

From Matrix-Vector Multiplication to Matrix-Matrix Multiplication

There are a LOT of programming assignments this week.

- They are meant to help clarify “slicing and dicing”.
- They show that the right abstractions in the mathematics, when reflected in how we program, allow one to implement algorithms very quickly.
- They help you understand special properties of matrices.

Practice as much as you think will benefit your understanding of the material. There is no need to do them all!

4.1 Opening Remarks

4.1.1 Predicting the Weather

A (very) naive way for predicting the weather...

		Today		
		sunny	cloudy	rainy
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

[View at edX](#)

The following table tells us how the weather for any day (e.g., today) predicts the weather for the next day (e.g., tomorrow):

		Today		
		sunny	cloudy	rainy
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

This table is interpreted as follows: If today is rainy, then the probability that it will be cloudy tomorrow is 0.6, etc.

Homework 4.1.1.1 If today is cloudy, what is the probability that tomorrow is

- sunny?
- cloudy?
- rainy?

A (very) naive way for predicting the weather...

		Today		
		sunny	cloudy	rainy
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

If today is cloudy, what is the probability that tomorrow is

- sunny? 0.3
- cloudy? 0.3
- rainy? 0.6

[View at edX](#)

Homework 4.1.1.2 If today is sunny, what is the probability that the day after tomorrow is sunny? cloudy? rainy?

Try this! If today is cloudy, what is the probability that a week from today it is sunny? cloudy? rainy?

Think about this for at most two minutes, and then look at the answer.

Notation:

- $\chi_s^{(k)}$: probability that it will be sunny k days from now.
- $\chi_c^{(k)}$: probability that it will be cloudy k days from now.
- $\chi_r^{(k)}$: probability that it will be rainy k days from now.

[View at edX](#)

When things get messy, it helps to introduce some notation.

- Let $\chi_s^{(k)}$ denote the probability that it will be sunny k days from now (on day k).
- Let $\chi_c^{(k)}$ denote the probability that it will be cloudy k days from now.
- Let $\chi_r^{(k)}$ denote the probability that it will be rainy k days from now.

The discussion so far motivate the equations

$$\begin{aligned}\chi_s^{(k+1)} &= 0.4 \times \chi_s^{(k)} + 0.3 \times \chi_c^{(k)} + 0.1 \times \chi_r^{(k)} \\ \chi_c^{(k+1)} &= 0.4 \times \chi_s^{(k)} + 0.3 \times \chi_c^{(k)} + 0.6 \times \chi_r^{(k)} \\ \chi_r^{(k+1)} &= 0.2 \times \chi_s^{(k)} + 0.4 \times \chi_c^{(k)} + 0.3 \times \chi_r^{(k)}.\end{aligned}$$

The probabilities that denote what the weather may be on day k and the table that summarizes the probabilities are often represented as a (*state*) *vector*, $x^{(k)}$, and (*transition*) *matrix*, P , respectively:

$$x^{(k)} = \begin{pmatrix} \chi_s^{(k)} \\ \chi_c^{(k)} \\ \chi_r^{(k)} \end{pmatrix} \quad \text{and} \quad P = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix}.$$

The transition from day k to day $k+1$ is then written as the matrix-vector product (multiplication)

$$\begin{pmatrix} \chi_s^{(k+1)} \\ \chi_c^{(k+1)} \\ \chi_r^{(k+1)} \end{pmatrix} = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(k)} \\ \chi_c^{(k)} \\ \chi_r^{(k)} \end{pmatrix}$$

or $x^{(k+1)} = Px^{(k)}$, which is simply a more compact representation (way of writing) the system of linear equations.

What this demonstrates is that matrix-vector multiplication can also be used to compactly write a set of simultaneous linear equations.

Assume again that today is cloudy so that the probability that it is sunny, cloudy, or rainy today is 0, 1, and 0, respectively:

$$x^{(0)} = \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

(If we KNOW today is cloudy, then the probability that is is sunny today is zero, etc.)

Ah! Our friend the unit basis vector reappears!

Then the vector of probabilities for tomorrow's weather, $x^{(1)}$, is given by

$$\begin{aligned}\begin{pmatrix} \chi_s^{(1)} \\ \chi_c^{(1)} \\ \chi_r^{(1)} \end{pmatrix} &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix} = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0.4 \times 0 + 0.3 \times 1 + 0.1 \times 0 \\ 0.4 \times 0 + 0.3 \times 1 + 0.6 \times 0 \\ 0.2 \times 0 + 0.4 \times 1 + 0.3 \times 0 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix}.\end{aligned}$$

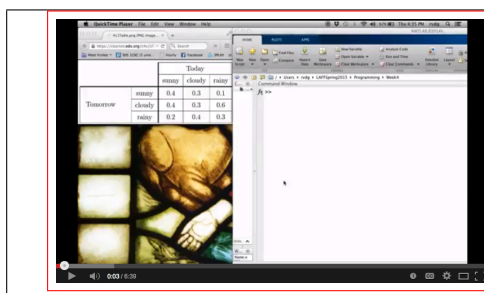
Ah! $Pe_1 = p_1$, where p_1 is the second column in matrix P . You should not be surprised!

The vector of probabilities for the day after tomorrow, $x^{(2)}$, is given by

$$\begin{aligned} \begin{pmatrix} \chi_s^{(2)} \\ \chi_c^{(2)} \\ \chi_r^{(2)} \end{pmatrix} &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(1)} \\ \chi_c^{(1)} \\ \chi_r^{(1)} \end{pmatrix} = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix} \\ &= \begin{pmatrix} 0.4 \times 0.3 + 0.3 \times 0.3 + 0.1 \times 0.4 \\ 0.4 \times 0.3 + 0.3 \times 0.3 + 0.6 \times 0.4 \\ 0.2 \times 0.3 + 0.4 \times 0.3 + 0.3 \times 0.4 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.45 \\ 0.30 \end{pmatrix}. \end{aligned}$$

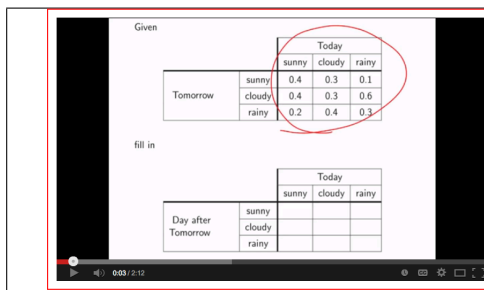
Repeating this process (preferably using Python rather than by hand), we can find the probabilities for the weather for the next seven days, under the assumption that today is cloudy:

	k							
	0	1	2	3	4	5	6	7
$x^{(k)}$	$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix}$	$\begin{pmatrix} 0.25 \\ 0.45 \\ 0.30 \end{pmatrix}$	$\begin{pmatrix} 0.265 \\ 0.415 \\ 0.320 \end{pmatrix}$	$\begin{pmatrix} 0.2625 \\ 0.4225 \\ 0.3150 \end{pmatrix}$	$\begin{pmatrix} 0.26325 \\ 0.42075 \\ 0.31600 \end{pmatrix}$	$\begin{pmatrix} 0.26312 \\ 0.42112 \\ 0.31575 \end{pmatrix}$	$\begin{pmatrix} 0.26316 \\ 0.42104 \\ 0.31580 \end{pmatrix}$



👉 View at edX

Homework 4.1.1.3 Follow the instructions in the above video.



👉 View at edX

We could build a table that tells us how to predict the weather for the day after tomorrow from the weather today:

		Today		
		sunny	cloudy	rainy
Day after Tomorrow	sunny			
	cloudy			
	rainy			

One way you can do this is to observe that

$$\begin{aligned}
 \begin{pmatrix} \chi_s^{(2)} \\ \chi_c^{(2)} \\ \chi_r^{(2)} \end{pmatrix} &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(1)} \\ \chi_c^{(1)} \\ \chi_r^{(1)} \end{pmatrix} \\
 &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \left(\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix} \right) = Q \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix},
 \end{aligned}$$

where Q is the transition matrix that tells us how the weather today predicts the weather the day after tomorrow. (Well, actually, we don't yet know that applying a matrix to a vector twice is a linear transformation... We'll learn that later this week.)

Now, just like P is simply the matrix of values from the original table that showed how the weather tomorrow is predicted from today's weather, Q is the matrix of values for the above table.

Homework 4.1.1.4 Given

		Today		
		sunny	cloudy	rainy
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

fill in the following table, which predicts the weather the day after tomorrow given the weather today:

		Today		
		sunny	cloudy	rainy
Day after Tomorrow	sunny			
	cloudy			
	rainy			

Now here is the hard part: Do so without using your knowledge about how to perform a matrix-matrix multiplication, since you won't learn about that until later this week... May we suggest that you instead use MATLAB to perform the necessary calculations.

4.1.2 Outline

4.1. Opening Remarks	145
4.1.1. Predicting the Weather	145
4.1.2. Outline	151
4.1.3. What You Will Learn	152
4.2. Preparation	153
4.2.1. Partitioned Matrix-Vector Multiplication	153
4.2.2. Transposing a Partitioned Matrix	156
4.2.3. Matrix-Vector Multiplication, Again	161
4.3. Matrix-Vector Multiplication with Special Matrices	165
4.3.1. Transpose Matrix-Vector Multiplication	165
4.3.2. Triangular Matrix-Vector Multiplication	167
4.3.3. Symmetric Matrix-Vector Multiplication	176
4.4. Matrix-Matrix Multiplication (Product)	180
4.4.1. Motivation	180
4.4.2. From Composing Linear Transformations to Matrix-Matrix Multiplication	182
4.4.3. Computing the Matrix-Matrix Product	183
4.4.4. Special Shapes	186
4.4.5. Cost	195
4.5. Enrichment	195
4.5.1. Markov Chains: Their Application	195
4.6. Wrap Up	196
4.6.1. Homework	196
4.6.2. Summary	197

4.1.3 What You Will Learn

Upon completion of this unit, you should be able to

- Apply matrix vector multiplication to predict the probability of future states in a Markov process.
 - Make use of partitioning to perform matrix vector multiplication.
 - Transpose a partitioned matrix.
 - Partition conformally, ensuring that the size of the matrices and vectors match so that matrix-vector multiplication works.
 - Take advantage of special structures to perform matrix-vector multiplication with triangular and symmetric matrices.
 - Express and implement various matrix-vector multiplication algorithms using the FLAME notation and FlamePy.
 - Make connections between the composition of linear transformations and matrix-matrix multiplication.
 - Compute a matrix-matrix multiplication.
 - Recognize scalars and column/row vectors as special cases of matrices.
 - Compute common vector-vector and matrix-vector operations as special cases of matrix-matrix multiplication.
 - Compute an outer product xy^T as a special case of matrix-matrix multiplication and recognize that
 - The rows of the resulting matrix are scalar multiples of y^T .
 - The columns of the resulting matrix are scalar multiples of x .
-

4.2 Preparation

4.2.1 Partitioned Matrix-Vector Multiplication

Motivation

Consider

$$A = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{cc|cc|cc} -1 & 2 & 4 & 1 & 0 & \\ \hline 1 & 0 & -1 & -2 & 1 & \\ \hline 2 & -1 & 3 & 1 & 2 & \\ \hline 1 & 2 & 3 & 4 & 3 & \\ \hline -1 & -2 & 0 & 1 & 2 & \end{array} \right),$$

$$x = \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right) = \left(\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \right), \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right),$$

where $y_0, y_2 \in \mathbb{R}^2$. Then $y = Ax$ means that

$$\begin{aligned} y &= \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right) = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right) = \left(\begin{array}{c} A_{00}x_0 + a_{01}\chi_1 + A_{02}x_2 \\ a_{10}^T x_0 + \alpha_{11}\chi_1 + a_{12}^T x_2 \\ A_{20}x_0 + a_{21}\chi_1 + A_{22}x_2 \end{array} \right) \\ &= \left(\begin{array}{c} \left(\begin{array}{cc} -1 & 2 \\ 1 & 0 \end{array} \right) \left(\begin{array}{c} 1 \\ 2 \end{array} \right) + \left(\begin{array}{c} 4 \\ -1 \end{array} \right) 3 + \left(\begin{array}{cc} 1 & 0 \\ -2 & 1 \end{array} \right) \left(\begin{array}{c} 4 \\ 5 \end{array} \right) \\ \hline \left(\begin{array}{cc} 2 & -1 \end{array} \right) \left(\begin{array}{c} 1 \\ 2 \end{array} \right) + \left(\begin{array}{c} 3 \end{array} \right) 3 + \left(\begin{array}{cc} 1 & 2 \end{array} \right) \left(\begin{array}{c} 4 \\ 5 \end{array} \right) \\ \hline \left(\begin{array}{cc} 1 & 2 \\ -1 & -2 \end{array} \right) \left(\begin{array}{c} 1 \\ 2 \end{array} \right) + \left(\begin{array}{c} 3 \\ 0 \end{array} \right) 3 + \left(\begin{array}{cc} 4 & 3 \\ 1 & 2 \end{array} \right) \left(\begin{array}{c} 4 \\ 5 \end{array} \right) \end{array} \right) = \end{aligned}$$

$$\begin{pmatrix} \frac{\begin{pmatrix} (-1) \times (1) + (2) \times (2) \\ (1) \times (1) + (0) \times (2) \end{pmatrix} + \begin{pmatrix} (4) \times (3) \\ (-1) \times (3) \end{pmatrix} + \begin{pmatrix} (1) \times (4) + (0) \times (5) \\ (-2) \times (4) + (1) \times (5) \end{pmatrix}}{(2) \times (1) + (-1) \times (2) + (3) \times (3) + (1) \times (4) + (2) \times (5)} \\ \frac{\begin{pmatrix} (1) \times (1) + (2) \times (2) \\ (-1) \times (1) + (-2) \times (2) \end{pmatrix} + \begin{pmatrix} (3) \times 3 \\ (0) \times 3 \end{pmatrix} + \begin{pmatrix} (4) \times (4) + (3) \times (5) \\ (1) \times (4) + (2) \times (5) \end{pmatrix}}{(1) \times (1) + (2) \times (2) + (3) \times (3) + (4) \times (4) + (3) \times (5)} \\ \frac{\begin{pmatrix} (-1) \times (1) + (2) \times (2) + (4) \times (3) + (1) \times (4) + (0) \times (5) \\ (1) \times (1) + (0) \times (2) + (-1) \times (3) + (-2) \times (4) + (1) \times (5) \end{pmatrix}}{(2) \times (1) + (-1) \times (2) + (3) \times (3) + (1) \times (4) + (2) \times (5)} \\ \frac{\begin{pmatrix} (-1) \times (1) + (2) \times (2) + (4) \times (3) + (1) \times (4) + (0) \times (5) \\ (1) \times (1) + (0) \times (2) + (-1) \times (3) + (-2) \times (4) + (1) \times (5) \end{pmatrix}}{(2) \times (1) + (-1) \times (2) + (3) \times (3) + (1) \times (4) + (2) \times (5)} \\ \frac{\begin{pmatrix} (-1) \times (1) + (2) \times (2) + (4) \times (3) + (1) \times (4) + (0) \times (5) \\ (1) \times (1) + (0) \times (2) + (-1) \times (3) + (-2) \times (4) + (1) \times (5) \end{pmatrix}}{(2) \times (1) + (-1) \times (2) + (3) \times (3) + (1) \times (4) + (2) \times (5)} \\ \frac{\begin{pmatrix} (-1) \times (1) + (2) \times (2) + (4) \times (3) + (1) \times (4) + (0) \times (5) \\ (1) \times (1) + (0) \times (2) + (-1) \times (3) + (-2) \times (4) + (1) \times (5) \end{pmatrix}}{(2) \times (1) + (-1) \times (2) + (3) \times (3) + (1) \times (4) + (2) \times (5)} \end{pmatrix} = \begin{pmatrix} 19 \\ -5 \\ 23 \\ 45 \\ 9 \end{pmatrix}$$

Homework 4.2.1.1 Consider

$$A = \begin{pmatrix} -1 & 2 & 4 & 1 & 0 \\ 1 & 0 & -1 & -2 & 1 \\ 2 & -1 & 3 & 1 & 2 \\ 1 & 2 & 3 & 4 & 3 \\ -1 & -2 & 0 & 1 & 2 \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix},$$

and partition these into submatrices (regions) as follows:

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) \quad \text{and} \quad \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ \hline x_2 \end{array} \right),$$

where $A_{00} \in \mathbb{R}^{3 \times 3}$, $x_0 \in \mathbb{R}^3$, α_{11} is a scalar, and χ_1 is a scalar. Show with lines how A and x are partitioned:

$$\left(\begin{array}{ccccc} -1 & 2 & 4 & 1 & 0 \\ 1 & 0 & -1 & -2 & 1 \\ 2 & -1 & 3 & 1 & 2 \\ 1 & 2 & 3 & 4 & 3 \\ -1 & -2 & 0 & 1 & 2 \end{array} \right) \quad \left(\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \right).$$

Homework 4.2.1.2 With the partitioning of matrices A and x in the above exercise, repeat the partitioned matrix-vector multiplication, similar to how this unit started.

Theory

Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$. Partition

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right), \quad x = \left(\begin{array}{c} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{array} \right), \quad \text{and} \quad y = \left(\begin{array}{c} y_0 \\ y_1 \\ \vdots \\ y_{M-1} \end{array} \right)$$

where

- $m = m_0 + m_1 + \cdots + m_{M-1}$,
- $m_i \geq 0$ for $i = 0, \dots, M-1$,
- $n = n_0 + n_1 + \cdots + n_{N-1}$,
- $n_j \geq 0$ for $j = 0, \dots, N-1$, and
- $A_{i,j} \in \mathbb{R}^{m_i \times n_j}$, $x_j \in \mathbb{R}^{n_j}$, and $y_i \in \mathbb{R}^{m_i}$.

If $y = Ax$ then

$$\begin{aligned} & \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right) \left(\begin{array}{c} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{array} \right) \\ &= \left(\begin{array}{c} A_{0,0}x_0 + A_{0,1}x_1 + \cdots + A_{0,N-1}x_{N-1} \\ \hline A_{1,0}x_0 + A_{1,1}x_1 + \cdots + A_{1,N-1}x_{N-1} \\ \hline \vdots \\ \hline A_{M-1,0}x_0 + A_{M-1,1}x_1 + \cdots + A_{M-1,N-1}x_{N-1} \end{array} \right). \end{aligned}$$

In other words,

$$y_i = \sum_{j=0}^{N-1} A_{i,j}x_j.$$

This is intuitively true and messy to prove carefully. Therefore we will not give its proof, relying on the many examples we will encounter in subsequent units instead.

If one partitions matrix A , vector x , and vector y into blocks, **and** one makes sure the dimensions match up, **then** blocked matrix-vector multiplication proceeds exactly as does a regular matrix-vector multiplication **except** that individual multiplications of scalars commute while (in general) individual multiplications with matrix and vector blocks (submatrices and subvectors) do not.

The labeling of the submatrices and subvectors in this unit was carefully chosen to convey information. Consider

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

The letters that are used convey information about the shapes. For example, for a_{01} and a_{21} the use of a lowercase Roman letter indicates they are column vectors while the T s in a_{10}^T and a_{12}^T indicate that they are row vectors. Symbols α_{11} and χ_1 indicate these are scalars. We will use these conventions consistently to enhance readability.

Notice that the partitioning of matrix A and vectors x and y has to be “conformal”. The simplest way to understand this is that matrix-vector multiplication only works if the sizes of matrices and vectors being multiply match. So, a partitioning of A , x , and y , when performing a given operation, is conformal if the suboperations with submatrices and subvectors that are encountered make sense.

4.2.2 Transposing a Partitioned Matrix

Look at the different submatrices:

$$A = \left(\begin{array}{c|c|c} 1 & -1 & 3 \\ \hline 2 & -2 & 1 \\ \hline 0 & -4 & 3 \end{array} \right), \quad A^T = \left(\begin{array}{c|c|c} 1 & 2 & 0 \\ \hline -1 & -2 & -4 \\ \hline 3 & 1 & 3 \end{array} \right)$$

$$\boxed{A} = \left(\begin{array}{c|c|c} 1 & -1 & 3 \\ \hline 2 & -2 & 1 \\ \hline 0 & -4 & 3 \end{array} \right), \quad \boxed{A^T} = \left(\begin{array}{c|c|c} (1 \ -1 \ 3)^T & (2 \ -2 \ 0)^T & (0 \ -4 \ 3)^T \\ \hline (2 \ -2 \ 1)^T & (3 \ 1 \ 3)^T & (3 \ 2 \ 2)^T \end{array} \right)$$

[View at edX](#)

Motivation

Consider

$$\begin{aligned} \left(\begin{array}{c|c} 1 & -1 & 3 & 2 \\ \hline 2 & -2 & 1 & 0 \\ \hline 0 & -4 & 3 & 2 \end{array} \right)^T &= \left(\begin{array}{c|c} \left(\begin{array}{ccc} 1 & -1 & 3 \end{array} \right) & \left(\begin{array}{c} 2 \\ 0 \end{array} \right) \\ \hline \left(\begin{array}{ccc} 2 & -2 & 1 \end{array} \right) & \left(\begin{array}{c} 2 \end{array} \right) \end{array} \right)^T \\ &= \left(\begin{array}{c|c} \left(\begin{array}{ccc} 1 & -1 & 3 \end{array} \right)^T & \left(\begin{array}{ccc} 0 & -4 & 3 \end{array} \right)^T \\ \hline \left(\begin{array}{c} 2 \\ 0 \end{array} \right)^T & \left(\begin{array}{c} 2 \end{array} \right)^T \end{array} \right) \end{aligned}$$

$$= \left(\left(\begin{array}{cc|c} 1 & 2 & 0 \\ -1 & -2 & -4 \\ 3 & 1 & 3 \\ \hline 2 & 0 & 2 \end{array} \right) \right) = \left(\begin{array}{cc|c} 1 & 2 & 0 \\ -1 & -2 & -4 \\ 3 & 1 & 3 \\ \hline 2 & 0 & 2 \end{array} \right).$$

This example illustrates a general rule: When transposing a partitioned matrix (matrix partitioned into submatrices), you transpose the matrix of blocks, and then you transpose each block.

Homework 4.2.2.1 Show, step-by-step, how to transpose

$$\left(\begin{array}{cc|cc} 1 & -1 & 3 & 2 \\ 2 & -2 & 1 & 0 \\ \hline 0 & -4 & 3 & 2 \end{array} \right)$$

Theory

Let $A \in \mathbb{R}^{m \times n}$ be partitioned as follows:

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{array} \right),$$

where $A_{i,j} \in \mathbb{R}^{m_i \times n_j}$. Then

$$A^T = \left(\begin{array}{c|c|c|c} A_{0,0}^T & A_{1,0}^T & \cdots & A_{M-1,0}^T \\ \hline A_{0,1}^T & A_{1,1}^T & \cdots & A_{M-1,1}^T \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{0,N-1}^T & A_{1,N-1}^T & \cdots & A_{M-1,N-1}^T \end{array} \right).$$

Transposing a partitioned matrix means that you view each submatrix as if it is a scalar, and you then transpose the matrix as if it is a matrix of scalars. But then you recognize that each of those scalars is actually a submatrix and you also transpose that submatrix.

Special cases

We now discuss a number of special cases that you may encounter.

Each submatrix is a scalar. If

$$A = \left(\begin{array}{c|c|c|c} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,N-1} \\ \hline \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{M-1,0} & \alpha_{M-1,1} & \cdots & \alpha_{M-1,N-1} \end{array} \right)$$

then

$$A^T = \left(\begin{array}{c|c|c|c} \alpha_{0,0}^T & \alpha_{1,0}^T & \cdots & \alpha_{M-1,0}^T \\ \hline \alpha_{0,1}^T & \alpha_{1,1}^T & \cdots & \alpha_{M-1,1}^T \\ \hline \vdots & \vdots & & \vdots \\ \hline \alpha_{0,N-1}^T & \alpha_{1,N-1}^T & \cdots & \alpha_{M-1,N-1}^T \end{array} \right) = \left(\begin{array}{cccc} \alpha_{0,0} & \alpha_{1,0} & \cdots & \alpha_{M-1,0} \\ \alpha_{0,1} & \alpha_{1,1} & \cdots & \alpha_{M-1,1} \\ \vdots & \vdots & & \vdots \\ \alpha_{0,N-1} & \alpha_{1,N-1} & \cdots & \alpha_{M-1,N-1} \end{array} \right).$$

This is because the transpose of a scalar is just that scalar.

The matrix is partitioned by rows. If

$$A = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array} \right),$$

where each \tilde{a}_i^T is a row of A , then

$$A^T = \left(\begin{array}{c} \tilde{a}_0^T \\ \hline \tilde{a}_1^T \\ \hline \vdots \\ \hline \tilde{a}_{m-1}^T \end{array} \right)^T = \left((\tilde{a}_0^T)^T \mid (\tilde{a}_1^T)^T \mid \cdots \mid (\tilde{a}_{m-1}^T)^T \right) = \left(\tilde{a}_0 \mid \tilde{a}_1 \mid \cdots \mid \tilde{a}_{m-1} \right).$$

This shows that rows of A , \tilde{a}_i^T , become columns of A^T : \tilde{a}_i .

The matrix is partitioned by columns. If

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

where each a_j is a column of A , then

$$A^T = \left(a_0 \mid a_1 \mid \cdots \mid a_{n-1} \right)^T = \left(\begin{array}{c} a_0^T \\ \hline a_1^T \\ \hline \vdots \\ \hline a_{n-1}^T \end{array} \right).$$

This shows that columns of A , a_j , become rows of A^T : a_j^T .

2×2 **blocked partitioning.** If

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

then

$$A^T = \left(\begin{array}{c|c} A_{TL}^T & A_{BL}^T \\ \hline A_{TR}^T & A_{BR}^T \end{array} \right).$$

3×3 **blocked partitioning.** If

$$A = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right),$$

then

$$A^T = \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right)^T = \left(\begin{array}{c|c|c} A_{00}^T & (a_{10}^T)^T & A_{20}^T \\ \hline a_{01}^T & \alpha_{11}^T & a_{12}^T \\ \hline A_{02}^T & (a_{21}^T)^T & A_{22}^T \end{array} \right) = \left(\begin{array}{c|c|c} A_{00}^T & a_{10} & A_{20}^T \\ \hline a_{01}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02}^T & a_{12} & A_{22}^T \end{array} \right).$$

Anyway, you get the idea!!!

Homework 4.2.2.2 Transpose the following matrices:

1. $\begin{pmatrix} 3 \end{pmatrix}$

2. $\begin{pmatrix} 3 \\ \frac{1}{\frac{1}{8}} \end{pmatrix}$

3. $\left(\begin{array}{cc|c|c} 3 & 1 & 1 & 8 \end{array} \right)$

4. $\left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{array} \right)$

5. $\left(\begin{array}{c|c|c} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{array} \right)$

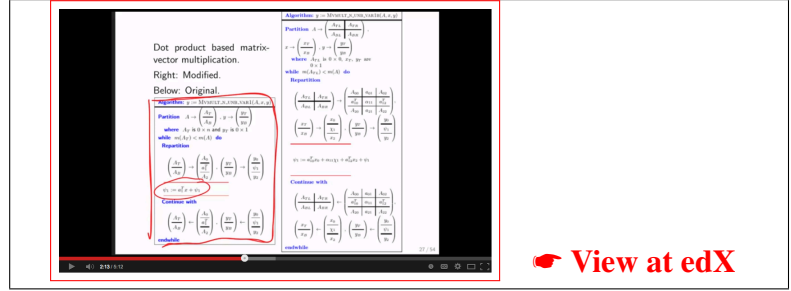
6. $\left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \end{array} \right)$

7. $\left(\left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \end{array} \right)^T \right)$

For any matrix $A \in \mathbb{R}^{m \times n}$,

$$A^{TT} = (A^T)^T = A$$

4.2.3 Matrix-Vector Multiplication, Again



Motivation

In the next few units, we will modify the matrix-vector multiplication algorithms from last week so that they can take advantage of matrices with special structure (e.g., triangular or symmetric matrices).

Now, what makes a triangular or symmetric matrix special? For one thing, it is square. For another, it only requires one triangle of a matrix to be stored. It was for this reason that we ended up with “algorithm skeletons” that looked like the one in Figure 4.1 when we presented algorithms for “triangularizing” or “symmetrizing” a matrix.

Now, consider a typical partitioning of a matrix that is encountered in such an algorithm:

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{01} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{cc|ccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{array} \right),$$

where each \times represents an entry in the matrix (in this case 6×6). If, for example, the matrix is lower triangular,

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{01} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right) = \left(\begin{array}{cc|ccc} \times & 0 & 0 & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 & 0 \\ \hline \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \\ \times & \times & \times & \times & \times & \times \end{array} \right),$$

then $a_{01} = 0$, $A_{02} = 0$, and $a_{12}^T = 0$. (Remember: the “0” is a matrix or vector “of appropriate size”.) If instead the matrix is symmetric with only the lower triangular part stored, then $a_{01} = (a_{10}^T)^T = a_{10}$, $A_{02} = A_{20}^T$, and $a_{12}^T = a_{21}^T$.

The above observation leads us to express the matrix-vector multiplication algorithms for computing $y := Ax + y$ given in Figure 4.2. Note:

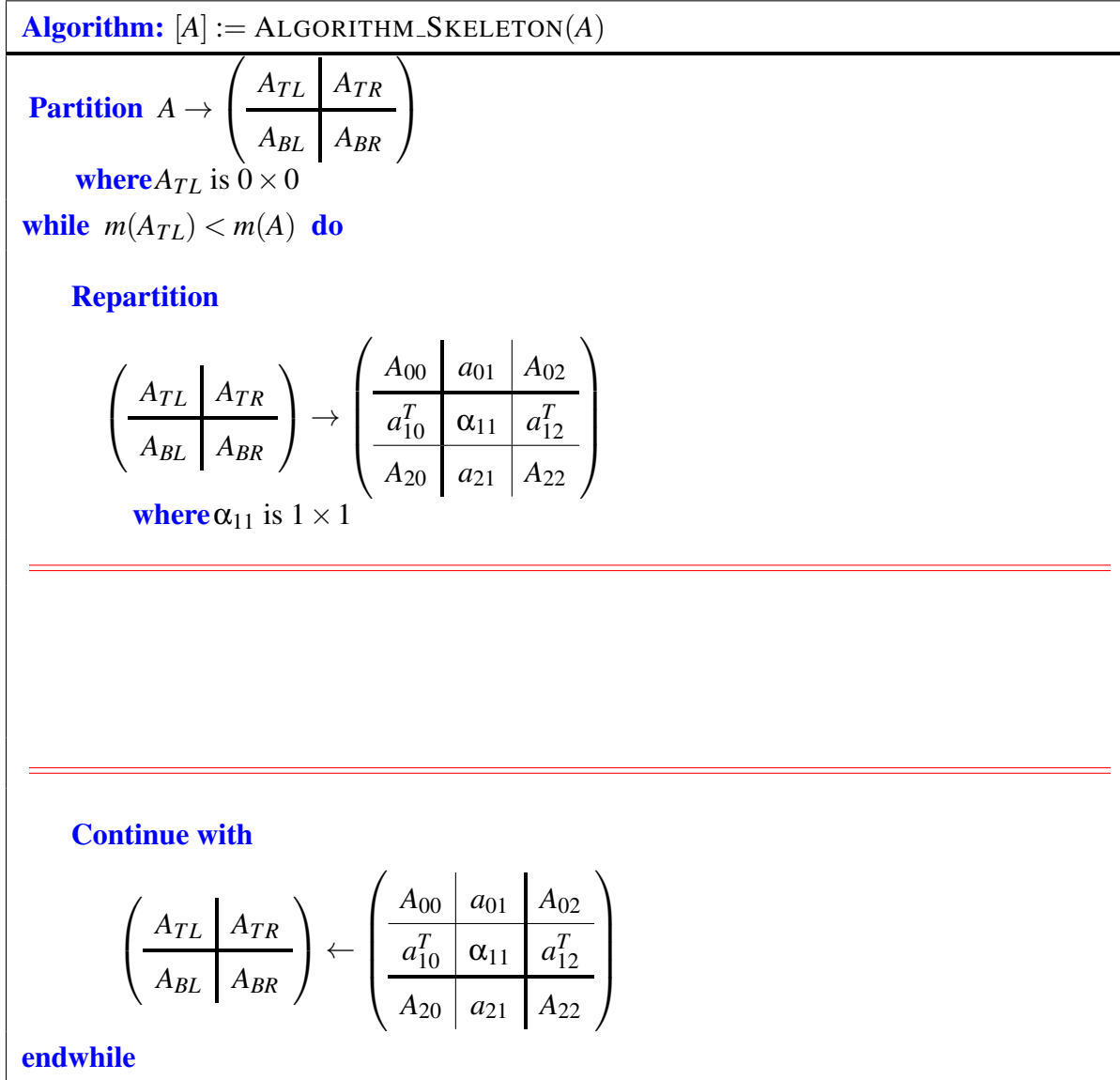


Figure 4.1: Code skeleton for algorithms when matrices are triangular or symmetric.

- For the left algorithm, what was previously the “current” row in matrix A , a_1^T , is now viewed as consisting of three parts:

$$a_1^T = \left(a_{10}^T \mid \alpha_{11} \mid a_{12}^T \right)$$

while the vector x is now also partitioned into three parts:

$$x = \left(\begin{array}{c} x_0 \\ \hline \chi_1 \\ x_1 \end{array} \right).$$

<p>Algorithm: $y := \text{MVMULT_N_UNB_VAR1B}(A, x, y)$</p> <p>Partition $A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right),$</p> <p>$x \rightarrow \left(\begin{array}{c} x_T \\ x_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$</p> <p>where A_{TL} is 0×0, x_T, y_T are 0×1</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p>Repartition</p> $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right),$ $\left(\begin{array}{c} x_T \\ x_B \end{array} \right) \rightarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$ <hr/> $\psi_1 := a_{10}^T x_0 + \alpha_{11} \chi_1 + a_{12}^T x_2 + \psi_1$ <hr/> <p>Continue with</p> $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right),$ $\left(\begin{array}{c} x_T \\ x_B \end{array} \right) \leftarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$ <p>endwhile</p>	<p>Algorithm: $y := \text{MVMULT_N_UNB_VAR2B}(A, x, y)$</p> <p>Partition $A \rightarrow \left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right),$</p> <p>$x \rightarrow \left(\begin{array}{c} x_T \\ x_B \end{array} \right), y \rightarrow \left(\begin{array}{c} y_T \\ y_B \end{array} \right)$</p> <p>where A_{TL} is 0×0, x_T, y_T are 0×1</p> <p>while $m(A_{TL}) < m(A)$ do</p> <p>Repartition</p> $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \rightarrow \left(\begin{array}{c c c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right),$ $\left(\begin{array}{c} x_T \\ x_B \end{array} \right) \rightarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \rightarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$ <hr/> $y_0 := \chi_1 a_{01} + y_0$ $\psi_1 := \chi_1 \alpha_{11} + \psi_1$ $y_2 := \chi_1 a_{21} + y_2$ <hr/> <p>Continue with</p> $\left(\begin{array}{c c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array} \right) \leftarrow \left(\begin{array}{c c c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right),$ $\left(\begin{array}{c} x_T \\ x_B \end{array} \right) \leftarrow \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_2 \end{array} \right), \left(\begin{array}{c} y_T \\ y_B \end{array} \right) \leftarrow \left(\begin{array}{c} y_0 \\ \psi_1 \\ y_2 \end{array} \right)$ <p>endwhile</p>
--	--

Figure 4.2: Alternative algorithms for matrix-vector multiplication.

As we saw in the first week, the partitioned dot product becomes

$$a_1^T x = \left(a_{10}^T \mid \alpha_{11} \mid a_{12}^T \right) \left(\begin{array}{c} x_0 \\ \chi_1 \\ x_1 \end{array} \right) = a_{10}^T x_0 + \alpha_{11} \chi_1 + a_{12}^T x_2,$$

which explains why the update

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

is now

$$\psi_1 := a_{10}^T x_0 + \alpha_{11} \chi_1 + a_{12}^T x_2 + \psi_1.$$

- Similar, for the algorithm on the right, based on the matrix-vector multiplication algorithm that uses the AXPY operations, we note that

$$y := \chi_1 a_1 + y$$

is replaced by

$$\begin{pmatrix} \frac{y_0}{\psi_1} \\ y_2 \end{pmatrix} := \chi_1 \begin{pmatrix} \frac{a_{01}}{\alpha_{11}} \\ a_{21} \end{pmatrix} + \begin{pmatrix} \frac{y_0}{\psi_1} \\ y_2 \end{pmatrix}$$

which equals

$$\begin{pmatrix} \frac{y_0}{\psi_1} \\ y_2 \end{pmatrix} := \begin{pmatrix} \frac{\chi_1 a_{01} + y_0}{\chi_1 \alpha_{11} + \psi_1} \\ \chi_1 a_{21} + y_2 \end{pmatrix}.$$

This explains the update

$$\begin{aligned} y_0 &:= \chi_1 a_{01} + y_0 \\ \psi_1 &:= \chi_1 \alpha_{11} + \psi_1 \\ y_2 &:= \chi_1 a_{21} + y_2. \end{aligned}$$

Now, for matrix-vector multiplication $y := Ax + y$, it is not beneficial to break the computation up in this way. Typically, a dot product is more efficient than multiple operations with the subvectors. Similarly, typically one AXPY is more efficient than multiple AXPYs. But the observations in this unit lay the foundation for modifying the algorithms to take advantage of special structure in the matrix, later this week.

Homework 4.2.3.1 Implement routines

- `[y_out] = Mvmult_n_unb_var1B(A, x, y);` and
- `[y_out] = Mvmult_n_unb_var2B(A, x, y)`

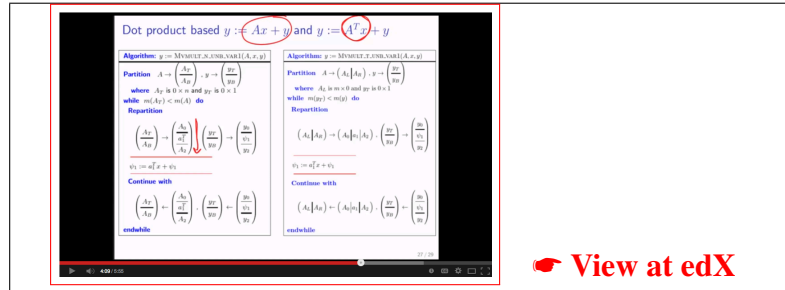
that compute $y := Ax + y$ via the algorithms in Figure 4.2.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

4.3 Matrix-Vector Multiplication with Special Matrices

4.3.1 Transpose Matrix-Vector Multiplication



Motivation

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

$$A^T x = \begin{pmatrix} 1 & -2 & 0 \\ 2 & -1 & 1 \\ 1 & 2 & 3 \end{pmatrix}^T \begin{pmatrix} -1 \\ 2 \\ -3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ -2 & -1 & 2 \\ 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ -3 \end{pmatrix} = \begin{pmatrix} 0 \\ -6 \\ -7 \end{pmatrix}.$$

The thing to notice is that what was a column in A becomes a row in A^T .

Algorithms

Let us consider how to compute $y := A^T x + y$.

It would be possible to explicitly transpose matrix A into a new matrix B (using, for example, the transpose function you wrote in Week 3) and to then compute $y := Bx + y$. This approach has at least two drawbacks:

- You will need space for the matrix B . Computational scientists tend to push the limits of available memory, and hence are always hesitant to use large amounts of space that isn't absolutely necessary.
- Transposing A into B takes time. A matrix-vector multiplication requires $2mn$ flops. Transposing a matrix requires $2mn$ memops (mn reads from memory and mn writes to memory). Memory operations are very slow relative to floating point operations... So, you will spend all your time transposing the matrix.

Now, the motivation for this unit suggest that we can simply use columns of A for the dot products in the dot product based algorithm for $y := Ax + y$. This suggests the algorithm in FLAME notation in Figure 4.3 (left). Alternatively, one can exploit the fact that columns in A become rows of A^T to change the algorithm for computing $y := Ax + y$ that is based on AXPY operations into an algorithm for computing $y := A^T x + y$, as shown in Figure 4.3 (right).

<p>Algorithm: $y := \text{MVMULT_T_UNB_VAR1}(A, x, y)$</p> <p>Partition $A \rightarrow \left(A_L \middle A_R \right), y \rightarrow \begin{pmatrix} y_T \\ y_B \end{pmatrix}$ where A_L is $m \times 0$ and y_T is 0×1 while $m(y_T) < m(y)$ do Repartition $\left(A_L \middle A_R \right) \rightarrow \left(A_0 \middle a_1 \middle A_2 \right), \begin{pmatrix} y_T \\ y_B \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ \frac{\psi_1}{y_2} \end{pmatrix}$ <hr/> $\psi_1 := a_1^T x + \psi_1$ <hr/> Continue with $\left(A_L \middle A_R \right) \leftarrow \left(A_0 \middle a_1 \middle A_2 \right), \begin{pmatrix} y_T \\ y_B \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ \frac{\psi_1}{y_2} \end{pmatrix}$ endwhile</p>	<p>Algorithm: $y := \text{MVMULT_T_UNB_VAR2}(A, x, y)$</p> <p>Partition $A \rightarrow \begin{pmatrix} A_T \\ A_B \end{pmatrix}, x \rightarrow \begin{pmatrix} x_T \\ x_B \end{pmatrix}$ where A_T is $0 \times n$ and x_T is 0×1 while $m(A_T) < m(A)$ do Repartition $\begin{pmatrix} A_T \\ A_B \end{pmatrix} \rightarrow \begin{pmatrix} A_0 \\ \frac{a_1^T}{A_2} \end{pmatrix}, \begin{pmatrix} x_T \\ x_B \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ \frac{\chi_1}{x_2} \end{pmatrix}$ <hr/> $y := \chi_1 a_1 + y$ <hr/> Continue with $\begin{pmatrix} A_T \\ A_B \end{pmatrix} \leftarrow \begin{pmatrix} A_0 \\ \frac{a_1^T}{A_2} \end{pmatrix}, \begin{pmatrix} x_T \\ x_B \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ \frac{\chi_1}{x_2} \end{pmatrix}$ endwhile</p>
---	---

Figure 4.3: Algorithms for computing $y := A^T x + y$.

Implementation

Homework 4.3.1.1 Implement the routines

- `[y_out] = Mvmult_t_unb_var1(A, x, y);` and
- `[y_out] = Mvmult_t_unb_var2(A, x, y)`

that compute $y := A^T x + y$ via the algorithms in Figure 4.3.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Homework 4.3.1.2 Implementations achieve better performance (finish faster) if one accesses data consecutively in memory. Now, most scientific computing codes store matrices in “column-major order” which means that the first column of a matrix is stored consecutively in memory, then the second column, and so forth. Now, this means that an algorithm that accesses a matrix by columns tends to be faster than an algorithm that accesses a matrix by rows. That, in turn, means that when one is presented with more than one algorithm, one should pick the algorithm that accesses the matrix by columns.

Our FLAME notation makes it easy to recognize algorithms that access the matrix by columns.

- For the matrix-vector multiplication $y := Ax + y$, would you recommend the algorithm that uses dot products or the algorithm that uses `axpy` operations?
- For the matrix-vector multiplication $y := A^T x + y$, would you recommend the algorithm that uses dot products or the algorithm that uses `axpy` operations?

The point of this last exercise is to make you aware of the fact that knowing more than one algorithm can give you a performance edge. (Useful if you pay \$30 million for a supercomputer and you want to get the most out of its use.)

4.3.2 Triangular Matrix-Vector Multiplication

Partition $U \rightarrow \begin{pmatrix} U_{00} & U_{01} & U_{02} \\ U_{10} & U_{11} & U_{12} \\ U_{20} & U_{21} & U_{22} \end{pmatrix}$, $x \rightarrow \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$, $y \rightarrow \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}$
 where U_{ij} is 0×0 , x_i , y_i are 0×1
 while $m(U_{11}) < m(U)$ do
 Repartition
 $\begin{pmatrix} U_{00} & U_{01} & U_{02} \\ U_{10} & U_{11} & U_{12} \\ U_{20} & U_{21} & U_{22} \end{pmatrix} \rightarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}$, $\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$, $\begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} \rightarrow \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}$
 where v_{11} , x_1 , and y_1 are scalars
 $y_1 := u_{10}^T x_0 + v_{11} x_1 + u_{12}^T x_2 + y_1$
 Continue with
 $\begin{pmatrix} U_{00} & U_{01} & U_{02} \\ U_{10} & U_{11} & U_{12} \\ U_{20} & U_{21} & U_{22} \end{pmatrix} \leftarrow \begin{pmatrix} U_{00} & u_{01} & U_{02} \\ u_{10}^T & v_{11} & u_{12}^T \\ U_{20} & u_{21} & U_{22} \end{pmatrix}$, $\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \leftarrow \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$, $\begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} \leftarrow \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}$
 endwhile

[View at edX](#)

Motivation

Let $U \in \mathbb{R}^{n \times n}$ be an upper triangular matrix and $x \in \mathbb{R}^n$ be a vector. Consider

$$Ux = \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & v_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \left(\begin{array}{c|c|c} -1 & 2 & 4 \\ 0 & 0 & -1 \\ \hline 0 & 0 & 3 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 2 \end{array} \right) \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

$$= \left(\begin{array}{c} \star \\ \star \\ \hline \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} 1 \\ 2 \end{pmatrix} + (3)(3) + \begin{pmatrix} 1 \\ 2 \end{pmatrix}^T \begin{pmatrix} 4 \\ 5 \end{pmatrix} \\ \hline \star \\ \star \end{array} \right) = \left(\begin{array}{c} \star \\ \star \\ \hline (3)(3) + \begin{pmatrix} 1 \\ 2 \end{pmatrix}^T \begin{pmatrix} 4 \\ 5 \end{pmatrix} \\ \hline \star \\ \star \end{array} \right),$$

where \star s indicate components of the result that aren't important in our discussion right now. We notice that $u_{10}^T = 0$ (a vector of two zeroes) and hence we need not compute with it.

Theory

If

$$U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline U_{BL} & U_{BR} \end{array} \right) = \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline u_{10}^T & \mathbf{v}_{11} & u_{12}^T \\ \hline U_{20} & u_{21} & U_{22} \end{array} \right),$$

where U_{TL} and U_{00} are square matrices. Then

- $U_{BL} = 0$, $u_{10}^T = 0$, $U_{20} = 0$, and $u_{21} = 0$, where 0 indicates a matrix or vector of the appropriate dimensions.
- U_{TL} and U_{BR} are upper triangular matrices.

We will just state this as “intuitively obvious”.

Similarly, if

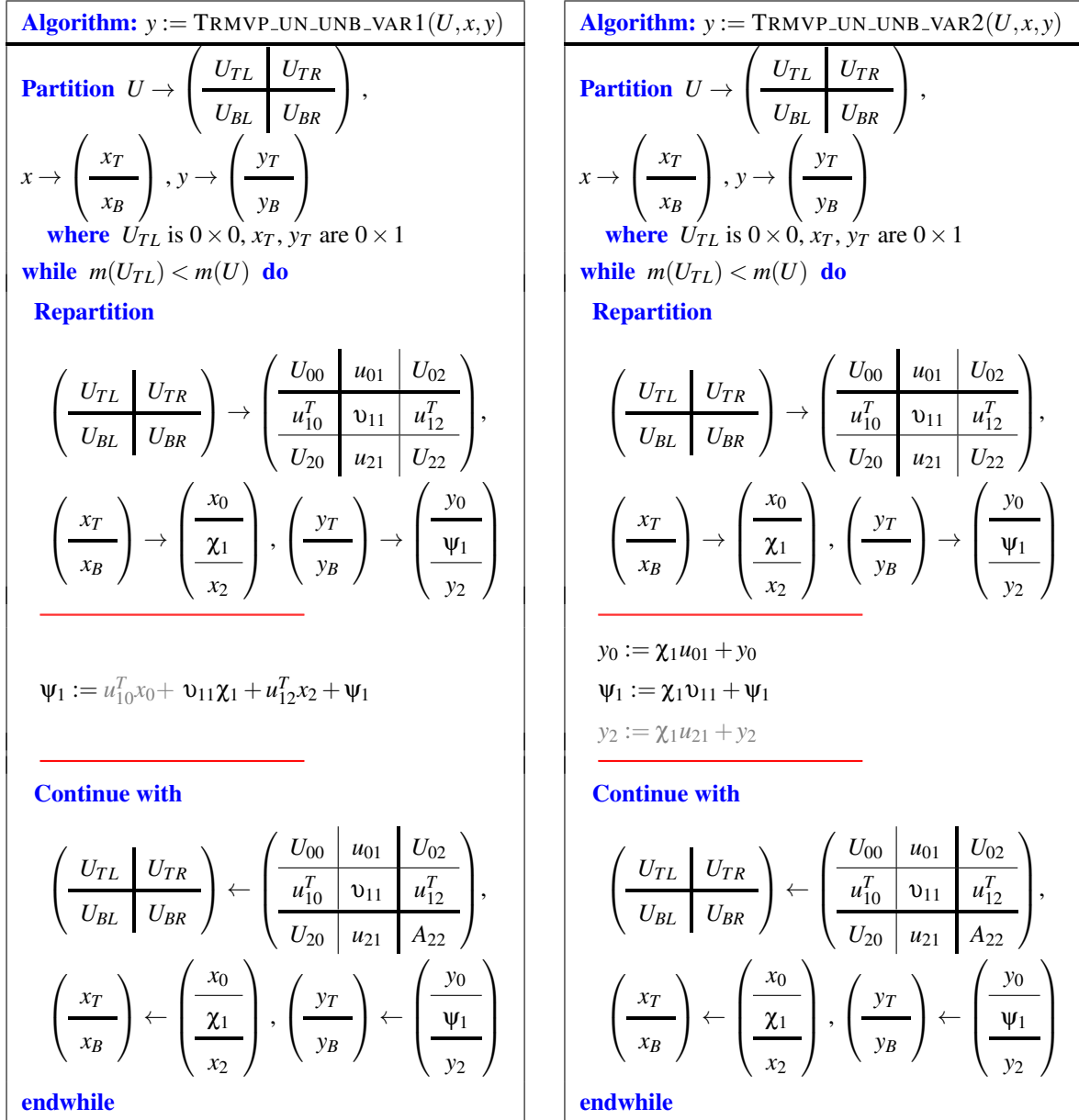
$$L \rightarrow \left(\begin{array}{c|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array} \right) = \left(\begin{array}{c|c|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array} \right),$$

where L_{TL} and L_{00} are square matrices, then

- $L_{TR} = 0$, $l_{01} = 0$, $L_{02} = 0$, and $l_{12}^T = 0$, where 0 indicates a matrix or vector of the appropriate dimensions.
- L_{TL} and L_{BR} are lower triangular matrices.

Algorithms

Let us start by focusing on $y := Ux + y$, where U is upper triangular. The algorithms from the previous section can be restated as in Figure 4.4, replacing A by U . Now, notice the parts in gray. Since $u_{10}^T = 0$ and $u_{21} = 0$, those computations need not be performed! Bingo, we have two algorithms that take advantage of the zeroes below the diagonal. We probably should explain the names of the routines:

Figure 4.4: Algorithms for computing $y := Ux + y$, where U is upper triangular.

TRMVP_UN_UNB_VAR1: Triangular matrix-vector multiply plus (y), with upper triangular matrix that is not transposed, unblocked variant 1.

(Yes, a bit convoluted, but such is life.)

Homework 4.3.2.1 Write routines

- `[y_out] = Trmvp_un_unb_var1 (U, x, y);` and
- `[y_out] = Trmvp_un_unb_var2 (U, x, y)`

that implement the algorithms in Figure 4.4 that compute $y := Ux + y$.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Homework 4.3.2.2 Modify the algorithms in Figure 4.5 so that they compute $y := Lx + y$, where L is a lower triangular matrix: (Just strike out the parts that evaluate to zero. We suggest you do this homework in conjunction with the next one.)

Homework 4.3.2.3 Write the functions

- `[y_out] = Trmvp_ln_unb_var1 (L, x, y);` and
- `[y_out] = Trmvp_ln_unb_var2 (L, x, y)`

that implement the algorithms for computing $y := Lx + y$ from Homework 4.3.2.2.

Homework 4.3.2.4 Modify the algorithms in Figure 4.6 to compute $x := Ux$, where U is an upper triangular matrix. **You may not use y . You have to overwrite x without using work space.** Hint: Think carefully about the order in which elements of x are computed and overwritten. You may want to do this exercise hand-in-hand with the implementation in the next homework.

Homework 4.3.2.5 Write routines

- `[x_out] = Trmv_un_unb_var1 (U, x);` and
- `[x_out] = Trmv_un_unb_var2 (U, x)`

that implement the algorithms for computing $x := Ux$ from Homework 4.3.2.4.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

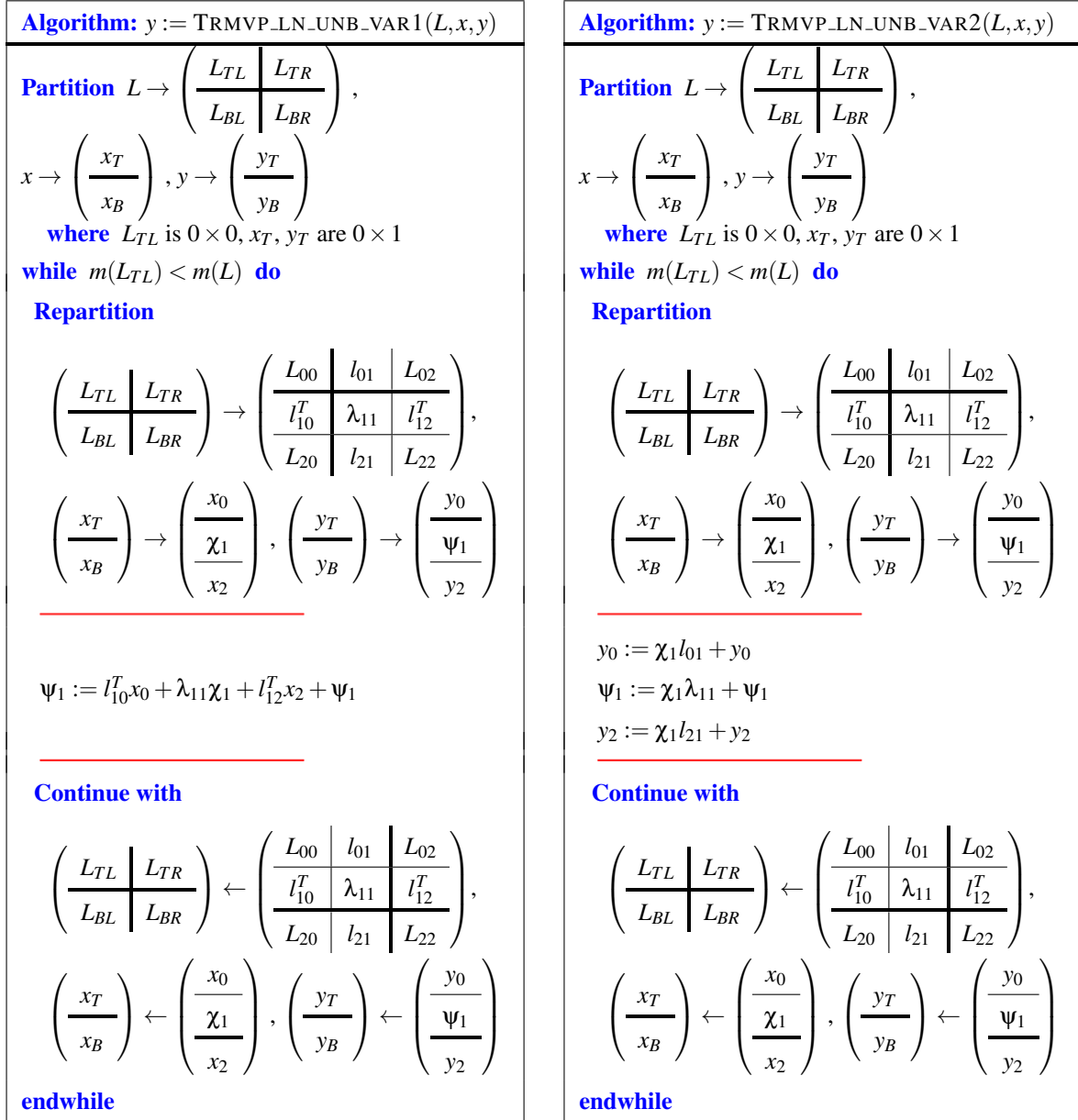


Figure 4.5: Algorithms to be used in Homework 4.3.2.2.

Homework 4.3.2.6 Modify the algorithms in Figure 4.7 to compute $x := Lx$, where L is a lower triangular matrix. **You may not use y . You have to overwrite x without using work space.** Hint: Think carefully about the order in which elements of x are computed and overwritten. This question is VERY tricky... You may want to do this exercise hand-in-hand with the implementation in the next homework.

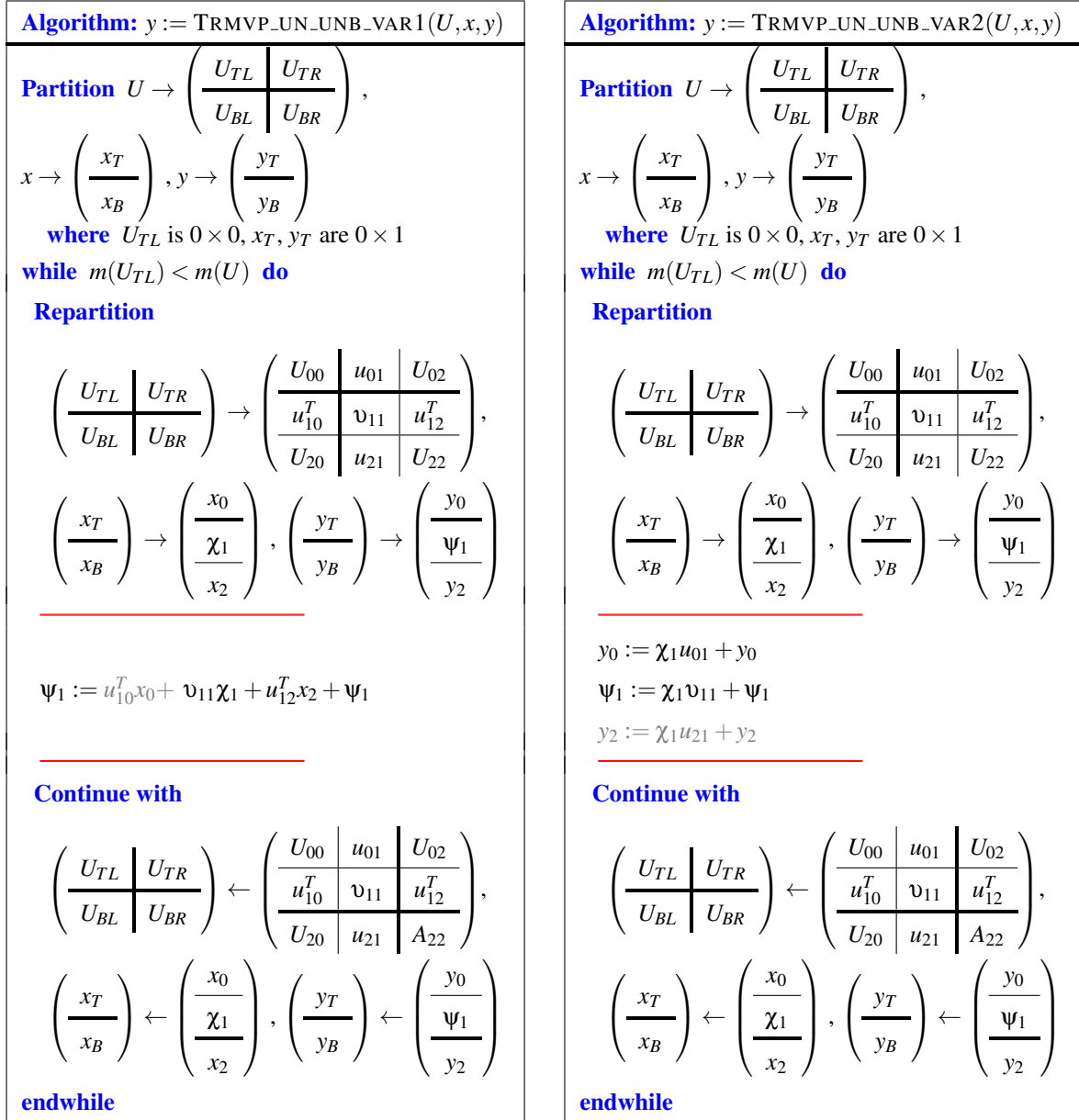


Figure 4.6: Algorithms to be used in Homework 4.3.2.4.

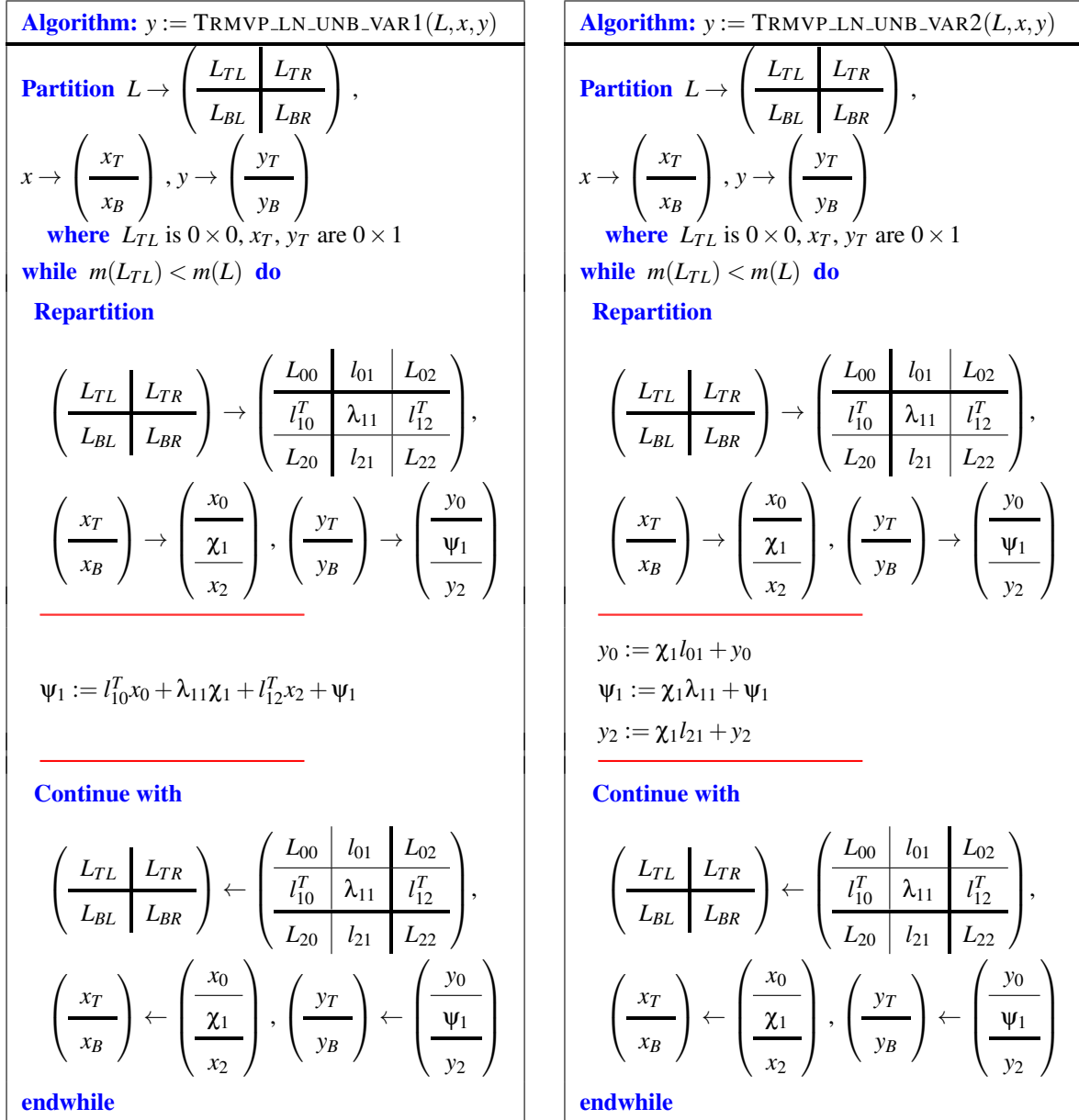


Figure 4.7: Algorithms to be used in Homework 4.3.2.6.

Homework 4.3.2.7 Write routines

- `[y_out] = Trmv_ln_unb_var1 (L, x);` and
- `[y_out] = Trmv_ln_unb_var2(L, x)`

that implement the algorithms from Homework 4.3.2.6 for computing $x := Lx$.
Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Homework 4.3.2.8 Develop algorithms for computing $y := U^T x + y$ and $y := L^T x + y$, where U and L are respectively upper triangular and lower triangular. Do not explicitly transpose matrices U and L . Write routines

- `[y_out] = Trmvp_ut_unb_var1 (U, x, y);` and
- `[y_out] = Trmvp_ut_unb_var2(U, x, y)`
- `[y_out] = Trmvp_lt_unb_var1 (L, x, y);` and
- `[y_out] = Trmvp_ln_unb_var2(L, x, y)`

that implement these algorithms.
Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Homework 4.3.2.9 Develop algorithms for computing $x := U^T x$ and $x := L^T x$, where U and L are respectively upper triangular and lower triangular. Do not explicitly transpose matrices U and L . Write routines

- `[y_out] = Trmv_ut_unb_var1 (U, x);` and
- `[y_out] = Trmv_ut_unb_var2(U, x)`
- `[y_out] = Trmv_lt_unb_var1 (L, x);` and
- `[y_out] = Trmv_ln_unb_var2(L, x)`

that implement these algorithms.
Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Cost

Let us analyze the algorithms for computing $y := Ux + y$. (The analysis of all the other algorithms is very similar.)

For the dot product based algorithm, the cost is in the update $\psi_1 := v_{11}x_1 + u_{12}^T x_2 + \psi_1$ which is typically computed in two steps:

- $\psi_1 := v_{11}x_1 + \psi_1$; followed by
- a dot product $\psi_1 := u_{12}^T x_2 + \psi_1$.

Now, during the first iteration, u_{12}^T and x_2 are of length $n - 1$, so that that iteration requires $2(n - 1) + 2 = 2n$ flops for the first step. During the k th iteration (starting with $k = 0$), u_{12}^T and x_2 are of length $(n - k - 1)$ so that the cost of that iteration is $2(n - k)$ flops. Thus, if A is an $n \times n$ matrix, then the total cost is given by

$$\sum_{k=0}^{n-1} [2(n - k)] = 2 \sum_{k=0}^{n-1} (n - k) = 2(n + (n - 1) + \cdots + 1) = 2 \sum_{k=1}^n k = 2(n + 1)n/2.$$

flops. (Recall that we proved in the second week that $\sum_{i=1}^n i = \frac{n(n+1)}{2}$.)

Homework 4.3.2.10 Compute the cost, in flops, of the algorithm for computing $y := Lx + y$ that uses AXPY s.

Homework 4.3.2.11 As hinted at before: Implementations achieve better performance (finish faster) if one accesses data consecutively in memory. Now, most scientific computing codes store matrices in “column-major order” which means that the first column of a matrix is stored consecutively in memory, then the second column, and so forth. Now, this means that an algorithm that accesses a matrix by columns tends to be faster than an algorithm that accesses a matrix by rows. That, in turn, means that when one is presented with more than one algorithm, one should pick the algorithm that accesses the matrix by columns.

Our FLAME notation makes it easy to recognize algorithms that access the matrix by columns. For example, in this unit, if the algorithm accesses submatrix a_{01} or a_{21} then it accesses columns. If it accesses submatrix a_{10}^T or a_{12}^T , then it accesses the matrix by rows.

For each of these, which algorithm accesses the matrix by columns:

- For $y := Ux + y$, TRSVP_UN_UNB_VAR1 or TRSVP_UN_UNB_VAR2?
Does the better algorithm use a dot or an axpy?
- For $y := Lx + y$, TRSVP_LN_UNB_VAR1 or TRSVP_LN_UNB_VAR2?
Does the better algorithm use a dot or an axpy?
- For $y := U^T x + y$, TRSVP_UT_UNB_VAR1 or TRSVP_UT_UNB_VAR2?
Does the better algorithm use a dot or an axpy?
- For $y := L^T x + y$, TRSVP_LT_UNB_VAR1 or TRSVP_LT_UNB_VAR2?
Does the better algorithm use a dot or an axpy?

4.3.3 Symmetric Matrix-Vector Multiplication

Motivation

Consider

$$\left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array} \right), = \left(\begin{array}{cc|cc|c} -1 & 2 & 4 & 1 & 0 \\ 2 & 0 & -1 & -2 & 1 \\ \hline 4 & -1 & 3 & 1 & 2 \\ \hline 1 & -2 & 1 & 4 & 3 \\ 0 & 1 & 2 & 3 & 2 \end{array} \right).$$

Here we purposely chose the matrix on the right to be symmetric. We notice that $a_{10}^T = a_{01}$, $A_{20}^T = A_{02}$, and $a_{12}^T = a_{21}$. A moment of reflection will convince you that this is a general principle, when A_{00} is square. Moreover, notice that A_{00} and A_{22} are then symmetric as well.

Theory

Consider

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

where A_{TL} and A_{00} are square matrices. If A is symmetric then

- A_{TL} , A_{BR} , A_{00} , and A_{22} are symmetric;
- $a_{10}^T = a_{01}^T$ and $a_{12}^T = a_{21}^T$; and
- $A_{20} = A_{02}^T$.

We will just state this as “intuitively obvious”.

Algorithms

Consider computing $y := Ax + y$ where A is a symmetric matrix. Since the upper and lower triangular part of a symmetric matrix are simply the transpose of each other, it is only necessary to store half the matrix: only the upper triangular part or only the lower triangular part. In Figure 4.8 we repeat the algorithms for matrix-vector multiplication from an earlier unit, and annotate them for the case where A is symmetric and only stored in the upper triangle. The change is simple: a_{10} and a_{21} are not stored and thus

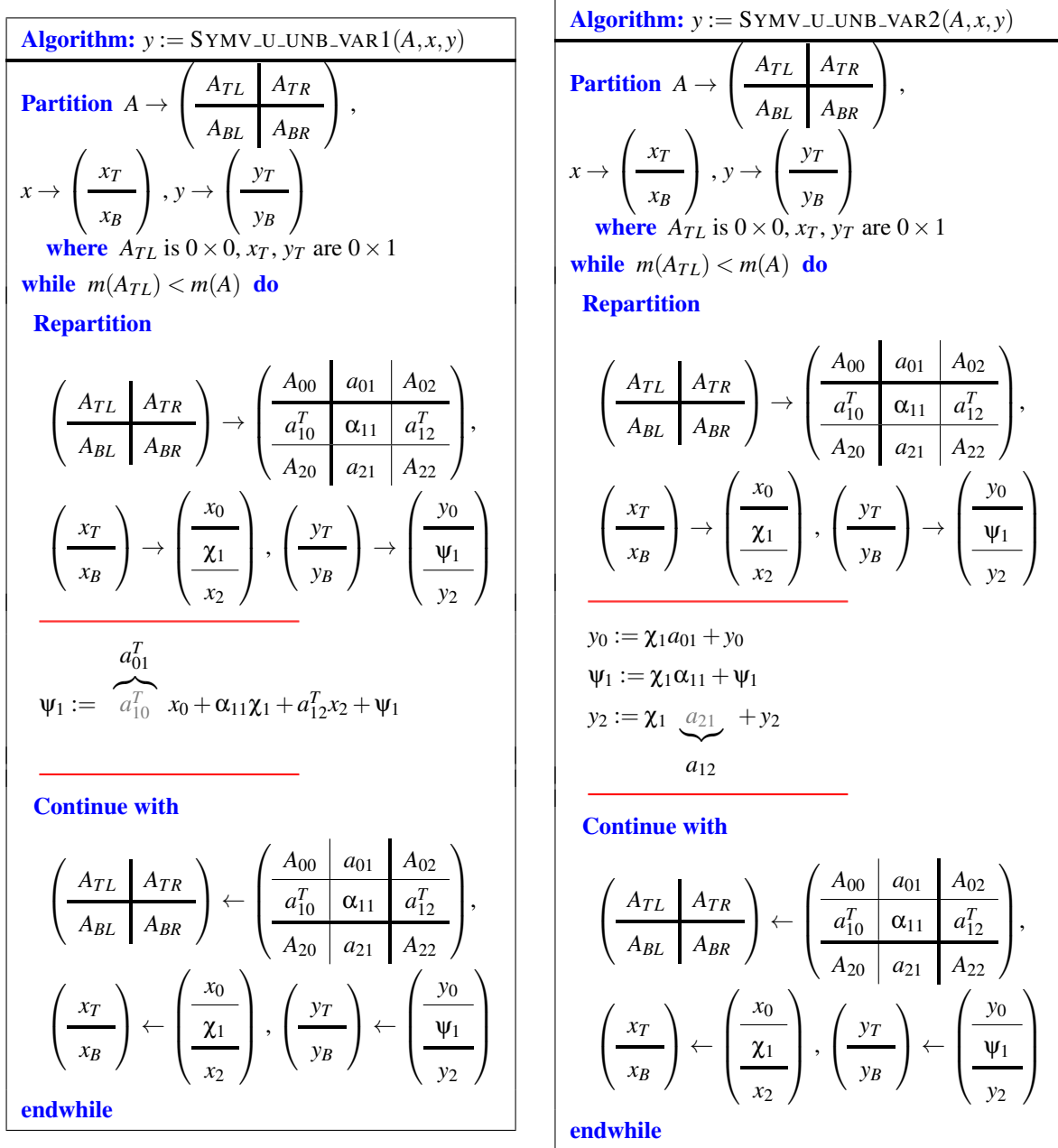


Figure 4.8: Algorithms for computing $y := Ax + y$ where A is symmetric, where only the upper triangular part of A is stored.

- For the left algorithm, the update $\psi_1 := a_{10}^T x_0 + \alpha_{11} \chi_1 + a_{12}^T x_2 + \psi_1$ must be changed to $\psi_1 := a_{01}^T x_0 + \alpha_{11} \chi_1 + a_{12}^T x_2 + \psi_1$.
- For the algorithm on the right, the update $y_2 := \chi_1 a_{21} + y_2$ must be changed to $y_2 := \chi_1 a_{12} + y_2$ (or, more precisely, $y_2 := \chi_1 (a_{12}^T)^T + y_2$ since a_{12}^T is the label for part of a row).

Homework 4.3.3.1 Write routines

- `[y_out] = Symv_u_unb_var1 (A, x, y);` and
- `[y_out] = Symv_u_unb_var2 (A, x, y)`

that implement the algorithms in Figure 4.8.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

Homework 4.3.3.2 Modify the algorithms in Figure 4.9 to compute $y := Ax + y$, where A is symmetric and stored in the lower triangular part of matrix. You may want to do this in conjunction with the next exercise.

Homework 4.3.3.3 Write routines

- `[y_out] = Symv_l_unb_var1 (A, x, y);` and
- `[y_out] = Symv_l_unb_var2 (A, x, y)`

that implement the algorithms from the previous homework.

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

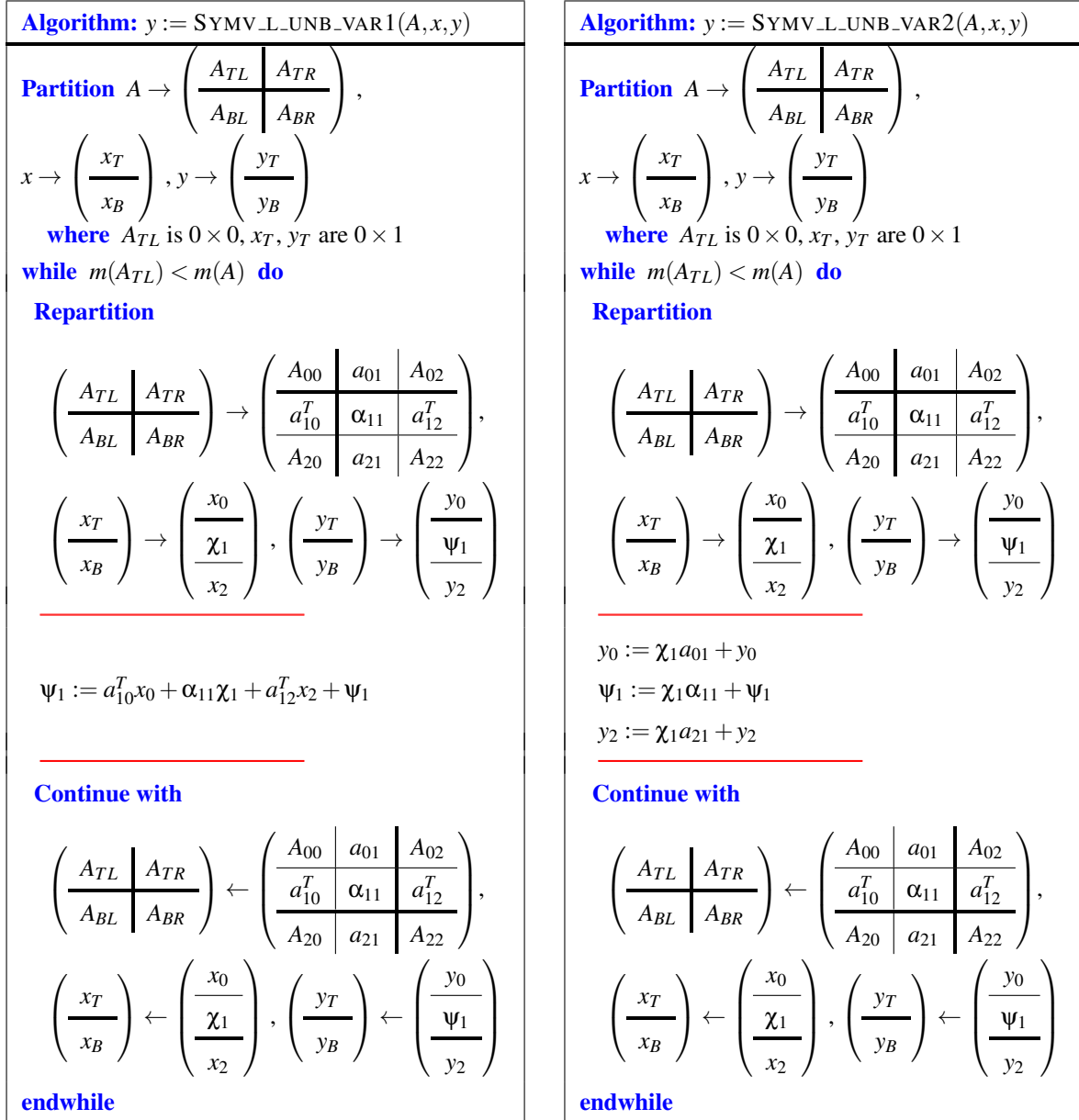


Figure 4.9: Algorithms for Homework 4.3.3.2

Homework 4.3.3.4 Challenge question! As hinted at before: Implementations achieve better performance (finish faster) if one accesses data consecutively in memory. Now, most scientific computing codes store matrices in “column-major order” which means that the first column of a matrix is stored consecutively in memory, then the second column, and so forth. Now, this means that an algorithm that accesses a matrix by columns tends to be faster than an algorithm that accesses a matrix by rows. That, in turn, means that when one is presented with more than one algorithm, one should pick the algorithm that accesses the matrix by columns. Our FLAME notation makes it easy to recognize algorithms that access the matrix by columns.

The problem with the algorithms in this unit is that all of them access both part of a row AND part of a column. So, your challenge is to devise an algorithm for computing $y := Ax + y$ where A is symmetric and only stored in one half of the matrix that only accesses parts of columns. We will call these “variant 3”. Then, write routines

- `[y_out] = Symv_u_unb_var3 (A, x, y);` and
- `[y_out] = Symv_l_unb_var3(A, x, y)`

Hint: (Let’s focus on the case where only the lower triangular part of A is stored.)

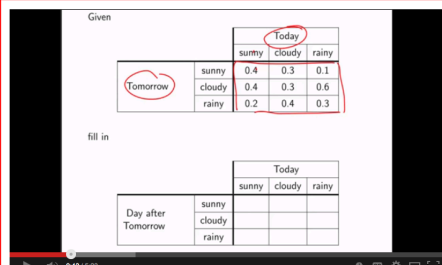
- If A is symmetric, then $A = L + \hat{L}^T$ where L is the lower triangular part of A and \hat{L} is the strictly lower triangular part of A .
- Identify an algorithm for $y := Lx + y$ that accesses matrix A by columns.
- Identify an algorithm for $y := \hat{L}^T x + y$ that accesses matrix A by columns.
- You now have two loops that together compute $y := Ax + y = (L + \hat{L}^T)x + y = Lx + \hat{L}^T x + y$.
- Can you “merge” the loops into one loop?

Some links that will come in handy:

-  [Spark](#). or  [Local Spark](#)
-  [PictureFLAME](#) or  [PictureFLAME](#)

4.4 Matrix-Matrix Multiplication (Product)

4.4.1 Motivation




Given

	Today			
	sunny	cloudy	rainy	
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

fill in

	Today			
	sunny	cloudy	rainy	
Day after Tomorrow	sunny			
	cloudy			
	rainy			

 [View at edX](#)

The first unit of the week, in which we discussed a simple model for prediction the weather, finished with the following exercise:

Given

		Today		
		sunny	cloudy	rainy
Tomorrow	sunny	0.4	0.3	0.1
	cloudy	0.4	0.3	0.6
	rainy	0.2	0.4	0.3

fill in the following table, which predicts the weather the day after tomorrow given the weather today:

		Today		
		sunny	cloudy	rainy
Day after Tomorrow	sunny			
	cloudy			
	rainy			

Now here is the hard part: Do so without using your knowledge about how to perform a matrix-matrix multiplication, since you won't learn about that until later this week...

The entries in the table turn out to be the entries in the transition matrix Q that was described just above the exercise:

$$\begin{aligned}
 \begin{pmatrix} \chi_s^{(2)} \\ \chi_c^{(2)} \\ \chi_r^{(2)} \end{pmatrix} &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(1)} \\ \chi_c^{(1)} \\ \chi_r^{(1)} \end{pmatrix} \\
 &= \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \left(\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix} \right) = Q \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix},
 \end{aligned}$$

Now, those of you who remembered from, for example, some other course that

$$\begin{aligned}
 &\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \left(\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix} \right) \\
 &= \left(\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \right) \begin{pmatrix} \chi_s^{(0)} \\ \chi_c^{(0)} \\ \chi_r^{(0)} \end{pmatrix}
 \end{aligned}$$

would recognize that

$$Q = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix}.$$

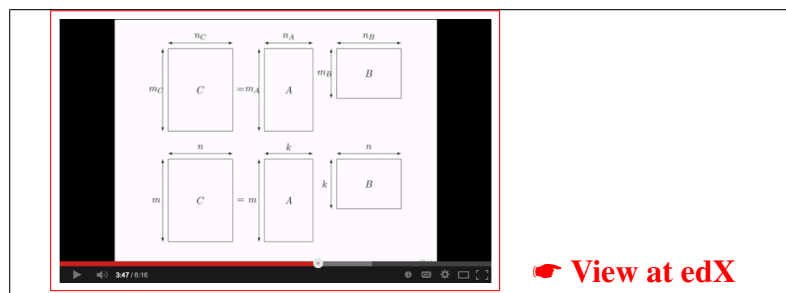
And, if you then remembered how to perform a matrix-matrix multiplication (or you did `P * P` in Python), you would have deduced that

$$Q = \begin{pmatrix} 0.3 & 0.25 & 0.25 \\ 0.4 & 0.45 & 0.4 \\ 0.3 & 0.3 & 0.35 \end{pmatrix}.$$

These then become the entries in the table. If you knew all the above, well, GOOD FOR YOU!

However, there are all kinds of issues that one really should discuss. How do you know such a matrix exists? Why is matrix-matrix multiplication defined this way? We answer that in the next few units.

4.4.2 From Composing Linear Transformations to Matrix-Matrix Multiplication



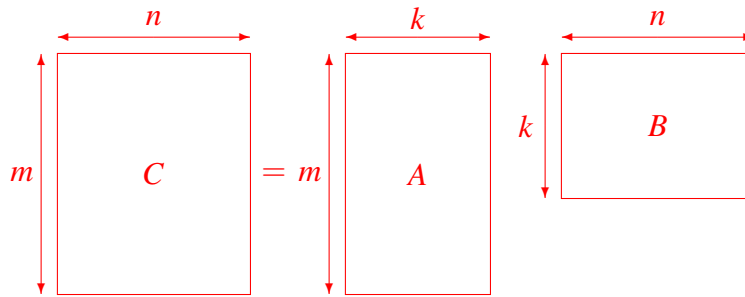
Homework 4.4.2.1 Let $L_A : \mathbb{R}^k \rightarrow \mathbb{R}^m$ and $L_B : \mathbb{R}^n \rightarrow \mathbb{R}^k$ both be linear transformations and, for all $x \in \mathbb{R}^n$, define the function $L_C : \mathbb{R}^n \rightarrow \mathbb{R}^m$ by $L_C(x) = L_A(L_B(x))$. $L_C(x)$ is a linear transformations.

Always/Sometimes/Never

Now, let linear transformations L_A , L_B , and L_C be represented by matrices $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$, and $C \in \mathbb{R}^{m \times n}$, respectively. (You know such matrices exist since L_A , L_B , and L_C are linear transformations.) Then $Cx = L_C(x) = L_A(L_B(x)) = A(Bx)$.

The matrix-matrix multiplication (product) is defined as the matrix C such that, for all vectors x , $Cx = A(B(x))$. The notation used to denote that matrix is $C = A \times B$ or, equivalently, $C = AB$. The operation AB is called a matrix-matrix multiplication or product.

If A is $m_A \times n_A$ matrix, B is $m_B \times n_B$ matrix, and C is $m_C \times n_C$ matrix, then for $C = AB$ to hold it must be the case that $m_C = m_A$, $n_C = n_B$, and $n_A = m_B$. Usually, the integers m and n are used for the sizes of C : $C \in \mathbb{R}^{m \times n}$ and k is used for the “other size”: $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$:



Homework 4.4.2.2 Let $A \in \mathbb{R}^{m \times n}$. $A^T A$ is well-defined. (By well-defined we mean that $A^T A$ makes sense. In this particular case this means that the dimensions of A^T and A are such that $A^T A$ can be computed.)

Always/Sometimes/Never

Homework 4.4.2.3 Let $A \in \mathbb{R}^{m \times n}$. AA^T is well-defined.

Always/Sometimes/Never

4.4.3 Computing the Matrix-Matrix Product

How to compute $C = AB$?

• $C = \begin{pmatrix} c_{0j} & c_{1j} & \dots & c_{m-1,j} \end{pmatrix}$ and $B = \begin{pmatrix} b_{0j} & b_{1j} & \dots & b_{n-1,j} \end{pmatrix}$.

• $c_j = C e_j = L_C(e_j) = L_A(L_B(e_j)) = A(B e_j) = A b_j$.

• $\begin{pmatrix} \gamma_{0,j} \\ \gamma_{1,j} \\ \vdots \\ \gamma_{m-1,j} \end{pmatrix} = \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix} b_j = \begin{pmatrix} \tilde{a}_0^T b_j \\ \tilde{a}_1^T b_j \\ \vdots \\ \tilde{a}_{m-1}^T b_j \end{pmatrix}$.

• $\gamma_{i,j} = \tilde{a}_i^T b_j = \begin{pmatrix} \alpha_{i,0} & \alpha_{i,1} & \dots & \alpha_{i,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,j} \\ \beta_{1,j} \\ \vdots \\ \beta_{k-1,j} \end{pmatrix} = \alpha_{i,0}\beta_{0,j} + \alpha_{i,1}\beta_{1,j} + \dots + \alpha_{i,k-1}\beta_{k-1,j} = \sum_{p=0}^{k-1} \alpha_{i,p}\beta_{p,j}$.

[View at edX](#)

The question now becomes how to compute C given matrices A and B . For this, we are going to use and abuse the unit basis vectors e_j .

Consider the following. Let

- $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$; and
- $C = AB$; and
- $L_C : \mathbb{R}^n \rightarrow \mathbb{R}^m$ equal the linear transformation such that $L_C(x) = Cx$; and
- $L_A : \mathbb{R}^k \rightarrow \mathbb{R}^m$ equal the linear transformation such that $L_A(x) = Ax$.
- $L_B : \mathbb{R}^n \rightarrow \mathbb{R}^k$ equal the linear transformation such that $L_B(x) = Bx$; and
- e_j denote the j th unit basis vector; and
- c_j denote the j th column of C ; and

- b_j denote the j th column of B .

Then

$$c_j = Ce_j = L_C(e_j) = L_A(L_B(e_j)) = L_A(Be_j) = L_A(b_j) = Ab_j.$$

From this we learn that

If $C = AB$ then the j th column of C , c_j , equals Ab_j , where b_j is the j th column of B .

Since by now you should be very comfortable with partitioning matrices by columns, we can summarize this as

$$\left(c_0 \mid c_1 \mid \cdots \mid c_{n-1} \right) = C = AB = A \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right) = \left(Ab_0 \mid Ab_1 \mid \cdots \mid Ab_{n-1} \right).$$

Now, let's expose the elements of C , A , and B .

$$C = \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{pmatrix}, \quad A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix},$$

$$\text{and } B = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{k-1,0} & \beta_{k-1,1} & \cdots & \beta_{k-1,n-1} \end{pmatrix}.$$

We are going to show that

$$\gamma_{i,j} = \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j},$$

which you may have learned in a high school algebra course.

We reasoned that $c_j = Ab_j$:

$$\begin{pmatrix} \gamma_{0,j} \\ \gamma_{1,j} \\ \vdots \\ \gamma_{i,j} \\ \vdots \\ \gamma_{m-1,j} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{i,0} & \alpha_{i,1} & \cdots & \alpha_{i,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,j} \\ \beta_{1,j} \\ \vdots \\ \beta_{k-1,j} \end{pmatrix}.$$

Here we highlight the i th element of c_j , $\gamma_{i,j}$, and the i th row of A . We recall that the i th element of Ax equals the dot product of the i th row of A with the vector x . Thus, $\gamma_{i,j}$ equals the dot product of the i th row of A with the vector b_j :

$$\gamma_{i,j} = \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j}.$$

Let $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$, and $C \in \mathbb{R}^{m \times n}$. Then the matrix-matrix multiplication (product) $C = AB$ is computed by

$$\gamma_{i,j} = \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j} = \alpha_{i,0} \beta_{0,j} + \alpha_{i,1} \beta_{1,j} + \cdots + \alpha_{i,k-1} \beta_{k-1,j}.$$

As a result of this definition $Cx = A(Bx) = (AB)x$ and can drop the parentheses, unless they are useful for clarity: $Cx = ABx$ and $C = AB$.

Homework 4.4.3.1 Compute

$$Q = P \times P = \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix}$$

We emphasize that for matrix-matrix multiplication to be a legal operations, the row and column dimensions of the matrices must obey certain constraints. Whenever we talk about dimensions being *conformal*, we mean that the dimensions are such that the encountered matrix multiplications are valid operations.

Homework 4.4.3.2 Let $A = \begin{pmatrix} 2 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & 3 & 1 \\ -1 & 1 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 2 & 1 & 2 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$. Compute

- $AB =$
- $BA =$

Homework 4.4.3.3 Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$ and $AB = BA$. A and B are square matrices.
Always/Sometimes/Never

Homework 4.4.3.4 Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$.

$$AB = BA.$$

Always/Sometimes/Never

Homework 4.4.3.5 Let $A, B \in \mathbb{R}^{n \times n}$. $AB = BA$.

Always/Sometimes/Never

Homework 4.4.3.6 A^2 is defined as AA . Similarly $A^k = \underbrace{AA \cdots A}_k$. Consistent with k occurrences of A

this, $A^0 = I$ so that $A^k = A^{k-1}A$ for $k > 0$.

A^k is well-defined only if A is a square matrix.

True/False

Homework 4.4.3.7 Let A, B, C be matrix “of appropriate size” so that $(AB)C$ is well defined. $A(BC)$ is well defined.

Always/Sometimes/Never

4.4.4 Special Shapes

We now show that if one treats scalars, column vectors, and row vectors as special cases of matrices, then many (all?) operations we encountered previously become simply special cases of matrix-matrix multiplication. In the below discussion, consider $C = AB$ where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$.

$m = n = k = 1$ (**scalar multiplication**)

$$1 \updownarrow \overset{1}{\boxed{C}} = 1 \updownarrow \overset{1}{\boxed{A}} \quad 1 \updownarrow \overset{1}{\boxed{B}}$$

In this case, all three matrices are actually scalars:

$$\begin{pmatrix} \gamma_{0,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} \end{pmatrix} \begin{pmatrix} \beta_{0,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} \beta_{0,0} \end{pmatrix}$$

so that matrix-matrix multiplication becomes scalar multiplication.

Homework 4.4.4.1 Let $A = \begin{pmatrix} 4 \end{pmatrix}$ and $B = \begin{pmatrix} 3 \end{pmatrix}$. Then $AB = \underline{\hspace{1cm}}$.

$n = 1, k = 1$ (SCAL)

$$\begin{array}{c}
 \begin{array}{|c|} \hline 1 \\ \hline \end{array} \\
 \updownarrow \\
 \begin{array}{|c|} \hline m \\ \hline \end{array}
 \end{array}
 C
 =
 \begin{array}{c}
 \begin{array}{|c|} \hline 1 \\ \hline \end{array} \\
 \updownarrow \\
 \begin{array}{|c|} \hline m \\ \hline \end{array}
 \end{array}
 A
 \quad
 \begin{array}{c}
 \begin{array}{|c|} \hline 1 \\ \hline \end{array} \\
 \updownarrow \\
 \begin{array}{|c|} \hline 1 \\ \hline \end{array}
 \end{array}
 B$$

Now the matrices look like

$$\begin{pmatrix} \gamma_{0,0} \\ \gamma_{1,0} \\ \vdots \\ \gamma_{m-1,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \vdots \\ \alpha_{m-1,0} \end{pmatrix} \begin{pmatrix} \beta_{0,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0}\beta_{0,0} \\ \alpha_{1,0}\beta_{0,0} \\ \vdots \\ \alpha_{m-1,0}\beta_{0,0} \end{pmatrix} = \begin{pmatrix} \beta_{0,0}\alpha_{0,0} \\ \beta_{0,0}\alpha_{1,0} \\ \vdots \\ \beta_{0,0}\alpha_{m-1,0} \end{pmatrix} = \beta_{0,0} \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \vdots \\ \alpha_{m-1,0} \end{pmatrix}.$$

In other words, C and A are vectors, B is a scalar, and the matrix-matrix multiplication becomes scaling of a vector.

Homework 4.4.4.2 Let $A = \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$ and $B = \begin{pmatrix} 4 \end{pmatrix}$. Then $AB =$.

Homework 4.4.4.3 This problem talks about IPython Notebooks and Python. It points out an interesting problem with the numpy package, which one can use with Python to do matrix computations. In MATLAB, the described behavior is not observed, so we can't create an equivalent homework for MATLAB. We left the problem here, because it points out interesting behavior when one considers a scalar to be a 11 matrix.

Start up a new IPython Notebook and try this:

```
import numpy as np

x = np.matrix( '1;2;3' )
print( x )

alpha = np.matrix( '-2' )
print( alpha )

print( x * alpha )
```

Notice how x , α , and $x * \alpha$ are created as matrices. Now try

```
print( alpha * x )
```

This causes an error! Why? Because numpy checks the sizes of matrices α and x and deduces that they don't match. Hence the operation is illegal. This is an artifact of how numpy is implemented.

Now, for us a 1×1 matrix and a scalar are one and the same thing, and that therefore $\alpha x = x\alpha$. Indeed, our `laff.scal` routine does just fine:

```
import laff
laff.scal( alpha, x )
print( x )
```

yields the desired result. This means that you can use the `laff.scal` routine for both update $x := \alpha x$ and $x := x\alpha$.

$m = 1, k = 1$ (SCAL)

$$1 \updownarrow \overbrace{\boxed{C}}^n = 1 \updownarrow \overbrace{\boxed{A}}^1 1 \updownarrow \overbrace{\boxed{B}}^n$$

Now the matrices look like

$$\begin{aligned} \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \end{pmatrix} &= \begin{pmatrix} \alpha_{0,0} \end{pmatrix} \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \end{pmatrix} \\ &= \alpha_{0,0} \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \end{pmatrix} \\ &= \begin{pmatrix} \alpha_{0,0}\beta_{0,0} & \alpha_{0,0}\beta_{0,1} & \cdots & \alpha_{0,0}\beta_{0,n-1} \end{pmatrix}. \end{aligned}$$

In other words, C and B are just row vectors and A is a scalar. The vector C is computed by scaling the row vector B by the scalar A .

Homework 4.4.4.4 Let $A = \begin{pmatrix} 4 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & -3 & 2 \end{pmatrix}$. Then $AB =$.

Homework 4.4.4.5 Like Homework 4.4.4.3, this problem talks about IPython Notebooks and Python. It points out an interesting problem with the numpy package, which one can use with Python to do matrix computations. In MATLAB, the described behavior is not observed, so we can't create an equivalent homework for MATLAB. We left the problem here, because it points out interesting behavior when one considers a scalar to be a 1×1 matrix.

Start up a new IPython Notebook and try this:

```
import numpy as np

xt = np.matrix( '1,2,3' )
print( xt )

alpha = np.matrix( '-2' )
print( alpha )

print( alpha * xt )
```

Again, notice how `xt`, `alpha`, and `alpha * xt` are create at matrices. Now try

```
print( xt * alpha )
```

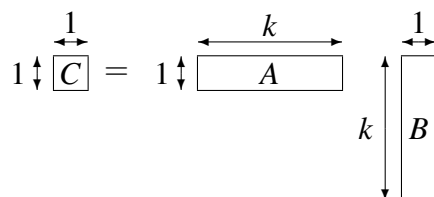
This causes an error! Why? Because numpy checks the sizes of matrices `alpha` and `x` and deduces that they don't match. Hence the operation is illegal. Again, this is an artifact of how numpy is implemented.

Now, for us a 1×1 matrix and a scalar are one and the same thing, and that therefore $\alpha x^T = x^T \alpha$. Indeed, our `laff.scal` routine does just fine:

```
import laff
laff.scal( alpha, xt )
print( xt )
```

yields the desired result. This means that you can use the `laff.scal` routine for both update $x^T := \alpha x^T$ and $x^T := x^T \alpha$.

$m = 1, n = 1$ (DOT)



The matrices look like

$$\begin{pmatrix} \gamma_{0,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,0} \\ \beta_{1,0} \\ \vdots \\ \beta_{k-1,0} \end{pmatrix} = \sum_{p=0}^{k-1} \alpha_{0,p} \beta_{p,0}.$$

In other words, C is a scalar that is computed by taking the dot product of the one row that is A and the one column that is B .

Homework 4.4.4.6 Let $A = \begin{pmatrix} 1 & -3 & 2 \end{pmatrix}$ and $B = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$. Then $AB =$

Homework 4.4.4.7 Try this in MATLAB:

```
>> xt = [ 1 2 3 ]

>> y = [
-1
0
2
]

>> xt * y

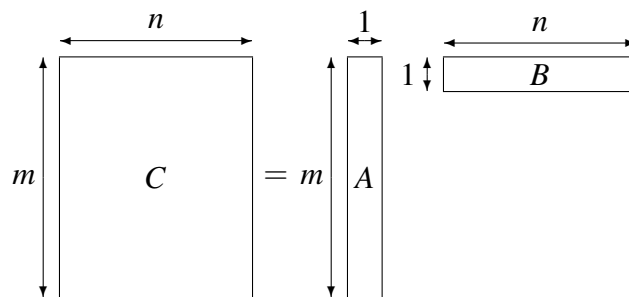
>> laff_dot( x, y )
```

The point is that

xt can be thought of as a 1×3 matrix or a row vector. y can be thought of as a 3×1 matrix or a column vector. $xt * y$ (matrix-matrix multiplication) computes the same as `laff_dot(x, y)`.

We prefer using our `laff_dot` and `laff_dots` routines, which don't care about whether x and y are rows or columns, making the adjustment automatically. This is in part because it explicitly tells us we are performing a dot product of two vectors, because of the names of the routines. In addition, when we use these routines in a code that uses the FLAME@lab API, we can use `PictureFLAME` to visualize the algorithm executing.

$k = 1$ (outer product)



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

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

Homework 4.4.4.8 Let $A = \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$ and $B = \begin{pmatrix} -1 & -2 \end{pmatrix}$. Then $AB =$

Homework 4.4.4.9 Let $a = \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix}$ and $b^T = \begin{pmatrix} -1 & -2 \end{pmatrix}$ and $C = ab^T$. Partition C by columns and by rows:

$$C = \left(c_0 \mid c_1 \right) \quad \text{and} \quad C = \begin{pmatrix} \tilde{c}_0^T \\ \tilde{c}_1^T \\ \tilde{c}_2^T \end{pmatrix}$$

Then

$$\bullet \quad c_0 = (-1) \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix} = \begin{pmatrix} (-1) \times (1) \\ (-1) \times (-3) \\ (-1) \times (2) \end{pmatrix} \quad \text{True/False}$$

$$\bullet \quad c_1 = (-2) \begin{pmatrix} 1 \\ -3 \\ 2 \end{pmatrix} = \begin{pmatrix} (-2) \times (1) \\ (-2) \times (-3) \\ (-2) \times (2) \end{pmatrix} \quad \text{True/False}$$

$$\bullet \quad C = \left(\begin{array}{c|c} (-1) \times (1) & (-2) \times (1) \\ (-1) \times (-3) & (-2) \times (-3) \\ (-1) \times (2) & (-2) \times (2) \end{array} \right) \quad \text{True/False}$$

$$\bullet \quad \tilde{c}_0^T = (1) \begin{pmatrix} -1 & -2 \end{pmatrix} = \begin{pmatrix} (1) \times (-1) & (1) \times (-2) \end{pmatrix} \quad \text{True/False}$$

$$\bullet \quad \tilde{c}_1^T = (-3) \begin{pmatrix} -1 & -2 \end{pmatrix} = \begin{pmatrix} (-3) \times (-1) & (-3) \times (-2) \end{pmatrix} \quad \text{True/False}$$

$$\bullet \quad \tilde{c}_2^T = (2) \begin{pmatrix} -1 & -2 \end{pmatrix} = \begin{pmatrix} (2) \times (-1) & (2) \times (-2) \end{pmatrix} \quad \text{True/False}$$

$$\bullet \quad C = \left(\begin{array}{cc} \frac{(-1) \times (1)}{(-1) \times (-3)} & \frac{(-2) \times (1)}{(-2) \times (-3)} \\ (-1) \times (2) & (-2) \times (2) \end{array} \right) \quad \text{True/False}$$

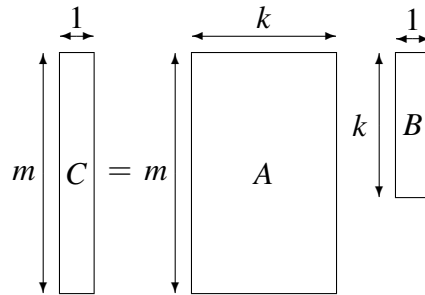
Homework 4.4.4.10 Fill in the boxes:

$$\begin{pmatrix} \square \\ \square \\ \square \\ \square \end{pmatrix} \begin{pmatrix} 2 & -1 & 3 \end{pmatrix} = \begin{pmatrix} 4 & \square & \square \\ -2 & \square & \square \\ 2 & \square & \square \\ 6 & \square & \square \end{pmatrix}$$

Homework 4.4.4.11 Fill in the boxes:

$$\begin{pmatrix} 2 \\ -1 \\ 1 \\ 3 \end{pmatrix} \begin{pmatrix} \square & \square & \square \end{pmatrix} = \begin{pmatrix} 4 & -2 & 6 \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix}$$

$n = 1$ (matrix-vector product)



$$\begin{pmatrix} \gamma_{0,0} \\ \gamma_{1,0} \\ \vdots \\ \gamma_{m-1,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,0} \\ \beta_{1,0} \\ \vdots \\ \beta_{k-1,0} \end{pmatrix}$$

We have studied this special case in great detail. To emphasize how it relates to have matrix-matrix multiplication is computed, consider the following:

$$\begin{pmatrix} \gamma_{0,0} \\ \vdots \\ \boxed{\gamma_{i,0}} \\ \vdots \\ \gamma_{m-1,0} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ \boxed{\alpha_{i,0} \quad \alpha_{i,1} \quad \cdots \quad \alpha_{i,k-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,0} \\ \beta_{1,0} \\ \vdots \\ \boxed{\beta_{k-1,0}} \end{pmatrix}$$

$m = 1$ (row vector-matrix product)

$$\begin{array}{c} \overleftarrow{n} \\ \boxed{C} \\ \overrightarrow{1} \end{array} = \begin{array}{c} \overleftarrow{k} \\ \boxed{A} \\ \overrightarrow{1} \end{array} \begin{array}{c} \overleftarrow{n} \\ \boxed{B} \\ \overrightarrow{k} \end{array}$$

$$\begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{k-1,0} & \beta_{k-1,1} & \cdots & \beta_{k-1,n-1} \end{pmatrix}$$

so that $\gamma_{0,j} = \sum_{p=0}^{k-1} \alpha_{0,p} \beta_{p,j}$. To emphasize how it relates to how matrix-matrix multiplication is computed, consider the following:

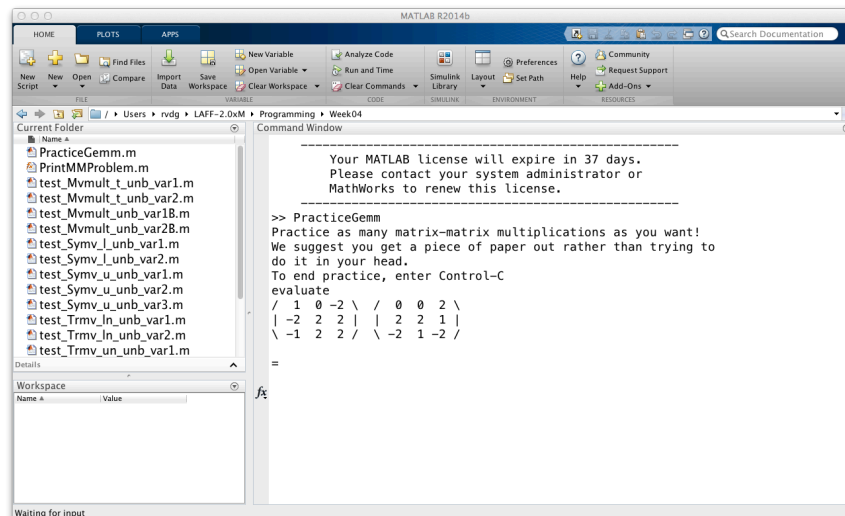
$$\begin{pmatrix} \gamma_{0,0} & \cdots & \boxed{\gamma_{0,j}} & \cdots & \gamma_{0,n-1} \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \end{pmatrix} \begin{pmatrix} \beta_{0,0} & \cdots & \boxed{\beta_{0,j}} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \cdots & \boxed{\beta_{1,j}} & \cdots & \beta_{1,n-1} \\ \vdots & & \boxed{\vdots} & & \vdots \\ \beta_{k-1,0} & \cdots & \boxed{\beta_{k-1,j}} & \cdots & \beta_{k-1,n-1} \end{pmatrix}.$$

Homework 4.4.4.12 Let $A = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & -2 & 2 \\ 4 & 2 & 0 \\ 1 & 2 & 3 \end{pmatrix}$. Then $AB =$

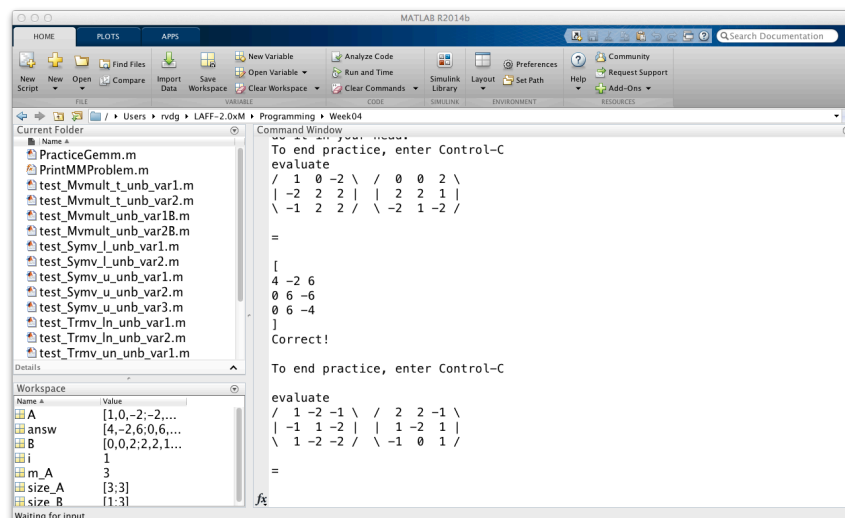
Homework 4.4.4.13 Let $e_i \in \mathbb{R}^m$ equal the i th unit basis vector and $A \in \mathbb{R}^{m \times n}$. Then $e_i^T A = \check{a}_i^T$, the i th row of A .

Always/Sometimes/Never

Homework 4.4.4.14 In Programming/Week04 you will find a script PracticeGemm. Execute it in the MATLAB Command Window to practice matrix-matrix multiplication. When you start, you will see something like

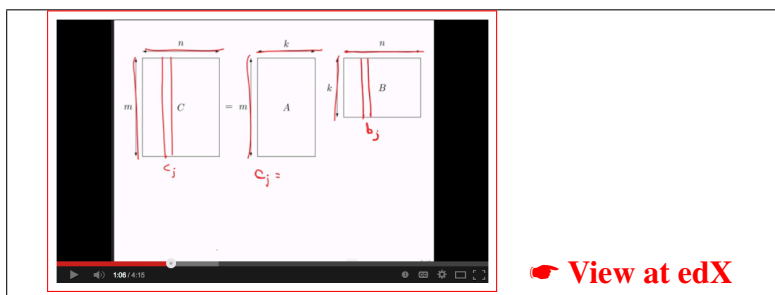


which you would answer like



If you understand how to perform a matrix-matrix multiplication, then you know how to perform all other operations with matrices and vectors that we have encountered so far.

4.4.5 Cost



Consider the matrix-matrix multiplication $C = AB$ where $C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$. Let us examine what the cost of this operation is:

- We argued that, by definition, the j th column of C , c_j , is computed by the matrix-vector multiplication Ab_j , where b_j is the j th column of B .
- Last week we learned that a matrix-vector multiplication of a $m \times k$ matrix times a vector of size k requires $2mk$ floating point operations (flops).
- C has n columns (since it is a $m \times n$ matrix.).

Putting all these observations together yields a cost of

$$n \times (2mk) = 2mnk \text{ flops.}$$

Try this! Recall that the dot product of two vectors of size k requires (approximately) $2k$ flops. We learned in the previous units that if $C = AB$ then $\gamma_{i,j}$ equals the dot product of the i th row of A and the j th column of B . Use this to give an alternative justification that a matrix multiplication requires $2mnk$ flops.

4.5 Enrichment

4.5.1 Markov Chains: Their Application

Matrices have many real world applications. As we have seen this week, one noteworthy use is connected to Markov chains. There are many, many examples of the use of Markov chains. You can find a brief look at some significant applications in **THE FIVE GREATEST APPLICATIONS OF MARKOV CHAINS** by Philipp von Hilgers and Amy N. Langville. (<http://langvillea.people.cofc.edu/MCapps7.pdf>).

4.6 Wrap Up

4.6.1 Homework

Homework 4.6.1.1 Let $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$. Then $(Ax)^T = x^T A^T$.

Always/Sometimes/Never

Homework 4.6.1.2 Our `laff` library has a routine

```
laff_gemv( trans, alpha, A, x, beta, y )
```

that has the following property

- `laff_gemv('No transpose', alpha, A, x, beta, y)` computes $y := \alpha Ax + \beta y$.
- `laff_gemv('Transpose', alpha, A, x, beta, y)` computes $y := \alpha A^T x + \beta y$.

The routine works regardless of whether x and/or y are column and/or row vectors.

Our library does NOT include a routine to compute $y^T := x^T A$. What call could you use to compute $y^T := x^T A$ if y^T is stored in `yt` and x^T in `xt`?

- `laff_gemv('No transpose', 1.0, A, xt, 0.0, yt)`.
- `laff_gemv('No transpose', 1.0, A, xt, 1.0, yt)`.
- `laff_gemv('Transpose', 1.0, A, xt, 1.0, yt)`.
- `laff_gemv('Transpose', 1.0, A, xt, 0.0, yt)`.

Homework 4.6.1.3 Let $A = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$. Compute

- $A^2 =$
- $A^3 =$
- For $k > 1$, $A^k =$

Homework 4.6.1.4 Let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

- $A^2 =$
- $A^3 =$
- For $n \geq 0$, $A^{2n} =$
- For $n \geq 0$, $A^{2n+1} =$

Homework 4.6.1.5 Let $A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.

- $A^2 =$
- $A^3 =$
- For $n \geq 0$, $A^{4n} =$
- For $n \geq 0$, $A^{4n+1} =$

Homework 4.6.1.6 Let A be a square matrix. If $AA = 0$ (the zero matrix) then A is a zero matrix. (AA is often written as A^2 .)

True/False

Homework 4.6.1.7 There exists a real valued matrix A such that $A^2 = -I$. (Recall: I is the identity)

True/False

Homework 4.6.1.8 There exists a matrix A that is not diagonal such that $A^2 = I$.

True/False

4.6.2 Summary

Partitioned matrix-vector multiplication

$$\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} A_{0,0}x_0 + A_{0,1}x_1 + \cdots + A_{0,N-1}x_{N-1} \\ A_{1,0}x_0 + A_{1,1}x_1 + \cdots + A_{1,N-1}x_{N-1} \\ \vdots \\ A_{M-1,0}x_0 + A_{M-1,1}x_1 + \cdots + A_{M-1,N-1}x_{N-1} \end{pmatrix}.$$

Transposing a partitioned matrix

$$\begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M-1,0} & A_{M-1,1} & \cdots & A_{M-1,N-1} \end{pmatrix}^T = \begin{pmatrix} A_{0,0}^T & A_{1,0}^T & \cdots & A_{M-1,0}^T \\ A_{0,1}^T & A_{1,1}^T & \cdots & A_{M-1,1}^T \\ \vdots & \vdots & \ddots & \vdots \\ A_{0,N-1}^T & A_{1,N-1}^T & \cdots & A_{M-1,N-1}^T \end{pmatrix}.$$

Composing linear transformations

Let $L_A : \mathbb{R}^k \rightarrow \mathbb{R}^m$ and $L_B : \mathbb{R}^n \rightarrow \mathbb{R}^k$ both be linear transformations and, for all $x \in \mathbb{R}^n$, define the function $L_C : \mathbb{R}^n \rightarrow \mathbb{R}^m$ by $L_C(x) = L_A(L_B(x))$. Then $L_C(x)$ is a linear transformations.

Matrix-matrix multiplication

$$AB = A \left(b_0 \mid b_1 \mid \cdots \mid b_{n-1} \right) = \left(Ab_0 \mid Ab_1 \mid \cdots \mid Ab_{n-1} \right).$$

If

$$C = \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,n-1} \end{pmatrix}, \quad A = \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,k-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,k-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \cdots & \alpha_{m-1,k-1} \end{pmatrix},$$

$$\text{and } B = \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \cdots & \beta_{0,n-1} \\ \beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{k-1,0} & \beta_{k-1,1} & \cdots & \beta_{k-1,n-1} \end{pmatrix}.$$

then $C = AB$ means that $\gamma_{i,j} = \sum_{p=0}^{k-1} \alpha_{i,p} \beta_{p,j}$.

A table of matrix-matrix multiplications with matrices of special shape is given at the end of this week.

Outer product

Let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. Then the *outer product* of x and y is given by xy^T . Notice that this yields an $m \times n$ matrix:

$$\begin{aligned} xy^T &= \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{m-1} \end{pmatrix} \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{n-1} \end{pmatrix}^T = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \vdots \\ \chi_{m-1} \end{pmatrix} \left(\psi_0 \mid \psi_1 \mid \cdots \mid \psi_{n-1} \right) \\ &= \begin{pmatrix} \chi_0\psi_0 & \chi_0\psi_1 & \cdots & \chi_0\psi_{n-1} \\ \chi_1\psi_0 & \chi_1\psi_1 & \cdots & \chi_1\psi_{n-1} \\ \vdots & \vdots & & \vdots \\ \chi_{m-1}\psi_0 & \chi_{m-1}\psi_1 & \cdots & \chi_{m-1}\psi_{n-1} \end{pmatrix}. \end{aligned}$$

m	n	k	Shape	Comment
1	1	1	$1 \updownarrow \boxed{C} = 1 \updownarrow \boxed{A} \quad 1 \updownarrow \boxed{B}$	Scalar multiplication
m	1	1	$m \updownarrow \boxed{C} = m \updownarrow \boxed{A} \quad 1 \updownarrow \boxed{B}$	Vector times scalar = scalar times vector
1	n	1	$1 \updownarrow \boxed{C} = 1 \updownarrow \boxed{A} \quad 1 \updownarrow \boxed{B}$	Scalar times row vector
1	1	k	$1 \updownarrow \boxed{C} = 1 \updownarrow \boxed{A} \quad k \updownarrow \boxed{B}$	Dot product (with row and column)
m	n	1	$m \updownarrow \boxed{C} = m \updownarrow \boxed{A} \quad 1 \updownarrow \boxed{B}$	Outer product
m	1	k	$m \updownarrow \boxed{C} = m \updownarrow \boxed{A} \quad k \updownarrow \boxed{B}$	Matrix-vector multiplication
1	n	k	$1 \updownarrow \boxed{C} = 1 \updownarrow \boxed{A} \quad k \updownarrow \boxed{B}$	Row vector times matrix multiply

LAFF routines

Operation Abbrev.	Definition	Function laff_	Approx. cost	
			flops	memops
Vector-vector operations				
Copy (COPY)	$y := x$	copy(x, y)	0	$2n$
Vector scaling (SCAL)	$x := \alpha x$	scal(alpha, x)	n	$2n$
Vector scaling (SCAL)	$x := x/\alpha$	invscal(alpha, x)	n	$2n$
Scaled addition (AXPY)	$y := \alpha x + y$	axpy(alpha, x, y)	$2n$	$3n$
Dot product (DOT)	$\alpha := x^T y$	alpha = dot(x, y)	$2n$	$2n$
Dot product (DOTS)	$\alpha := x^T y + \alpha$	dots(x, y, alpha)	$2n$	$2n$
Length (NORM2)	$\alpha := \ x\ _2$	alpha = norm2(x)	$2n$	n
Matrix-vector operations				
General matrix-vector	$y := \alpha Ax + \beta y$	gemv('No transpose', alpha, A, x, beta, y)	$2mn$	mn
multiplication (GEMV)	$y := \alpha A^T x + \beta y$	gemv('Transpose', alpha, A, x, beta, y)	$2mn$	mn
Rank-1 update (GER)	$A := \alpha xy^T + A$	ger(alpha, x, y, A)	$2mn$	mn

