Design Final Report

Annie Ding and Meghan Furton

5/16/14

Part I
Problem Definition

Introduction

For the semester of Spring 2014, we have worked on the Materials List code for the entire plant design. We will build on the progress made in Fall 2012 and Spring 2013 by team members Kira Gordon and Annie Ding respectively. Kira's report thoroughly explains the details of how she put together all the information needed in order to write the code. Annie's report focuses on her modifications to Kira's code and her work to optimize the lances cutting algorithm. Once the entire design challenge has been completed, the materials list code should integrate what we know about the materials available in the region with knowledge of the materials necessary to build the entire plant. The resulting materials list will include the following hydraulic and structural components:

- Hydraulic design materials
  - Pipes
  - Pipe fittings
  - Flocculator baffle sheets
  - Sedimentation plate settlers
  - Chemical dose controller pieces
- Structural design materials
  - Brick
  - Rebar
  - Cement (for plant walls)
  - River rock (for entrance tank hoppers)

As the only information we currently have is a list of hydraulic components from San Nicolas plant and a budget for all materials used or the Alauca plant, this will primarily be useful to construction teams in Honduras and for estimating how much a specific plant design would cost. The code should automatically output a list of materials needed to build any plant specified by the user and should include number of items needed, costs and should be in both Spanish and English.

Current Focus

The first component the code will assess is the total number of pipes needed for the plant. The code that will eventually output the list of pipes was started by a previous design team member. It turns out that this problem is a typical cutting stock problem.The cutting stock algorithm for optimizing the number of lances, which Kira began to write, was first proposed by Andrew Hart. This method was adopted by Annie for the Spring 2013 semester. During this semester, the algorithm was modified in order to facilitate evaluation by MathCAD. In addition, progress was made on calling the correct values in the code to input into the algorithm code.

Design Details

Lances Calculation

PVC pipes in Honduras are available as 6.1 meter sections called lances. The completed code, which will be in a new MathCAD file that will be called after the plant drawing is finished, should find the minimum number of lances needed to build the plant. Using this information, the overall cost of the pipes needed to build the plant can be found given specific costs per pipe specification. The first part of the code finds and categorizes all the pipes based on their nominal diameter and pipe specification in a series of matrices. The second part of the code will use this information to optimize the number of lances needed to reduce waste. The final step will be to calculate actual costs given individual prices for each type of lance.

Relationship to Existing Code

The code to find matrices which will store pipe information depends on the current PipeF and PipeSUBF functions which create scripts to draw pipes in AutoCAD, as well as additional inputs of N. The variable N is the number of similar pipes drawn, which is calculated based on how many times a pipe is arrayed or mirrored. PipeMatrix is a new variable that will hold the following four values for each type of pipe:

- Length
- NominalDiameter (ND)

- Enumerated pipe schedule type or pipe specification (EN)
- Number of similar pipes (N)

In order to implement this, the entire AutoCAD drawing must be searched to find each time that a pipe is called so that the new code for determining N can be added.

Constraints and Assumptions

Several assumptions were made before starting to write the code to calculate the optimal number of lances, most of which were discussed first with Andrew Hart. First, we are assuming that the construction crews may make mistakes in cutting the pipe and that there will be differences between the design and the actual construction lengths in real life. Because of this, we are building in a buffer of about 60 cm into the code so that our amount and price estimates will take this into account. This will mean that our actual usable pipe length will be $6.1 - 0.6 = 5.5$m.

Significant figures needed for the pipe lengths should be given in millimeters. Although actual construction crews in the Honduras do not build most parts of the plant with millimeter precision, lengths should be given in millimeters anyway since a few parts of the plant, like the spacer pipes in the plate modules, need to be built with millimeter precision in mind.

Couplings between pipes do not need to be taken into account (for pipes that will need to be longer than the lances) since the lances include a “campana,” which allows lances to slide into one another without need for a coupling. However, the buffer will be accounting for the fact that the “campanas” can be of differing lengths.

Solution Approach

The following outlines the original cutting optimization procedure started by Kira in the Fall 2012 semester. The algorithm was modified by Annie over the Spring 2013 and Spring 2014 semester.

1. The code:

    (a) Construct PipeMatrix (see Relationship to Existing Code section)

    (b) There will be lengths of pipe in the PipeMatrix that will be longer than the length of a lance; separate these lengths into a new matrix (LongPipesMatrix). This matrix will be put through all the same dividing codes as the original PipeMatrix (1(c)-(d)).

    (c) Separate the PipeMatrix into different matrices that each only contain pipes of a certain nominal diameter and pipe specification. These different matrices will be contained in a large matrix, or cell array (A) , to keep them in one place.

(d) Group together all pipes with the same length in the same row, adding the appropriate number onto the value of N, the number of times pipes of that length, ND and EN appear in the drawing.

(e) Optimizing cutting of lances:

   i. For each of the rows in the LongPipesMatrix:

      A. Determine the number of full lances needed to construct each pipe in A

      B. Find the length of pipe leftover (t) for each pipe and replace the previous lengths in the arrays with the leftover lengths.

      C. Add the number of lances (n) used to a new matrix, LanceMatrix, with the corresponding pipe specification and nominal diameter. LanceMatrix will be added to and used later in the calculation of the cost of all the pipes.

      D. With all the leftover lengths of pipes that are not covered by the lengths of full lances, each row should contain 4 values: a length less than that of a lance, ND, EN, and N. These rows are to be treated in the same manner as the original (short) PipeMatrix; to simplify the process later, the information in these rows is consolidated into one matrix with the values from original PipeMatrix.

   ii. For each of the rows in each matrix in the now consolidated PipeMatrix array:

      A. Find the longest pipe length (p) for the first matrix (which contains pipes with the same ND and EN values) in PipesMatrix.

      B. Find $t = l - p$, where l is the length of a lance.

      C. Make a new matrix called CutMatrix (just the first time; the same matrix will be used for all following pipes p in all arrays A) to hold the lengths of pipes subtracted from this lance. Each row of CutMatrix should stand for one specific pipe and it will look like LanceMatrix, except instead of n there will be a column that will hold a matrix (per pipe) stating the lengths that will be cut out of that lance and the number of times that length will be cut out of a pipe.

      D. Add the length p onto the vector of the appropriate row in CutMatrix.

      E. Add 1 onto n in the row in LanceMatrix corresponding to the pipe specification and nominal diameter of A (if there is no existing row, stack a new row with the proper information onto the matrix).

   iii. From the remaining lengths in A, find the next longest length that is shorter than the remaining lance t:

      A. Loop through all the lengths in A to find the maximum of the lengths (m) that is still less than t.

B. Subtract this number m from the current t to find the new t.

C. Repeat i. through iii. until t is less than the smallest length available in A.

iv. Subtract 1 from the N associated with that particular pipe length in A to make sure we don't accidentally use lances for pipes already accounted for.

v. Repeat (i) though (iv) until all arrays are accounted for. This process is shown in Figure 1.

(f) The cost of each of the lances can be found using costs from previous plants built in Honduras, then the number of lances needed for each ND and EN can be multiplied by the corresponding costs, in their own matrices, to find the total cost for all the pipes in the plant.

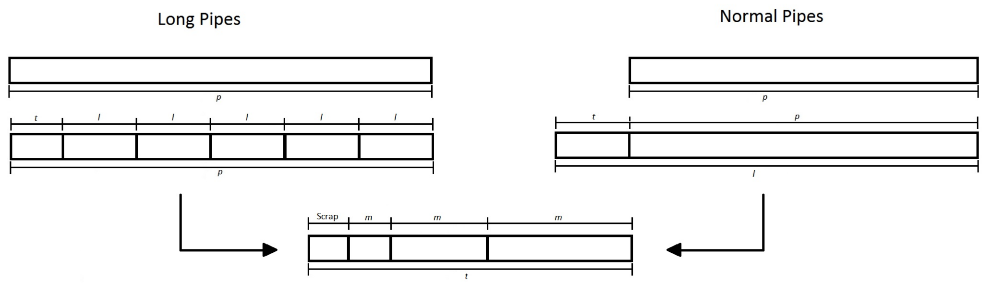


Figure 1:

2. Testing the code:

   (a) After the coding has been completed, all PipeMatrices called from the different sections of code corresponding to the different elements of the plant must be run through the code to make sure the code works for all cases.

   (b) We can then test the code by hand to make sure it does what it's supposed to do. A small matrix, with about 20 rows, will be used to test the matrix and the appropriate number of lances needed will be calculated by hand as well in order to see if the code is correctly optimizing the number of lances needed.

   (c) We can then test the code on a plant design of the same flow rate as the San Nicolas plant, which we have the pipe data for. If the numbers for the number of pipes (and therefore the cost) don't match up, assumptions may have to be reassessed and the code will have to be thoroughly re-examined.

Due to MathCAD not being able to fully evaluate the code written for the algorithm given in step one, the algorithm was revised to be simpler, with less calculations involved so that it could be fully evaluated by MathCAD. The revised algorithm only changes sections 1.(e)ii.-iv. of the algorithm above; the rest of the process remains the same. The following steps can be substituted into the previous algorithm:

ii. For each of the rows in each matrix in the now consolidated PipeMatrix array:

   A. For the first row in the matrix, subtract the length associated with the row (a) from the length of lance available to use (5.5m) to find the length of pipe left to use once a is cut out of the pipe ($MaxLengthUse$).

   B. Go through the matrix and find the largest length in the matrix that is less than or equal to $MaxLengthUse$ (b). If there is no length left in the matrix that is less than or equal to$MaxLengthUse$, $b = 0$ and skip to step 1.(e)ii.D.

   C. Pull out the n values corresponding to both of these lengths (a and b correspond to n_a and n_b).

      1. If the two values are the same ($n = n_a = n_b$),

         a. Add n as N to the CutMatrix with the corresponding ND and EN from the matrix.

         b. If the length of the pipe is less than half of the length of a lance, then there can be at least two pipes cut out of a lance. Find the number of times you can cut the length out of the lance (e.g. a 2m long pipe can be cut out of a 5.5m lance 2 times). Divide this number from the number of pipes needed for this length and specification (n) to find the total number of lances needed (N_{Use}). Add N_{Use} as N to the LanceMatrix with the corresponding ND and EN from the matrix.

         c. If the length of the pipe is more than half of the length of a lance, you can only cut out one pipe per lance. Add n to LanceMatrix with the corresponding ND and EN values.

         d. Set both n_a and n_b within the original matrix to zero: these pipes have all been taken into account, so their numbers to be calculated are now zero.

      2. If the two values are different ($n_a \neq n_b$),

         a. Set the lower value of the two to N_{Use}; subtract N_{Use} from both n_a and n_b within the matrix in the cell array. The value that had previously been lower should now read zero, while the value that had been higher should now reflect a number of pipes that have not yet been taken into account.

b. Add N_{Use} into the CutMatrix twice: once with a length of a and once with a length of b, both times with the corresponding ND and EN values.

c. Add $\frac{N_{Use}}{2}$ as the N into the LanceMatrix, with the corresponding ND and EN values (there are two pipes getting cut out of each lance, so the number of lances used should be half the number of pipes cut).

D. If it was found that $b = 0$ in step 1.(e)ii.B, add just the length a into the CutMatrix and LanceMatrix with the n_a as the n value in both cases. Set n_a in the original matrix to zero.

E. Move to the next row in the matrix and repeat 1.(e).ii.A-D. If the n value corresponding to this row is zero, move to the row after that and repeat.

F. Repeat 1.(e).ii.A-E until all n values within the matrix are zero. This means that every pipe within the matrix has been taken into account in CutMatrix and LanceMatrix.

iii. Repeat 1.(e).ii until all arrays are accounted for.

Other Materials

When the code for pipes has been completed, the next step will be to analyze other components and determine what parameters will be used to sort the required materials.

Plant Drawing

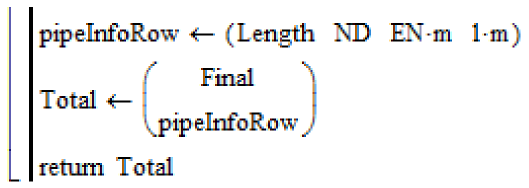
The completed codes for determining the lists of materials will need to be incorporated into the plant drawing functions for each plant component. Unlike the plant design specifications, which are output as a Microsoft Word file, the best way to output the variables for the materials is to use a Microsoft Excel file. These two files would be sent to the user who requests the design. The code can be tested by comparing the actual materials list for one of the recently completed plants in Honduras with the design output for equivalent specifications. The amount of pipe used in the actual construction of the plant should be similar to the amount found using the code. There are too many ways that actual numbers could vary, so it needs to be made clear in the file that the numbers provided are just an estimate. The procedure for pipe lengths is detailed above.

Part II

Documented Progress

Pipe Matrix Extraction

The method used to extract pipe data from the drawing script is to modify the pipe drawing functions PipeF and PipeSUBF to produce a cell array. The first cell will contain the unchanged AutoCAD drawing script while the last cell will contain the pipe data that is a given value when the function is called as well as an N value of 1. The added lines to the pipe drawing functions are shown below in Figure 2. Note that each entry in the pipe matrix has units of meters.

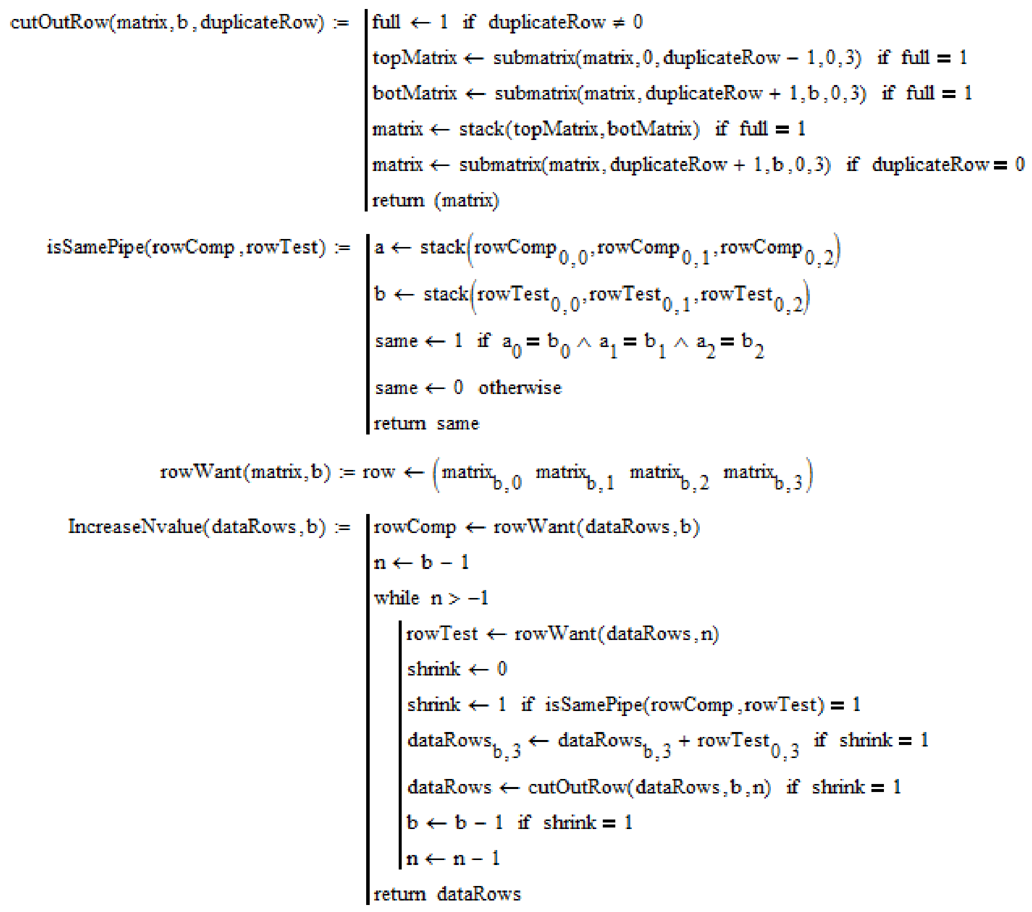


```
pipeInfoRow ← (Length  ND  EN·m  1·m)
Total ← (Final ; pipeInfoRow)
return Total
```

Figure 2: Pipe Drawing Functions Modified

Adding to a Growing Pipe Matrix

The functions below in Figure 3 are ultimately used in a function which will change the N value of pipes when a pipe of previously used specifications is added to the pipe matrix. The function "cut out row" removes the desired row and is called when the pipe data matches as indicated by the previous helper. The "Is Same Pipe" function compares two pipes given to it and returns a binary result of either 1 or 0. The "row want" function simply creates a row vector from the index of a matrix. Finally, the Increase N function only compares the most recently added line to the rest of the Pipe Matrix. The b input is the index of the most recently added row. It also relies on helper functions discussed above.

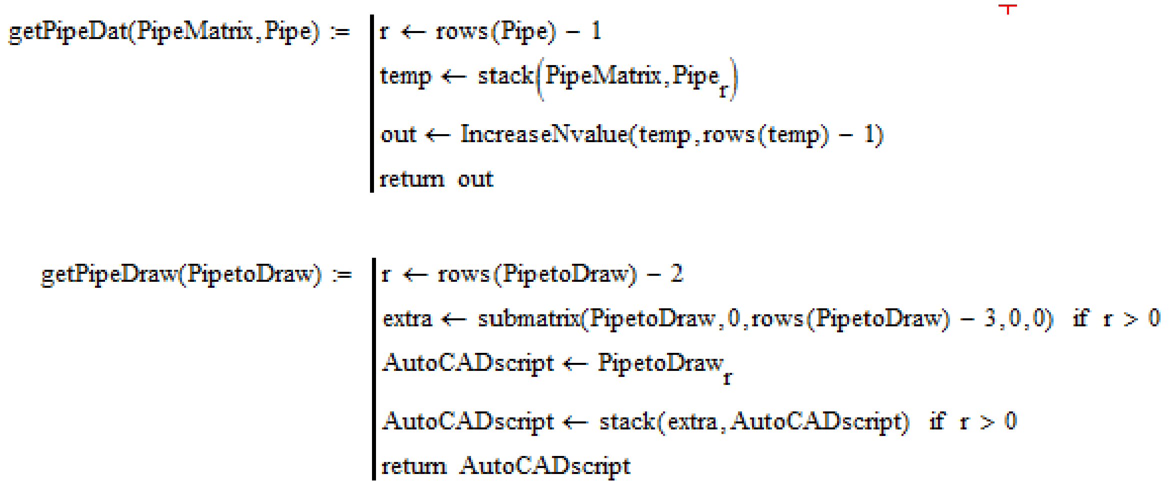


```
cutOutRow(matrix, b, duplicateRow) := | full ← 1  if  duplicateRow ≠ 0
                                      | topMatrix ← submatrix(matrix, 0, duplicateRow − 1, 0, 3)  if  full = 1
                                      | botMatrix ← submatrix(matrix, duplicateRow + 1, b, 0, 3)  if  full = 1
                                      | matrix ← stack(topMatrix, botMatrix)  if  full = 1
                                      | matrix ← submatrix(matrix, duplicateRow + 1, b, 0, 3)  if  duplicateRow = 0
                                      | return (matrix)

isSamePipe(rowComp, rowTest) := | a ← stack($(rowComp_{0,0}, rowComp_{0,1}, rowComp_{0,2})$)
                                | b ← stack($(rowTest_{0,0}, rowTest_{0,1}, rowTest_{0,2})$)
                                | same ← 1  if  $a_0 = b_0 \wedge a_1 = b_1 \wedge a_2 = b_2$
                                | same ← 0  otherwise
                                | return same

rowWant(matrix, b) := row ← $(matrix_{b,0}\ \ matrix_{b,1}\ \ matrix_{b,2}\ \ matrix_{b,3})$

IncreaseNvalue(dataRows, b) := | rowComp ← rowWant(dataRows, b)
                               | n ← b − 1
                               | while  n > −1
                               |     | rowTest ← rowWant(dataRows, n)
                               |     | shrink ← 0
                               |     | shrink ← 1  if  isSamePipe(rowComp, rowTest) = 1
                               |     | $dataRows_{b,3}$ ← $dataRows_{b,3}$ + $rowTest_{0,3}$  if  shrink = 1
                               |     | dataRows ← cutOutRow(dataRows, b, n)  if  shrink = 1
                               |     | b ← b − 1  if  shrink = 1
                               |     | n ← n − 1
                               | return dataRows
```

Figure 3: Helper Functions for Pipe Matrix Additions

Implementing Pipe Matrix Extraction Code

Every time that a pipe is stacked into a larger section of script, this pipe is assumed to be used in the drawing. At the same time that the pipe is added to a stack, its data will be added to the growing Pipe Matrix. The two codes below in Figure 4 show the functions that will modify the pipe to take only the necessary cell: the top cell for the drawing and the bottom cell for the Pipe Matrix. Note that these functions allow for pipes which contain extraneous drawing code above the line which specifically draws a pipe.



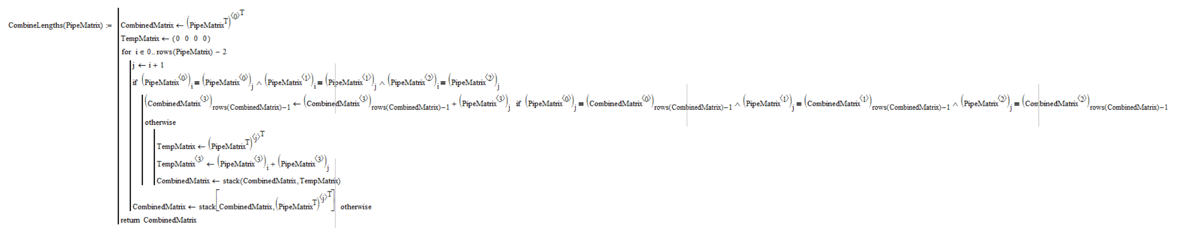
```
getPipeDat(PipeMatrix, Pipe) :=  | r ← rows(Pipe) − 1
                                 | temp ← stack(PipeMatrix, Pipe_r)
                                 | out ← IncreaseNvalue(temp, rows(temp) − 1)
                                 | return out

getPipeDraw(PipetoDraw) :=  | r ← rows(PipetoDraw) − 2
                            | extra ← submatrix(PipetoDraw, 0, rows(PipetoDraw) − 3, 0, 0)  if r > 0
                            | AutoCADscript ← PipetoDraw_r
                            | AutoCADscript ← stack(extra, AutoCADscript)  if r > 0
                            | return AutoCADscript
```

Figure 4: getPipe Functions

These functions cannot be called explicitly every time a pipe appears in a stack. When a pipe is followed by an array or mirror script, the pipe's data must be added to the pipe matrix an additional amount of times depending on the array and mirror script. The implementation involves copying these values into loops. A second type of complication is the conditional stack of pipes. A simple way to implement this is to mimic the format of the drawing stack for the pipe matrix additions. This appears in the drawing code for the Entrance Tank. Multiple pipes are created, but at the end of the drawing code, only some are stacked into the final drawing script. This means that the general method of pairing a Pipe Matrix addition with the first stack of a pipe is not useful because about half of the pipes stacked are not used. Instead, the Pipe Matrix is added to only at the end of the code, following the same logic as the drawing stack. Here, it is important to keep track of the individual pipes included in smaller stacks. Ultimately, the Pipe Matrix or Matrices from individual sections will be referenced by the lance cutting code.

Combining Rows

The CombineLengths code was written with simplification in mind. The code takes an array matrix containing pipe information and simplifies it by combining any rows that have the same length, ND and EN values. The n values for all rows are added together. This code is used especially after the PipeMatrix containing shorter pipe information and the matrix leftover from cutting full lances out of longer pipes are combined (as in 1(e)i.D above). There had been code written for this purpose in previous versions of the file, but the code was not working properly; the code has been fixed to what is seen in Figure 5.

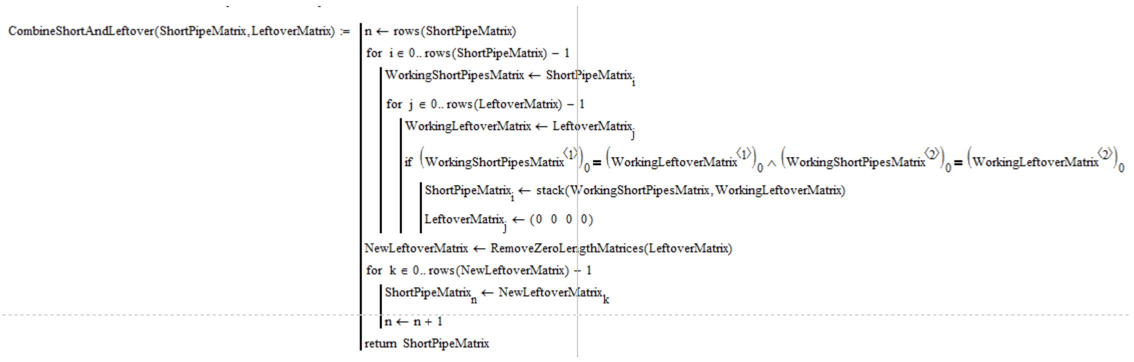


```
CombineLengths(PipeMatrix) := | CombinedMatrix ← $((PipeMatrix^T)^{\langle 0\rangle})^T$
                              | TempMatrix ← (0 0 0 0)
                              | for i ∈ 0.. rows(PipeMatrix) − 2
                              |   | j ← i + 1
                              |   | if $(PipeMatrix^{\langle 0\rangle})_i = (PipeMatrix^{\langle 0\rangle})_j \wedge (PipeMatrix^{\langle 1\rangle})_i = (PipeMatrix^{\langle 1\rangle})_j \wedge (PipeMatrix^{\langle 2\rangle})_i = (PipeMatrix^{\langle 2\rangle})_j$
                              |   |   | $(CombinedMatrix^{\langle 3\rangle})_{rows(CombinedMatrix)-1} \leftarrow (CombinedMatrix^{\langle 3\rangle})_{rows(CombinedMatrix)-1} + (PipeMatrix^{\langle 3\rangle})_j$ if $(PipeMatrix^{\langle 0\rangle})_j = (CombinedMatrix^{\langle 0\rangle})_{rows(CombinedMatrix)-1} \wedge (PipeMatrix^{\langle 1\rangle})_j = (CombinedMatrix^{\langle 1\rangle})_{rows(CombinedMatrix)-1} \wedge (PipeMatrix^{\langle 2\rangle})_j = (CombinedMatrix^{\langle 2\rangle})_{rows(CombinedMatrix)-1}$
                              |   |   | otherwise
                              |   |   |   | TempMatrix ← $((PipeMatrix^T)^{\langle j\rangle})^T$
                              |   |   |   | $TempMatrix^{\langle 3\rangle} \leftarrow (PipeMatrix^{\langle 3\rangle})_i + (PipeMatrix^{\langle 3\rangle})_j$
                              |   |   |   | CombinedMatrix ← stack(CombinedMatrix, TempMatrix)
                              |   | CombinedMatrix ← stack$\left[CombinedMatrix, ((PipeMatrix^T)^{\langle j\rangle})^T\right]$ otherwise
                              | return CombinedMatrix
```

Figure 5:

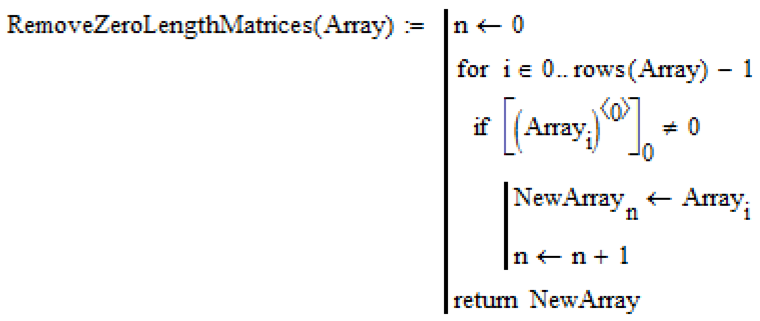
Combining Pipe Matrices

The step for combining two matrices as detailed in step 1(e)i.D of the algorithm was not started as of the end of the Spring 2013 semester. The code in Figure 6 (and for the subfunction in Figure 7) was written to combine the two array pipe matrices for use in the cutting optimization code.



```
CombineShortAndLeftover(ShortPipeMatrix, LeftoverMatrix) := | n ← rows(ShortPipeMatrix)
                                                            | for i ∈ 0.. rows(ShortPipeMatrix) − 1
                                                            |   | WorkingShortPipesMatrix ← $ShortPipeMatrix_i$
                                                            |   | for j ∈ 0.. rows(LeftoverMatrix) − 1
                                                            |   |   | WorkingLeftoverMatrix ← $LeftoverMatrix_j$
                                                            |   |   | if $(WorkingShortPipesMatrix^{\langle 1\rangle})_0 = (WorkingLeftoverMatrix^{\langle 1\rangle})_0 \wedge (WorkingShortPipesMatrix^{\langle 2\rangle})_0 = (WorkingLeftoverMatrix^{\langle 2\rangle})_0$
                                                            |   |   |   | $ShortPipeMatrix_i$ ← stack(WorkingShortPipesMatrix, WorkingLeftoverMatrix)
                                                            |   |   |   | $LeftoverMatrix_j$ ← (0 0 0 0)
                                                            | NewLeftoverMatrix ← RemoveZeroLengthMatrices(LeftoverMatrix)
                                                            | for k ∈ 0.. rows(NewLeftoverMatrix) − 1
                                                            |   | $ShortPipeMatrix_n$ ← $NewLeftoverMatrix_k$
                                                            |   | n ← n + 1
                                                            | return ShortPipeMatrix
```

Figure 6:

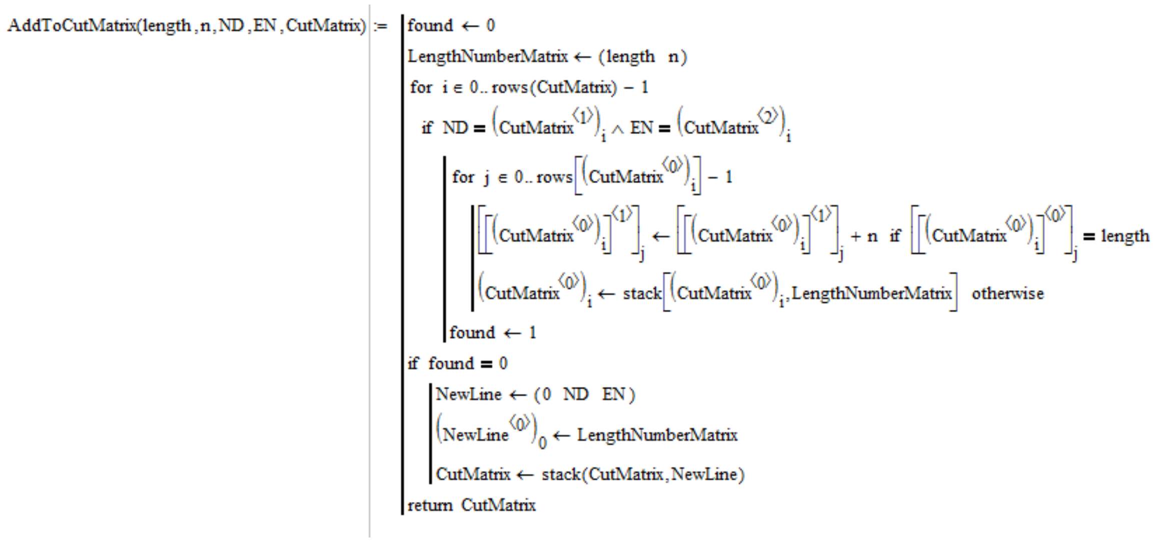


```
RemoveZeroLengthMatrices(Array) :=
    $n \leftarrow 0$
    for $i \in 0..\mathrm{rows}(Array) - 1$
      if $\left[\left(Array_i\right)^{\langle 0 \rangle}\right]_0 \neq 0$
          $NewArray_n \leftarrow Array_i$
          $n \leftarrow n + 1$
    return NewArray
```

Figure 7:

CutMatrix Additions

After struggling to debug the cutting optimization code during the Spring 2013 semester, it was decided that the best course of action would be to start from scratch. This especially seemed prudent since the CutMatrix concept was rethought and it was decided that the column that was originally supposed to hold a vector containing lengths that were cut out of the lance would need to be altered to be a matrix with both the lengths and a number of said lengths n, since there will be differing numbers of pipes of a particular length needed (depending on the original number of lengths needed). The code in Figure 8 was created for this purpose, which will simplify the overall cutting optimization code.

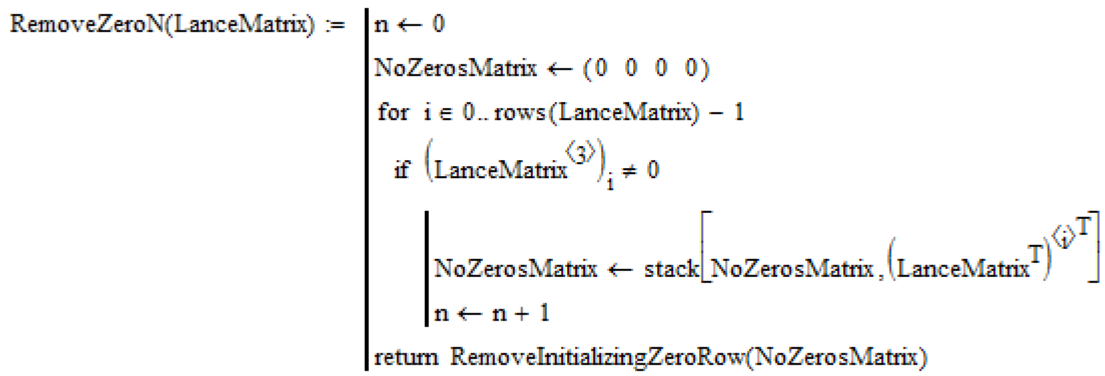


```
AddToCutMatrix(length, n, ND, EN, CutMatrix) :=
    $found \leftarrow 0$
    $LengthNumberMatrix \leftarrow (length \;\; n)$
    for $i \in 0..\mathrm{rows}(CutMatrix) - 1$
      if $ND = \left(CutMatrix^{\langle 1 \rangle}\right)_i \wedge EN = \left(CutMatrix^{\langle 2 \rangle}\right)_i$
          for $j \in 0..\mathrm{rows}\left[\left(CutMatrix^{\langle 0 \rangle}\right)_i\right] - 1$
              $\left[\left[\left(CutMatrix^{\langle 0 \rangle}\right)_i\right]^{\langle 1 \rangle}\right]_j \leftarrow \left[\left[\left(CutMatrix^{\langle 0 \rangle}\right)_i\right]^{\langle 1 \rangle}\right]_j + n$ if $\left[\left[\left(CutMatrix^{\langle 0 \rangle}\right)_i\right]^{\langle 0 \rangle}\right]_j = length$
              $\left(CutMatrix^{\langle 0 \rangle}\right)_i \leftarrow \mathrm{stack}\left[\left(CutMatrix^{\langle 0 \rangle}\right)_i, LengthNumberMatrix\right]$ otherwise
          $found \leftarrow 1$
    if $found = 0$
        $NewLine \leftarrow (0 \;\; ND \;\; EN)$
        $\left(NewLine^{\langle 0 \rangle}\right)_0 \leftarrow LengthNumberMatrix$
        $CutMatrix \leftarrow \mathrm{stack}(CutMatrix, NewLine)$
    return CutMatrix
```

Figure 8:

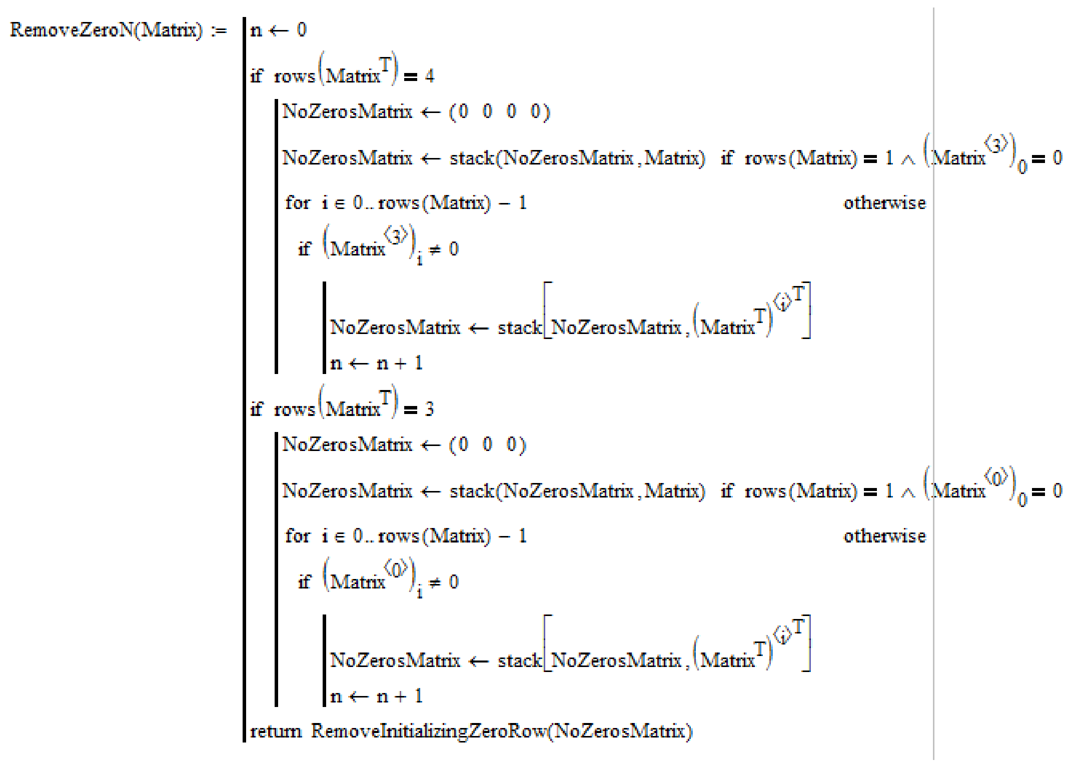
Redundancy Function Modification

There is a previously existing helping function that is used in the cutting optimization code that removes rows with no pipes left ($N = 0$) so that the code doesn't use the corresponding specifications by accident. The code (shown in Figure 9) has been modified (as in Figure 10) to work for both pipe matrices (with length, ND, EN and N columns) as well as for the CutMatrix (with its array of three columns). This will hopefully help to simplify the cutting optimization code.



```
RemoveZeroN(LanceMatrix) := | n ← 0
                            | NoZerosMatrix ← (0  0  0  0)
                            | for  i ∈ 0.. rows(LanceMatrix) − 1
                            |   if  $(LanceMatrix^{\langle 3 \rangle})_i \neq 0$
                            |      | NoZerosMatrix ← stack$\left[NoZerosMatrix, \left((LanceMatrix^{T})^{\langle i \rangle}\right)^{T}\right]$
                            |      | n ← n + 1
                            | return  RemoveInitializingZeroRow(NoZerosMatrix)
```

Figure 9:

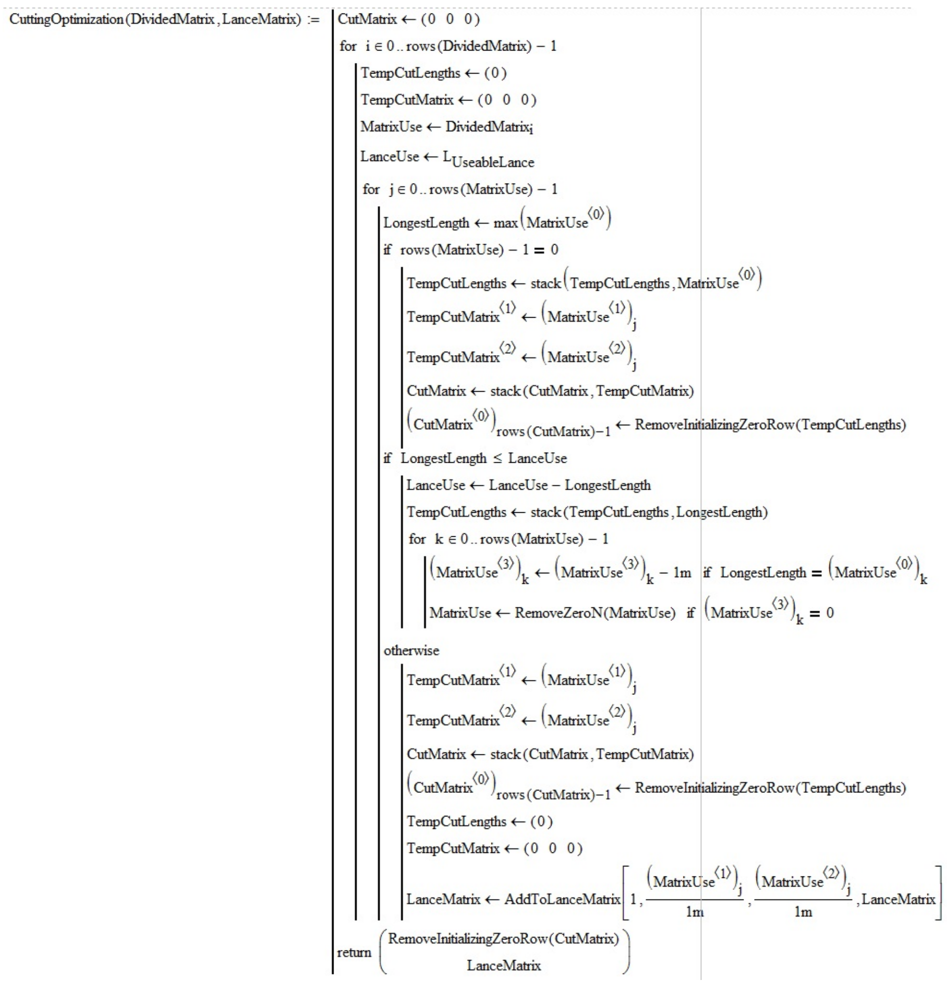


```
RemoveZeroN(Matrix) :=
  n ← 0
  if rows(Matrix^T) = 4
     NoZerosMatrix ← (0  0  0  0)
     NoZerosMatrix ← stack(NoZerosMatrix, Matrix)  if rows(Matrix) = 1 ∧ (Matrix^⟨3⟩)_0 = 0
     for i ∈ 0..rows(Matrix) − 1                   otherwise
       if (Matrix^⟨3⟩)_i ≠ 0
          NoZerosMatrix ← stack[NoZerosMatrix, ((Matrix^T)^⟨i⟩)^T]
          n ← n + 1
  if rows(Matrix^T) = 3
     NoZerosMatrix ← (0  0  0)
     NoZerosMatrix ← stack(NoZerosMatrix, Matrix)  if rows(Matrix) = 1 ∧ (Matrix^⟨0⟩)_0 = 0
     for i ∈ 0..rows(Matrix) − 1                   otherwise
       if (Matrix^⟨0⟩)_i ≠ 0
          NoZerosMatrix ← stack[NoZerosMatrix, ((Matrix^T)^⟨i⟩)^T]
          n ← n + 1
  return RemoveInitializingZeroRow(NoZerosMatrix)
```

Figure 10:

Cutting Optimization Code Debugging

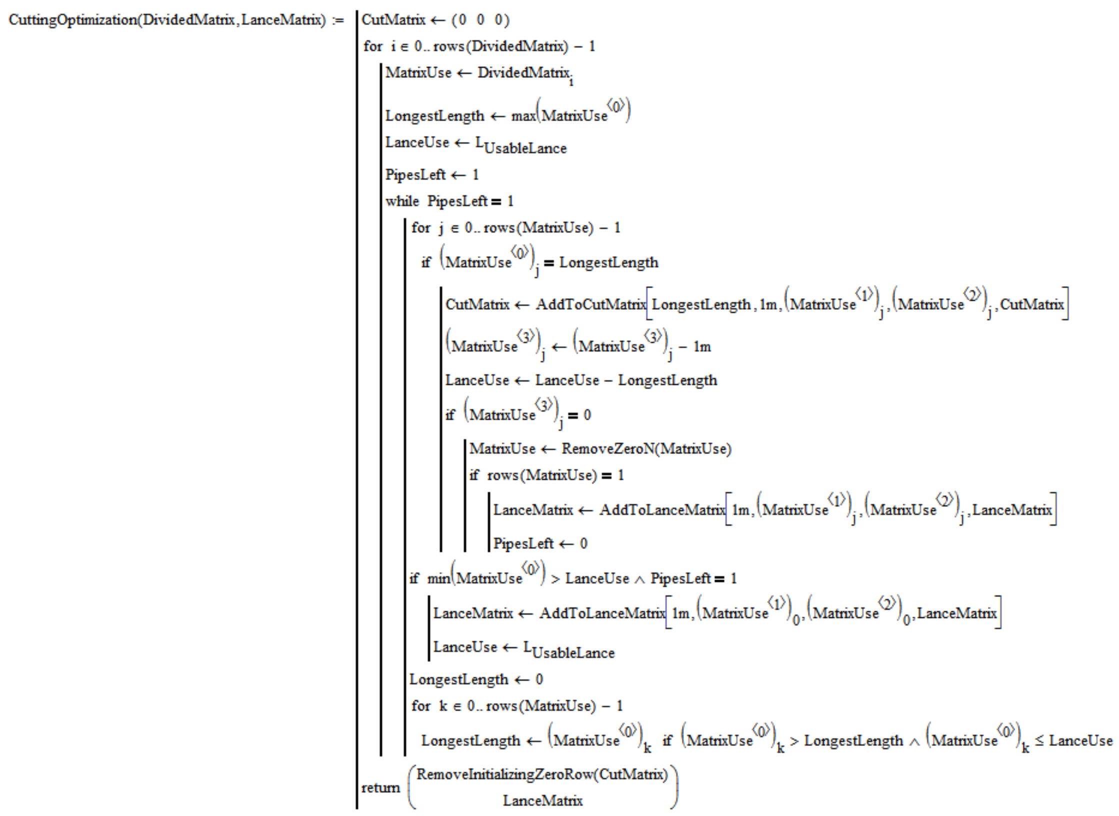
Over the course of the semester, several different versions of the cutting optimization code have been tested and worked on. The first was a carryover from previous work in Spring 2013, as shown in Figure 11. It proved to be difficult to debug this code, so it was scrapped and coding was started over from scratch.



```
CuttingOptimization(DividedMatrix, LanceMatrix) :=
  CutMatrix ← (0 0 0)
  for i ∈ 0..rows(DividedMatrix) − 1
    TempCutLengths ← (0)
    TempCutMatrix ← (0 0 0)
    MatrixUse ← $\text{DividedMatrix}_i$
    LanceUse ← $L_{\text{UseableLance}}$
    for j ∈ 0..rows(MatrixUse) − 1
      LongestLength ← $\max\left(\text{MatrixUse}^{\langle 0\rangle}\right)$
      if rows(MatrixUse) − 1 = 0
        TempCutLengths ← $\text{stack}\left(\text{TempCutLengths}, \text{MatrixUse}^{\langle 0\rangle}\right)$
        $\text{TempCutMatrix}^{\langle 1\rangle} \leftarrow \left(\text{MatrixUse}^{\langle 1\rangle}\right)_j$
        $\text{TempCutMatrix}^{\langle 2\rangle} \leftarrow \left(\text{MatrixUse}^{\langle 2\rangle}\right)_j$
        CutMatrix ← stack(CutMatrix, TempCutMatrix)
        $\left(\text{CutMatrix}^{\langle 0\rangle}\right)_{\text{rows(CutMatrix)}-1} \leftarrow$ RemoveInitializingZeroRow(TempCutLengths)
      if LongestLength ≤ LanceUse
        LanceUse ← LanceUse − LongestLength
        TempCutLengths ← stack(TempCutLengths, LongestLength)
        for k ∈ 0..rows(MatrixUse) − 1
          $\left(\text{MatrixUse}^{\langle 3\rangle}\right)_k \leftarrow \left(\text{MatrixUse}^{\langle 3\rangle}\right)_k - 1\text{m}$ if $\text{LongestLength} = \left(\text{MatrixUse}^{\langle 0\rangle}\right)_k$
          MatrixUse ← RemoveZeroN(MatrixUse) if $\left(\text{MatrixUse}^{\langle 3\rangle}\right)_k = 0$
      otherwise
        $\text{TempCutMatrix}^{\langle 1\rangle} \leftarrow \left(\text{MatrixUse}^{\langle 1\rangle}\right)_j$
        $\text{TempCutMatrix}^{\langle 2\rangle} \leftarrow \left(\text{MatrixUse}^{\langle 2\rangle}\right)_j$
        CutMatrix ← stack(CutMatrix, TempCutMatrix)
        $\left(\text{CutMatrix}^{\langle 0\rangle}\right)_{\text{rows(CutMatrix)}-1} \leftarrow$ RemoveInitializingZeroRow(TempCutLengths)
        TempCutLengths ← (0)
        TempCutMatrix ← (0 0 0)
        LanceMatrix ← $\text{AddToLanceMatrix}\left[1, \frac{\left(\text{MatrixUse}^{\langle 1\rangle}\right)_j}{1\text{m}}, \frac{\left(\text{MatrixUse}^{\langle 2\rangle}\right)_j}{1\text{m}}, \text{LanceMatrix}\right]$
  return $\begin{pmatrix}\text{RemoveInitializingZeroRow(CutMatrix)}\\ \text{LanceMatrix}\end{pmatrix}$
```

Figure 11:

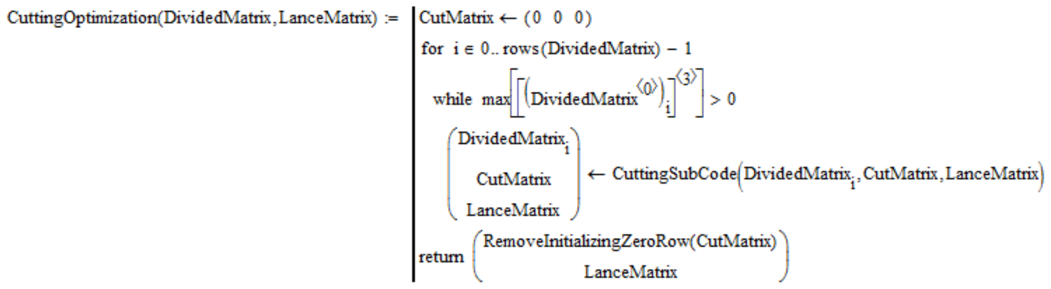
The next attempt to code the original algorithm resulted in a function (Figure 12) that never seemed to evaluate. It's unknown as to why the code doesn't work; after testing other cases, it seems that the incorporation of a while loop in this particular code caused the problem, but we could not come up with an alternative.



```
CuttingOptimization(DividedMatrix, LanceMatrix) :=
  CutMatrix ← (0 0 0)
  for i ∈ 0..rows(DividedMatrix) − 1
      MatrixUse ← $\text{DividedMatrix}_i$
      LongestLength ← $\max\left(\text{MatrixUse}^{\langle 0\rangle}\right)$
      LanceUse ← $L_{\text{UsableLance}}$
      PipesLeft ← 1
      while PipesLeft = 1
          for j ∈ 0..rows(MatrixUse) − 1
            if $\left(\text{MatrixUse}^{\langle 0\rangle}\right)_j$ = LongestLength
                CutMatrix ← AddToCutMatrix[LongestLength, 1m, $\left(\text{MatrixUse}^{\langle 1\rangle}\right)_j$, $\left(\text{MatrixUse}^{\langle 2\rangle}\right)_j$, CutMatrix]
                $\left(\text{MatrixUse}^{\langle 3\rangle}\right)_j \leftarrow \left(\text{MatrixUse}^{\langle 3\rangle}\right)_j - 1\text{m}$
                LanceUse ← LanceUse − LongestLength
                if $\left(\text{MatrixUse}^{\langle 3\rangle}\right)_j$ = 0
                    MatrixUse ← RemoveZeroN(MatrixUse)
                    if rows(MatrixUse) = 1
                        LanceMatrix ← AddToLanceMatrix[1m, $\left(\text{MatrixUse}^{\langle 1\rangle}\right)_j$, $\left(\text{MatrixUse}^{\langle 2\rangle}\right)_j$, LanceMatrix]
                        PipesLeft ← 0
          if $\min\left(\text{MatrixUse}^{\langle 0\rangle}\right)$ > LanceUse ∧ PipesLeft = 1
              LanceMatrix ← AddToLanceMatrix[1m, $\left(\text{MatrixUse}^{\langle 1\rangle}\right)_0$, $\left(\text{MatrixUse}^{\langle 2\rangle}\right)_0$, LanceMatrix]
              LanceUse ← $L_{\text{UsableLance}}$
          LongestLength ← 0
          for k ∈ 0..rows(MatrixUse) − 1
            LongestLength ← $\left(\text{MatrixUse}^{\langle 0\rangle}\right)_k$ if $\left(\text{MatrixUse}^{\langle 0\rangle}\right)_k$ > LongestLength ∧ $\left(\text{MatrixUse}^{\langle 0\rangle}\right)_k$ ≤ LanceUse
  return ( RemoveInitializingZeroRow(CutMatrix)
           LanceMatrix )
```

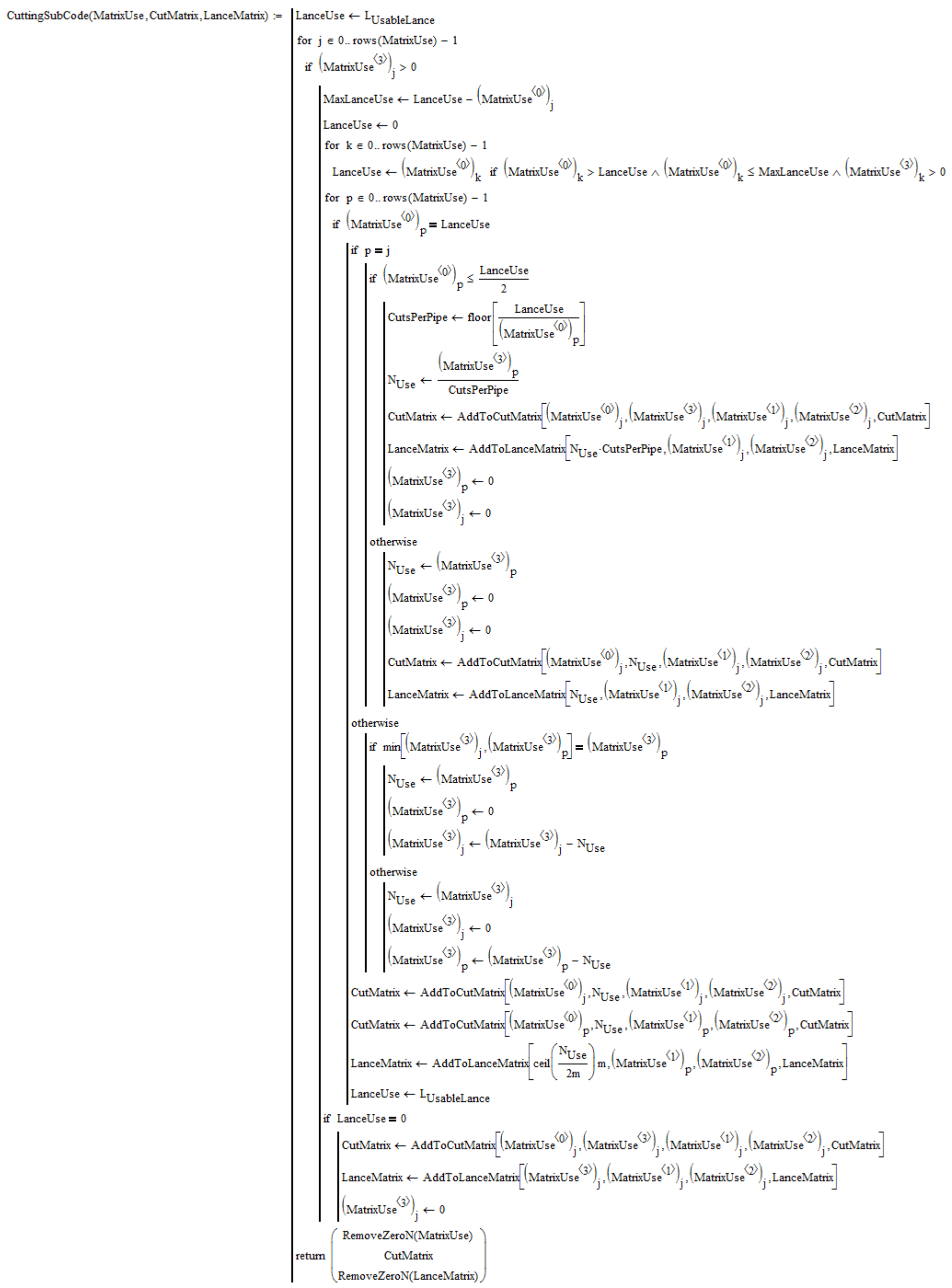
Figure 12:

Keeping the base looping parts of the code, we altered it and the algorithm to contain no while loops. As previously stated, the code is less accurate but needs less calculations overall, which is most likely why it works while previous attempts had not. The function has been separated (at least for now) into a subfunction that performs the actual algorithm and the code that iterates the cutting algorithm through the separated matrix (called CuttingSubCode and CuttingOptimization respectively at the moment). This was done both for simplicity's sake and so that the code would be easier to debug. The new version of the CuttingOptimization function is shown in Figure 13; it consists of the initialized variables and the for loop that goes through each matrix within the larger cell array. The CuttingSubCode function (shown in Figure 14) consists of the internal nested loops within the original, larger function that affect each individual matrix. To match it up with the algorithm stated in problem definition, Figure 14 corresponds to step 1.(e).ii and Figure 13 to step 1.(e).iii.



```
CuttingOptimization(DividedMatrix, LanceMatrix) :=
    CutMatrix ← (0  0  0)
    for  i ∈ 0.. rows(DividedMatrix) − 1
        while  max$\left[\left[\left(\text{DividedMatrix}^{\langle 0\rangle}\right)_i\right]^{\langle 3\rangle}\right] > 0$
            $\begin{pmatrix}\text{DividedMatrix}_i\\ \text{CutMatrix}\\ \text{LanceMatrix}\end{pmatrix}$ ← CuttingSubCode(DividedMatrix$_i$, CutMatrix, LanceMatrix)
    return $\begin{pmatrix}\text{RemoveInitializingZeroRow(CutMatrix)}\\ \text{LanceMatrix}\end{pmatrix}$
```

Figure 13:



```
$\text{CuttingSubCode}(\text{MatrixUse}, \text{CutMatrix}, \text{LanceMatrix}) :=$
  $\text{LanceUse} \leftarrow L_{\text{UsableLance}}$
  for $j \in 0..\text{rows}(\text{MatrixUse}) - 1$
    if $(\text{MatrixUse}^{\langle 3 \rangle})_j > 0$
      $\text{MaxLanceUse} \leftarrow \text{LanceUse} - (\text{MatrixUse}^{\langle 0 \rangle})_j$
      $\text{LanceUse} \leftarrow 0$
      for $k \in 0..\text{rows}(\text{MatrixUse}) - 1$
        $\text{LanceUse} \leftarrow (\text{MatrixUse}^{\langle 0 \rangle})_k$ if $(\text{MatrixUse}^{\langle 0 \rangle})_k > \text{LanceUse} \wedge (\text{MatrixUse}^{\langle 0 \rangle})_k \le \text{MaxLanceUse} \wedge (\text{MatrixUse}^{\langle 3 \rangle})_k > 0$
      for $p \in 0..\text{rows}(\text{MatrixUse}) - 1$
        if $(\text{MatrixUse}^{\langle 0 \rangle})_p = \text{LanceUse}$
          if $p = j$
            if $(\text{MatrixUse}^{\langle 0 \rangle})_p \le \frac{\text{LanceUse}}{2}$
              $\text{CutsPerPipe} \leftarrow \text{floor}\left[\frac{\text{LanceUse}}{(\text{MatrixUse}^{\langle 0 \rangle})_p}\right]$
              $N_{\text{Use}} \leftarrow \frac{(\text{MatrixUse}^{\langle 3 \rangle})_p}{\text{CutsPerPipe}}$
              $\text{CutMatrix} \leftarrow \text{AddToCutMatrix}\left[(\text{MatrixUse}^{\langle 0 \rangle})_j, (\text{MatrixUse}^{\langle 3 \rangle})_j, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{CutMatrix}\right]$
              $\text{LanceMatrix} \leftarrow \text{AddToLanceMatrix}\left[N_{\text{Use}} \cdot \text{CutsPerPipe}, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{LanceMatrix}\right]$
              $(\text{MatrixUse}^{\langle 3 \rangle})_p \leftarrow 0$
              $(\text{MatrixUse}^{\langle 3 \rangle})_j \leftarrow 0$
            otherwise
              $N_{\text{Use}} \leftarrow (\text{MatrixUse}^{\langle 3 \rangle})_p$
              $(\text{MatrixUse}^{\langle 3 \rangle})_p \leftarrow 0$
              $(\text{MatrixUse}^{\langle 3 \rangle})_j \leftarrow 0$
              $\text{CutMatrix} \leftarrow \text{AddToCutMatrix}\left[(\text{MatrixUse}^{\langle 0 \rangle})_j, N_{\text{Use}}, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{CutMatrix}\right]$
              $\text{LanceMatrix} \leftarrow \text{AddToLanceMatrix}\left[N_{\text{Use}}, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{LanceMatrix}\right]$
          otherwise
            if $\min\left[(\text{MatrixUse}^{\langle 3 \rangle})_j, (\text{MatrixUse}^{\langle 3 \rangle})_p\right] = (\text{MatrixUse}^{\langle 3 \rangle})_p$
              $N_{\text{Use}} \leftarrow (\text{MatrixUse}^{\langle 3 \rangle})_p$
              $(\text{MatrixUse}^{\langle 3 \rangle})_p \leftarrow 0$
              $(\text{MatrixUse}^{\langle 3 \rangle})_j \leftarrow (\text{MatrixUse}^{\langle 3 \rangle})_j - N_{\text{Use}}$
            otherwise
              $N_{\text{Use}} \leftarrow (\text{MatrixUse}^{\langle 3 \rangle})_j$
              $(\text{MatrixUse}^{\langle 3 \rangle})_j \leftarrow 0$
              $(\text{MatrixUse}^{\langle 3 \rangle})_p \leftarrow (\text{MatrixUse}^{\langle 3 \rangle})_p - N_{\text{Use}}$
            $\text{CutMatrix} \leftarrow \text{AddToCutMatrix}\left[(\text{MatrixUse}^{\langle 0 \rangle})_j, N_{\text{Use}}, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{CutMatrix}\right]$
            $\text{CutMatrix} \leftarrow \text{AddToCutMatrix}\left[(\text{MatrixUse}^{\langle 0 \rangle})_p, N_{\text{Use}}, (\text{MatrixUse}^{\langle 1 \rangle})_p, (\text{MatrixUse}^{\langle 2 \rangle})_p, \text{CutMatrix}\right]$
            $\text{LanceMatrix} \leftarrow \text{AddToLanceMatrix}\left[\text{ceil}\left(\frac{N_{\text{Use}}}{2m}\right)m, (\text{MatrixUse}^{\langle 1 \rangle})_p, (\text{MatrixUse}^{\langle 2 \rangle})_p, \text{LanceMatrix}\right]$
            $\text{LanceUse} \leftarrow L_{\text{UsableLance}}$
      if $\text{LanceUse} = 0$
        $\text{CutMatrix} \leftarrow \text{AddToCutMatrix}\left[(\text{MatrixUse}^{\langle 0 \rangle})_j, (\text{MatrixUse}^{\langle 3 \rangle})_j, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{CutMatrix}\right]$
        $\text{LanceMatrix} \leftarrow \text{AddToLanceMatrix}\left[(\text{MatrixUse}^{\langle 3 \rangle})_j, (\text{MatrixUse}^{\langle 1 \rangle})_j, (\text{MatrixUse}^{\langle 2 \rangle})_j, \text{LanceMatrix}\right]$
        $(\text{MatrixUse}^{\langle 3 \rangle})_j \leftarrow 0$
  return $\begin{pmatrix} \text{RemoveZeroN}(\text{MatrixUse}) \\ \text{CutMatrix} \\ \text{RemoveZeroN}(\text{LanceMatrix}) \end{pmatrix}$
```

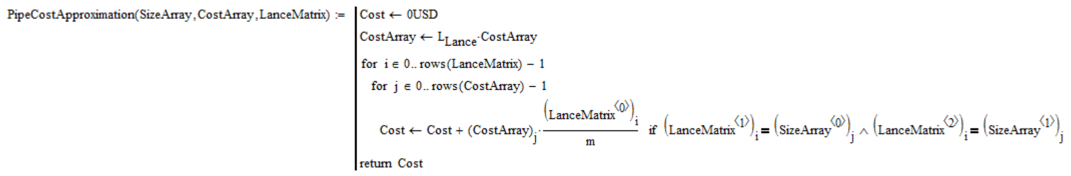
Figure 14:

Cost Function Implementation

After getting a LanceMatrix output from the cutting optimization code in Figure 14 and Figure 13, we can finally calculate the costs of the pipes in the PipeMatrix. Basing the cost arrays on Julia Morris's previous work on the cost of sedimentation tanks from the Spring 2012 semester, we can create two inputs given a cost array (such as the example as shown in Figure 15). The size array is a total list of different pipe specifications possible within the plant. Each row contains a unique combination of ND and EN values. The cost array corresponds with the size array one-to-one (i.e. the first row of the cost array is the cost of a pipe given in the first row of the size array) and gives pipe prices in USD per unit length. We can then use a function, as shown in Figure 16, to match up costs with numbers of lances needed to calculate first the costs of each type of pipe needed and then sum those values to determine the total cost of pipes. The output of the function is the total cost of all the pipes contained within the original PipeMatrix using the LanceMatrix calculated using the algorithm stated in the Problem Definition.

$$\mathrm{PipeSizeArray}_{PVC} := \begin{array}{c} \begin{array}{cc} \mathrm{ND} & \mathrm{EN} \end{array} \\ \begin{pmatrix} 1\mathrm{in} & 2\mathrm{m} \\ 1.5\mathrm{in} & 2\mathrm{m} \\ 2\mathrm{in} & 8\mathrm{m} \\ 3\mathrm{in} & 8\mathrm{m} \\ 4\mathrm{in} & 8\mathrm{m} \\ 6\mathrm{in} & 8\mathrm{m} \\ 8\mathrm{in} & 8\mathrm{m} \end{pmatrix} \end{array} \qquad \mathrm{CostArray}_{PVC} := \begin{pmatrix} \frac{5}{6} \\ \frac{10}{6} \\ \frac{15}{6} \\ \frac{20}{6} \\ \frac{25.41}{6} \\ \frac{70.83}{6} \\ \frac{127.96}{6} \end{pmatrix} \cdot \frac{\mathrm{USD}}{\mathrm{m}}$$

Figure 15:



```
PipeCostApproximation(SizeArray, CostArray, LanceMatrix) := | Cost ← 0USD
                                                            | CostArray ← $L_{Lance} \cdot$ CostArray
                                                            | for i ∈ 0.. rows(LanceMatrix) − 1
                                                            |   for j ∈ 0.. rows(CostArray) − 1
                                                            |     Cost ← Cost + $(CostArray)_j \cdot \frac{(LanceMatrix^{\langle 0 \rangle})_i}{m}$ if $(LanceMatrix^{\langle 1 \rangle})_i = (SizeArray^{\langle 0 \rangle})_j \wedge (LanceMatrix^{\langle 2 \rangle})_i = (SizeArray^{\langle 1 \rangle})_j$
                                                            | return Cost
```

Figure 16:

Pipe Matrix Extraction

The major progress in pipe matrix extraction has been to include the Array and Mirror functions in the addition of pipe addresses. The current method is to use if statements to add the pipes, and to treat arrays and mirrors simply as copies of pipes.

Part III
Future Work

Pipe Matrix Extraction

The future work for extracting the Pipe Matrix from the drawing code is to add relevant code into all plant components. More importantly, the Pipe Matrix entries need to be evaluated for correctness using data from field sites. Also, as discussed above, the Pipe Matrix entries are added using methods that mirror the drawing code in terms of arrays and conditional statements, so they are not flexible and need to be changed independently as the pipe drawing functions are changed. One possible simplification of the helper functions is to allow multiple pipes to be stacked at one call of the getPipeDat function. Therefore, in order to modify existing components, the Pipe Matrix code will also need to be checked.

Cutting Optimization

Ideally, future work will be able to alter the code for the original cutting optimization algorithm so that MathCAD is able to fully evaluate it. Even if that is not possible in the end, the current algorithm (coded in Figure 14 and Figure 13) can be further refined so that it more accurately reflects the most optimized method of cutting pipes. The current code is meant to be a stand-in so that an approximate price can be calculated, but will ideally be improved so that the price can better reflect the actual cost of building a plant in the real world.

Cost Inputs

The size and cost arrays shown in Figure 15 are based off of prices Julia Morris found in 2012, and may not accurately reflect prices of pipes currently. Her previous code also did not take pipe EN values into account, so it is not certain if the values shown in the figure even correspond to the correct types of pipes. There will have to be future research done to determine accurate prices for the pipes needed to build a plant as given in the drawing code.

Other Materials

The work detailed in this report only reflects the coding that has been completed relating to pipe costs. There is an entire list of other materials given in the Problem Definition section of this report that have yet to be looked into. This will be the work of team members in future semesters.