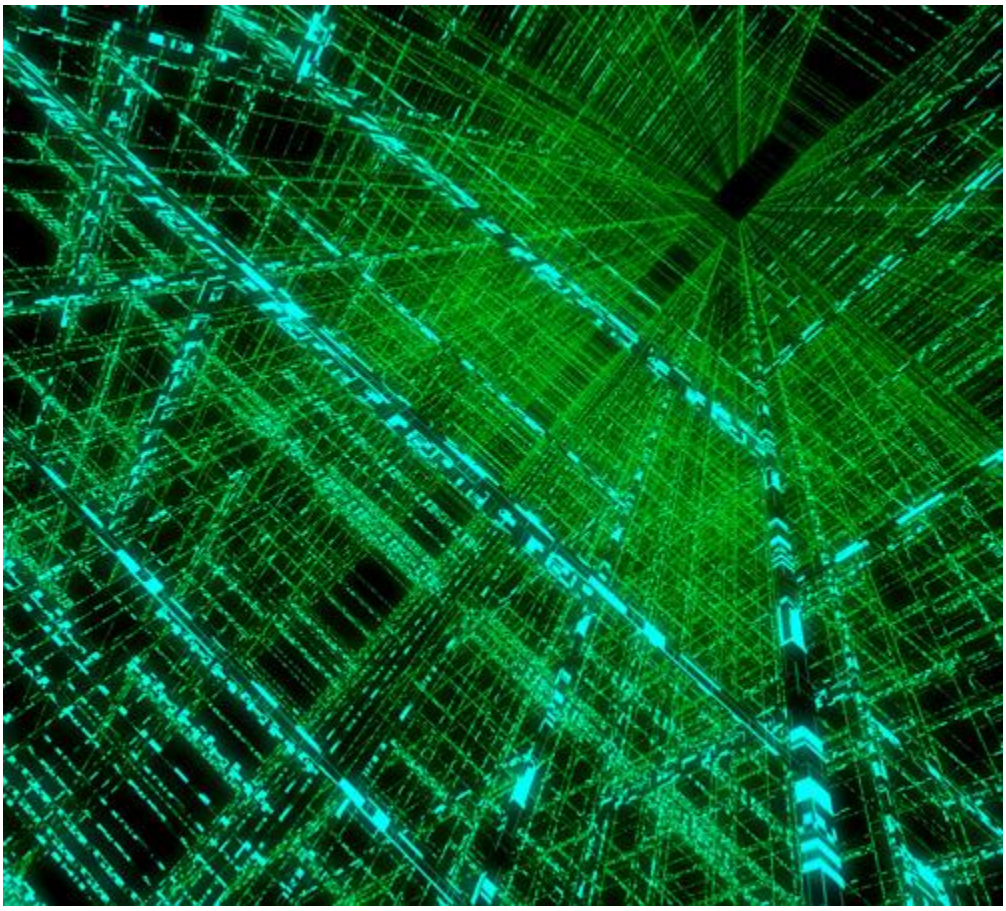
Ponderosa Computing

Linear Algebra Excel Add-in



Paul J. McClellan, Ph.D.

30 July 2019

Table of Contents

1 Ponderosa Computing Linear Algebra Excel Add-in

In spite of the apparent similarities between Microsoft Excel spreadsheet arrays and the matrix and vector elements of linear algebra, Microsoft Excel provides very little direct support of linear algebra functions and operations. The few Excel functions that do support linear algebra functions and operations are:

Array addition, subtraction, scalar multiplication	+, -, *
Inner product	SUMPRODUCT
Array transpose	TRANSPOSE
Matrix multiplication	MMULT
Matrix determinant	MDETERM
Matrix inverse	MINVERSE

Even this support is suspect. For example, it is not recommended to solve square systems of linear equations using matrix inversion and multiplication. It is more efficient and accurate to factor and solve systems of linear equations using triangularization methods. Nor is it recommended to solve linear least-square (regression) problems by solving corresponding normal equations. Instead, an orthogonal factorization method or singular value decomposition should be used for greater numerical stability.

LAPACK [1] is a freely-available, peer-reviewed computational linear algebra software package that provides routines for solving systems of simultaneous linear equations, computing least-squares solutions of linear systems of equations, and computing eigenvalue and singular value decompositions. The associated matrix factorizations (LU, Cholesky, LQ/QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as estimating condition numbers.

The purpose of the (32-bit) Ponderosa Computing Linear Algebra Excel Add-in is to provide linear algebra computations as scalar and array worksheet functions augmenting or replacing those provided as built-in Excel functions. This Excel Add-in:

1. Uses peer-reviewed and publicly documented LAPACK linear algebra algorithms.
2. Uses the CLAPACK implementation [3] of the LAPACK software package [2].
3. Supports double-precision scalars and matrices.
4. Supports Microsoft Windows platforms running Excel 2007 and later.
5. Tracks latest changes made in the LAPACK distribution addressing the latest bug reports and feature requests.

This document describes version 1.2 of the Ponderosa Computing Linear Algebra Excel Add-in.

2 Notation, Implementation, and Numerical Precision

2.1 Linear Algebra Notation

The Ponderosa Computing Linear Algebra Excel Add-in interprets Microsoft Excel spreadsheet arrays as linear algebra matrices or vectors.

A linear algebra **matrix** is a two-dimensional rectangular array of elements consisting of numbers, symbols, or expressions arranged in rows and columns. A matrix of m rows and n columns consists of **elements** x_{ij}, where *i* is the row location and *j* is the column location of the element:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}.$$

Numerical linear algebra operations are defined for matrices and vectors with all elements defined on the domain of complex numbers $\mathbb{C}$. Given that Microsoft Excel does not natively support complex numbers, we further restrict our consideration to matrices and vectors with all elements defined on the domain of real numbers $\mathbb{R} = (-\infty, +\infty)$.

A two-dimensional Excel spreadsheet array consisting of multiple rows and columns of adjacent cells can naturally be interpreted as a matrix and can be used as a matrix argument or return value by Ponderosa Computing Linear Algebra Excel Add-in worksheet functions.

The **transpose** operation $\boldsymbol{Y} = \boldsymbol{X^T}$ on an m x n matrix $\boldsymbol{X}$ creates an n x m matrix $\boldsymbol{Y}$ with rows and columns interchanged:

$$\boldsymbol{Y} = \boldsymbol{X^T} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}, \; where \; y_{ij} = \; x_{ji} \; .$$

A linear algebra **row vector** is a 1 x m matrix, consisting of a single row of m elements:

$$\boldsymbol{x} = \; [x_1 \quad x_2 \quad \ldots \quad x_m] \; .$$

An Excel spreadsheet array consisting of a single row of adjacent cells can naturally be interpreted as a row vector and can be used as a row vector argument or return value by Ponderosa Computing Linear Algebra Excel Add-in worksheet functions.

The transpose of a row vector is a **column vector**, an m x 1 matrix consisting of a single column of m elements:

$$\boldsymbol{x}^T = [x_1 \quad x_2 \quad \dots \quad x_m]^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}.$$

An Excel spreadsheet array consisting of a single column of adjacent cells can naturally be interpreted as a column vector and can be used as a column vector argument or return value by Ponderosa Computing Linear Algebra Excel Add-in worksheet functions.

2.2 This Add-in Uses the CLAPACK Implementation of LAPACK

LAPACK [1] is a freely-available, peer-reviewed numerical linear algebra software package that provides routines for solving systems of simultaneous linear equations, computing least-squares solutions of linear systems of equations, and computing eigenvalue and singular value decompositions. The associated matrix factorizations (LU, Cholesky, LQ/QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as estimating condition numbers.

The Netlib Repository provides a cross-platform Fortran source distribution of LAPACK [2] and a C source distributions of CLAPACK [3]. Instructions and tools for building CLAPACK on the Windows platform are available from the University of Tennessee Innovative Computing Laboratory [4]. The Ponderosa Computing Linear Algebra Excel Add-in uses the CLAPACK implementation of the LAPACK software package.

2.3 The Add-in Computations use IEEE 754 Double Precision Format

The numerical linear algebra computations are all defined for matrix and vector elements defined on the domain of real numbers $\mathbb{R}$ = (-∞, +∞). To describe limiting behavior of linear algebra computations it can be useful to define them for matrix and vector elements defined on the domain of the **affinely extended real numbers** [-∞, +∞], which adds the elements +∞ (positive infinity) and -∞ (negative infinity) to the real numbers.

The Ponderosa Computing Linear Algebra Excel Add-in implements its worksheet array functions using the double precision format defined by the IEEE Standard 754 for Binary Floating-Point Arithmetic [5]. This format includes representations of signed zeros and normalized, denormalized, and nonfinite (infinite and indeterminate) floating point numbers.

2.3.1 Machine Constants

We define here some double precision constants that appear later in this document.

eps : The relative machine precision, the distance from 1.0 to the next largest double-precision number. This number is $eps =$ dlamch_(P) $= 2^{-52} = 2.2204460492503131e-016$.

$safemin$: The minimum positive floating point value such that $1/safemin$ does not overflow. $safemin =$ dlamch_(S) $= 2.2250738585072014e-308$.

2.3.2 Worksheet Functions Do Not Support Nonfinite Matrix Elements

The CLAPACK implementation does not reliably handle nonfinite matrix or vector elements. For example, some computational loops are skipped when a factor is zero under the assumption that the skipped loop would have no impact on the computed results [7]. But this may fail to propagate nonfinite values or fail to detect invalid operations.

Microsoft Excel does not support nonfinite floating point matrix or vector elements [6] so the Ponderosa Computing Linear Algebra Excel Add-in worksheet functions will not encounter matrix or vector arguments containing nonfinite floating point elements.

2.3.3 Worksheet Function Finite Return Values May be Modified by Excel

Microsoft Excel does not support IEEE 754 denormalized numbers. As an example, for the matrix

$$\boldsymbol{A} = \begin{bmatrix} 1E-154 & 0 \\ 0 & 1E-154 \end{bmatrix},$$

the Ponderosa Computing Linear Algebra Excel Add-in worksheet function LA.DET() returns the denormalized value 1.000000000000e-308#DEN to Excel. Excel then truncates this denormalized value to 0.0.

Both Microsoft Excel and the Ponderosa Computing Linear Algebra Excel Add-in internally support normalized IEEE 754 double precision numbers with 53 binary digits of precision (equivalent to nearly 16 decimal digits of precision). However, Excel displays at most 15 decimal digits of precision in numeric values [6].

3 Worksheet Functions

The Ponderosa Computing Linear Algebra Excel Add-in implements most of its worksheet functions using version 3.2.1 of CLAPACK [3]. This implementation of CLAPACK was machine translated by Netlib from Fortran 77 to ANSI C using the f2c tool and version 3.2.1 of LAPACK [2]. The Ponderosa Computing Linear Algebra Excel Add-in uses the reference BLAS library included with this CLAPACK distribution.

3.1 Matrix Creation and Extraction

The Ponderosa Computing Linear Algebra Excel Add-in provides the following worksheet functions for creating matrices and extracting the diagonals of a matrix:

Create a constant matrix	LA.GEN.CON, LA.RGN.CON
Create a random matrix	LA.GEN.RAN, LA.RGN.RAN
Create an identity matrix	LA.GEN.IDN
Create a diagonal matrix	LA.GEN.DIAG
Extract diagonal elements	LA.XDIAG.C, LA.XDIAG.R

3.1.1 Create a Constant Matrix

The worksheet function LA.GEN.CON(m, n, value) returns an m x n matrix with entries all equal to value. If m < 1 or n < 1 then this worksheet function returns the #VALUE! error.

The worksheet function LA.RNG.CON(value) returns a region-filling matrix with entries all equal to value.

3.1.2 Create a Random Matrix

The worksheet function LA.GEN.RAN(m, n, minimum, maximum) returns an m x n matrix with random entries in [minimum, maximum). If m < 1 or n < 1 or maximum < minimum then this worksheet function returns the #VALUE! error.

The worksheet function LA.RNG.RAN(minimum, maximum) returns a region-filling matrix with entries all equal to value. If maximum < minimum then this worksheet function returns the #NUM! error.

These worksheet functions generate uniformly distributed random elements in the interval [minimum, maximum] using a Mersenne Twister pseudo-random generator of 32-bit numbers with a state size of 19937 bits provided by the C++11 standard library. This generator is initialized at library creation time, when Excel loads the Ponderosa Computing Linear Algebra Excel Add-in.

3.1.3 Create an Identity Matrix

The worksheet function LA.GEN.IDN(n) returns an n x n identity matrix, $\boldsymbol{I_n}$, of order n. If n < 1 then this worksheet function returns the #VALUE! error.

3.1.4 Create a Diagonal Matrix

The worksheet function LA.GEN.DIAG(D) returns an n x n diagonal matrix where n is the size of the argument matrix D and the diagonal elements of the returned matrix contain the elements of the matrix D in row-major order.

3.1.5 Extract Diagonal Elements

The worksheet functions LA.XDIAG.C(A) and LA.XDIAG.R(A) return the min(m, n) main diagonal elements of an m x n matrix A. The worksheet function LA.XDIAG.C(A) returns the diagonal elements as a min(m, n)-element column vector. The worksheet function LA.XDIAG.R(A) returns the diagonal elements as a min(m, n)-element row vector.

3.2 Scalar-Valued Matrix Functions

The Ponderosa Computing Linear Algebra Excel Add-in provides the following scalar-valued matrix functions:

1-norm (column norm)	LA.NORM1
Infinity-norm (row norm)	LA.NORMINF
Frobenius norm	LA.NORMF
2-norm (spectral norm)	LA.NORM2
Rank	LA.RANK
Spectral radius of a symmetric matrix	LA.SRAD
Trace of square matrix	LA.TRACE
1-norm inverse condition number estimate of square matrix	LA.ICNBR1
Infinity-norm inverse condition number estimate of square matrix	LA.ICNBRINF
Determinant of square matrix	LA.DET

3.2.1 1-Norm (Column Norm)

The worksheet function LA.NORM1(A) returns the 1-norm (column norm, $\|\boldsymbol{A}\|_1$) of a matrix A. This is the maximum absolute column sum of the elements of an m x n matrix A:

$$\|\boldsymbol{A}\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$$

This worksheet function computes the 1-norm of A using the routine `dlange_()` from CLAPACK v 3.2.1.

3.2.2 Infinity-Norm (Row Norm)

The worksheet function LA.NORMINF(A) returns the infinity-norm (row norm, $\|\boldsymbol{A}\|_\infty$) of a matrix A. This is the maximum absolute row sum of the elements of an m x n matrix A:

$$\|\boldsymbol{A}\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$$

This worksheet function computes the infinity-norm of A using routine `dlange_()` from CLAPACK v 3.2.1.

3.2.3 Frobenius-Norm

The worksheet function LA.NORMF(A) returns the Frobenius-norm ($\|\boldsymbol{A}\|_F$) of a matrix A. This is the square root of the sum of absolute squares (root mean square) of the elements of an m x n matrix A:

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

This worksheet function computes the Frobenius norm of A using routine `dlange_()` from CLAPACK v 3.2.1.

3.2.4 2-Norm (Spectral Norm)

The worksheet function LA.NORM2(A) returns the 2-norm (spectral norm, $\|\boldsymbol{A}\|_2$) of an m x n matrix A. This is the largest singular value of A:

$$\|\boldsymbol{A}\|_2 = \sigma_{\max}(\boldsymbol{A}) \, .$$

This worksheet function computes the min(m, n) singular values of A using routine `dgesvd_()` from CLAPACK v 3.2.1.

Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of A then this worksheet function returns the #NUM! error.

3.2.5 Rank

The **column rank** of an m x n matrix A is the maximum number of linearly independent column vectors of A. The **row rank** of an m x n matrix A is the maximum number of linearly independent row vectors of A. The column rank and the row rank are equal, and this is called the **rank** of the matrix A. The rank of an m x n matrix A is also the number of nonzero singular values of A.

The worksheet function LA.RANK(A) returns the rank of A. This function computes the min(m, n) singular values of A using routine `dgesvd_()` from CLAPACK v 3.2.1. The rank is then determined as the number of computed singular values that are significantly greater than zero.

The Ponderosa Computing Linear Algebra Excel Add-in uses the threshold value

$$threshold = \max(sfmin,\ \max(m,n) * eps * \sigma_{\max}(A)) \quad .$$

A computed singular value is considered significantly greater than zero if it exceeds this threshold. Here $safemin$ is the minimum positive floating point value such that $1/safemin$ does not overflow, eps is the relative machine precision, and $\sigma_{\max}(A)$ is the largest singular value of A.

Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of A then this worksheet function returns the #NUM! error.

3.2.6 Spectral Radius

Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the (real) eigenvalues of an n x n symmetric (real) matrix A. Then the spectral radius of A is:

$$\rho(\boldsymbol{A}) = \max_i(|\lambda_i|) \ .$$

The worksheet function LA.SRAD(A) returns the spectral radius of A. If A is not symmetric this worksheet function returns the #VALUE! error.

This worksheet function computes the eigenvalues of A using routine `dsyev_()` from CLAPACK v 3.2.1. Routine `dsyev_()` uses routine `dsterf_()` which computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or

QR algorithm. If this algorithm fails to find all of the eigenvalues of A in at most 30*n iterations then this worksheet function LA.SRAD() returns the #NUM! error.

3.2.7 Trace

The worksheet function LA.TRACE(A) returns the trace of an n x n matrix A. This is the sum of the main diagonal elements of A:

$$tr(\boldsymbol{A}) = \sum_{i=1}^{n} a_{ii}$$

This worksheet function computes the trace directly from its definition. If A is not square this worksheet function returns the #VALUE! error.

3.2.8 1-Norm Inverse Condition Number Estimate

The worksheet function LA.ICNBR1(A) returns the inverse of a 1-norm condition number estimate of an n x n matrix A. If A is not square this worksheet function returns the #VALUE! error.

LA.ICNBR1(A) starts by computing $\|\boldsymbol{A}\|_1$ using routine `dlange_()` from CLAPACK v 3.2.1 and returns zero if $\|\boldsymbol{A}\|_1 = 0$.

LA.ICNBR1(A) then uses routine `dgetrf_()` from CLAPACK v 3.2.1 to compute the LU factorization A = PLU using partial pivoting with row interchanges. If routine `dgetrf_()`determines the matrix A is exactly singular then LA.ICNBR1(A) returns 0.

Otherwise, LA.ICNBR1(A) estimates $\left\|\boldsymbol{A^{-1}}\right\|_1$ using the LU factorization and routine `dgecon_()` from CLAPACK v 3.2.1 and returns zero if the $\left\|\boldsymbol{A^{-1}}\right\|_1$ estimate is zero. Otherwise LA.ICNBR1(A) returns the inverse condition number estimate

$$c = \frac{1}{\|\boldsymbol{A}\|_1 \|\boldsymbol{A^{-1}}\|_1}$$

LA.ICNBR1(A) returns the inverse of the 1-norm condition number estimate of a matrix, rather than the condition number estimate of the matrix, itself, to provide a zero return value for singular matrices.

3.2.9 Infinity-Norm Inverse Condition Number Estimate

The worksheet function LA.ICNBRINF(A) returns the inverse of an infinity-norm condition number estimate of an n x n matrix A. If A is not square this worksheet function returns the #VALUE! error.

LA.ICNBRINF(A) starts by computing $\|\boldsymbol{A}\|_\infty$ using routine `dlange_()` from CLAPACK v 3.2.1 and returns zero if $\|\boldsymbol{A}\|_\infty = 0$.

LA.ICNBRINF(A) then uses routine `dgetrf_()` from CLAPACK v 3.2.1 to compute the LU factorization A = PLU using partial pivoting with row interchanges. If routine `dgetrf_()`determines the matrix A is exactly singular then LA.ICNBRINF(A) returns 0.

Otherwise, LA.ICNBRINF(A) estimates $\left\|\boldsymbol{A^{-1}}\right\|_\infty$ using the LU factorization and routine `dgecon_()` from CLAPACK v 3.2.1 and returns zero if the $\left\|\boldsymbol{A^{-1}}\right\|_\infty$ estimate is zero. Otherwise LA.ICNBRINF(A) returns the inverse condition number estimate

$$c = \frac{1}{\|\boldsymbol{A}\|_\infty \|\boldsymbol{A^{-1}}\|_\infty}$$

LA.ICNBRINF(A) returns the inverse of the infinity-norm condition number estimate of a matrix, rather than the condition number estimate of the matrix, itself, to provide a zero return value for singular matrices.

3.2.10 Determinant

The worksheet function LA.DET(A) returns the determinant of a square matrix A. If A is not square this worksheet function returns the #VALUE! error.

LA.DET(A) computes the determinant of A by using routine `dgetrf_()` from CLAPACK v 3.2.1 to compute the LU factorization A = PLU using partial pivoting with row interchanges. If routine `dgetrf_()`determines the matrix A is exactly singular then LA.DET(A) returns 0. Otherwise, LA.DET(A) accumulates the product of the diagonal entries of U with sign adjustment according to pivot row interchanges and scaling to avoid intermediate overflow.

3.3 Linear System Solvers

The Ponderosa Computing Linear Algebra Excel Add-in provides the following linear system solvers:

Full rank square system solvers	LA.SYS.SLV, LA.SYS.SLVE
General least-squares system solvers (via LQ/QR)	LA.SYS.LS, LA.SYS.LSE
General least-squares system solvers (via SVD)	LA.SYS.LSS, LA.SYS.LSSE

3.3.1 Full Rank Square System Solvers

The worksheet functions LA.SYS.SLV(A,B) and LA.SYS.SLVE(A,B) return the solution matrix $\boldsymbol{X}$ to a real system of linear equations using equilibration and iterative refinement as needed:

$$\boldsymbol{A}\,\boldsymbol{X} \;=\; \boldsymbol{B}\;.$$

Here A is an n x n matrix and $\boldsymbol{B}$ is a n x p matrix. The solution matrix $\boldsymbol{X}$ is n x p. If A is not square or if matrix $\boldsymbol{B}$ does not have n rows these worksheet functions return the #VALUE! error.

The worksheet function LA.SYS.SLV(A,B) returns the matrix solution $\boldsymbol{X}$ as a n x p matrix.

The worksheet function LA.SYS.SLVE(A,B) returns the matrix solution $\boldsymbol{X}$ in the first n rows of an (n+1) x p augmented solution matrix $\boldsymbol{X_A}$. It also returns a forward error bound e for each column of the matrix solution $\boldsymbol{X}$ in the last row of the augmented matrix solution $\boldsymbol{X_A}$.

$$\boldsymbol{X_A} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \\ e_1 & e_2 & \cdots & e_p \end{bmatrix}.$$

For each column j of the augmented solution matrix, the forward error bound estimate e_j bounds the relative error in the computed solution column $\boldsymbol{x_j}$. If $\boldsymbol{x_j}$ is the computed solution column and $\boldsymbol{x_{jt}}$ is the true solution column, then the relative error for that column is bounded by:

$$\frac{\left\|\boldsymbol{x_j} - \boldsymbol{x_{jt}}\right\|_\infty}{\left\|\boldsymbol{x_j}\right\|_\infty} \leq e$$

That is, e_j is an estimated upper bound for the magnitude of the largest element in $\boldsymbol{x_j} - \boldsymbol{x_{jt}}$ divided by the magnitude of the largest element in $\boldsymbol{x_j}$. This estimate is almost always a slight overestimate of the true error.

These worksheet functions compute the solution matrix $\boldsymbol{X}$ by using routine `dgesvx_()` from CLAPACK v 3.2.1, using equilibration and iterative refinement as needed. If routine `dgesvx_()`determines the matrix A is exactly singular then these worksheet functions return the #NUM! error.

3.3.2 General Least-Squares System Solvers (via Orthogonal Factorizations)

The worksheet functions LA.SYS.LS(A,b) and LA.SYS.LSE(A,b) return the minimum-norm solution vector $\boldsymbol{x}$ to a real linear least squares problem:

$$\mathbf{min}_{\boldsymbol{x}}\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2 \;, with\ minimum\ \|\boldsymbol{x}\|_2\;.$$

These worksheet functions use a complete orthogonal factorization of $\boldsymbol{A}$. Here $\boldsymbol{A}$ is an m x n matrix which may be rank-deficient and $\boldsymbol{b}$ is an m-vector. The solution is the minimum-2-norm n-vector $\boldsymbol{x}$ achieving the minimum 2-norm residual.

These worksheet functions can solve for several (p) right hand side vectors $\boldsymbol{b_j}$ and solution vectors $\boldsymbol{x_j}$ in a single call. The p right hand side vectors $\boldsymbol{b_j}$ are stored as the columns of a m x p matrix $\boldsymbol{B}$ and LA.SYS.LS(A,B) and LA.SYS.LSE(A,B) return the p solution vectors $\boldsymbol{x_j}$ as the p columns of the n x p solution matrix $\boldsymbol{X}$. If matrix $\boldsymbol{B}$ does not have m rows this worksheet function returns the #VALUE! error.

These worksheet functions compute the solution matrix $\boldsymbol{X}$ by using routine `dgelsy_()` from CLAPACK v 3.2.1. This routine first computes a QR factorization of $\boldsymbol{A}$ with column pivoting. It then determines the effective rank r of $\boldsymbol{A}$ and then refactors a reduced system into an r x r triangular system which is then solved. If routine `dgelsy_()`determines the matrix $\boldsymbol{A}$ has effective rank 0 then these worksheet functions return the zero matrix.

The worksheet function LA.SYS.LS(A,B) returns the matrix solution $\boldsymbol{X}$ as a n x p matrix.

The worksheet function LA.SYS.LSE(A,B) returns the matrix solution $\boldsymbol{X}$ in the first n rows of an (n+1) x p augmented solution matrix $\boldsymbol{X_A}$. If the matrix $\boldsymbol{A}$ has m ≥ n (we have an overdetermined system) with full column rank then the worksheet function also returns a forward error bound for each column of the matrix solution $\boldsymbol{X}$ in the last row of the augmented matrix solution $\boldsymbol{X_A}$. Otherwise the worksheet function returns the values -1 in this last row.

The error bounds for the full column rank overdetermined case are computed in the manner described here:

http://www.netlib.org/lapack/lug/node82.html

3.3.3 General Least-Squares System Solvers (via SVD)

The worksheet functions LA.SYS.LSS(A,b) and LA.SYS.LSSE(A,b) return the minimum-norm solution vector $\boldsymbol{x}$ to a real linear least squares problem:

$$\mathbf{min}_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2 \ , with\ minimum\ \|\boldsymbol{x}\|_2 \ .$$

These worksheet functions use the singular value decomposition of $\boldsymbol{A}$. Here $\boldsymbol{A}$ is an m x n matrix which may be rank-deficient and $\boldsymbol{b}$ is an m-vector. The solution is the minimum-2-norm n-vector $\boldsymbol{x}$ achieving the minimum 2-norm residual.

These worksheet functions can solve for several (p) right hand side vectors $\boldsymbol{b_j}$ and solution vectors $\boldsymbol{x_j}$ in a single call. The p right hand side vectors $\boldsymbol{b_j}$ are stored as the columns of a m x p matrix $\boldsymbol{B}$ and LA.SYS.LSS(A,B) and LA.SYS.LSSE(A,B) return the p solution vectors $\boldsymbol{x_j}$ as the p columns of the n x p solution matrix $\boldsymbol{X}$. If matrix $\boldsymbol{B}$ does not have m rows this worksheet function returns the #VALUE! error.

These worksheet functions compute the solution matrix $\boldsymbol{X}$ by using routine `dgelss_()` from CLAPACK v 3.2.1. This routine first computes the singular value decomposition of $\boldsymbol{A}$. It then determines the effective rank r of $\boldsymbol{A}$, zeros out the insignificant portion of the SVD, and solves

the reduced system. If routine `dgelss_()`determines the matrix $\boldsymbol{A}$ has effective rank 0 then these worksheet functions return the zero matrix.

The worksheet function LA.SYS.LSS(A,B) returns the matrix solution $\boldsymbol{X}$ as a n x p matrix.

The worksheet function LA.SYS.LSSE(A,B) returns the matrix solution $\boldsymbol{X}$ in the first n rows of an (n+1) x p augmented solution matrix $\boldsymbol{X_A}$. If the matrix A has m ≥ n (we have an overdetermined system) with full column rank then the worksheet function also returns a forward error bound for each column of the matrix solution $\boldsymbol{X}$ in the last row of the augmented matrix solution $\boldsymbol{X_A}$. Otherwise the worksheet function returns the values -1 in this last row.

The error bounds for the full column rank overdetermined case are computed in the manner described here:

http://www.netlib.org/lapack/lug/node82.html

3.4 Singular Value Decomposition

The **singular value decomposition** of an m x n (real) matrix $\boldsymbol{A}$ is a factorization of $\boldsymbol{A}$ into the form:

$$\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{\Sigma}\,\boldsymbol{V}^T$$

where $\boldsymbol{U}$ is a m x m matrix, $\boldsymbol{V}^T$ is an n x n matrix, and $\boldsymbol{\Sigma}$ is a m x n rectangular diagonal matrix with p = min(m,n) nonnegative entries on the main diagonal.

The nonnegative diagonal entries of $\boldsymbol{\Sigma}$ are the **singular values** of $\boldsymbol{A}$, $\{\sigma_1, \sigma_2, \ldots \sigma_{\min(m,n)}\}$, arranged in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$.

The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal:

$$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{U}\,\boldsymbol{U}^T = \boldsymbol{I_m}\,, \qquad \boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{V}\,\boldsymbol{V}^T = \boldsymbol{I_n}\,,$$

where $\boldsymbol{I_m}$ and $\boldsymbol{I_n}$ denote the identity matrices of order m and n, respectively. The m columns of $\boldsymbol{U}$, $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \boldsymbol{u}_m\}$, are called the **left singular vectors** of $\boldsymbol{A}$. The n columns of $\boldsymbol{V}$, $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$, are called the **right singular vectors** of $\boldsymbol{A}$.

Since $\boldsymbol{\Sigma}$ is a m x n rectangular diagonal matrix with p = min(m,n), the singular value decomposition can be represented as the **thin singular value decomposition**:

$$\boldsymbol{A} = \boldsymbol{U_p}\,\boldsymbol{\Sigma_p}\,\boldsymbol{V_p}^T\,,$$

Where $\boldsymbol{U_p}$ is the m x p matrix consisting of the first p columns of $\boldsymbol{U}$, $\boldsymbol{\Sigma_p}$ is the p x p upper diagonal portion of $\boldsymbol{\Sigma}$, and $\boldsymbol{V_p}$ is the n x p matrix consisting of the first p columns of $\boldsymbol{V}$. The p columns of $\boldsymbol{U_p}$ and the p columns of $\boldsymbol{V_p}$ are orthonormal:

$$\boldsymbol{U_p}^T\boldsymbol{U_p} = \boldsymbol{I_p}\,, \qquad \boldsymbol{V_p}^T\boldsymbol{V_p} = \boldsymbol{I_p}\,,$$

where $\boldsymbol{I_p}$ denote the identity matrix of order p.

The Ponderosa Computing Linear Algebra Excel Add-in provides the following worksheet functions computing the thin singular value decomposition or parts thereof:

Singular value decomposition	LA.SVD
Singular values	LA.SNGVL.C, LA.SNGVL.R
Singular values with error bound	LA.SNGVLE.C, LA.SNGVLE.R
Left or right singular vectors	LA.LSNGVC, LA.RSNGVC

3.4.1 Singular Value Decomposition

The worksheet function LA.SVD(A) returns the p = min(m,n) singular values, their associated left singular vectors, and their associated right singular vectors of $\boldsymbol{A}$ in the p columns of a (m+n+1) x p matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \begin{bmatrix} \sigma_1 & \sigma_2 & \cdots & \sigma_p \\ u_{11} & u_{21} & \cdots & u_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1m} & u_{2m} & \cdots & u_{pm} \\ v_{11} & v_{21} & \cdots & v_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{1n} & v_{2n} & \cdots & v_{pn} \end{bmatrix}$$

The first row of $\boldsymbol{S}$ contains the p singular values of $\boldsymbol{A}$ in descending order, $\{\sigma_1, \sigma_2, \ldots \sigma_p\}$. The columns of the next m rows contain the first p left singular vectors, $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \boldsymbol{u}_p\}$. The columns of the last n rows contain the first p right singular vectors, $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_p\}$.

The worksheet function LA.SVD(A) computes the singular value decomposition of $\boldsymbol{A}$ using routine `dgesvd_()` from CLAPACK v 3.2.1. Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values and singular vectors of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of $\boldsymbol{A}$ then this worksheet function returns the #NUM! error.

3.4.2 Singular Values

The worksheet functions LA.SNGVL.C(A) and LA.SNGVL.R(A) return the singular values of a m x n matrix $\boldsymbol{A}$ in descending order. The worksheet function LA.SNGVL.C(A) returns the p =

min(m,n) singular values of $\boldsymbol{A}$, $\{\sigma_1, \sigma_2, \ldots \sigma_p\}$, in descending order as an p-element column vector $\boldsymbol{s}$:

$$\boldsymbol{s} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_p \end{bmatrix} .$$

The worksheet function LA.SNGVL.R(A) returns the singular values as an p-element row vector.

These worksheet functions compute the singular values of $\boldsymbol{A}$ using routine `dgesvd_()` from CLAPACK v 3.2.1. Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of $\boldsymbol{A}$ then these worksheet functions return the #NUM! error.

3.4.3 Singular Values With Error Bound

Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$ be the p = min(m,n) singular values of an m x n matrix A.

The worksheet functions LA.SNGVLE.C(A) and LA.SNGVLE.R(A) return the singular values of a m x n matrix $\boldsymbol{A}$ in descending order followed by a computation error bound. The worksheet function LA.SNGVLE.C(A) returns the p = min(m,n) singular values of $\boldsymbol{A}$, $\{\sigma_1, \sigma_2, \ldots \sigma_p\}$, in descending order by a forward error bound estimate e as an (p+1)-element column vector s_A:

$$\boldsymbol{s_A} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_p \\ e \end{bmatrix} .$$

The worksheet function LA.SNGVLE.R(A) returns the singular values and forward error bound as an (p+1)-element row vector.

The forward error bound estimate e bounds the absolute error in the computed singular values. If σ_i is the i-th computed singular value and σ_{it} is the i-th true singular value, then the absolute error in σ_i is bounded by:

$$|\sigma_i - \sigma_{it}| \leq e$$

This estimate is almost always a slight overestimate of the true error.

These worksheet functions compute the singular values of $\boldsymbol{A}$ using routine `dgesvd_()` from CLAPACK v 3.2.1. Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this

algorithm fails to find all the singular values of $\boldsymbol{A}$ then these worksheet functions return the #NUM! error.

The singular values forward error bound e is:

$$e = \max(sfmin\,, \max(m,n) * eps * \sigma_1) \ .$$

This error bound is described here: http://www.netlib.org/lapack/lug/node97.html, but this methods uses p(m,n) = max(m,n) in place of p(m,n) = 1 as the "modestly growing function of n" and uses machine epsilon dlamch_("P") in place of dlamch_("E"). The computation of e is similar to that found in the example program: http://www.nag.com/lapack-ex/node128.html .

3.4.4 Left Singular Vectors

The worksheet function LA.LSNGVC(A) returns the p = min(m,n) singular values and first p left singular vectors of a m x n matrix $\boldsymbol{A}$ in the p columns of a (m+1) x p matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \begin{bmatrix} \sigma_1 & \sigma_2 & \cdots & \sigma_p \\ u_{11} & u_{21} & \cdots & u_{p1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1m} & u_{2m} & \cdots & u_{pm} \end{bmatrix}$$

The first row of $\boldsymbol{S}$ contains the p singular values of $\boldsymbol{A}$, $\{\sigma_1, \sigma_2, \ldots \sigma_p\}$, in descending order. The columns of the remaining m rows contain the first p left singular vectors, $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \boldsymbol{u}_p\}$.

The worksheet function LA.LSNGVC(A) computes the singular value decomposition of $\boldsymbol{A}$ using routine `dgesvd_()` from CLAPACK v 3.2.1. Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values and singular vectors of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of $\boldsymbol{A}$ then this worksheet function returns the #NUM! error.

3.4.5 Right Singular Vectors

The worksheet function LA.RSNGVC(A) returns the p = min(m,n) singular values and first p right singular vectors of a m x n matrix $\boldsymbol{A}$ in the p rows of a p x (n+1) matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \begin{bmatrix} \sigma_1 & v_{11} & v_{12} & \cdots & v_{1n} \\ \sigma_2 & v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_p & v_{p1} & v_{p2} & \cdots & v_{pn} \end{bmatrix}$$

The first column of $\boldsymbol{S}$ contains the p singular values of $\boldsymbol{A}$ in descending order. The rows of the remaining n columns of $\boldsymbol{S}$ contain the first p right singular vectors, $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_p\}$.

The worksheet function LA.RSNGVC(A) computes the singular value decomposition of $\boldsymbol{A}$ using routine `dgesvd_()` from CLAPACK v 3.2.1. Routine `dgesvd_()` uses routine `dbdsqr_()` to compute the singular values and singular vectors of an (upper or lower) bidiagonal matrix using the implicit zero-shift QR algorithm. If this algorithm fails to find all the singular values of $\boldsymbol{A}$ then this worksheet function returns the #NUM! error.

3.5 Eigenvalues and Eigenvectors

A **right eigenvector** of an n x n symmetric (real) matrix $\boldsymbol{A}$ is a nonzero n-vector $\boldsymbol{v}$ such that the matrix product:

$$\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v} \ .$$

Here λ is a real number (scalar) called the **eigenvalue** of $\boldsymbol{A}$ corresponding to the right eigenvector $\boldsymbol{v}$. The set of all eigenvectors of $\boldsymbol{A}$, $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$, each paired with its corresponding eigenvalue, $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$, is called the **eigensystem** of the matrix $\boldsymbol{A}$. It can be expressed as:

$$\boldsymbol{A}\,\boldsymbol{V} = \boldsymbol{V}\,\boldsymbol{\Lambda}.$$

Here $\boldsymbol{\Lambda}$ is a n x n diagonal matrix with diagonal entries $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$ and $\boldsymbol{V}$ is a n x n matrix with columns $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$.

The Ponderosa Computing Linear Algebra Excel Add-in provides the following worksheet functions computing the eigenvalues and right eigenvectors:

Eigenvalues	LA.EGNVL.C, LA.EGNVL.R
Eigenvalues with error bound	LA.EGNVLE.C, LA.EGNVLE.R
Eigenvalues and right eigenvectors	LA.REGNVC

3.5.1 Eigenvalues

The worksheet functions LA.EGNVL.C(A) and LA.EGNVL.R(A) return the (real) eigenvalues of an n x n symmetric (real) matrix $\boldsymbol{A}$. The worksheet function LA.EGNVL.C(A) returns the eigenvalues of $\boldsymbol{A}$, $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$, in ascending order, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, as an n-element column vector $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} .$$

The worksheet function LA.EGNVL.R(A) returns the eigenvalues of $\boldsymbol{A}$ as an n-element row vector. If $\boldsymbol{A}$ is not symmetric these worksheet functions return the #NUM! error.

These worksheet functions compute the eigenvalues of $\boldsymbol{A}$ using routine `dsyev_()` from CLAPACK v 3.2.1. Routine `dsyev_()` uses routine `dsterf_()` which computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm. If this algorithm fails to find all of the eigenvalues of $\boldsymbol{A}$ in at most 30*n iterations then these worksheet functions return the #NUM! error.

3.5.2 Eigenvalues with Error Bounds

The worksheet functions LA.EGNVLE.C(A) and LA.EGNVLE.R(A) return the (real) eigenvalues of an n x n symmetric (real) matrix $\boldsymbol{A}$ followed by a forward error bound e. The worksheet function LA.EGNVLE.C(A) returns the eigenvalues of $\boldsymbol{A}$, $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$, in ascending order, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, followed by a forward error bound estimate e as an (n+1)-element column vector $\boldsymbol{\lambda_A}$:

$$\boldsymbol{\lambda_A} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ e \end{bmatrix} .$$

The worksheet function LA.EGNVLE.R(A) returns the eigenvalues and error bound as an (n+1)-element row vector. If $\boldsymbol{A}$ is not symmetric these worksheet functions return the #NUM! error.

The forward error bound estimate e bounds the absolute error in the computed eigenvalues. If λ_i is the i-th computed eigenvalue and λ_{it} is the i-th true eigenvalue, then the absolute error in λ_i is bounded by:

$$|\lambda_i - \lambda_{it}| \leq e$$

This estimate is almost always a slight overestimate of the true error.

These worksheet functions compute the eigenvalues of $\boldsymbol{A}$ using routine `dsyev_()` from CLAPACK v 3.2.1. Routine `dsyev_()` uses routine `dsterf_()` which computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm. If this algorithm fails to find all of the eigenvalues of $\boldsymbol{A}$ in at most 30*n iterations then these worksheet functions return the #NUM! error.

The Ponderosa Computing Linear Algebra Excel Add-in uses the eigenvalues error bound e is:

$$e = \max(sfmin\,,\ n * eps * \max_i(|\lambda_i|)) \quad .$$

This error bound is described here: http://www.netlib.org/lapack/lug/node90.html, but this method uses p(n) = n in place of p(n) = 1 as the "modestly growing function of n" and uses machine epsilon dlamch_("P") in place of dlamch_("E"). The computation of e is similar to that found in the example program: http://www.nag.com/lapack-ex/node71.html.

3.5.3 Eigenvalues and Right Eigenvectors

The worksheet function LA.REGNVC(A) returns the eigenvalues and right eigenvectors of an n x n symmetric matrix A in the n columns of the (n+1) x n matrix matrix $\boldsymbol{V_A}$.

$$\boldsymbol{V_A} = \begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_n \\ v_{11} & v_{21} & \cdots & v_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{1n} & v_{2n} & \cdots & v_{nn} \end{bmatrix}$$

The first row of $\boldsymbol{V_A}$ contains the n eigenvalues of $\boldsymbol{A}$, $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$, in ascending order, $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The columns of the remaining n rows of $\boldsymbol{V_A}$ contain the corresponding n right eigenvectors of $\boldsymbol{A}$, $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$. If $\boldsymbol{A}$ is not symmetric this worksheet function returns the #NUM! error.

This worksheet function computes the eigenvalues of $\boldsymbol{A}$ using routine `dsyev_()` from CLAPACK v 3.2.1. Routine `dsyev_()` uses routine `dsterf_()` which computes all eigenvalues of a symmetric tridiagonal matrix using the Pal-Walker-Kahan variant of the QL or QR algorithm. If this algorithm fails to find all of the eigenvalues of $\boldsymbol{A}$ in at most 30*n iterations then this worksheet function returns the #NUM! error.

3.6 Cholesky Factorization

The **upper triangular Cholesky factorization** (or decomposition) of an n x n symmetric positive definite matrix A is the unique factorization of $\boldsymbol{A}$ into

$$\boldsymbol{A} = \boldsymbol{U}^T \boldsymbol{U}$$

where $\boldsymbol{U}$ is an n x n upper triangular matrix with positive entries on the diagonal.

The **lower triangular Cholesky factorization** (or decomposition) of an n x n symmetric positive definite matrix A is the unique factorization of $\boldsymbol{A}$ into

$$\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{L}^T$$

where $\boldsymbol{L}$ is an n x n lower triangular matrix with positive entries on the diagonal.

The Ponderosa Computing Linear Algebra Excel Add-in provides the following worksheet functions computing the Cholesky factorizations of a symmetric positive definite matrix:

Upper Cholesky factorization	LA.CHOL.U
Lower Cholesky factorization	LA.CHOL.L

3.6.1 Upper and Lower Cholesky Factorizations

The worksheet function LA.CHOL.U(A) returns the upper triangular Cholesky factorization of an n x n symmetric positive definite matrix ***A*** as an n x n upper triangular matrix ***U*** with positive entries on the diagonal.

The worksheet function LA.CHOL.L(A) returns the lower triangular Cholesky factorization of an n x n symmetric positive definite matrix ***A*** as an n x n lower triangular matrix ***L*** with positive entries on the diagonal.

These worksheet functions compute the Cholesky factorization of ***A*** using routine `dpotrf_()` from CLAPACK v 3.2.1. If this algorithm determines the matrix ***A*** is not positive definite then these worksheet functions return the #NUM! error.

3.7 Alternative Matrix Multiplication and Transformation Operations

The Ponderosa Computing Linear Algebra Excel Add-in provides the following worksheet functions serving as alternatives to the built-in Excel matrix multiplication and transpose array functions:

Matrix multiplication	LA.MUL
Matrix transpose multiplication	LA.TMUL, LA.MULT
Matrix transpose	LA.TRN

3.7.1 Matrix Multiplication

The worksheet function LA.MUL(A,B) returns the product of two matrices. The matrix ***A*** must be m x n and the matrix B must be n x p with m, n, p > 0. Otherwise this worksheet function returns the #VALUE! error.

This worksheet function computes the matrix product using routine `dgemm_()` from the CLAPACK v 3.2.1 BLAS package. LA.MUL(A,B) is an alternative to the Excel MMULT(A,B) built-in function.

3.7.2 Matrix Transpose Multiplication

The worksheet function LA.TMUL(A,B) returns the product of the transpose of the first matrix A times the second matrix B. The matrix A must be n x m and the matrix B must be n x p with m, n, p > 0. Otherwise this worksheet function returns the #VALUE! error.

The worksheet function LA.MULT(A,B) returns the product of the first matrix A times the transpose of the second matrix B. The matrix A must be m x n and the matrix B must be p x n with m, n, p > 0. Otherwise this worksheet function returns the #VALUE! error.

These worksheet functions compute the matrix product using routine `dgemm_()` from the CLAPACK v 3.2.1 BLAS package.

3.7.3 Matrix Transpose

The worksheet function LA.TRN(A) returns the transpose of an m x n matrix A. Both m and n must be positive or this worksheet function returns the #VALUE! error. This worksheet function is an alternative to the Excel TRANSPOSE(A) built-in function.

4 Installing and Activating this Excel Add-in

There are three steps to enabling the Ponderosa Computing Linear Algebra Excel add-in to be used in Excel spreadsheets. These are (1) installing the add-in on your computer, (2) registering the add-in with Excel, and (3) activating the add-in in Excel.

Installing the Excel Add-In on your Computer

1. Go to the Ponderosa Computing downloads page (http://www.ponderosacomputing.com/downloads/) and identify the installation package for the Ponderosa Computing Linear Algebra Excel add-in.
2. Click on the link to the Excel Add-in installation package. In the **Opening** dialog box that appears click **Save File**, and save the installation package at a convenient location on your computer.
3. On your computer navigate to the location of the saved installation package and double-click on the file.
4. In the **Open File – Security Warning** dialog you can click on the Publisher link to verify the Digital Signature. Then click **Run**.
5. The installation program will guide you through the rest of the process of installing the Excel Add-in on your computer. The add-in XLL will be installed in C:\Program Files (x86)\Ponderosa Computing\PcLinAlgXLL.
6. You may delete the downloaded installation package.

Registering and Activating the Excel Add-In in Excel

1. Open a 32-bit version of Microsoft Excel.
2. Click the **File** tab, click **Options**, and then click the **Add-Ins** category.
3. In the Manage box, select **Excel Add-ins**, and then click **Go**.
4. The **Add-Ins** dialog box appears. Click **Browse**, locate and select the Excel Add-in XLL C:\Program Files (x86)\Ponderosa Computing\PcLinAlgXLL\PcLinAlgXLL.xll, and click **OK**.
5. The Excel Add-in has now been registered with Excel. The **Add-Ins** dialog box reappears and now contains a checked entry for the add-in. If you uncheck the entry for the add-in and click **OK**, or if you click **Cancel** the add-in will remain registered, but not activated. If you leave the entry checked and click **OK** the add-in will also be activated and a confirmation message will appear.

Activating a Registered Excel Add-In in Excel

If the Excel Add-in has been registered with, but not activated in Excel, then follow these steps to activate the add-in so it is accessible from Excel worksheets:

1. Open Excel (if not already open).
2. Click the **File** tab, click **Options**, and then click the **Add-Ins** category.
3. In the **Manage** box, click **Excel Add-ins**, and then click **Go**.

4. The **Add-Ins** dialog box appears. Select the check box next to the add-in you want to activate, and click **OK**.
5. A confirmation message will appear. Dismiss it.

The Excel Add-in is now loaded into Excel.

The activated Excel Add-in will be loaded into Excel whenever Excel is started until it is inactivated. Also, the Excel Add-in command menu will appear in the Menu Commands section of the Add-Ins tab on the Ribbon.

5 Inactivating and Uninstalling this Excel Add-in

Should you decide to inactivate, unregister, uninstall, or remove Ponderosa Computing Linear Algebra Excel Add-in from Excel or your computer, you can do so by following these instructions.

Inactivating the Excel Add-In

If an activated Excel Add-in is inactivated, it will remain registered with Excel, but not loaded when Excel starts. Follow these steps to inactivate the Excel Add-in:

1. Open Excel (if not already open).
2. Click the **File** tab, click **Options**, and then click the **Add-Ins** category.
3. In the **Manage** box, click **Excel Add-ins**, and then click **Go**.
4. In the **Add-Ins** dialog box clear the check box next to the add-in that you want to inactivate and then click **OK**.
5. A confirmation message will appear. Dismiss it.

The command menu for an inactivated Excel Add-in will not appear in the Menu Commands section of the Add-Ins tab on the Ribbon. You can reactivate a registered Excel Add-in using the instructions above.

Unregistering the Excel Add-In from Excel

Inactivating the Excel Add-in does not unregister it from Excel. To unregister the add-in from Excel you must first uninstall the add-in from your computer and then remove it from Excel. These steps are described next.

Uninstalling the Excel Add-In from your Computer

Inactivating the Excel Add-in does not remove it from your computer. To remove the add-in from your computer, you must uninstall it.

1. Exit Excel (click the **File** tab, and then click **Exit**).
2. Open the **Control Panel**.
3. In the Control Panel, click **Programs** and **Uninstall a program**.
4. Click the name of the Excel Add-in in the list of installed programs, and then click the **Uninstall** button.
5. The **Windows Installer** dialog will appear. Click **Yes**.
6. The installation program will guide you through the rest of the process of uninstalling the Excel add-in from your computer.

Removing an Uninstalled Add-In from Excel

1. Open Excel.
2. Click the **File** tab, click **Options**, and then click the **Add-Ins** category.
3. In the **Manage** box, click **Excel Add-ins**, and then click **Go**.

4. The **Add-Ins** dialog box appears. Select the check box next to the add-in you want to unregister.
5. Excel will advise you that it cannot find the add-in and ask you whether to delete it from the list. Click **Yes**.
6. The add-in will be removed from the list of available add-ins.

6 References

[1] Anderson, E. et al., LAPACK Users' Guide, Third Edition (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999) ISBN 0-89871-447-8.

[2] LAPACK on the Netlib Repository at UTK and ORNL. http://www.netlib.org/lapack/

[3] CLAPACK (f2c'ed version of LAPACK) version 3.2.1 on the Netlib Repository at UTK and ORNL. http://www.netlib.org/clapack/. See clapack-3.2.1-CMAKE.tgz.

[4] CLAPACK for Windows at the University of Tennessee Innovative Computing Laboratory. http://icl.cs.utk.edu/lapack-for-windows/clapack/

[5] Wikipedia: IEEE floating point. This discussion includes references to the standard publications.

[6] Microsoft. "Floating-point arithmetic may give inaccurate results in Excel." link

[7] LAPACK/ScaLAPACK Development Forum. http://icl.cs.utk.edu/lapack-forum/

7 License Notice

The Ponderosa Computing Linear Algebra Excel Add-in was built using the CLAPACK implementation of LAPACK, a freely-available software package from netlib at http://www.netlib.org/lapack. The license used for the LAPACK software is the modified BSD license:

```
Copyright (c) 1992-2013 The University of Tennessee and The University
                        of Tennessee Research Foundation.  All rights
                        reserved.
Copyright (c) 2000-2013 The University of California Berkeley. All
                        rights reserved.
Copyright (c) 2006-2013 The University of Colorado Denver.  All rights
                        reserved.

$COPYRIGHT$

Additional copyrights may follow

$HEADER$

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer listed
  in this license in the documentation and/or other materials
  provided with the distribution.

- Neither the name of the copyright holders nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

The copyright holders provide no reassurances that the source code
provided does not infringe any patent, copyright, or any other
intellectual property rights of third parties.  The copyright holders
disclaim any liability to any recipient for claims brought against
recipient by any third party for infringement of that parties
intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

