3

Matrix-Vector Operations

3.1 Opening Remarks
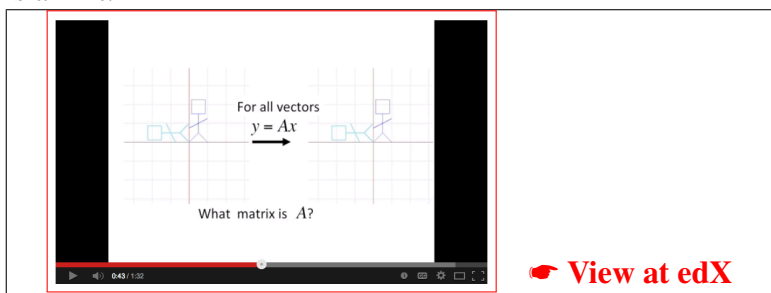
3.1.1 Timmy Two Space

☛ **View at edX**
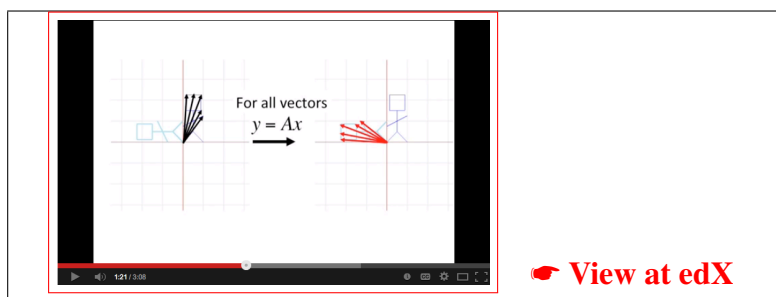
Homework 3.1.1.1 Click on the below link to open a browser window with the "Timmy Two Space" exercise. This exercise was suggested to us by our colleague Prof. Alan Cline. It was first implemented using an IPython Notebook by Ben Holder. During the Spring 2014 offering of LAFF on the edX platform, one of the participants, Ed McCardell, rewrote the activity as the below activity:

- ☛ Timmy! on the web.

- ☛ Timmy!.

If you get really frustrated, here is a hint:

☛ **View at edX**

99

3.1.2 Outline Week 3

3.1.3 What You Will Learn

Upon completion of this unit, you should be able to

- Recognize matrix-vector multiplication as a linear combination of the columns of the matrix.

- Given a linear transformation, determine the matrix that represents it.

- Given a matrix, determine the linear transformation that it represents.

- Connect special linear transformations to special matrices.

- Identify special matrices such as the zero matrix, the identity matrix, diagonal matrices, triangular matrices, and symmetric matrices.

- Transpose a matrix.

- Scale and add matrices.

- Exploit properties of special matrices.

- Extrapolate from concrete computation to algorithms for matrix-vector multiplication.

- Partition (slice and dice) matrices with and without special properties.

- Use partitioned matrices and vectors to represent algorithms for matrix-vector multiplication.

- Use partitioned matrices and vectors to represent algorithms in code.

3.2 Special Matrices

3.2.1 The Zero Matrix

☞ **View at edX**

Homework 3.2.1.1 Let $L_0 : \mathbb{R}^n \to \mathbb{R}^m$ be the function defined for every $x \in \mathbb{R}^n$ as $L_0(x) = 0$, where 0 denotes the zero vector "of appropriate size". L_0 is a linear transformation.

True/False

We will denote the matrix that represents L_0 by 0, where we typically know what its row and column sizes are from context (in this case, $0 \in \mathbb{R}^{m \times n}$). If it is not obvious, we may use a subscript ($0_{m \times n}$) to indicate its size, that is, m rows and n columns.

By the definition of a matrix, the jth column of matrix 0 is given by $L_0(e_j) = 0$ (a vector with m zero components). Thus, the matrix that represents L_0, which we will call the zero matrix, is given by the $m \times n$ matrix

$$0 = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

It is easy to check that for any $x \in \mathbb{R}^n$, $0_{m \times n} x_n = 0_m$.

Definition 3.1 *A matrix $A \in \mathbb{R}^{m \times n}$ equals the $m \times n$ zero matrix if all of its elements equal zero.*

Througout this course, we will use the number 0 to indicate a scalar, vector, or matrix of "appropriate size".

In Figure 3.1, we give an algorithm that, given an $m \times n$ matrix A, sets it to zero. Notice that it exposes columns one at a time, setting the exposed column to zero.

MATLAB provides the function "zeros" that returns a zero matrix of indicated size. Your are going to write your own, to helps you understand the material.

Algorithm: $[A] := \text{SET\_TO\_ZERO}(A)$

Partition $A \to \left(\begin{array}{c|c} A_L & A_R \end{array} \right)$

$\quad$ **where** A_L has 0 columns

while $n(A_L) < n(A)$ **do**

$\quad$ **Repartition**

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \to \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$$

$\quad$ **where** a_1 has 1 column

$\quad\quad a_1 := 0$ $\qquad\qquad$ (Set the current column to zero)

$\quad$ **Continue with**

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$$

endwhile

Figure 3.1: Algorithm for setting matrix A to the zero matrix.

Homework 3.2.1.2 With the FLAME API for MATLAB (FLAME@lab) implement the algorithm in Figure 3.1. You will use the function `laff_zerov( x )`, which returns a zero vector of the same size and shape (column or row) as input vector x. Since you are still getting used to programming with M-script and FLAME@lab, you may want to follow the instructions in this video:
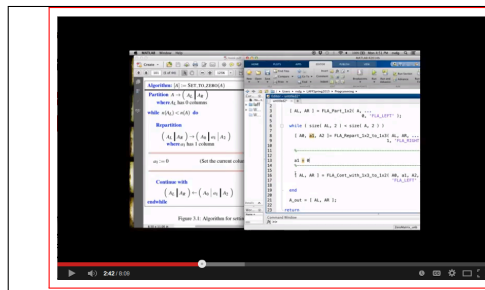
☛ **View at edX**

Some links that will come in handy:

- ☛ **Spark**. or ☛ `Local Spark`

- ☛ **PictureFLAME** or ☛ `PictureFLAME`

Homework 3.2.1.3 In the MATLAB Command Window, type

```
A = zeros( 5,4 )
```

What is the result?

Homework 3.2.1.4 Apply the zero matrix to ☞ Timmy Two Space. What happens?

1. Timmy shifts off the grid.

2. Timmy disappears into the origin.

3. Timmy becomes a line on the x-axis.

4. Timmy becomes a line on the y-axis.

5. Timmy doesn't change at all.

3.2.2 The Identity Matrix

☞ View at edX

Homework 3.2.2.1 Let $L_I : \mathbb{R}^n \to \mathbb{R}^n$ be the function defined for every $x \in \mathbb{R}^n$ as $L_I(x) = x$. L_I is a linear transformation.

True/False

We will denote the matrix that represents L_I by the letter I (capital "I") and call it the identity matrix. Usually, the size of the identity matrix is obvious from context. If not, we may use a subscript, I_n, to indicate the size, that is: a matrix that has n rows and n columns (and is hence a "square matrix").

Again, by the definition of a matrix, the jth column of I is given by $L_I(e_j) = e_j$. Thus, the identity matrix is given by

$$I = \left(\begin{array}{c|c|c|c} e_0 & e_1 & \cdots & e_{n-1} \end{array} \right) = \left(\begin{array}{c|c|c|c} 1 & 0 & \cdots & 0 \\ \hline 0 & 1 & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & 1 \end{array} \right).$$

Here, and frequently in the future, we use vertical lines to indicate a partitioning of a matrix into its columns. (Slicing and dicing again!) It is easy to check that $Ix = x$.

Definition 3.2 *A matrix $I \in \mathbb{R}^{n \times n}$ equals the $n \times n$ identity matrix if all its elements equal zero, except for the elements on the diagonal, which all equal one.*

The diagonal of a matrix A consists of the entries $\alpha_{0,0}$, $\alpha_{1,1}$, etc. In other words, all elements $\alpha_{i,i}$.

Througout this course, we will use the capital letter I to indicate an identity matrix "of appropriate size".

We now motivate an algorithm that, given an $n \times n$ matrix A, sets it to the identity matrix. We'll start by trying to closely mirror the `Set_to_zero` algorithm from the previous unit:

Algorithm: $[A] := \text{SET\_TO\_IDENTITY}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right)$
    where A_L has 0 columns

while $n(A_L) < n(A)$ **do**

   Repartition

   $\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$
       where a_1 has 1 column

   $a_1 := e_j$ (Set the current column to the correct unit basis vector)

   Continue with

   $\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$

endwhile

The problem is that our notation doesn't keep track of the column index, j. Another problem is that we don't have a routine to set a vector to the jth unit basis vector.

To overcome this, we recognize that the jth column of A, which in our algorithm above appears as a_1, and the jth unit basis vector can each be partitioned into three parts:

$$a_1 = a_j = \left(\begin{array}{c} a_{01} \\ \alpha_{11} \\ a_{21} \end{array} \right) \quad \text{and} \quad e_j = \left(\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right),$$

where the 0's refer to vectors of zeroes of appropriate size. To then set a_1 $(= a_j)$ to the unit basis vector, we can make the assignments

$$\begin{array}{rcl} a_{01} & := & 0 \\ \alpha_{11} & := & 1 \\ a_{21} & := & 0 \end{array}$$

The algorithm in Figure 3.2 very naturally exposes exactly these parts of the current column.

Algorithm: $[A] := \text{SET\_TO\_IDENTITY}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$

    where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

    Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

        where α_{11} is 1×1

---

    set current column to appropriate unit basis vector

    $a_{01} := 0$ set a_{01}'s components to zero

    $\alpha_{11} := 1$

    $a_{21} := 0$ set a_{21}'s components to zero

    Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

endwhile

Figure 3.2: Algorithm for setting matrix A to the identity matrix.

Why is it guaranteed that α_{11} refers to the diagonal element of the current column?

Answer: A_{TL} starts as a 0×0 matrix, and is expanded by a row and a column in every iteration. Hence, it is always square. This guarantees that α_{11} is on the diagonal.

MATLAB provides the routine "eye" that returns an identity matrix of indicated size. But we will write our own.

Homework 3.2.2.2   With the FLAME API for MATLAB (FLAME@lab) implement the algorithm in Figure 3.2. You will use the functions `laff_zerov( x )` and `laff_onev( x )`, which return a zero vector and vector of all ones of the same size and shape (column or row) as input vector x, respectively. Try it yourself! (Hint: in Spark, you will want to pick Direction `TL->BR`.) Feel free to look at the below video if you get stuck.

Some links that will come in handy:

- ☛ Spark.   or ☛ Local Spark

- ☛ PictureFLAME or ☛ PictureFLAME
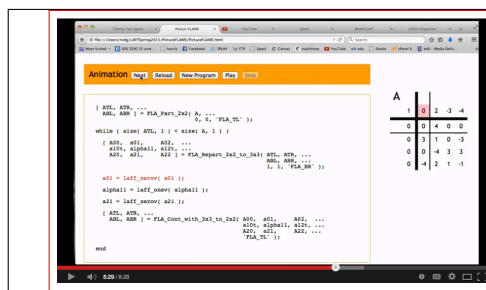
☛ **View at edX**

Homework 3.2.2.3 In the MATLAB Command Window, type

`A = eye( 4,4 )`

What is the result?

Homework 3.2.2.4 Apply the identity matrix to ☛ Timmy Two Space. What happens?

1. Timmy shifts off the grid.

2. Timmy disappears into the origin.

3. Timmy becomes a line on the x-axis.

4. Timmy becomes a line on the y-axis.

5. Timmy doesn't change at all.

Homework 3.2.2.5 The trace of a matrix equals the sum of the diagonal elements. What is the trace of the identity $I \in \mathbb{R}^{n \times n}$?

3.2.3 Diagonal Matrices

☛ **View at edX**

Let $L_D : \mathbb{R}^n \to \mathbb{R}^n$ be the function defined for every $x \in \mathbb{R}^n$ as

$$
L\left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{array}\right) = \left(\begin{array}{c} \delta_0 \chi_0 \\ \delta_1 \chi_1 \\ \vdots \\ \delta_{n-1} \chi_{n-1} \end{array}\right),
$$

where $\delta_0, \ldots, \delta_{n-1}$ are constants.

Here, we will denote the matrix that represents L_D by the letter D. Once again, by the definition of a matrix, the jth column of D is given by

$$
L_D(e_j) = L_D\left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{array}\right) = \left(\begin{array}{c} \delta_0 \times 0 \\ \vdots \\ \delta_{j-1} \times 0 \\ \delta_j \times 1 \\ \delta_{j+1} \times 0 \\ \vdots \\ \delta_{n-1} \times 0 \end{array}\right) = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ \delta_j \times 1 \\ 0 \\ \vdots \\ 0 \end{array}\right) = \delta_j \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{array}\right) = \delta_j e_j.
$$

This means that

$$
D = \left(\begin{array}{c|c|c|c} \delta_0 e_0 & \delta_1 e_1 & \cdots & \delta_{n-1} e_{n-1} \end{array}\right) = \left(\begin{array}{cccc} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{n-1} \end{array}\right).
$$

Definition 3.3 *A matrix* $A \in \mathbb{R}^{n \times n}$ *is said to be diagonal if* $\alpha_{i,j} = 0$ *for all* $i \neq j$ *so that*

$$
A = \left(\begin{array}{cccc} \alpha_{0,0} & 0 & \cdots & 0 \\ 0 & \alpha_{1,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{n-1,n-1} \end{array}\right).
$$

Homework 3.2.3.1 Let $A = \begin{pmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$ and $x = \begin{pmatrix} 2 \\ 1 \\ -2 \end{pmatrix}$. Evaluate Ax.

Homework 3.2.3.2 Let $D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -1 \end{pmatrix}$. What linear transformation, L, does this matrix represent? In particular, answer the following questions:

- $L : \mathbb{R}^n \to \mathbb{R}^m$. What are m and n?

- A linear transformation can be described by how it transforms the unit basis vectors:

$$L(e_0) = \begin{pmatrix} \\ \\ \end{pmatrix} ; L(e_1) = \begin{pmatrix} \\ \\ \end{pmatrix} ; L(e_2) = \begin{pmatrix} \\ \\ \end{pmatrix}$$

- $L(\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}) = \begin{pmatrix} \\ \\ \end{pmatrix}$

An algorithm that sets a given square matrix A to a diagonal matrix that has as its ith diagonal entry the ith entry of vector x ig given in Figure 3.3.

Homework 3.2.3.3 Implement a function

$$[\ A\_out\ ] = DiagonalMatrix\_unb(\ A,\ x\ )$$

based on Figure 3.3.
Some links that will come in handy:

- ☛ Spark.   or ☛ Local Spark

- ☛ PictureFLAME or ☛ PictureFLAME

Homework 3.2.3.4 In the MATLAB Command Window, type

```
x = [ -1; 2; -3 ]
A = diag( x )
```

What is the result?

Algorithm: $[A] := \text{SET\_TO\_DIAGONAL\_MATRIX}(A, x)$

Partition $A \to \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$, $x \to \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right)$

where A_{TL} is 0×0, x_T has 0 elements

while $m(A_{TL}) < m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \to \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right) \to \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{array} \right)$$

where α_{11} is 1×1, χ_1 is a scalar

$a_{01} := 0$

$\alpha_{11} := \chi_1$

$a_{21} := 0$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), \left(\begin{array}{c} x_T \\ \hline x_B \end{array} \right) \leftarrow \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{array} \right)$$

endwhile

Figure 3.3: Algorithm that sets A to a diagonal matrix with the entries of x on its diagonal.

In linear algebra an element-wise vector-vector product is not a meaningful operation: when $x, y \in \mathbb{R}^n$ the product xy has no meaning. However, MATLAB has an "element-wise multiplication" operator ".*'". Try

```
x = [-1; 2; -3]
y = [1; -1; 2]
x .* y
diag( x ) * y
```

Conclude that element-wise multiplication by a vector is the same as multiplication by a diagonal matrix with diagonal elements equal to the elements of that vector.

Homework 3.2.3.5 Apply the diagonal matrix $\begin{pmatrix} -1 & 0 \\ 0 & 2 \end{pmatrix}$ to ☛ Timmy Two Space. What happens?

1. Timmy shifts off the grid.

2. Timmy is rotated.

3. Timmy doesn't change at all.

4. Timmy is flipped with respect to the vertical axis.

5. Timmy is stretched by a factor two in the vertical direction.

Homework 3.2.3.6 Compute the trace of $\begin{pmatrix} -1 & 0 \\ 0 & 2 \end{pmatrix}$.

3.2.4 Triangular Matrices
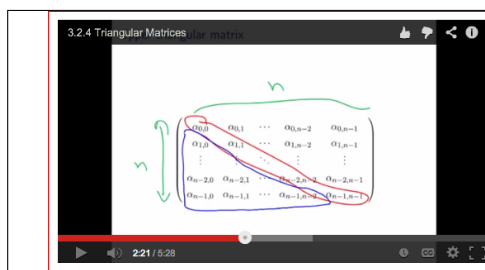
☛ **View at edX**

Homework 3.2.4.1 Let $L_U : \mathbb{R}^3 \to \mathbb{R}^3$ be defined as $L_U(\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}) = \begin{pmatrix} 2\chi_0 - \chi_1 + \chi_2 \\ 3\chi_1 - \chi_2 \\ -2\chi_2 \end{pmatrix}$. We have proven for similar functions that they are linear transformations, so we will skip that part. What matrix, U, represents this linear transformation?

A matrix like U in the above practice is called a triangular matrix. In particular, it is an *upper* triangular matrix.

In Figure 3.4 a number of different special cases of triangular matrices are defined.

Homework 3.2.4.2 A matrix that is both lower and upper triangular is, in fact, a diagonal matrix.

Always/Sometimes/Never

Homework 3.2.4.3 A matrix that is both strictly lower and strictly upper triangular is, in fact, a zero matrix.

Always/Sometimes/Never

Definition 3.4 (Triangular matrix)
A matrix $A \in \mathbb{R}^{n \times n}$ is said to be

lower triangular	$\alpha_{i,j} = 0$ *if* $i < j$	$\begin{pmatrix} \alpha_{0,0} & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}$
strictly lower triangular	$\alpha_{i,j} = 0$ *if* $i \leq j$	$\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & 0 & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & 0 \end{pmatrix}$
unit lower triangular	$\alpha_{i,j} = \begin{cases} 0 & \text{if } i < j \\ 1 & \text{if } i = j \end{cases}$	$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & 1 & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & 1 \end{pmatrix}$
upper triangular	$\alpha_{i,j} = 0$ *if* $i > j$	$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & \alpha_{1,1} & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \alpha_{n-2,n-2} & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & \alpha_{n-1,n-1} \end{pmatrix}$
strictly upper triangular	$\alpha_{i,j} = 0$ *if* $i \geq j$	$\begin{pmatrix} 0 & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & 0 & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$
unit upper triangular	$\alpha_{i,j} = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \end{cases}$	$\begin{pmatrix} 1 & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & 1 & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$

If a matrix is either lower or upper triangular, it is said to be triangular.

Figure 3.4: Definition of various triangular matrices

The algorithm in Figure 3.5 sets a given matrix $A \in \mathbb{R}^{n \times n}$ to its lower triangular part (zeroing the elements above the diagonal).

Homework 3.2.4.4 In the above algorithm you could have replaced $a_{01} := 0$ with $a_{12}^T := 0$.

Always/Sometimes/Never

Homework 3.2.4.5 Consider the algorithm in Figure 3.6. Change the ????? in the above algorithm so that it sets A to its

- Upper triangular part. (Set_to_upper_triangular_matrix_unb)

- Strictly upper triangular part. (Set_to_strictly_upper_triangular_matrix_unb)

- Unit upper triangular part. (Set_to_unit_upper_triangular_matrix_unb)

- Strictly lower triangular part. (Set_to_strictly_lower_triangular_matrix_unb)

- Unit lower triangular part. (Set_to_unit_lower_triangular_matrix_unb)

The MATLAB functions tril and triu, when given an $n \times n$ matrix A, return the lower and upper triangular parts of A, respectively. The strictly lower and strictly upper triangular parts of A can be extracted by the calls tril(A, -1) and triu(A, 1), respectively. We now write our own routines that sets the appropriate entries in a matrix to zero.

Homework 3.2.4.6 Implement functions for each of the algorithms from the last homework. In other words, implement functions that, given a matrix A, return a matrix equal to

- the upper triangular part. (Set_to_upper_triangular_matrix)

- the strictly upper triangular part. (Set_to_strictly_upper_triangular_matrix)

- the unit upper triangular part. (Set_to_unit_upper_triangular_matrix)

- strictly lower triangular part. (Set_to_strictly_lower_triangular_matrix)

- unit lower triangular part. (Set_to_unit_lower_triangular_matrix)

(Implement as many as you enjoy implementing. Then move on.)
Some links that will come in handy:

- ☛ Spark.　or ☛ Local Spark

- ☛ PictureFLAME or ☛ PictureFLAME

Algorithm: $[A] := \text{SET\_TO\_LOWER\_TRIANGULAR\_MATRIX}(A)$

Partition $A \to \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$

  where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

  Repartition

  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \to \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$

    where α_{11} is 1×1

  set the elements of the current column above the diagonal to zero

  $a_{01} := 0 \qquad\qquad$ set a_{01}'s components to zero

  Continue with

  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$

endwhile

Figure 3.5: Algorithm for making a matrix A a lower triangular matrix by setting the entries above the diagonal to zero.

Algorithm: $[A] := \text{SET\_TO\_???\_TRIANGULAR\_MATRIX}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$

  where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

  Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

    where α_{11} is 1×1

?????

  Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

endwhile

Figure 3.6: Algorithm skeleton for setting a square matrix to a triangular matrix.

Homework 3.2.4.7 In MATLAB try this:

```
A = [ 1,2,3;4,5,6;7,8,9 ]
tril( A )
tril( A, -1 )
tril( A, -1 ) + eye( size( A ) )
triu( A )
triu( A, 1 )
triu( A, 1 ) + eye( size( A ) )
```

Homework 3.2.4.8 Apply $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ to ☛ Timmy Two Space. What happens to Timmy?

1. Timmy shifts off the grid.

2. Timmy becomes a line on the x-axis.

3. Timmy becomes a line on the y-axis.

4. Timmy is skewed to the right.

5. Timmy doesn't change at all.

3.2.5 Transpose Matrix

☛ **View at edX**

Definition 3.5 *Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times m}$. Then B is said to be the transpose of A if, for $0 \le i < m$ and $0 \le j < n$, $\beta_{j,i} = \alpha_{i,j}$. The transpose of a matrix A is denoted by A^T so that $B = A^T$.*

We have already used ^{T} to indicate a row vector, which is consistent with the above definition: it is a column vector that has been transposed.

Homework 3.2.5.1 Let $A = \begin{pmatrix} -1 & 0 & 2 & 1 \\ 2 & -1 & 1 & 2 \\ 3 & 1 & -1 & 3 \end{pmatrix}$ and $x = \begin{pmatrix} -1 \\ 2 \\ 4 \end{pmatrix}$. What are A^T and x^T?

Clearly, $(A^T)^T = A$.

Notice that the columns of matrix A become the rows of matrix A^T. Similarly, the rows of matrix A become the columns of matrix A^T.

The following algorithm sets a given matrix $B \in \mathbb{R}^{n \times m}$ to the transpose of a given matrix $A \in \mathbb{R}^{m \times n}$:

Algorithm: $[B] := \text{TRANSPOSE}(A, B)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right), B \rightarrow \left(\dfrac{B_T}{B_B} \right)$

    where A_L has 0 columns, B_T has 0 rows

while $n(A_L) < n(A)$ **do**

    Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\dfrac{B_T}{B_B} \right) \rightarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

      where a_1 has 1 column, b_1 has 1 row

  $b_1^T := a_1^T$ (Set the current row of B to the current column of A)

    Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\dfrac{B_T}{B_B} \right) \leftarrow \left(\begin{array}{c} B_0 \\ \hline b_1^T \\ \hline B_2 \end{array} \right)$$

endwhile

The ^{T} in b_1^T is part of indicating that b_1^T is a row. The ^{T} in a_1^T in the assignment changes the column vector a_1 into a row vector so that it can be assigned to b_1^T.

Homework 3.2.5.2 Consider the following algorithm.

Algorithm: $[B] := \text{TRANSPOSE\_ALTERNATIVE}(A, B)$

Partition $A \rightarrow \left(\dfrac{A_T}{A_B} \right)$, $B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array} \right)$

    where A_T has 0 rows, B_L has 0 columns

while $m(A_T) < m(A)$ **do**

    Repartition

$$\left(\dfrac{A_T}{A_B} \right) \rightarrow \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array} \right) , \left(\begin{array}{c|c} B_L & B_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right)$$

    where a_1 has 1 row, b_1 has 1 column

    Continue with

$$\left(\dfrac{A_T}{A_B} \right) \leftarrow \left(\begin{array}{c} A_0 \\ \hline a_1^T \\ \hline A_2 \end{array} \right) , \left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right)$$

endwhile

Modify the above algorithm so that it copies rows of A into columns of B.

Homework 3.2.5.3 Implement functions

- `Transpose_unb( A, B )`

- `Transpose_alternative_unb( A, B )`

Some links that will come in handy:

- ☞ Spark. or ☞ `Local Spark`

- ☞ PictureFLAME or ☞ `PictureFLAME`

Homework 3.2.5.4 The transpose of a lower triangular matrix is an upper triangular matrix.

Always/Sometimes/Never

Homework 3.2.5.5 The transpose of a strictly upper triangular matrix is a strictly lower triangular matrix.

Always/Sometimes/Never

Homework 3.2.5.6 The transpose of the identity is the identity.

Always/Sometimes/Never

Homework 3.2.5.7 Evaluate

- $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^T =$

- $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^T =$

Homework 3.2.5.8 If $A = A^T$ then $A = I$ (the identity).

True/False

3.2.6 Symmetric Matrices

☞ **View at edX**

A matrix $A \in \mathbb{R}^{n \times n}$ is said to be symmetric if $A = A^T$.

In other words, if $A \in \mathbb{R}^{n \times n}$ is symmetric, then $\alpha_{i,j} = \alpha_{j,i}$ for all $0 \leq i, j < n$. Another way of expressing this is that

$$
A = \begin{pmatrix}
\alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\
\alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{0,n-2} & \alpha_{1,n-2} & \cdots & \alpha_{n-2,n-2} & \alpha_{n-2,n-1} \\
\alpha_{0,n-1} & \alpha_{1,n-1} & \cdots & \alpha_{n-2,n-1} & \alpha_{n-1,n-1}
\end{pmatrix}
$$

and

$$
A = \begin{pmatrix}
\alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{n-2,0} & \alpha_{n-1,0} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{n-2,1} & \alpha_{n-1,1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1}
\end{pmatrix}.
$$

Homework 3.2.6.1 Assume the below matrices are symmetric. Fill in the remaining elements.

$$
\begin{pmatrix}
2 & \square & -1 \\
-2 & 1 & -3 \\
\square & \square & -1
\end{pmatrix} ; \quad
\begin{pmatrix}
2 & \square & \square \\
-2 & 1 & \square \\
-1 & 3 & -1
\end{pmatrix} ; \quad
\begin{pmatrix}
2 & 1 & -1 \\
\square & 1 & -3 \\
\square & \square & -1
\end{pmatrix}.
$$

Homework 3.2.6.2 A triangular matrix that is also symmetric is, in fact, a diagonal matrix.
Always/Sometimes/Never

The nice thing about symmetric matrices is that only approximately half of the entries need to be stored. Often, only the lower triangular or only the upper triangular part of a symmetric matrix is stored. Indeed: Let A be symmetric, let L be the lower triangular matrix stored in the lower triangular part of A, and let $\widetilde{L}$ is the strictly lower triangular matrix stored in the strictly lower triangular part of A. Then $A = L + \widetilde{L}^T$:

$$
\begin{aligned}
A &= \begin{pmatrix}
\alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{n-2,0} & \alpha_{n-1,0} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{n-2,1} & \alpha_{n-1,1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1}
\end{pmatrix} \\
&= \begin{pmatrix}
\alpha_{0,0} & 0 & \cdots & 0 & 0 \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & 0 \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1}
\end{pmatrix} + \begin{pmatrix}
0 & \alpha_{1,0} & \cdots & \alpha_{n-2,0} & \alpha_{n-1,0} \\
0 & 0 & \cdots & \alpha_{n-2,1} & \alpha_{n-1,1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & \alpha_{n-1,n-2} \\
0 & 0 & \cdots & 0 & 0
\end{pmatrix} \\
&= \begin{pmatrix}
\alpha_{0,0} & 0 & \cdots & 0 & 0 \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & 0 \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1}
\end{pmatrix} + \begin{pmatrix}
0 & 0 & \cdots & 0 & 0 \\
\alpha_{1,0} & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & 0 & 0 \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & 0
\end{pmatrix}^T.
\end{aligned}
$$

Let A be symmetric and assume that $A = L + \widetilde{L}^T$ as discussed above. Assume that only L is stored in A and that we would like to also set the upper triangular parts of A to their correct values (in other words,

---

Algorithm: $[A] := \text{SYMMETRIZE\_FROM\_LOWER\_TRIANGLE}(A)$

Partition $A \to \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$

    where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

    Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \to \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

        where α_{11} is 1×1

(set a_{01}'s components to their symmetric parts below the diagonal)

$a_{01} := (a_{10}^T)^T$

    Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \gets \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

endwhile

Figure 3.7: Algorithm that "symmetrizes" a matrix, assuming initially only the lower triangular part is stored.

set the strictly upper triangular part of A to $\widetilde{L}$). The algorithm in Figure 3.7 performs this operation, which we will call "symmetrizing" A.

Homework 3.2.6.3 In the algorithm in Figure 3.7 one can replace $a_{01} := a_{10}^T$ by $a_{12}^T = a_{21}$.

Always/Sometimes/Never

Homework 3.2.6.4 Consider the following algorithm.

Algorithm: $[A] := \text{SYMMETRIZE\_FROM\_UPPER\_TRIANGLE}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right)$

    where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ **do**

    Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

      where α_{11} is 1×1

?????

    Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)$$

endwhile

What commands need to be introduced between the lines in order to "symmetrize" A assuming that only its upper triangular part is stored initially.
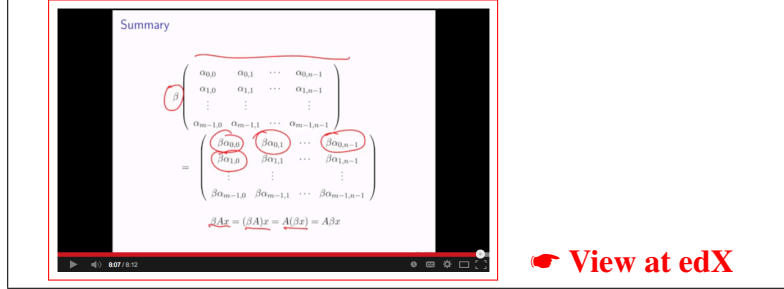
Homework 3.2.6.5 Implement functions

- `Symmetrize_from_lower_triangle_unb( A, B )`

- `Symmetrize_from_upper_triangle_unb( A, B )`

Some links that will come in handy:

- ☛ Spark.   or ☛ Local Spark

- ☛ PictureFLAME or ☛ PictureFLAME

3.3 Operations with Matrices

3.3.1 Scaling a Matrix

☞ **View at edX**

Theorem 3.6 *Let $L_A : \mathbb{R}^n \to \mathbb{R}^m$ be a linear transformation and, for all $x \in \mathbb{R}^n$, define the function $L_B : \mathbb{R}^n \to \mathbb{R}^m$ by $L_B(x) = \beta L_A(x)$, where β is a scalar. Then $L_B(x)$ is a linear transformation.*

Homework 3.3.1.1 Prove the above theorem.

Let A be the matrix that represents L_A. Then, for all $x \in \mathbb{R}^n$, $\beta(Ax) = \beta L_A(x) = L_B(x)$. Since L_B is a linear transformation, there should be a matrix B such that, for all $x \in \mathbb{R}^n$, $Bx = L_B(x) = \beta(Ax)$. Recall that $b_j = Be_j$, the jth column of B. Thus, $b_j = Be_j = \beta(Ae_j) = \beta a_j$, where a_j equals the jth column of A. We conclude that B is computed from A by scaling each column by β. But that simply means that each element of B is scaled by β.

The above motivates the following definition.

If $A \in \mathbb{R}^{m \times n}$ and $\beta \in \mathbb{R}$, then

$$\beta \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} = \begin{pmatrix} \beta\alpha_{0,0} & \beta\alpha_{0,1} & \cdots & \beta\alpha_{0,n-1} \\ \beta\alpha_{1,0} & \beta\alpha_{1,1} & \cdots & \beta\alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \beta\alpha_{m-1,0} & \beta\alpha_{m-1,1} & \cdots & \beta\alpha_{m-1,n-1} \end{pmatrix}.$$

An alternative motivation for this definition is to consider

$$\beta(Ax) = \beta \begin{pmatrix} \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + & \cdots + & \alpha_{0,n-1}\chi_{n-1} \\ \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + & \cdots + & \alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + & \cdots + & \alpha_{m-1,n-1}\chi_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \beta(\alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + & \cdots + & \alpha_{0,n-1}\chi_{n-1}) \\ \beta(\alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + & \cdots + & \alpha_{1,n-1}\chi_{n-1}) \\ \vdots & \vdots & \vdots & \vdots \\ \beta(\alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + & \cdots + & \alpha_{m-1,n-1}\chi_{n-1}) \end{pmatrix}$$

Algorithm: $[A] := \text{SCALE\_MATRIX}(\beta, A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_L & A_R \end{array} \right)$

    where A_L has 0 columns

while $n(A_L) < n(A)$ **do**

    Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \rightarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$$

        where a_1 has 1 column

   $a_1 := \beta a_1$ (Scale the current column of A)

    Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right)$$

endwhile

Figure 3.8: Algorithm that updates $A := \beta A$.

$$= \left(\begin{array}{cccc} \beta\alpha_{0,0}\chi_0 + & \beta\alpha_{0,1}\chi_1 + & \cdots + & \beta\alpha_{0,n-1}\chi_{n-1} \\ \beta\alpha_{1,0}\chi_0 + & \beta\alpha_{1,1}\chi_1 + & \cdots + & \beta\alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \beta\alpha_{m-1,0}\chi_0 + & \beta\alpha_{m-1,1}\chi_1 + & \cdots + & \beta\alpha_{m-1,n-1}\chi_{n-1} \end{array} \right)$$

$$= \left(\begin{array}{cccc} \beta\alpha_{0,0} & \beta\alpha_{0,1} & \cdots & \beta\alpha_{0,n-1} \\ \beta\alpha_{1,0} & \beta\alpha_{1,1} & \cdots & \beta\alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta\alpha_{m-1,0} & \beta\alpha_{m-1,1} & \cdots & \beta\alpha_{m-1,n-1} \end{array} \right) \left(\begin{array}{c} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{array} \right) = (\beta A)x.$$

Since, by design, $\beta(Ax) = (\beta A)x$ we can drop the parentheses and write βAx (which also equals $A(\beta x)$ since $L(x) = Ax$ is a linear transformation).

Given matrices $\beta \in \mathbb{R}$ and $A \in \mathbb{R}^{m \times n}$, the algorithm in Figure 3.8 scales A by β.

Homework 3.3.1.2 Consider the following algorithm.

Algorithm: $[A] := \text{SCALE\_MATRIX\_ALTERNATIVE}(\beta, A)$

Partition $A \rightarrow \left(\frac{A_T}{A_B} \right)$

    where A_T has 0 rows

while $m(A_T) < m(A)$ **do**

    Repartition

$$\left(\frac{A_T}{A_B} \right) \rightarrow \left(\frac{A_0}{\frac{a_1^T}{A_2}} \right)$$

        where a_1 has 1 row

?????

    Continue with

$$\left(\frac{A_T}{A_B} \right) \leftarrow \left(\frac{A_0}{\frac{a_1^T}{A_2}} \right)$$

endwhile

What update will scale A one row at a time?

With MATLAB, when `beta` is a scalar and `A` is a matrix, the simple command `beta * A` will scale A by alpha.

Homework 3.3.1.3 Implement function `Scale_matrix_unb( beta, A )`.
Some links that will come in handy:

- ☛ Spark.   or ☛ `Local Spark`

- ☛ PictureFLAME or ☛ `PictureFLAME`

Homework 3.3.1.4
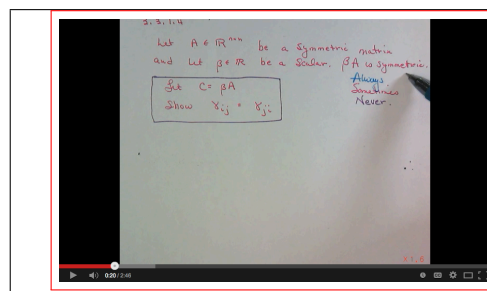
☛ **View at edX**

Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix and $\beta \in \mathbb{R}$ a scalar, βA is symmetric.

Always/Sometimes/Never

Homework 3.3.1.5

☛ **View at edX**

Let $A \in \mathbb{R}^{n \times n}$ be a lower triangular matrix and $\beta \in \mathbb{R}$ a scalar, βA is a lower triangular matrix.

Always/Sometimes/Never

Homework 3.3.1.6 Let $A \in \mathbb{R}^{n \times n}$ be a diagonal matrix and $\beta \in \mathbb{R}$ a scalar, βA is a diagonal matrix.

Always/Sometimes/Never

Homework 3.3.1.7 Let $A \in \mathbb{R}^{m \times n}$ be a matrix and $\beta \in \mathbb{R}$ a scalar, $(\beta A)^T = \beta A^T$.

Always/Sometimes/Never

3.3.2 Adding Matrices
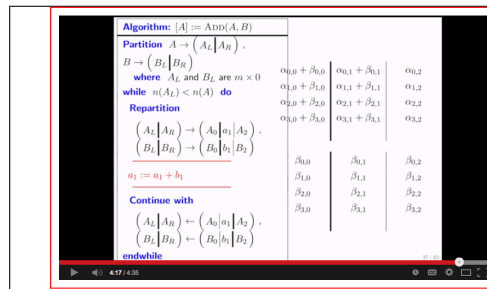
☛ **View at edX**

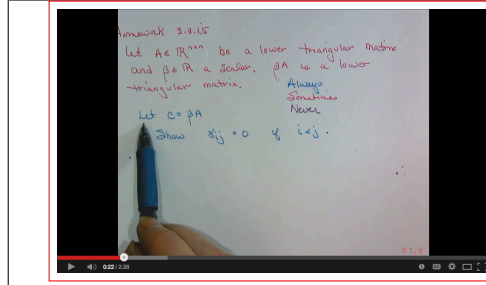**Homework 3.3.2.1** The sum of two linear transformations is a linear transformation. More formally: Let $L_A : \mathbb{R}^n \to \mathbb{R}^m$ and $L_B : \mathbb{R}^n \to \mathbb{R}^m$ be two linear transformations. Let $L_C : \mathbb{R}^n \to \mathbb{R}^m$ be defined by $L_C(x) = L_A(x) + L_B(x)$. L is a linear transformation.

Always/Sometimes/Never

Now, let A, B, and C be the matrices that represent L_A, L_B, and L_C in the above theorem, respectively. Then, for all $x \in \mathbb{R}^n$, $Cx = L_C(x) = L_A(x) + L_B(x)$. What does c_j, the jth column of C, equal?

$$c_j = Ce_j = L_C(e_j) = L_A(e_j) + L_B(e_j) = Ae_j + Be_j = a_j + b_j,$$

where a_j, b_j, and c_j equal the jth columns of A, B, and C, respectively. Thus, the jth column of C equals the sum of the corresponding columns of A and B. That simply means that each element of C equals the sum of the corresponding elements of A and B.

Algorithm: $[A] := \text{ADD\_MATRICES}(A, B)$

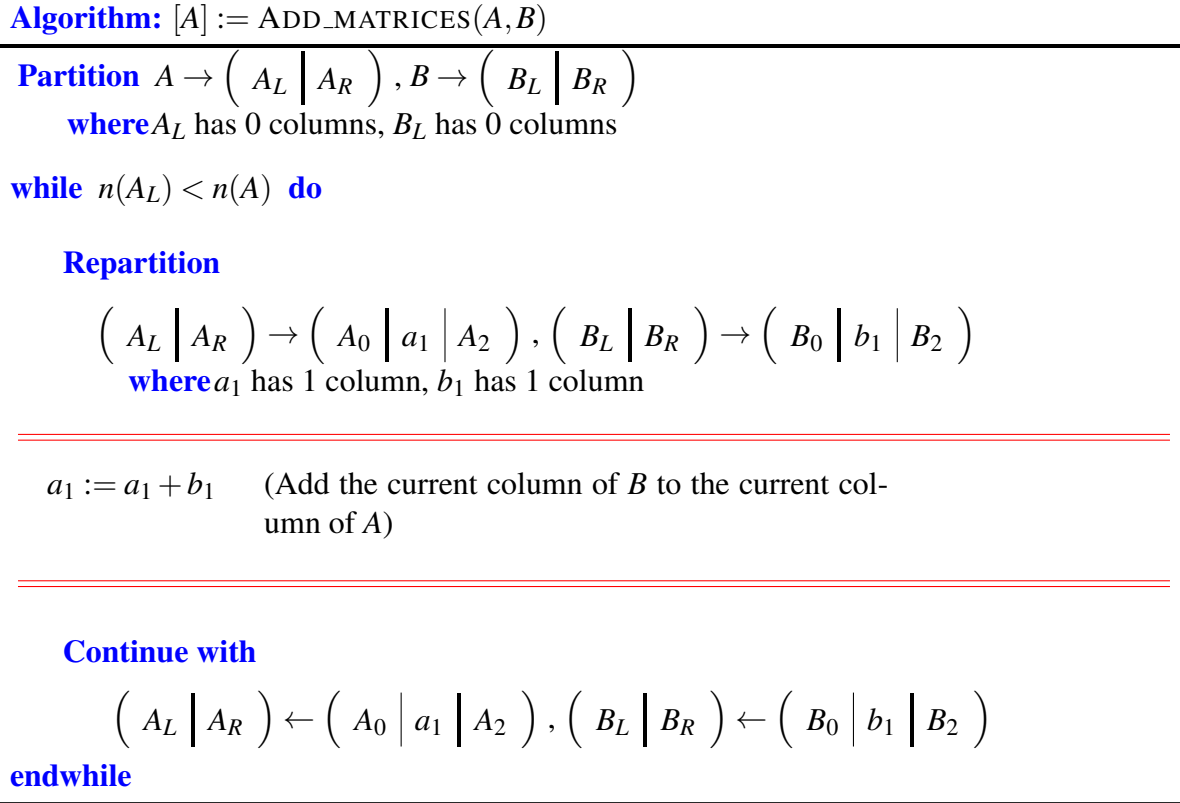**Partition** $A \to \left(\begin{array}{c|c} A_L & A_R \end{array} \right), B \to \left(\begin{array}{c|c} B_L & B_R \end{array} \right)$
where A_L has 0 columns, B_L has 0 columns

while $n(A_L) < n(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \to \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c|c} B_L & B_R \end{array} \right) \to \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right)$$
where a_1 has 1 column, b_1 has 1 column

$a_1 := a_1 + b_1$ (Add the current column of B to the current column of A)

Continue with

$$\left(\begin{array}{c|c} A_L & A_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} A_0 & a_1 & A_2 \end{array} \right), \left(\begin{array}{c|c} B_L & B_R \end{array} \right) \leftarrow \left(\begin{array}{c|c|c} B_0 & b_1 & B_2 \end{array} \right)$$

endwhile

Figure 3.9: Algorithm for adding two matrices by columns.

If $A, B \in \mathbb{R}^{m \times n}$, then

$$A + B = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} + \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \beta_{m-1,0} & \beta_{m-1,1} & \cdots & \beta_{m-1,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \alpha_{0,0} + \beta_{0,0} & \alpha_{0,1} + \beta_{0,1} & \cdots & \alpha_{0,n-1} + \beta_{0,n-1} \\ \alpha_{1,0} + \beta_{1,0} & \alpha_{1,1} + \beta_{1,1} & \cdots & \alpha_{1,n-1} + \beta_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} + \beta_{m-1,0} & \alpha_{m-1,1} + \beta_{m-1,1} & \cdots & \alpha_{m-1,n-1} + \beta_{m-1,n-1} \end{pmatrix}.$$

Given matrices $A, B \in \mathbb{R}^{m \times n}$, the algorithm in Figure 3.9 adds B to A.

Homework 3.3.2.2 Consider the algorithm in Figure 3.10.
What update will add B to A one row at a time, overwriting A with the result?

When A and B are created as matrices of the same size, MATLAB adds two matrices with the simple command A + B . We'll just use that when we need it!

Algorithm: $[A] := \text{ADD\_MATRICES\_ALTERNATIVE}(A,B)$

Partition $A \to \left(\dfrac{A_T}{A_B} \right)$, $B \to \left(\dfrac{B_T}{B_B} \right)$

   where A_T has 0 rows, B_T has 0 rows

while $m(A_T) < m(A)$ **do**

   Repartition

   $$\left(\dfrac{A_T}{A_B} \right) \to \left(\dfrac{A_0}{\dfrac{a_1^T}{A_2}} \right), \left(\dfrac{B_T}{B_B} \right) \to \left(\dfrac{B_0}{\dfrac{b_1^T}{B_2}} \right)$$

      where a_1 has 1 row, b_1 has 1 row

   Continue with

   $$\left(\dfrac{A_T}{A_B} \right) \leftarrow \left(\dfrac{A_0}{\dfrac{a_1^T}{A_2}} \right), \left(\dfrac{B_T}{B_B} \right) \leftarrow \left(\dfrac{B_0}{\dfrac{b_1^T}{B_2}} \right)$$

endwhile

Figure 3.10: Algorithm skeleton for adding two matrices by rows.

Try this! In MATLAB execute

```
A = [ 1,2;3,4;5,6 ]
B = [ -1,2;3,-4;5,6 ]
C = A + B
```

Homework 3.3.2.3 Let $A, B \in \mathbb{R}^{m \times n}$. $A + B = B + A$.

Always/Sometimes/Never

Homework 3.3.2.4 Let $A, B, C \in \mathbb{R}^{m \times n}$. $(A + B) + C = A + (B + C)$.

Always/Sometimes/Never

Homework 3.3.2.5 Let $A, B \in \mathbb{R}^{m \times n}$ and $\gamma \in \mathbb{R}$. $\gamma(A + B) = \gamma A + \gamma B$.

Always/Sometimes/Never

Homework 3.3.2.6 Let $A \in \mathbb{R}^{m \times n}$ and $\beta, \gamma \in \mathbb{R}$. $(\beta + \gamma)A = \beta A + \gamma A$.

Always/Sometimes/Never

Homework 3.3.2.7 Let $A, B \in \mathbb{R}^{n \times n}$. $(A + B)^T = A^T + B^T$.

Always/Sometimes/Never

Homework 3.3.2.8 Let $A, B \in \mathbb{R}^{n \times n}$ be symmetric matrices. $A + B$ is symmetric.

Always/Sometimes/Never

Homework 3.3.2.9 Let $A, B \in \mathbb{R}^{n \times n}$ be symmetric matrices. $A - B$ is symmetric.

Always/Sometimes/Never

Homework 3.3.2.10 Let $A, B \in \mathbb{R}^{n \times n}$ be symmetric matrices and $\alpha, \beta \in \mathbb{R}$. $\alpha A + \beta B$ is symmetric.

Always/Sometimes/Never

Homework 3.3.2.11 Let $A, B \in \mathbb{R}^{n \times n}$.

If A and B are lower triangular matrices then $A + B$ is lower triangular.

True/False

If A and B are strictly lower triangular matrices then $A + B$ is strictly lower triangular.

True/False

If A and B are unit lower triangular matrices then $A + B$ is unit lower triangular.

True/False

If A and B are upper triangular matrices then $A + B$ is upper triangular.

True/False

If A and B are strictly upper triangular matrices then $A + B$ is strictly upper triangular.

True/False

If A and B are unit upper triangular matrices then $A + B$ is unit upper triangular.

True/False

Homework 3.3.2.12 Let $A, B \in \mathbb{R}^{n \times n}$.

If A and B are lower triangular matrices then $A - B$ is lower triangular.

True/False

If A and B are strictly lower triangular matrices then $A - B$ is strictly lower triangular. True/False

If A and B are unit lower triangular matrices then $A - B$ is *strictly* lower triangular. True/False
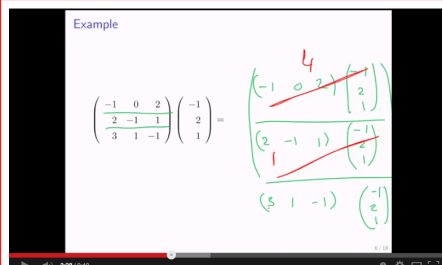
If A and B are upper triangular matrices then $A - B$ is upper triangular. True/False

If A and B are strictly upper triangular matrices then $A - B$ is strictly upper triangular. True/False

If A and B are unit upper triangular matrices then $A - B$ is unit upper triangular. True/False

3.4 Matrix-Vector Multiplication Algorithms

3.4.1 Via Dot Products

☛ **View at edX**

Motivation

Recall that if $y = Ax$, where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, then

$$y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + & \cdots + & \alpha_{0,n-1}\chi_{n-1} \\ \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + & \cdots + & \alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + & \cdots + & \alpha_{m-1,n-1}\chi_{n-1} \end{pmatrix}.$$

If one looks at a typical row,

$$\alpha_{i,0}\chi_0 + \quad \alpha_{i,1}\chi_1 + \quad \cdots + \quad \alpha_{i,n-1}\chi_{n-1}$$

one notices that this is just the dot product of vectors

$$\widetilde{a}_i = \begin{pmatrix} \alpha_{i,0} \\ \alpha_{i,1} \\ \vdots \\ \alpha_{i,n-1} \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_n - 1 \end{pmatrix}.$$

In other words, the dot product of the ith row of A, viewed as a column vector, with the vector x, which one can visualize as

$$\begin{pmatrix} \psi_0 \\ \vdots \\ \boxed{\psi_i} \\ \vdots \\ \psi_{m-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \vdots & \vdots & & \vdots \\ \boxed{\alpha_{i,0} \quad \alpha_{i,1} \quad \cdots \quad \alpha_{i,n-1}} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix}$$

The above argument starts to expain why we write the dot product of vectors x and y as $x^T y$.

Example 3.7 Let $A = \begin{pmatrix} -1 & 0 & 2 \\ 2 & -1 & 1 \\ 3 & 1 & -1 \end{pmatrix}$ and $x = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$. Then

$$
Ax = \begin{pmatrix} -1 & 0 & 2 \\ 2 & -1 & 1 \\ 3 & 1 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} -1 & 0 & 2 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 3 & 1 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \end{pmatrix}
$$

$$
= \begin{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 2 \end{pmatrix}^T \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}^T \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 3 \\ 1 \\ -1 \end{pmatrix}^T \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} (-1)(-1) + (0)(2) + (2)(1) \\ (2)(-1) + (-1)(2) + (1)(1) \\ (3)(-1) + (1)(2) + (-1)(1) \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ -2 \end{pmatrix}
$$

Algorithm (traditional notation)

An algorithm for computing $y := Ax + y$ (notice that we add the result of Ax to y) via dot products is given by

$$
\begin{aligned}
&\textbf{for } i = 0, \ldots, m-1 \\
&\quad \textbf{for } j = 0, \ldots, n-1 \\
&\quad\quad \psi_i := \psi_i + \alpha_{i,j}\chi_j \\
&\quad \textbf{endfor} \\
&\textbf{endfor}
\end{aligned}
$$

If initially $y = 0$, then it computes $y := Ax$.

Now, let us revisit the fact that the matrix-vector multiply can be computed as dot products of the rows of A with the vector x. Think of the matrix A as individual rows:

$$
A = \begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix},
$$

where $\widetilde{a}_i$ is the (column) vector which, when transposed, becomes the ith row of the matrix. Then

$$
Ax = \begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix} x = \begin{pmatrix} \widetilde{a}_0^T x \\ \widetilde{a}_1^T x \\ \vdots \\ \widetilde{a}_{m-1}^T x \end{pmatrix},
$$

which is exactly what we reasoned before. To emphasize this, the algorithm can then be annotated as follows:

$$
\left.
\begin{aligned}
&\textbf{for } i = 0, \ldots, m-1 \\
&\qquad \textbf{for } j = 0, \ldots, n-1 \\
&\qquad\qquad \psi_i := \psi_i + \alpha_{i,j}\chi_j \\
&\qquad \textbf{endfor} \\
&\textbf{endfor}
\end{aligned}
\right\} \psi_i := \psi_i + \widetilde{a}_i^T x
$$

Algorithm (FLAME notation)

In Figure 3.11 we now present the algorithm that casts matrix-vector multiplication in terms of dot products using the FLAME notation with which you became familiar earlier this week.

> **Homework 3.4.1.1** Implement the algorithm in Figure 3.11 yielding function
>
> $$\texttt{Mvmult\_n\_unb\_var1(A, x, y)}.$$
>
> Some links that will come in handy:
>
> - ☛ Spark.   or ☛ `Local Spark`
>
> - ☛ PictureFLAME or ☛ `PictureFLAME`

Algorithm: $y := \text{MVMULT\_N\_UNB\_VAR1}(A, x, y)$

Partition $A \rightarrow \left(\dfrac{A_T}{A_B} \right)$, $y \rightarrow \left(\dfrac{y_T}{y_B} \right)$

where A_T is $0 \times n$ and y_T is 0×1

while $m(A_T) < m(A)$ **do**

   Repartition

$$\left(\frac{A_T}{A_B} \right) \rightarrow \left(\frac{A_0}{\frac{a_1^T}{A_2}} \right) , \left(\frac{y_T}{y_B} \right) \rightarrow \left(\frac{y_0}{\frac{\psi_1}{y_2}} \right)$$

   where a_1 is a row

$\quad \psi_1 := a_1^T x + \psi_1$

   Continue with

$$\left(\frac{A_T}{A_B} \right) \leftarrow \left(\frac{A_0}{\frac{a_1^T}{A_2}} \right) , \left(\frac{y_T}{y_B} \right) \leftarrow \left(\frac{y_0}{\frac{\psi_1}{y_2}} \right)$$

endwhile

Figure 3.11: Algorithm that computes $y := Ax + y$ via dot products.

3.4.2 Via AXPY Operations

☛ **View at edX**

Motivation

Note that, by definition,

$$Ax = \begin{pmatrix} \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + & \cdots + & \alpha_{0,n-1}\chi_{n-1} \\ \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + & \cdots + & \alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + & \cdots + & \alpha_{m-1,n-1}\chi_{n-1} \end{pmatrix} =$$

$$\chi_0 \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \vdots \\ \alpha_{m-1,0} \end{pmatrix} + \chi_1 \begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \vdots \\ \alpha_{m-1,1} \end{pmatrix} + \cdots + \chi_{n-1} \begin{pmatrix} \alpha_{0,n-1} \\ \alpha_{1,n-1} \\ \vdots \\ \alpha_{m-1,n-1} \end{pmatrix}.$$

Example 3.8 Let $A = \begin{pmatrix} -1 & 0 & 2 \\ 2 & -1 & 1 \\ 3 & 1 & -1 \end{pmatrix}$ and $x = \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}$. Then

$$\begin{aligned}
Ax &= \begin{pmatrix} -1 & 0 & 2 \\ 2 & -1 & 1 \\ 3 & 1 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} = (-1) \begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix} + (2) \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} + (1) \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix} \\
&= \begin{pmatrix} (-1)(-1) \\ (-1)(2) \\ (-1)(3) \end{pmatrix} + \begin{pmatrix} (2)(0) \\ (2)(-1) \\ (2)(1) \end{pmatrix} + \begin{pmatrix} (1)(2) \\ (1)(1) \\ (1)(-1) \end{pmatrix} \\
&= \begin{pmatrix} (-1)(-1) + (0)(2) + (2)(1) \\ (2)(-1) + (-1)(2) + (1)(1) \\ (3)(-1) + (1)(2) + (-1)(1) \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ -2 \end{pmatrix}
\end{aligned}$$

Algorithm (traditional notation)

The above suggests the alternative algorithm for computing $y := Ax + y$ given by

$$\textbf{for } j = 0, \ldots, n-1$$
$$\textbf{for } i = 0, \ldots, m-1$$
$$\psi_i := \psi_i + \alpha_{i,j}\chi_j$$
$$\textbf{endfor}$$
$$\textbf{endfor}$$

If we let a_j denote the vector that equals the jth column of A, then

$$A = \left(\begin{array}{c|c|c|c} a_0 & a_1 & \cdots & a_{n-1} \end{array} \right)$$

and

$$Ax = \chi_0 \underbrace{\begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \vdots \\ \alpha_{m-1,0} \end{pmatrix}}_{a_0} + \chi_1 \underbrace{\begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \vdots \\ \alpha_{m-1,1} \end{pmatrix}}_{a_1} + \cdots + \chi_{n-1} \underbrace{\begin{pmatrix} \alpha_{0,n-1} \\ \alpha_{1,n-1} \\ \vdots \\ \alpha_{m-1,n-1} \end{pmatrix}}_{a_{n-1}}$$

$$= \chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1}.$$

This is emphasized by annotating the algorithm as follows:

$$\textbf{for } j = 0, \ldots, n-1$$
$$\left. \begin{array}{l} \textbf{for } i = 0, \ldots, m-1 \\ \quad \psi_i := \psi_i + \alpha_{i,j}\chi_j \\ \textbf{endfor} \end{array} \right\} y := \chi_j a_j + y$$
$$\textbf{endfor}$$

Algorithm (FLAME notation)

The algorithm that casts matrix-vector multiplication in terms of AXPYs using the FLAME notation is given in Figure 3.12.

Homework 3.4.2.1 Implement the algorithm in Figure 3.12 yielding function

$$\texttt{Mvmult\_n\_unb\_var2(A, x, y)}.$$

(Hint: use the function $\texttt{laff\_axpy(alpha, x, y)}$ that updates $y := \alpha x + y$.)
Some links that will come in handy:

- ☞ Spark.   or ☞ Local Spark

- ☞ PictureFLAME or ☞ PictureFLAME

Algorithm: $y := \text{MVMULT\_N\_UNB\_VAR2}(A, x, y)$

Partition $A \rightarrow \left(A_L \,\middle|\, A_R \right)$, $x \rightarrow \left(\dfrac{x_T}{x_B} \right)$

  where A_L is $m \times 0$ and x_T is 0×1

while $m(x_T) < m(x)$ **do**

  Repartition

$$\left(A_L \,\middle|\, A_R \right) \rightarrow \left(A_0 \,\middle|\, a_1 \,\middle|\, A_2 \right), \quad \left(\dfrac{x_T}{x_B} \right) \rightarrow \left(\dfrac{x_0}{\dfrac{\chi_1}{x_2}} \right)$$

  where a_1 is a column

$y := \chi_1 a_1 + y$

  Continue with

$$\left(A_L \,\middle|\, A_R \right) \leftarrow \left(A_0 \,\middle|\, a_1 \,\middle|\, A_2 \right), \quad \left(\dfrac{x_T}{x_B} \right) \leftarrow \left(\dfrac{x_0}{\dfrac{\chi_1}{x_2}} \right)$$

endwhile

Figure 3.12: Algorithm for computing $y := Ax + y$ via AXPYs.

3.4.3 Compare and Contrast

☛ **View at edX**

Motivation

It is always useful to compare and contrast different algorithms for the same operation.

Algorithms (traditional notation)

Let us put the two algorithms that compute $y := Ax + y$ via "double nested loops" next to each other:

$$\textbf{for } j = 0, \ldots, n-1$$
$$\quad \textbf{for } i = 0, \ldots, m-1$$
$$\quad\quad \psi_i := \psi_i + \alpha_{i,j}\chi_j$$
$$\quad \textbf{endfor}$$
$$\textbf{endfor}$$

$$\textbf{for } i = 0, \ldots, m-1$$
$$\quad \textbf{for } j = 0, \ldots, n-1$$
$$\quad\quad \psi_i := \psi_i + \alpha_{i,j}\chi_j$$
$$\quad \textbf{endfor}$$
$$\textbf{endfor}$$

On the left is the algorithm based on the dot product and on the right the one based on AXPY operations. Notice that these loops differ only in that the order of the two loops are interchanged. This is known as "interchanging loops" and is sometimes used by compilers to optimize nested loops. In the enrichment section of this week we will discuss why you may prefer one ordering of the loops over another.

> The above explains, in part, why we chose to look at $y := Ax + y$ rather than $y := Ax$. For $y := Ax + y$, the two algorithms differ only in the order in which the loops appear. To compute $y := Ax$, one would have to initialize each component of y to zero, $\psi_i := 0$. Depending on where in the algorithm that happens, transforming an algorithm that computes $y := Ax$ elements of y at a time (the inner loop implements a dot product) into an algorithm that computes with columns of A (the inner loop implements an AXPY operation) gets trickier.

Algorithms (FLAME notation)

Now let us place the two algorithms presented using the FLAME notation next to each other:

Algorithm: $y := \text{MVMULT\_N\_UNB\_VAR1}(A,x,y)$	**Algorithm:** $y := \text{MVMULT\_N\_UNB\_VAR2}(A,x,y)$
Partition $A \to \left(\dfrac{A_T}{A_B}\right), y \to \left(\dfrac{y_T}{y_B}\right)$ **where** A_T is $0 \times n$ and y_T is 0×1	**Partition** $A \to \left(A_L \mid A_R\right), x \to \left(\dfrac{x_T}{x_B}\right)$ **where** A_L is $m \times 0$ and x_T is 0×1
while $m(A_T) < m(A)$ **do**	**while** $m(x_T) < m(x)$ **do**
Repartition	**Repartition**
$\left(\dfrac{A_T}{A_B}\right) \to \left(\dfrac{A_0}{\dfrac{a_1^T}{A_2}}\right), \left(\dfrac{y_T}{y_B}\right) \to \left(\dfrac{y_0}{\dfrac{\psi_1}{y_2}}\right)$	$\left(A_L \mid A_R\right) \to \left(A_0 \mid a_1 \mid A_2\right), \left(\dfrac{x_T}{x_B}\right) \to \left(\dfrac{x_0}{\dfrac{\chi_1}{x_2}}\right)$
$\psi_1 := a_1^T x + \psi_1$	$y := \chi_1 a_1 + y$
Continue with	**Continue with**
$\left(\dfrac{A_T}{A_B}\right) \leftarrow \left(\dfrac{A_0}{\dfrac{a_1^T}{A_2}}\right), \left(\dfrac{y_T}{y_B}\right) \leftarrow \left(\dfrac{y_0}{\dfrac{\psi_1}{y_2}}\right)$	$\left(A_L \mid A_R\right) \leftarrow \left(A_0 \mid a_1 \mid A_2\right), \left(\dfrac{x_T}{x_B}\right) \leftarrow \left(\dfrac{x_0}{\dfrac{\chi_1}{x_2}}\right)$
endwhile	**endwhile**

The algorithm on the left clearly accesses the matrix by rows while the algorithm on the right accesses it by columns. Again, this is important to note, and will be discussed in enrichment for this week.

3.4.4   Cost of Matrix-Vector Multiplication

☞ **View at edX**

Consider $y := Ax + y$, where $A \in \mathbb{R}^{m \times n}$:

$$
y = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{m-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + & \cdots + & \alpha_{0,n-1}\chi_{n-1} + & \psi_0 \\ \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + & \cdots + & \alpha_{1,n-1}\chi_{n-1} + & \psi_2 \\ \vdots & \vdots & \vdots & & \vdots \\ \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + & \cdots + & \alpha_{m-1,n-1}\chi_{n-1} + & \psi_{m-1} \end{pmatrix}.
$$

Notice that there is a multiply and an add for every element of A. Since A has $m \times n = mn$ elements, $y := Ax + y$, requires *mn* multiplies and *mn* adds, for a total of $2mn$ floating point operations (flops). This count is the same regardless of the order of the loops (i.e., regardless of whether the matrix-vector multiply is organized by computing dot operations with the rows or axpy operations with the columns).

3.5   Wrap Up

3.5.1   Homework

No additional homework this week. You have done enough...

3.5.2 Summary

Special Matrices

Name	Represents linear transformation	Has entries
Zero matrix, $0_{m \times n} \in \mathbb{R}^{m \times n}$	$L_0 : \mathbb{R}^n \to \mathbb{R}^m$ $L_0(x) = 0$ for all x	$0 = 0_{m \times n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$
Identity matrix, $I \in \mathbb{R}^{n \times n}$	$L_I : \mathbb{R}^n \to \mathbb{R}^n$ $L_I(x) = x$ for all x	$I = I_{n \times n} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$
Diagonal matrix, $D \in \mathbb{R}^{n \times n}$	$L_D : \mathbb{R}^n \to \mathbb{R}^n$ if $y = L_D(x)$ then $\psi_i = \delta_i \chi_i$	$D = \begin{pmatrix} \delta_0 & 0 & \cdots & 0 \\ 0 & \delta_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_{n-1} \end{pmatrix}$

Triangular matrices

$A \in \mathbb{R}^{n \times n}$ is said to be...	if ...	
lower triangular	$\alpha_{i,j} = 0$ if $i < j$	$\begin{pmatrix} \alpha_{0,0} & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1} \end{pmatrix}$
strictly lower triangular	$\alpha_{i,j} = 0$ if $i \leq j$	$\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & 0 & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & 0 \end{pmatrix}$
unit lower triangular	$\alpha_{i,j} = \begin{cases} 0 & \text{if } i < j \\ 1 & \text{if } i = j \end{cases}$	$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ \alpha_{1,0} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & 1 & 0 \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & 1 \end{pmatrix}$
upper triangular	$\alpha_{i,j} = 0$ if $i > j$	$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & \alpha_{1,1} & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \alpha_{n-2,n-2} & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & \alpha_{n-1,n-1} \end{pmatrix}$
strictly upper triangular	$\alpha_{i,j} = 0$ if $i \geq j$	$\begin{pmatrix} 0 & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & 0 & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$
unit upper triangular	$\alpha_{i,j} = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \end{cases}$	$\begin{pmatrix} 1 & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\ 0 & 1 & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \alpha_{n-2,n-1} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$

Transpose matrix

$$
\begin{pmatrix}
\alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\
\vdots & \vdots & & \vdots & \vdots \\
\alpha_{m-2,0} & \alpha_{m-2,1} & \cdots & \alpha_{m-2,n-2} & \alpha_{m-2,n-1} \\
\alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-2} & \alpha_{m-1,n-1}
\end{pmatrix}^T
=
\begin{pmatrix}
\alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{m-2,0} & \alpha_{m-1,0} \\
\alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{m-2,1} & \alpha_{m-1,1} \\
\vdots & \vdots & & \vdots & \vdots \\
\alpha_{0,n-2} & \alpha_{1,n-2} & \cdots & \alpha_{m-2,n-2} & \alpha_{m-1,n-2} \\
\alpha_{0,n-1} & \alpha_{1,n-1} & \cdots & \alpha_{m-2,n-1} & \alpha_{m-1,n-1}
\end{pmatrix}
$$

Symmetric matrix

Matrix $A \in \mathbb{R}^{n \times n}$ is symmetric if and only if $A = A^T$:

$$
A =
\begin{pmatrix}
\alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-2} & \alpha_{0,n-1} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-2} & \alpha_{1,n-1} \\
\vdots & \vdots & & \vdots & \vdots \\
\alpha_{n-2,0} & \alpha_{n-2,1} & \cdots & \alpha_{n-2,n-2} & \alpha_{n-2,n-1} \\
\alpha_{n-1,0} & \alpha_{n-1,1} & \cdots & \alpha_{n-1,n-2} & \alpha_{n-1,n-1}
\end{pmatrix}
=
\begin{pmatrix}
\alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{n-2,0} & \alpha_{n-1,0} \\
\alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{n-2,1} & \alpha_{n-1,1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\alpha_{0,n-2} & \alpha_{1,n-2} & \cdots & \alpha_{n-2,n-2} & \alpha_{n-1,n-2} \\
\alpha_{0,n-1} & \alpha_{1,n-1} & \cdots & \alpha_{n-2,n-1} & \alpha_{n-1,n-1}
\end{pmatrix}
= A^T
$$

Scaling a matrix

Let $\beta \in \mathbb{R}$ and $A \in \mathbb{R}^{m \times n}$. Then

$$
\beta A = \beta \left(a_0 \,\middle|\, a_1 \,\middle|\, \cdots \,\middle|\, a_{n-1} \right) = \left(\beta a_0 \,\middle|\, \beta a_1 \,\middle|\, \cdots \,\middle|\, \beta a_{n-1} \right)
$$

$$
= \beta
\begin{pmatrix}
\alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\
\vdots & \vdots & & \vdots \\
\alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1}
\end{pmatrix}
=
\begin{pmatrix}
\beta\alpha_{0,0} & \beta\alpha_{0,1} & \cdots & \beta\alpha_{0,n-1} \\
\beta\alpha_{1,0} & \beta\alpha_{1,1} & \cdots & \beta\alpha_{1,n-1} \\
\vdots & \vdots & & \vdots \\
\beta\alpha_{m-1,0} & \beta\alpha_{m-1,1} & \cdots & \beta\alpha_{m-1,n-1}
\end{pmatrix}
$$

Adding matrices

Let $A, B \in \mathbb{R}^{m \times n}$. Then

$$
A + B = \left(a_0 \,\middle|\, a_1 \,\middle|\, \cdots \,\middle|\, a_{n-1} \right) + \left(b_0 \,\middle|\, b_1 \,\middle|\, \cdots \,\middle|\, b_{n-1} \right) = \left(a_0 + b_0 \,\middle|\, a_1 + b_1 \,\middle|\, \cdots \,\middle|\, a_{n-1} + b_{n-1} \right)
$$

$$
=
\begin{pmatrix}
\alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\
\alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\
\vdots & \vdots & & \vdots \\
\alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1}
\end{pmatrix}
+
\begin{pmatrix}
\beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\
\beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\
\vdots & \vdots & & \vdots \\
\beta_{m-1,0} & \beta_{m-1,1} & \cdots & \beta_{m-1,n-1}
\end{pmatrix}
$$

$$= \begin{pmatrix} \alpha_{0,0} + \beta_{0,0} & \alpha_{0,1} + \beta_{0,1} & \cdots & \alpha_{0,n-1} + \beta_{0,n-1} \\ \alpha_{1,0} + \beta_{1,0} & \alpha_{1,1} + \beta_{1,1} & \cdots & \alpha_{1,n-1} + \beta_{1,n-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{m-1,0} + \beta_{m-1,0} & \alpha_{m-1,1} + \beta_{m-1,1} & \cdots & \alpha_{m-1,n-1} + \beta_{m-1,n-1} \end{pmatrix}$$

- Matrix addition commutes: $A + B = B + A$.

- Matrix addition is associative: $(A + B) + C = A + (B + C)$.

- $(A + B)^T = A^T + B^T$.

Matrix-vector multiplication

$$\begin{aligned} Ax &= \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,n-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0}\chi_0 + & \alpha_{0,1}\chi_1 + \cdots + & \alpha_{0,n-1}\chi_{n-1} \\ \alpha_{1,0}\chi_0 + & \alpha_{1,1}\chi_1 + \cdots + & \alpha_{1,n-1}\chi_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0}\chi_0 + & \alpha_{m-1,1}\chi_1 + \cdots + & \alpha_{m-1,n-1}\chi_{n-1} \end{pmatrix} \\[2em] &= \begin{pmatrix} a_0 & \big| & a_1 & \big| & \cdots & \big| & a_{n-1} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{n-1} \end{pmatrix} = \chi_0 a_0 + \chi_1 a_1 + \cdots + \chi_{n-1} a_{n-1} \\[2em] &= \begin{pmatrix} \widetilde{a}_0^T \\ \widetilde{a}_1^T \\ \vdots \\ \widetilde{a}_{m-1}^T \end{pmatrix} x = \begin{pmatrix} \widetilde{a}_0^T x \\ \widetilde{a}_1^T x \\ \vdots \\ \widetilde{a}_{m-1}^T x \end{pmatrix} \end{aligned}$$