AMCM043 XML and Structured Information

Course Work 1

Student ID:089583717

Contents

Introduction

This document forms part of the documentation for the AMCM043 Course Work 1. The objective is to give background information of the problem, details on the testing strategy and formal documentation on the produced Extensible Stylesheet Transformation artefact. Much of the theory behind the course work has been taken from the book "Introduction To Algorithms: Second Edition, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein". Due credit is given to their explanation of the LUP Decomposition method for solving linear equations.

Linear Equation Overview

Content for this section is taken from the above stated "Introduction To Algorithms: Second Edition" book.

A set of linear equations in *n* unknowns $x_1, x_2, \ldots, x_n$ can be represented as:

$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,$

$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$

.

.

.

$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$

A set of values for $x_1, x_2, \ldots, x_n$ that satisfy all of the above equations simultaneously is said to be a solution to these equations.

Furthermore, the above set of equations can be represented in a matrix format as a matrix-vector equation:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \; \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Now, let $A = (a_{ij})$, $x = (x_i)$, and $b = (b_i)$ and we can re-state the above as:

$Ax = b$

Overview of LUP Decomposition Method for determining Linear Equations

The concept behind the LUP decomposition is to find three $n \times n$ matrices, L, U, and P such that:

$P\,A = L\,U$

where

- A is the matrix described in the above linear equation overview,
- L is a unit lower-triangular matrix,

- U is an upper-triangular matrix, and
- P is a permutation matrix.

The idea behind this is that converting the set of linear equations into a triangular set of matrices makes the calculation of the equations easier. This is achieved through the following algebraic manipulation.

If Ax = b and P A = L U, then

P Ax = Pb.

Substituting L U for P A, we have

L U x = Pb

Now, letting y = Ux, we have

Ly = Pb

So, by solving the unit lower-triangular system, L, and then using forward substitution to calculate y we are in a position to produce a solution set for the linear equations given that y = Ux (where x represents the set of solutions to the linear equation set).

The full explanation and theory behind the LUP algorithm can be found within the referenced book “Introduction To Algorithms: Second Edition”, and the interest reader is directed there for further reading.

Course Work Structure

The course work contains the following artefacts:

- Explanation document (this document)
- XSLT style sheet “linearEquation.xsl” that performs the linear equation calculation
- Set of test XML documents used in the testing of the above linearEquation XSLT

All of the technical artefacts are contained within their own file but also shown in this document appendix for reference.

linearEquation Explanation

This section provides formal technical documentation for the linearEquation extensible style sheet transformation artefact. All transformations using style sheet are done under the Saxon-B Version 9 transformation standard.

There is a given XML Schema for the input XML document, but a summary of how this format maps to the above linear equation format is also given here.

Input Document

The input XML consists of the one root linearEquation node. This contains one LHS node and one RHS node. The LHS node contains multiple row nodes, each having multiple entry nodes. The RHS only contains entry nodes. An example of this is:

```
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

Figure 1 – Example XML Input Document

In this example the LHS contains the A type matrix entries, and the RHS the values, or b matrix. From earlier, recall that linear equations can be written in the form:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Ax=b

In this format the sample XML document would resolve to

$$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$$

where

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$b = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$$

This gives the following set of linear equations:

$x_1 + 2x_2 = 3$

$3x_1 + 4x_2 + + 4x_3 = 7$

$5x_1 + 6x_2 + 3x_3 = 8$

Initial Validation

The transformation starts by matching at the root document level. The input XML document that represents the linear equation to be solved is first passed through a validation routine; named template "validation-routine". This style sheet uses a "firewall" type approach to validation. Meaning that prior to any LUP decomposition processing the input XML document is tested against several scenarios to check for validity. If any of the validity checks find an error, then the style sheet will output an error message relevant to the error found and no further processing will continue. This type of validation has the benefits of allowing a common validation routine where all XML document validation logic can be grouped and of keeping the remaining named template routines free from validation requirements. This allocation of responsibility helps to keep the code readable and assist in future maintenance.

The validation routine is explored in more detail within the Testing Strategy section, but a listing of the validation messages is also given here:

- There should be one, and only one, linearEquation node.
- There should be one and only one LHS node.
- There should be one and only one RHS node.
- The number of entries in the LHS rows should be equal.
- The number of entries in the LHS columns should be equal to those in the rows.

- The number of LHS rows should be equal to the number of RHS columns.
- Each entry node should contain a numeric attribute value.
- LHS Columns containing only zeros are not allowed.
- LHS Rows containing only zeros are not allowed.
- Singular Matrices are not allowed.

Permutation Matrix Processing

The next step adds in the Permutation Matrix processing, required to keep track of any swapping of rows. The style sheet does this by adding on an additional index type column to the left of the A Matrix. This allows the procedure to keep track of any changes to the row positions. The result of this operation is held in variable “Matrix-A”.

Save Matrix b

The values given on the RHS node are saved in variable “Matrix-b”.

Transform the coefficient, A Matrix.

This is one of the main routines, carried out by named template “transform-coefficient-matrix”, that performs the transformation of the given A Matrix. The result of this procedure is used to produce the two triangular matrices, L & U.

The procedure receives as an input parameter the Matrix-A variable, which now contains the permutation index column. There is an initial check to determine if the matrix is fully processed, and if so it is returned. If not, the following processing takes place.

The input matrix is adjusted for the pivot position, taking the largest absolute value of the A matrix column one values as that pivot. Next variables v and w are created to hold the upper most row vector and left most (excluding the permutation index column) column. Both these variables do not contain the upper left hand matrix entry.

The procedure then performs the LUP decomposition as outlined in “Introduction To Algorithms: Second Edition”. The return variable is the transformed matrix A, still including the permutation index column. In addition, internal program documentation is available within the linearEquation style sheet.

Create Unit Lower-Triangular Matrix L

The processed A Matrix is used to create the unit lower-triangular matrix L.

Create Upper-Triangular Matrix U

The processed A Matrix is used to create the upper-triangular matrix U.

Calculate the Product Pb

The permutation is applied to the matrix b to give the equation term Pb (recall that L U x = Pb).

Calculate the Y Matrix

Recall that Ly = Pb. By using forward substitution of the y variable, the unit-lower triangular matrix L and the above product Pb are used to derive a matrix value for Y.

Calculate the X Matrix

Recall that Ux = y. By using backward substitution of the x variable, the upper-triangular matrix U and the Y Matrix are used to derive the solution matrix for the linear equation set.

Presentation of results

The course work gives no clear guidance on the presentation of the results; therefore the following structure is suggested:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation>
   <solution>
      <entry value="-1.4"/>
      <entry value="2.2"/>
      <entry value="0.6"/>
   </solution>
</linearEquation>
```

The template can be easily modified to accommodate any different type of results presentation. It is also noted here that the transformation process can introduce small rounding errors. To cater for this, the results are rounded to five decimal positions. This is easily removable, or maintainable if needed, by changing the variable $rounding-factor.

The actual figures given are the solution values to the example matrices given earlier.

Testing Strategy

The testing strategy is divided into two broad categories:

- Validation of the input XML document, and
- Verification of the results given by the LUP decomposition linearEquation extensible style sheet transformation artefact.

Validation Routine

Several scenarios have been identified that would result in an invalid XML input document being given to the transformation routine. This section describes each of these scenarios, giving also a sample XML input document used to test the validation scenario.

Scenario 1

Validation Scenario:	There should be one, and only one, linearEquation node.
Justification:	Although this should be covered in the schema validation, the routine none the less checks for this as it is possible to submit an XML input document without schema checks, although the transformation *should* pick this up as a not well formed input document.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 2

Validation Scenario:	There should be one and only one LHS node
Justification:	The set of linear equations are represented by the equation Ax=b. A second LHS node would imply A_1A_2x=b, which could be reduced to Ax=b prior to input.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 3

Validation Scenario:	There should be one and only one RHS node
Justification:	The set of linear equations are represented by the equation Ax=b. A second LHS node would imply Ax=b_1b_2, which could be reduced to Ax=b prior to input.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}\begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 4

Validation Scenario:	The number of entries in the LHS rows should be equal
Justification:	The LUP decomposition is designed to find three *n* x *n* matrices L, U and P to help solve the system of linear equations. Each row on the LHS matrix represents the coefficients given to the *x* values. To ensure a satisfactory outcome the number of elements in each row should be *n*.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 & \\ 3 & 4 & & \\ 5 & 6 & 3 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 5

Validation Scenario:	The number of entries in the LHS columns should be equal to those in the rows.
Justification:	This is really a specialization of the Scenario 4 error checking, included for completeness. This checks that the given A Matrix is indeed an *n* x *n* type.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 6

Validation Scenario:	The number of LHS rows should be equal to the number of RHS columns
Justification:	Given that the matrix representation of a set of linear equations is equivalent to: $a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,$ $a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$ . . . $a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$ This validation routine checks that the above set of equations have been correctly encoded within the set of matrices.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$

Scenario 7

Validation Scenario:	Each entry node should contain a numeric attribute value
Justification:	The "value" attribute is used to hold the numeric value of the linear equation encoding. This validation checks that each of these attribute entries holds a numeric value. Part of this validation should be also done by validation against the supporting schema. But it may still be possible to bypass this, therefore a further validation check is given. Assumptions could be made with null entries, for example treat null entries as zero, but this introduces a level of ambiguity that may not be appropriate.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 2 & 0 \\ 3 & & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 8

Validation Scenario:	LHS Columns containing only zeros are not allowed
Justification:	If any column is zero, then the determinant of the matrix is also 0 leading to a singular matrix. The LUP decomposition only works for non-singular matrices. This test could have been rolled into Scenario 10 below, the general test for a singular matrix. However, this particular type of property of a singular matrix would be easier for a user to recognize.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 1 & 0 & 0 \\ 3 & 0 & 4 \\ 5 & 0 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 9

Validation Scenario:	LHS Rows containing only zeros are not allowed
Justification:	If any row is zero, then the determinant of the matrix is also 0 leading to a singular matrix. The LUP decomposition only works for non-singular matrices. This test could have been rolled into Scenario 10 below, the general test for a singular matrix. However, this particular type of property of a singular matrix would be easier for a user to recognize.
XML Test File	
Input Matrix Representation:	$\begin{pmatrix} 0 & 0 & 0 \\ 3 & 4 & 4 \\ 5 & 6 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Scenario 10

Validation Scenario:	Singular Matrices are not allowed
Justification:	This is a general test for singular matrices. The LUP decomposition only works for non-singular matrices.
XML Test File	eqtest10.xml

Input Matrix Representation:	$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 0 & 0 \\ 4 & 6 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 7 \\ 8 \end{pmatrix}$

Verification Sample Files

This section gives some sample files that have passed validation and have been solved using the given linearEquation style sheet. All file contents can be found in Appendix D – Sample Valid Files.

Sample 1

This is a trivial one-row sample using file eqpass1.xml. This sample represents:

$4x_1 = 10$

Where:

$x_1 = 2.5$

Sample 2

This is a 2 x 2 matrix using file eqpass2.xml. This sample represents:

$x_2 = 5.3$
$x_1 + 2x_2 = 2.8$

Where:

$x_1 = -7.8$
$x_2 = 5.3$

Sample 3

This is a 3 x 3 matrix using file eqpass3.xml. This sample reprsents:

$6x_1 - 3x_2 +6x_3 = 72$
$x_1 + 11x_2 + x_3 = -57$
$4x_1 + 9x_2 + 2x_3 = -28$

Where

$x_1 = 4$
$x_2 = -6$
$x_3 = 5$

Sample 4

This is a 4 x 4 matrix using eqpass4.xml. This sample represents:

$2x_1 + 11x_2 + 7x_3 + 9x_4 = -204$
$9x_1 + x_2 + 5x_3 -13x_4 = 147$
$-10x_1 + 11x_2 – x_3 + 10x_4 = -228$
$-3x_1 - 6x_2 + 12x_3 + 11x_4 = 21$

Where

$x_1 = -3$
$x_2 = -12$
$x_3 = 6$

$x_4 = -12$

Appendix A – File Inventory

The course work consists of several files. The inventory list of these is listed below.

File Name	Description
AMCM043CourseWork1.pdf	The course work documentation.
linearEquation.xsl	The Extensible Style Sheet Transformation file that performs the LUP decomposition method for solving linear equations.
eqtest1.xml	The XML input file to test validation scenario 1.
eqtest2.xml	The XML input file to test validation scenario 2.
eqtest3.xml	The XML input file to test validation scenario 3.
eqtest4.xml	The XML input file to test validation scenario 4.
eqtest5.xml	The XML input file to test validation scenario 5.
eqtest6.xml	The XML input file to test validation scenario 6.
eqtest7.xml	The XML input file to test validation scenario 7.
eqtest8.xml	The XML input file to test validation scenario 8.
eqtest9.xml	The XML input file to test validation scenario 9.
eqtest10.xml	The XML input file to test validation scenario 10.
eqpass1.xml	The XML input file for verification pass 1.
eqpass2.xml	The XML input file for verification pass 2.
eqpass3.xml	The XML input file for verification pass 3.
eqpass4.xml	The XML input file for verification pass 4.

Appendix B – linearEquation Source Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   exclude-result-prefixes="xs" xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <!--
      ******************************** Start of Routine ********************************
      Name: Main Processing Routine - Template match on root.

      Description: Performs the main line processing for the linearEquation XSLT.

      Student ID: 089583717

      Creation Date: 29 November 2008

      Modification History

      Parameters in use


      Variables in use
      A-posn: An xs:integer based variable used to store the position of A-Matrix whilst constructing Matrix-U.
      Error: A string based variable used to return any error found within the "validation-routine".
      index-posn: An xs:integer holding the current position of the Matrix-Y row to be evaluated.
      Matrix-A: A row based variable that contains the coefficient Matrix to be processed.
      Matrix-b: A RHS based variable used to store the results Matrix b.
      Matrix-L: A row based variable used to store the contents of the unit lower-triangular Matrix L.
      Matrix-Pb: A RHS based parameter that hold the product of the Permuatation Matrix and the Linear Equation results matrix, b.
      Matrix-U: A row based variable used to store the contents of the upper-triangular Matrix U.
```

```
    Matrix-X-initial: An entry based variable that holds the initial value of the Matrix-X.
    Matrix-Y-initial: An entry based variable that holds the initial value of the Matrix-Y.
    No-Error: Default no error found return code.
    rounding-factor: An xs:integer based variable used to control the degree of rounding in the presented equation solutions.
  -->

  <xsl:template match="/">

    <!-- This variable control the degree of rounding to be performed on the returned solutions of the linear equations
      The factor of 10 equates to the number of decimal positions that the solutions are rounded to.  This is required since some
      rounding errors are introduced during the LUP decomposition.
    -->
    <xsl:variable name="rounding-factor" as="xs:integer">
      100000
    </xsl:variable>

    <!-- Validation Routine -->
    <xsl:variable name="Error" as="xs:string">
      <xsl:call-template name="validation-routine">
        <xsl:with-param name="linearEquation" select="//linearEquation"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:variable name="No-Error" as="xs:string"> "No-Error" </xsl:variable>

    <xsl:choose>
      <xsl:when test="$Error != $No-Error">
        <!-- Error found -->
        <error>
          <xsl:copy-of select="$Error"/>
        </error>
      </xsl:when>
```

```
<xsl:otherwise>
   <!-- No error, therefore continue processing -->

   <!-- Add on permutation column as an index column on the coefficient Matrix A -->
   <xsl:variable name="Matrix-A" as="element(row)*">
      <xsl:for-each select="//row">
         <row>
            <xsl:variable name="posn" as="xs:integer">
               <xsl:value-of select="position()"/>
            </xsl:variable>
            <entry value="{$posn}"/>
            <xsl:for-each select="entry">
               <entry value="{@value}"/>
            </xsl:for-each>
         </row>
      </xsl:for-each>
   </xsl:variable>

   <!-- Save results Matrix b -->
   <xsl:variable name="Matrix-b" as="element(RHS)">
      <xsl:copy-of select="//RHS"/>
   </xsl:variable>

   <!-- Transform the Matrix A -->
   <xsl:variable name="A-Matrix" as="element(row)*">
      <xsl:call-template name="transform-coefficient-matrix">
         <xsl:with-param name="Matrix-A1" select="$Matrix-A"/>
      </xsl:call-template>
   </xsl:variable>

   <!-- Create L Matrix -->
```

```
<xsl:variable name="Matrix-L" as="element(row)*">
   <xsl:for-each select="$A-Matrix">
      <xsl:variable name="A7-posn" as="xs:integer">
         <xsl:value-of select="(position())"/>
      </xsl:variable>
      <row>
         <xsl:for-each select="entry[position()!=1]">
            <xsl:choose>
               <xsl:when test="position()&gt;$A7-posn">
                  <entry value="{0}"/>
               </xsl:when>
               <xsl:when test="position()=$A7-posn">
                  <entry value="{1}"/>
               </xsl:when>
               <xsl:otherwise>
                  <entry value="{@value}"/>
               </xsl:otherwise>
            </xsl:choose>
         </xsl:for-each>
      </row>
   </xsl:for-each>
</xsl:variable>

<!-- Create U Matrix -->
<xsl:variable name="Matrix-U" as="element(row)*">
   <xsl:for-each select="$A-Matrix">
      <xsl:variable name="A-posn" as="xs:integer">
         <xsl:value-of select="(position())"/>
      </xsl:variable>
      <row>
         <xsl:for-each select="entry[position()!=1]">
            <xsl:choose>
```

```
                    <xsl:when test="position()&lt;$A-posn">
                        <entry value="{0}"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <entry value="{@value}"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:for-each>
        </row>
    </xsl:for-each>
</xsl:variable>

<!-- Calculate Pb -->
<xsl:variable name="Matrix-Pb" as="element(RHS)">
    <RHS>
        <xsl:for-each select="$A-Matrix/entry[1]">
            <xsl:variable name="index" as="xs:integer">
                <xsl:value-of select="@value"/>
            </xsl:variable>
            <entry value="{$Matrix-b/entry[position()=$index]/@value}"/>
        </xsl:for-each>
    </RHS>
</xsl:variable>

<!-- Calculate Matrix Y -->
<xsl:variable name="Matrix-y-initial" as="element(entry)*">
    <!-- Setup "starter" Matrix X -->
    <entry value="{$Matrix-Pb/entry[1]/@value}"/>
</xsl:variable>

<xsl:variable name="Matrix-Y" as="element(entry)*">
    <xsl:call-template name="calculate-Matrix-Y">
```

```
            <xsl:with-param name="Matrix-L" select="$Matrix-L"/>
            <xsl:with-param name="Matrix-Pb" select="$Matrix-Pb"/>
            <xsl:with-param name="Matrix-Y-initial" select="$Matrix-y-initial"/>
            <xsl:with-param name="index-posn" select="2"/>
        </xsl:call-template>
    </xsl:variable>

    <!-- Calculate Matrix X -->
    <xsl:variable name="Matrix-X-initial" as="element(entry)*">
        <!-- Setup "starter" Matrix X -->
        <xsl:for-each select="$Matrix-Y">
            <xsl:choose>
                <xsl:when test="position()=count($Matrix-Y)">
                    <entry
                        value="{($Matrix-Y[count($Matrix-Y)]/@value) div ($Matrix-U[count($Matrix-Y)]/entry[count($Matrix-Y)]/@value)}"
                    />
                </xsl:when>
                <xsl:otherwise>
                    <entry/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:for-each>
    </xsl:variable>

    <xsl:variable name="Matrix-X" as="element(entry)*">
        <xsl:call-template name="calculate-Matrix-X">
            <xsl:with-param name="Matrix-U" select="$Matrix-U"/>
            <xsl:with-param name="Matrix-Y" select="$Matrix-Y"/>
            <xsl:with-param name="Matrix-X-initial" select="$Matrix-X-initial"/>
            <xsl:with-param name="index-posn" select="(count($Matrix-Y) - 1)"/>
        </xsl:call-template>
    </xsl:variable>
```

```
            <!-- List the results -->
            <linearEquation>
               <solution>
                  <xsl:for-each select="$Matrix-X">
                     <!-- The transformation process can introduce small rounding errors, therefore the results are rounded to 5dp -->
                     <entry value="{round((@value * $rounding-factor)) div $rounding-factor}"/>
                  </xsl:for-each>
               </solution>
            </linearEquation>

         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>

   <!--
      ******************************** Start of Routine ********************************
      Name: validation-routine

      Description: Performs the pre-processing validation for the XSLT.  This is a "firewall" type approach to validation.
      If the input XML document passes this routine then it is assumed to be valid.

      Student ID: 089583717

      Creation Date: 29 November 2008

      Modification History

      Parameters in use
      linearEquation: A linearEquation based variable used to hold the given matrix details.
```

```
    Variables in use
    errnnn: String variables used to hold error message content.
    row-check-equal: Used to check "The number of entries in the LHS rows should be equal".
    LHS-attr-numeric-check: Used to check "Each entry node should contain a numeric attribute value".
    RHS-attr-numeric-check: Used to check "Each entry node should contain a numeric attribute value".
    zero-col-check: Used to check "LHS Columns containing only zeros are not allowed".
    zero-row-check: Used to check "LHS Rows containing only zeros are not allowed".
    No-Error: Default no error found return code.
-->

<xsl:template name="validation-routine" as="xs:string">
    <xsl:param name="linearEquation" as="element(linearEquation)*"/>

    <!-- Set standard error messages -->
    <xsl:variable name="err001" as="xs:string"> "There should be one, and only one,
        linearEquation node" </xsl:variable>
    <xsl:variable name="err002" as="xs:string"> "There should be one and only one LHS node" </xsl:variable>
    <xsl:variable name="err003" as="xs:string"> "There should be one and only one RHS node" </xsl:variable>
    <xsl:variable name="err004" as="xs:string"> "The number of entries in the LHS rows should be
        equal" </xsl:variable>
    <xsl:variable name="err005" as="xs:string"> "The number of entries in the LHS columns should
        be equal to those in the rows" </xsl:variable>
    <xsl:variable name="err006" as="xs:string"> "The number of LHS rows should be equal to the
        number of RHS columns" </xsl:variable>
    <xsl:variable name="err007" as="xs:string"> "Each entry node should contain a numeric
        attribute value" </xsl:variable>
    <xsl:variable name="err008" as="xs:string"> "LHS Columns containing only zeros are not
        allowed" </xsl:variable>
    <xsl:variable name="err009" as="xs:string"> "LHS Rows containing only zeros are not allowed" </xsl:variable>
    <xsl:variable name="err010" as="xs:string"> "Singular Matrices are not allowed" </xsl:variable>

    <xsl:variable name="No-Error" as="xs:string"> "No-Error" </xsl:variable>
```

```
<!-- Define variables used for validation -->

<!--  Used to check "The number of entries in the LHS rows should be equal"  -->
<xsl:variable name="row-check-equal" as="element(entry)*">
   <xsl:for-each select="$linearEquation/LHS/row">
      <xsl:if test="count(entry)!=count($linearEquation/LHS/row[1]/entry)">
         <entry/>
      </xsl:if>
   </xsl:for-each>
</xsl:variable>

<!--  Used to check "Each entry node should contain a numeric
   attribute value"  -->
<xsl:variable name="LHS-attr-numeric-check" as="element(entry)*">
   <xsl:for-each select="$linearEquation/LHS/row/entry">
      <xsl:if test="not (@value castable as xs:double)">
         <entry/>
      </xsl:if>
   </xsl:for-each>
</xsl:variable>

<!--  Used to check "Each entry node should contain a numeric
   attribute value"  -->
<xsl:variable name="RHS-attr-numeric-check" as="element(entry)*">
   <xsl:for-each select="$linearEquation/RHS//entry">
      <xsl:if test="not (@value castable as xs:double)">
         <entry/>
      </xsl:if>
   </xsl:for-each>
</xsl:variable>
```

```
<!--  Used to check "LHS Columns containing only zeros are not allowed"  -->
<xsl:variable name="zero-col-check" as="element(entry)*">
   <xsl:for-each select="$linearEquation/LHS/row">
      <xsl:variable name="col-nbr" as="xs:integer" select="position()"/>
      <xsl:if test="(count($linearEquation/LHS/row/entry[$col-nbr]/@value[.=0])) = 3">
         <entry/>
      </xsl:if>
   </xsl:for-each>
</xsl:variable>

<!--  Used to check "LHS Rows containing only zeros are not allowed"  -->
<xsl:variable name="zero-row-check" as="element(entry)*">
   <xsl:for-each select="$linearEquation/LHS/row">
      <xsl:if test="(count(entry/@value[.=0])) = 3">
         <entry/>
      </xsl:if>
   </xsl:for-each>
</xsl:variable>

<!--  Used to check "Singular Matrices are not allowed"  -->
<xsl:variable name="matrix-determinant" as="xs:double">
   <xsl:call-template name="determinant-routine">
      <xsl:with-param name="Matrix-A1" select="$linearEquation/LHS/row"/>
   </xsl:call-template>
</xsl:variable>

<!-- Start of validation processing -->
<xsl:choose>

   <!-- There should be one, and only one, linearEquation node -->
   <xsl:when test="count($linearEquation)&gt;1">
      <xsl:sequence select="$err001"/>
```

```
</xsl:when>

<!-- There should be one and only one LHS node -->
<xsl:when test="count($linearEquation/LHS)&gt;1">
   <xsl:sequence select="$err002"/>
</xsl:when>

<!-- There should be one and only one RHS node -->
<xsl:when test="count($linearEquation/RHS)&gt;1">
   <xsl:sequence select="$err003"/>
</xsl:when>

<!--  The number of entries in the LHS rows should be equal -->
<xsl:when test="count($row-check-equal)!=0">
   <xsl:sequence select="$err004"/>
</xsl:when>


<!-- The number of entries in the LHS columns should be equal to those in the rows -->
<xsl:when
   test="count($linearEquation/LHS/row) != count($linearEquation/LHS/row[1]/entry)">
   <xsl:sequence select="$err005"/>
</xsl:when>

<!-- The number of LHS rows should be equal to the number of RHS columns -->
<xsl:when test="count($linearEquation/LHS/row) != count($linearEquation/RHS/entry)">
   <xsl:sequence select="$err006"/>
</xsl:when>

<!-- Each entry node should contain a numeric attribute value -->
<xsl:when test="count($LHS-attr-numeric-check)!=0">
   <xsl:sequence select="$err007"/>
```

```
        </xsl:when>

        <xsl:when test="count($RHS-attr-numeric-check)!=0">
          <xsl:sequence select="$err007"/>
        </xsl:when>

        <!-- LHS Columns containing only zeros are not allowed -->
        <xsl:when test="count($zero-col-check) != 0">
          <xsl:sequence select="$err008"/>
        </xsl:when>

        <!-- RHS Columns containing only zeros are not allowed -->
        <xsl:when test="count($zero-row-check) != 0">
          <xsl:sequence select="$err009"/>
        </xsl:when>

        <!-- Singular Matrices are not allowed -->
        <xsl:when test="$matrix-determinant = 0">
          <xsl:sequence select="$err010"/>
        </xsl:when>

        <xsl:otherwise>
          <!-- No error found -->
          <xsl:sequence select="$No-Error"/>
        </xsl:otherwise>
      </xsl:choose>

    </xsl:template>
    <!--    ******************************** End of Routine ******************************** -->


    <!--
```

******************************** Start of Routine ********************************
Name: transform-coefficient-matrix

Description: This routine accepts the starting coefficient Matrix A and performs the LUP decomposition processing to produce a new Matrix A. Because of the immutable nature of XSLT variables, this routine makes use of recursive programming and multi-variables.

Student ID: 089583717

Creation Date: 29 November 2008

Modification History

Parameters in use
Matrix-A1: A row based parameter that contains the coefficient Matrix to be processed.

Variables in use
A4-col-posn: An xs:integer variable used to hold the current position of the Matrix-A4 column (entry).
A4-row-posn: An xs:integer variable used to hold the current position of the Matrix-A4 row.
Matrix-A2: A row based parameter that contains the coefficient Matrix after the changes required for the pivot position.
Matrix-A3: A row based variable being the sub-matrix of the coefficient Matrix A, created by removing the top level row and left most column.
Matrix-A4: A row based variable being the difference between the existing sub-matrix A (Matrix-A3) and Matrix-Y.
Matrix-A5: A row based variable used to hold the result of the recursive calls to named
template "transform-coefficient-matrix".
Matrix-A6: A row based variable being the result of the matrix reconstruction sequence. Also, the return variable.
Matrix-A-pivot-value: An xs:double based variable that holds the value of the coefficient Matrix pivot position.
Matrix-X: A row based variable being the product of column vector v and row vector w.
Matrix-Y: A row based variable being the result of dividing Matrix-X by the Matrix A pivot value.
Matrix-Y-temp: An entry based variable created from Matrix-Y for the purpose of simplifying subsequent calculations involving Matrix-Y.
pivot-posn: An xs:integer based variable used to hold the current position of the pivot value.
v: A row based variable that holds the Matrix A left most true, non-index, column (excluding the pivot value).
v1-value: The attribute value of the relevant v vector used in the matrix reconstruction sequence.
v1: A row based variable that holds the new "v" variable updated for changes in the pivot position.

```
    w: A row based variable that holds the Matrix A top level row (excluding the pivot value).
-->
<xsl:template name="transform-coefficient-matrix" as="element(row)*">
    <xsl:param name="Matrix-A1" as="element(row)*"/>

    <!-- Only process if there is more than 1 row, otherwise matrix has been processed and should be returned -->
    <xsl:choose>
        <xsl:when test="count($Matrix-A1) &gt;1">

            <!-- Determine pivot row for matrix transformation -->
            <xsl:variable name="pivot-posn" as="xs:integer">
                <xsl:call-template name="determine-pivot-position">
                    <xsl:with-param name="Matrix-A1" select="$Matrix-A1"/>
                </xsl:call-template>
            </xsl:variable>

            <!-- Creat new Matrix A variable that results from the swapping of the rows for the pivot positioning -->
            <xsl:variable name="Matrix-A2" as="element(row)*">
                <xsl:for-each select="$Matrix-A1">
                    <xsl:choose>
                        <xsl:when test="position()=1">
                            <xsl:copy-of select="$Matrix-A1[$pivot-posn]"/>
                        </xsl:when>
                        <xsl:when test="position()=$pivot-posn">
                            <xsl:copy-of select="$Matrix-A1[1]"/>
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:copy-of select="."/>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:for-each>
            </xsl:variable>
```

```
<!-- Create variable to hold column vector v -->
<xsl:variable name="v" as="element(row)*">
   <xsl:for-each select="$Matrix-A2[position()!=1]">
      <row>
         <xsl:copy-of select="entry[position()&lt;3]"/>
      </row>
   </xsl:for-each>
</xsl:variable>

<!-- Create variable to hold row vector w -->
<xsl:variable name="w" as="element(row)">
   <xsl:for-each select="$Matrix-A2[1]">
      <row>
         <xsl:copy-of select="entry[position()!=2]"/>
      </row>
   </xsl:for-each>
</xsl:variable>

<!-- "Update" the v vector by dividing by the A Matrix pivot -->
<xsl:variable name="Matrix-A-pivot-value" as="xs:double"
   select="$Matrix-A2[1]/entry[2]/@value"/>
<xsl:variable name="v1" as="element(row)*">
   <xsl:for-each select="$v">
      <row>
         <xsl:for-each select="entry">
            <xsl:choose>
               <xsl:when test="position()=2">
                  <entry value="{@value div $Matrix-A-pivot-value}"/>
               </xsl:when>
               <xsl:otherwise>
                  <entry value="{@value}"/>
```

```
                </xsl:otherwise>
            </xsl:choose>
        </xsl:for-each>
    </row>
  </xsl:for-each>
</xsl:variable>

<!-- Now calculate A' sub-matrix.  This is a sub-matrix of the full A-Matrix, which represents a n-1 x n-1 dimension matrix-->
<xsl:variable name="Matrix-A3" as="element(row)*">
  <xsl:for-each select="$Matrix-A2[position()!=1]">
    <row>
      <xsl:copy-of select="./entry[position()!=2]"/>
    </row>
  </xsl:for-each>
</xsl:variable>

<!-- Start of Schur Complement calculations.  These are part of the decomposition processing. -->

<!-- Let Matrix-X be the product of column vector v and row vector w -->
<xsl:variable name="Matrix-X" as="element(row)*">
  <xsl:for-each select="$v/entry[2]/@value">
    <xsl:variable name="v-temp" as="xs:double">
      <xsl:value-of select="../@value"/>
    </xsl:variable>
    <row>
      <xsl:for-each select="$w/entry[position()!=1]/@value">
        <entry value="{$v-temp * (../@value)}"/>
      </xsl:for-each>
    </row>
  </xsl:for-each>
</xsl:variable>
```

```
        <!-- Let Matrix-Y be the result of dividing Matrix-X by the Matrix A pivot value -->
        <xsl:variable name="Matrix-Y" as="element(row)*">
          <xsl:for-each select="$Matrix-X">
            <row>
              <xsl:for-each select="./entry">
                <entry value="{@value div $Matrix-A-pivot-value}"/>
              </xsl:for-each>
            </row>
          </xsl:for-each>
        </xsl:variable>


        <!-- Let Matrix-A4, the difference between the existing sub-matrix A (Matrix-A3) and Matrix-Y, be the new sub-matrix for further
processing -->
        <!-- First, convert Matrix-Y into and entry object for easier processing -->
        <xsl:variable name="Matrix-Y-temp" as="element(entry)*">
          <xsl:copy-of select="$Matrix-Y/entry"/>
        </xsl:variable>

        <!-- Now give definition for variable Matrix-A4 -->
        <xsl:variable name="Matrix-A4" as="element(row)*">
          <xsl:for-each select="$Matrix-A3">
            <xsl:variable name="A4-row-posn" as="xs:integer">
              <xsl:value-of select="position()"/>
            </xsl:variable>
            <row>
              <xsl:for-each select="./entry">
                <xsl:variable name="A4-col-posn" as="xs:integer">
                  <xsl:value-of select="position()"/>
                </xsl:variable>
                <xsl:choose>
                  <xsl:when test="$A4-col-posn=1">
```

```
                    <entry value="{@value}"/>
                  </xsl:when>
                  <xsl:otherwise>
                    <entry
                      value="{@value - ($Matrix-Y-temp[position()=(count($Matrix-A3)*($A4-row-posn - 1)) + ($A4-col-posn - 1)]/@value)}"
                    />
                  </xsl:otherwise>
                </xsl:choose>
              </xsl:for-each>
            </row>
          </xsl:for-each>
        </xsl:variable>

        <!-- Create variable for use within the recursive calls of this template. -->
        <xsl:variable name="Matrix-A5" as="element(row)*">
          <xsl:call-template name="transform-coefficient-matrix">
            <xsl:with-param name="Matrix-A1" select="$Matrix-A4"/>
          </xsl:call-template>
        </xsl:variable>

        <!-- Reconstruct Matrix (see inline comments) -->
        <xsl:variable name="Matrix-A6" as="element(row)*">

          <row>
            <!-- First row being a combination of the pivot value and the top most row and left most column -->
            <xsl:for-each select="$w/entry">
              <xsl:choose>
                <xsl:when test="position()=1">
                  <entry value="{@value}"/>
                  <entry value="{$Matrix-A-pivot-value}"/>
                </xsl:when>
                <xsl:otherwise>
                <entry value="{@value}"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
</row>

<xsl:for-each select="$Matrix-A5">
    <row>
        <!-- Recursive calls to template "transform-coefficient-matrix" to determine the remaining matrix entries
        This will continue until a matrix of size 1 x 1 is produced.  At that point the template will traverse upwards
        through the call stack until the completed, new, A-Matrix is created.  Note the use of column 1 to determine the
        permutations that have taken place to determine the best decomposition pivot value.
        -->
        <xsl:for-each select="entry">
            <xsl:choose>
                <xsl:when test="position()=1">
                    <!-- Column 1 contains the Permutation Matrix "Index" value -->
                    <xsl:variable name="row-index" as="xs:double">
                        <xsl:value-of select="@value"/>
                    </xsl:variable>
                    <entry value="{@value}"/>
                    <xsl:variable name="v1-value" as="xs:double">
                        <!-- Select the matching index value of the column vector -->
                        <xsl:value-of
                            select="$v1/entry[position()=2 and preceding-sibling::node()/@value=$row-index]/@value"
                        />
                    </xsl:variable>
                    <entry value="{$v1-value}"/>
                </xsl:when>
                <xsl:otherwise>
                    <entry value="{@value}"/>
                </xsl:otherwise>
```

```
                    </xsl:choose>
                  </xsl:for-each>
                </row>
              </xsl:for-each>
            </xsl:variable>

            <!-- Returned matrix variable to calling template (returned back up the "calling stack") -->
            <xsl:sequence select="$Matrix-A6"/>
          </xsl:when>
          <xsl:otherwise>
            <!-- Relates to the fact that a 1 x 1 size matrix was passed into the routine -->
            <xsl:sequence select="$Matrix-A1"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:template>
      <!--    ******************************** End of Routine ******************************** -->

      <!--
        ******************************** Start of Routine ********************************
        Name: calculate-Matrix-Y

        Description: Calculates the Y Matrix using forward substitution

        Student ID: 089583717

        Creation Date: 29 November 2008

        Modification History

        Parameters in use
        Matrix-L: A row based parameter that contains the results of the calculation to produce the unit lower-triangular matrix.
        Matrix-Pb: A RHS based parameter that hold the product of the Permuatation Matrix and the Linear Equation results matrix, b.
```

```
    Matrix-Y-initial: An entry based parameter that holds the initial value of the Matrix-Y.
    index-posn: An integer holding the current position of the Matrix-Y row to be evaluated.

    Variables in use
    Y-temp: An entry based variable used to hold the product of the previous Y variables for substituting into the current Y variable evaluation.
    Y-posn: An xs:integer based variable used to hold the current position of the processing through the Matrix-Y-initial Matrix.
    Y-value: A xs:double based variable used to hold the value of the current Matrix-Y-initial attribute value.

-->

<xsl:template name="calculate-Matrix-Y" as="element(entry)*">
    <xsl:param name="Matrix-L" as="element(row)*"/>
    <xsl:param name="Matrix-Pb" as="element(RHS)"/>
    <xsl:param name="Matrix-Y-initial" as="element(entry)*"/>
    <xsl:param name="index-posn" as="xs:integer"/>

    <xsl:choose>
        <xsl:when test="count($Matrix-Pb/entry)&lt;($index-posn)">
            <!-- Return back calculated y value -->
            <xsl:sequence select="$Matrix-Y-initial"/>
        </xsl:when>

        <xsl:otherwise>
            <!-- Use forward substitution to calculate the y variable -->
            <xsl:variable name="Y-temp" as="element(entry)*">
                <xsl:for-each select="$Matrix-Y-initial">
                    <xsl:variable name="Y-posn" as="xs:integer">
                        <xsl:value-of select="position()"/>
                    </xsl:variable>
                    <xsl:variable name="Y-value" as="xs:double">
                        <xsl:value-of select="@value"/>
                    </xsl:variable>
                    <entry value="{($Matrix-L[$index-posn]/entry[$Y-posn]/@value) * $Y-value}"/>
                </xsl:for-each>
            </xsl:variable>

            <xsl:variable name="Matrix-Y-return" as="element(entry)*">
                <xsl:copy-of select="$Matrix-Y-initial"/>
                <entry value="{$Matrix-Pb/entry[$index-posn]/@value - (sum($Y-temp/@value))}"/>
            </xsl:variable>

            <xsl:call-template name="calculate-Matrix-Y">
                <xsl:with-param name="Matrix-L" select="$Matrix-L"/>
                <xsl:with-param name="Matrix-Pb" select="$Matrix-Pb"/>
                <xsl:with-param name="Matrix-Y-initial" select="$Matrix-Y-return"/>
                <xsl:with-param name="index-posn" select="($index-posn + 1)"/>
            </xsl:call-template>

        </xsl:otherwise>
    </xsl:choose>

</xsl:template>

<!--    ********************************* End of Routine ********************************* -->

<!--
    ********************************* Start of Routine *********************************
    Name: calculate-Matrix-X

    Description: Calculates the X Matrix using backward substitution

    Student ID: 089583717

    Creation Date: 29 November 2008
    Modification History

    Parameters in use
    Matrix-U: A row based parameter that contains the results of the calculation to produce the upper-triangular matrix.
    Matrix-Y: An entry based parameter that holds the value of the Y Matrix.
    Matrix-X-initial: An entry based parameter that holds the initial value of the Matrix-X.
    index-posn: An integer holding the current position of the Matrix-Y row to be evaluated.

    Variables in use
    Y-temp: An entry based variable used to hold the product of the previous Y variables for substituting into the current Y variable evaluation.
    Y-posn: An xs:integer based variable used to hold the current position of the processing through the Matrix-Y-initial Matrix.
    Y-value: A xs:double based variable used to hold the value of the current Matrix-Y-initial attribute value.

-->
<xsl:template name="calculate-Matrix-X" as="element(entry)*">
    <xsl:param name="Matrix-U" as="element(row)*"/>
    <xsl:param name="Matrix-Y" as="element(entry)*"/>
    <xsl:param name="Matrix-X-initial" as="element(entry)*"/>
    <xsl:param name="index-posn" as="xs:integer"/>

    <xsl:choose>
        <xsl:when test="$index-posn&lt;1">
            <!-- Return back calculated x value -->
            <xsl:sequence select="$Matrix-X-initial"/>
        </xsl:when>

        <xsl:otherwise>
            <!-- Use backward substitution to calculate the x variable -->
            <xsl:variable name="X-temp" as="element(entry)*">
                <xsl:for-each select="$Matrix-X-initial">
                    <xsl:variable name="X-posn" as="xs:integer">
                        <xsl:value-of select="position()"/>
                    </xsl:variable>
                    <xsl:if test="$X-posn&gt;$index-posn">
                        <xsl:variable name="X-value" as="xs:double">
                            <xsl:value-of select="@value"/>
                        </xsl:variable>
                        <entry
                            value="{($Matrix-U[$index-posn]/entry[$X-posn]/@value) * $X-value}"
                        />
                    </xsl:if>
                </xsl:for-each>
            </xsl:variable>

            <xsl:variable name="Matrix-X-return" as="element(entry)*">
                <xsl:for-each select="$Matrix-X-initial">
                    <xsl:choose>
                        <xsl:when test="position()=$index-posn">
                            <entry
                                value="{($Matrix-Y[$index-posn]/@value - (sum($X-temp/@value))) div ($Matrix-U[$index-posn]/entry[$index-
        posn]/@value)}"
                            />
                        </xsl:when>
                        <xsl:otherwise>
                            <xsl:copy-of select="."/>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:for-each>
            </xsl:variable>

            <xsl:call-template name="calculate-Matrix-X">
                <xsl:with-param name="Matrix-U" select="$Matrix-U"/>
                <xsl:with-param name="Matrix-Y" select="$Matrix-Y"/>
```

```
            <xsl:with-param name="Matrix-X-initial" select="$Matrix-X-return"/>
            <xsl:with-param name="index-posn" select="($index-posn - 1)"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>

    </xsl:template>

    <!--    ******************************** End of Routine ******************************** -->


    <!--
      ******************************** Start of Routine ********************************
      Name: determine-pivot-position

      Description: Determine the pivot position for the coefficient Matrix A

      Student ID: 089583717

      Creation Date: 29 November 2008

      Modification History

      Parameters in use
      Matrix-A1: A row based parameter that contains the coefficient Matrix A.

      Variables in use
      col-vector: A row based variable used to hold the first true, non-index, column of the coefficient Matrix A.
      multi-pivot-posn: An xs:integer based variable used to hold the current position(s) of the candidate pivot values.
      pivot-posn: An xs:integer based variable used to hold the current position of the pivot value.

    -->
```

```
<xsl:template name="determine-pivot-position" as="xs:integer">
   <xsl:param name="Matrix-A1" as="element(row)*"/>

   <xsl:variable name="col-vector" as="element(row)*">
      <xsl:for-each select="$Matrix-A1">
         <row>
            <!-- Note: entry[2] because entry[1] will hold the permutation index -->
            <xsl:copy-of select="entry[2]"/>
         </row>
      </xsl:for-each>
   </xsl:variable>

   <!-- Determine position of pivot row-->
   <xsl:variable name="multi-pivot-posn" as="xs:integer*">
      <xsl:for-each select="$col-vector/entry">
         <xsl:if test="abs(@value)=max($Matrix-A1/entry[2]/abs(@value))">
            <xsl:value-of select="position()"/>
         </xsl:if>
      </xsl:for-each>
   </xsl:variable>

   <!-- It is possible that the above variable holds multi-values, this happens when there are more than one
   value with the same absolute value.  In this case we need only one, therefore will select the first one found.
   -->
   <xsl:variable name="pivot-posn" as="xs:integer">
      <xsl:value-of select="$multi-pivot-posn[1]"/>
   </xsl:variable>

   <xsl:sequence select="$pivot-posn"/>

</xsl:template>
```

```
<!--   ******************************** End of Routine ******************************** -->
<!--
   ******************************** Start of Routine ********************************
   Name: determinant-routine

   Description: Calculate the determinant of a given n x n matrix

   Student ID: 089583717

   Creation Date: 29 November 2008

   Modification History

   Parameters in use
   Matrix-A1: A row based parameter that contains the coefficient Matrix A.

   Variables in use
   co-factor: An xs:double based variable used to hold the major matrix co-factor.
   minor-matrix: A row based varibale used to hold the value of the minor matrix.
   minor-matrix-determinant: An xs:double based variable used to hold the value of the minor matrix determinant.
   matrix-determinant: An xs:double based variable used to hold the resultant A-Matrix determinant.
   posn1: An xs:integer based variable used to track the position of a set during processing.
   posn2: An xs:integer based variable used to track the position of a set during processing.
   sign: An xs:integer based variable used to hold the sign for the summation of the minors.
-->

<xsl:template name="determinant-routine" as="xs:double*">
   <xsl:param name="Matrix-A1" as="element(row)*"/>

   <!-- Determinant overview
      The determinant of a matrix n x n, where n > 2 is calculated recursively by multiplying
      each the determinant of the minor-matrix (created by removing a row and a column from
    the original matrix) with the co-factor of the major matrix.
-->

<!-- Parameter for recursive calculation of the matrix determinant by the expansion of the minors -->
<xsl:variable name="matrix-determinant" as="xs:double*">
   <xsl:for-each select="$Matrix-A1[1]/entry">

      <!-- Hold the current co-factor position -->
      <xsl:variable name="posn1" as="xs:integer" select="position()"/>

      <!-- Hold the current co-factor value -->
      <xsl:variable name="co-factor" as="xs:double" select="./@value"/>

      <!-- Determine the sign to be applied when summing the elements -->
      <xsl:variable name="sign" as="xs:integer">
         <xsl:choose>
            <xsl:when test="($posn1 mod 2)=0"> -1 </xsl:when>
            <xsl:otherwise> 1 </xsl:otherwise>
         </xsl:choose>
      </xsl:variable>

      <!-- Create the minor matrix by removing row 1 and the column that equates to the co-factor position -->
      <xsl:variable name="minor-matrix" as="element(row)*">
         <xsl:for-each select="$Matrix-A1[position()!=1]">
            <xsl:variable name="posn2" as="xs:integer" select="position()"/>
            <row>
               <xsl:for-each
                  select="$Matrix-A1[position()=($posn2)+1]/entry[position()!=$posn1]">
                  <entry value="{@value}"/>
               </xsl:for-each>
            </row>
         </xsl:for-each>
```

```
            </xsl:variable>

            <!-- Use recursion to calculate the determinants of each level of minor matrices (continues unti we have a 2 x 2 matrix -->
            <xsl:variable name="minor-matrix-determinant" as="xs:double">
               <xsl:call-template name="determinant-routine">
                  <xsl:with-param name="Matrix-A1" select="$minor-matrix"/>
               </xsl:call-template>
            </xsl:variable>

            <!-- Give the calculated value to the variable "matrix-determinant" -->
            <xsl:value-of select="$sign * $co-factor * $minor-matrix-determinant"/>

         </xsl:for-each>
      </xsl:variable>

      <xsl:choose>
         <!-- Trivial determinant calculation for 1 x 1 matrices -->
         <xsl:when test="count($Matrix-A1) = 1">
            <xsl:sequence select="$Matrix-A1[1]/entry[1]/@value"/>
         </xsl:when>

         <!-- Standard determinant calcualtion for a 2 x 2 matrix -->
         <xsl:when test="count($Matrix-A1) = 2">
            <xsl:sequence
               select="(($Matrix-A1[1]/entry[1]/@value) * ($Matrix-A1[2]/entry[2]/@value)) - (($Matrix-A1[1]/entry[2]/@value) * ($Matrix-
A1[2]/entry[1]/@value))"
            />
         </xsl:when>

         <!-- Trigger the above recursive calculation -->
         <xsl:otherwise>
            <xsl:sequence select="sum($matrix-determinant)"/>
            </xsl:otherwise>

        </xsl:choose>

    </xsl:template>
    <!--    ******************************** End of Routine ******************************** -->
</xsl:stylesheet>
```

Appendix C – Validation Testing Input Files

eqtest1.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
         <entry value="4"/>
      </row>
      <row>
```

```
            <entry value="5"/>
            <entry value="6"/>
            <entry value="3"/>
        </row>
    </LHS>
    <RHS>
        <entry value="3"/>
        <entry value="7"/>
        <entry value="8"/>
    </RHS>
</linearEquation>
```

eqtest2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
    <LHS>
        <row>
            <entry value="1"/>
            <entry value="2"/>
            <entry value="0"/>
        </row>
        <row>
            <entry value="3"/>
            <entry value="4"/>
            <entry value="4"/>
        </row>
        <row>
            <entry value="5"/>
            <entry value="6"/>
            <entry value="3"/>
        </row>
    </LHS>
    <LHS>
        <row>
            <entry value="1"/>
            <entry value="2"/>
            <entry value="0"/>
        </row>
        <row>
            <entry value="3"/>
            <entry value="4"/>
            <entry value="4"/>
        </row>
        <row>
            <entry value="5"/>
            <entry value="6"/>
            <entry value="3"/>
        </row>
    </LHS>
    <RHS>
        <entry value="3"/>
        <entry value="7"/>
        <entry value="8"/>
    </RHS>
```

```
</linearEquation>
```

eqtest3.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="linearEquation.xsd">
  <LHS>
    <row>
      <entry value="1"/>
      <entry value="2"/>
      <entry value="0"/>
    </row>
    <row>
      <entry value="3"/>
      <entry value="4"/>
      <entry value="4"/>
    </row>
    <row>
      <entry value="5"/>
      <entry value="6"/>
      <entry value="3"/>
    </row>
  </LHS>
  <RHS>
    <entry value="3"/>
    <entry value="7"/>
    <entry value="8"/>
  </RHS>
  <RHS>
    <entry value="3"/>
    <entry value="7"/>
    <entry value="8"/>
  </RHS>
</linearEquation>
```

eqtest4.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

eqtest5.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

eqtest6.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
   </RHS>
</linearEquation>
```

equest7.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="2"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

equest8.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="0"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="0"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="0"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

eqtest9.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="0"/>
         <entry value="0"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="3"/>
         <entry value="4"/>
         <entry value="4"/>
      </row>
      <row>
         <entry value="5"/>
         <entry value="6"/>
         <entry value="3"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

eqtest10.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="1"/>
         <entry value="0"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="-2"/>
         <entry value="0"/>
         <entry value="0"/>
      </row>
      <row>
         <entry value="4"/>
         <entry value="6"/>
         <entry value="1"/>
      </row>
   </LHS>
   <RHS>
      <entry value="3"/>
      <entry value="7"/>
      <entry value="8"/>
   </RHS>
</linearEquation>
```

Appendix D – Sample Valid Files

eqpass1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="linearEquation.xsd">
  <LHS>
    <row>
      <entry value="4"/>
    </row>
  </LHS>
  <RHS>
    <entry value="10"/>
  </RHS>
</linearEquation>
```

eqpass2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="file:/homes/graham/teaching/xml/marks/coursework0809/
linearEquation.xsd">
  <LHS>
    <row>
      <entry value="0.0"/>
      <entry value="1.0"/>
    </row>
    <row>
      <entry value="1.0"/>
      <entry value="2.0"/>
    </row>
 </LHS>
  <RHS>
    <entry value="5.3"/>
    <entry value="2.8"/>
  </RHS>
</linearEquation>
```

eqpass3.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="linearEquation.xsd">
   <LHS>
      <row>
         <entry value="6"/>
         <entry value="-3"/>
         <entry value="6"/>
      </row>
      <row>
         <entry value="1"/>
         <entry value="11"/>
         <entry value="1"/>
      </row>
      <row>
         <entry value="4"/>
         <entry value="9"/>
         <entry value="2"/>
      </row>
   </LHS>
   <RHS>
      <entry value="72"/>
      <entry value="-57"/>
      <entry value="-28"/>
   </RHS>
</linearEquation>
```

eqpass4.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<linearEquation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="linearEquation.xsd">
  <LHS>
    <row>
      <entry value="2"/>
      <entry value="11"/>
      <entry value="7"/>
      <entry value="9"/>
    </row>
    <row>
      <entry value="9"/>
      <entry value="1"/>
      <entry value="5"/>
      <entry value="-13"/>
    </row>
    <row>
      <entry value="-10"/>
      <entry value="11"/>
      <entry value="-1"/>
      <entry value="10"/>
    </row>
    <row>
      <entry value="-3"/>
      <entry value="-6"/>
      <entry value="12"/>
      <entry value="11"/>
    </row>
  </LHS>
  <RHS>
    <entry value="-204"/>
    <entry value="147"/>
    <entry value="-228"/>
    <entry value="21"/>
  </RHS>
</linearEquation>
```