes the process of installing source code, third party libraries, and building test and example executables.

Several steps are required to build the C++ Tensor Toolbox. A quick outline is below, and full details for various platforms follow.

- 3.1 Unpack Tensor Toolbox source code.
- 3.2 Download and install CMake toolset.
- 3.3 Obtain or build an LAPACK library (optional, but recommended).
- 3.4 Build and test.

3.1 Unpack Tensor Toolbox Source Code

The software is supplied in compressed file form for Unix (tar.gz) or Windows (zip). The contents are the same, but the build procedure is slightly different.

Save the compressed file to any directory and unzip it. You should see a directory structure like the following:

```
TensorToolboxCPP-1.0-src
  doc
  TTB
  TTB_MathLibs
  TTB_PerfTests
  TTB_Test
  test.data
```

Files under these directories contain source code for the Tensor Toolbox libraries (TTB), unit tests (TTB_Test), performance tests (TTB_PerfTests), and some basic math routines (TTB_MathLibs).

This section will describe how to compile and build the Tensor Toolbox libraries, and test executables. In general, your application will use classes and methods in the libraries. The unit test and performance test executables are good places to look for examples of calling the libraries.

3.2 Download and Install CMake

CMake is a leading open-source build system that supports multiple operating systems. You need to download a CMake binary distribution (typically, 5-10 Mbytes in size) appropriate for your operating system and install it. The installation creates a CMake tool that will be used to construct platform-specific build scripts for compiling Tensor Toolbox source code. CMake is not a compiler; it assumes a C++ compiler is already installed.

Visit <http://cmake.org/> and find a recent release of CMake for your target operating system. The CMake release must be 2.6.2 or later. At the time this documentation was produced, the CMake distribution could be found by clicking on **RESOURCES** and then **Download** to reach <http://cmake.org/cmake/resources/software.html>. Only the binary distribution is needed (no CMake source code). For example, `cmake-2.8.7-Linux-i386.tar.gz` was the file name for an x86 Linux machine, and `cmake-2.8.7-win32-x86.zip` the file name for an x86 Windows machine.

Installation of CMake is very simple, and explained on the CMake download page. For example, on a Linux machine just unpack the file to any directory (the procedure does not require root privileges). It should create a new subdirectory tree with a name like `cmake-2.8.7-Linux-i386`. Just add the subdirectory `cmake-2.8.7-Linux-i386/bin` to `PATH`.

On Windows, run the CMake distribution file to start an installation wizard and follow the directions. By default, CMake will install at `C:\\Program Files\\CMake 2.6` and create a **Start Menu** entry that invokes the CMake GUI interface. If you prefer to run the command line version of CMake, then click a wizard button that adds CMake to `PATH`.

3.3 Build an LAPACK Library

A third party LAPACK (Linear Algebra PACKage) library is required for certain Tensor Toolbox capabilities. The software comes with a default implementation for quickly compiling and running the code, but a proper third party LAPACK library is recommended. The provided default contains only some of the LAPACK functions required, so certain Tensor Toolbox capabilities are not available until the package is linked with a proper LAPACK.

Your system may already have LAPACK installed. For instance, on some Linux distributions LAPACK is available in the file `/usr/lib/liblapack.a`. In this case CMake should find it automatically and no further effort is needed. Try building the software as described in Section 3.4; the CMake configuration will state clearly whether an LAPACK library was found.

If LAPACK was not found on your system, or you prefer a particular version, then the library must be installed. LAPACK libraries are available from many sources. Perhaps the most common version is from Netlib, freely available at <http://netlib.sandia.gov/lapack>. Other possibilities are vendor-provided libraries like the Intel MKL or AMD ACML, and tunable versions such as ATLAS.

LAPACK functions called by Tensor Toolbox are listed below. An asterisk indicates a function not available in the default implementation provided with the software.

<code>dasum</code>	<code>dger</code>
<code>daxpy</code>	<code>dgesv</code>
<code>dcopy</code>	<code>dnrm2</code>
<code>ddot</code>	<code>dscal</code>
<code>dgemm</code>	<code>idamax</code>
<code>dgemv</code>	

Make sure the library contains these functions and their dependents, or there will be unresolved symbols when linking executables. CMake will test for the presence of these functions when it configures Tensor Toolbox, and will halt with a warning message if it detects a problem.

Linux example of building Netlib. This example shows a particular case of building a Netlib version using the GNU compilers. Netlib produces two library files, one for BLAS functions such as `ddot` and one for LAPACK functions such as `dgesv`. The link order of these files matters; in the example below the LAPACK library must be listed before the BLAS library. The Netlib libraries are created using a Fortran compiler, so the Tensor Toolbox C++ executables must include a Fortran-to-C library (the CMake build process will try to do this automatically). Please note this is just one possible example and your build procedure may differ.

- Download `lapack-3.4.0.tgz` from <http://netlib.sandia.gov/lapack>
- Unpack the distribution (this example assumes the directory `/tmp` is used)
- Consult README and other documentation for Netlib instructions.
- For a Linux RHEL 5.5 machine build a minimal LAPACK as follows:
 - > `cd /tmp/lapack-3.4.0`
 - > `cp INSTALL/make.inc.gfortran make.inc`
 - > `make lib` (should produce file `liblapack.a`)
 - > `make blaslib` (should produce file `librefblas.a`)
- Add this option to the CMake command line when building Tensor Toolbox:
 - `-DLAPACK_LIBS="$LH/liblapack.a;$LH/librefblas.a"`
 - (where `$LH` is the LAPACK home `/tmp/lapack-3.4.0`)
- If CMake has trouble finding the native Fortran-to-C library, try adding:
 - `-DLAPACK_ADD_LIBS="gfortran"`

Windows example using Netlib CLAPACK with MSVC. This example uses a precompiled Netlib distribution made with the Microsoft Visual C++ compiler (MSVC). Netlib code is usually written in Fortran, but it is often more convenient to use the free MSVC compiler on Windows. Netlib provides a version of the source code that is translated to C, called CLAPACK. Symbol names are different, and an extra flag must be passed on the command line to inform CMake. Please note this is just one possible example and your build procedure may differ.

- Download `CLAPACK3-Windows.zip` from <http://netlib.sandia.gov/clapack>
- Unzip the distribution in any directory; here, assume `c:\temp` is used
- Add these options to the CMake command line:
 - `-DLAPACK_LIBS="c:\temp\CLAPACK3-Windows\CLAPACK\Release\clapack.lib"`
 - `-DHAVE_BLAS_F2C=yes`

Linux example of using Intel MKL. The Intel MKL contains routines for LAPACK and many other math functions that are specially tuned for Intel microprocessors. Assuming MKL version 10.3.7 is installed under `$MKL`, add the options below to the CMake command line (please note this is just one possible example and your build procedure may differ):

- Add these options to the CMake command line:
 - `-DLAPACK_LIBS="$MKL/mkl/lib/intel64/libmkl_rt.so"`
 - `-DLAPACK_ADD_LIBS="$MKL/lib/intel64/libiomp5.so"`

3.4 Build and Test Tensor Toolbox

The CMake tool constructs platform-specific build scripts for compiling and linking executables. We recommend making an “out of source” build, instead of building the object and executable files in the source directories. This is easy to do with CMake and allows the existence of multiple builds without conflict.

To create an out of source build, make a clean directory, change to it, and run CMake from this directory. CMake allows the build directories to be anywhere, but in the remainder of this section we assume a clean directory called `build` is created at the same level as `TensorToolboxCPP-1.0-src`. After successfully building, the directory structure will look like the following (on Windows the libraries will be named `ttb.lib` and `ttb_mathlibs.lib`):

TensorToolboxCPP-1.0-src	
doc	(provided)
TTB	(provided)
TTB_MathLibs	(provided)
TTB_PerfTests	(provided)
TTB_Test	(provided)
test_data	(provided)
TTB_build	(you create this)
libttb.a	(built by CMake)
libttb_mathlibs.a	(built by CMake)
bin	(built by CMake)
perf_CpAlsAminAcid	(built by CMake)
perf_CpAlsRandomKtensor	(built by CMake)
unit_tests	(built by CMake)
test_data	(files copied by CMake)

The examples below show how to run CMake on various platforms. In all cases, please run the `unit_tests` executable after building to verify correctness of the libraries. For help with CMake, refer to <http://www.cmake.org/Wiki/CMake>. For information on linking with an LAPACK library, see Section 3.3.

Linux. Start in the directory above `TensorToolboxCPP-1.0-src` and run the following commands:

```
> mkdir TTB_build
> cd TTB_build
> cmake ../TensorToolboxCPP-1.0-src
-- The CXX compiler identification is GNU
-- ...
-- Build files have been written to: ...
> make
> ./bin/unit_tests
```

The execution of `cmake` displays several lines of informational output, only a few of which are shown above. Its behavior is roughly similar to a Unix “autoconf” or “config” tool. It produces the subdirectory structure described above, plus directories called `CMakeFiles` and `CMakeInclude`, and a `Makefile` that works with the chosen compiler. Running `make` in the last step produces the

libraries and executables. You can run the executables from the current directory, as shown for `unit_tests`.

Windows using Visual Studio. CMake can generate a Microsoft Visual Studio project for the Tensor Toolbox source code, and then Visual Studio C++ can be used to compile the libraries and executables. This example uses the free *Microsoft Visual C++ 2010 (Express Edition)* with version 10.0 C++ compiler. A subsequent example describes how CMake can produce a set of **Makefile** files that work with the command line **nmake** tool in the Express Edition.

First, make sure environment variables are configured for the Microsoft compiler. If installed in its default location, this is accomplished (for version 10.0) by running:

```
> c:\Program Files\Microsoft Visual Studio 10.0\VC\bin\vcvars32.bat
```

CMake can execute in a GUI or from the command line. This example uses a Windows DOS-like command line console such as the one below.

```
c:\temp\build>cmake -G "Visual Studio 10" ..\TTB_cpp
-- Check for working C compiler using: Visual Studio 10
-- Check for working C compiler using: Visual Studio 10 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler using: Visual Studio 10
-- Check for working CXX compiler using: Visual Studio 10 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- TTB: Looking for LAPACK/BLAS libraries
-- TTB: LAPACK not found, will compile TTB LAPACK code.
-- TTB: Makefiles will compile with production flags.
-- Configuring done
-- Generating done
-- Build files have been written to: C:/temp/build
```

Start in the directory above `TensorToolboxCPP-1.0-src` and run the following commands:

```
> mkdir TTB_build
> cd TTB_build
> cmake -G "Visual Studio 10" ..\TensorToolboxCPP-1.0-src
-- Check for working C compiler: Visual Studio 10
-- ...
-- Build files have been written to: ...
```

The execution of **cmake** displays several lines of informational output, only a few of which are shown above. It produces a subdirectory structure similar to that of `TensorToolboxCPP-1.0-src`, and a file `ALL_BUILD.vcxproj` with the main Visual Studio project.

Now either start Visual Studio and open the file `ALL_BUILD.vcxproj`, or double-click directly on `ALL_BUILD.vcxproj` from Windows Explorer. Use Visual Studio to build `ALL_BUILD`, compiling and linking everything, including the libraries, unit test executable, and examples. A successful build is shown in the screen shot below.

Run the `unit_test` and example executables to verify proper installation.

Windows using NMake. CMake can generate a set of `Makefile` files that work with the Visual Studio command line `nmake` tool. This example uses the free *Microsoft Visual C++ 2010 (Express Edition)* with version 10.0 C++ compiler.

First, make sure environment variables are configured for the Microsoft compiler. If installed in its default location, this is accomplished (for version 10.0) by running:

```
> c:\Program Files\Microsoft Visual Studio 10.0\VC\bin\vcvars32.bat
```

CMake can execute in a GUI or from the command line. This example uses a Windows DOS-like command line console such as the one below.

Start in the directory above `TensorToolboxCPP-1.0-src` and run the following commands:

```
> mkdir TTB_build
> cd TTB_build
> cmake -G "NMake Makefiles" ..\TensorToolboxCPP-1.0-src
```



```
-- Building for: NMake Makefiles
-- The C compiler identification is MSVC
-- ...
-- Build files have been written to: ...
> nmake
```

The execution of `cmake` displays several lines of informational output, only a few of which are shown above. It produces a subdirectory structure similar to that of `TensorToolboxCPP-1.0-src`, with `Makefile` files that work with the chosen compiler. Running `nmake` in the last step produces the Tensor Toolbox libraries and executable. Run the `unit_test` and example executables to verify proper installation.

3.5 CMake Tips

Documentation for CMake is part of the CMake installation, or can be found on the CMake web site (<http://cmake.org/>).

When modifying CMake build options, it is usually best to completely erase the previous build directory and start over.

Files in the source distribution named `CMakeLists.txt` or files that end with the suffix `.cmake` were written for Tensor Toolbox. Some of these can be edited to alter CMake behavior. For instance, to enable more makefile output during compilation, edit `ConfigureBuildType.cmake` and uncomment the line

```
SET (CMAKE_VERBOSE_MAKEFILE ON)
```

Then you should call `cmake` in a clean build directory.

References

- [1] B. W. BADER AND T. G. KOLDA, *Efficient MATLAB computations with sparse and factored tensors*, SIAM Journal on Scientific Computing, 30 (2007), pp. 205–231.
- [2] J. D. CARROLL AND J. J. CHANG, *Analysis of individual differences in multidimensional scaling via an N -way generalization of “Eckart-Young” decomposition*, Psychometrika, 35 (1970), pp. 283–319.
- [3] E. C. CHI AND T. G. KOLDA, *On tensors, sparsity, and nonnegative factorizations*. arXiv:1112.2414 [math.NA], December 2011.
- [4] R. A. HARSHMAN, *Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis*, UCLA working papers in phonetics, 16 (1970), pp. 1–84. Available at <http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf>.
- [5] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

DISTRIBUTION:

1	MS 9159	Todd Plantenga, 8958
1	MS 9159	Tamara Kolda, 8966
1	MS 0899	Technical Library, 9536 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office, 1911

This page intentionally left blank.