**INTRODUCTION**

**Linear Algebra**

Linear algebra is a branch of mathematics, but the truth of it is that linear algebra is the mathematics of data. Matrices and vectors are the language of data.

Linear algebra is about linear combinations. That is, using arithmetic on columns of numbers called vectors and arrays of numbers called matrices, to create new columns and arrays of numbers. Linear algebra is the study of lines and planes, vector spaces and mappings that are required for linear transforms.

Linear algebra allows the analysis of [rotations](https://mathworld.wolfram.com/Rotation.html) in space, [least squares fitting](https://mathworld.wolfram.com/LeastSquaresFitting.html), solution of coupled differential equations, determination of a circle passing through three given points, as well as many other problems in mathematics, physics, and engineering.

**Polynomial Interpolation**

Polynomial interpolation is a method of estimating values between known data points. When graphical data contains a gap, but data is available on either side of the gap or at a few specific points within the gap, an estimate of values within the gap can be made by interpolation.

The simplest method of interpolation is to draw straight lines between the known data points and consider the function as the combination of those straight lines. This method, called linear interpolation, usually introduces considerable error. A more precise approach uses a polynomial function to connect the points. A [polynomial](https://whatis.techtarget.com/definition/polynomial) is a mathematical expression comprising a sum of terms, each term including a variable or variables raised to a power and multiplied by a [coefficient](https://whatis.techtarget.com/definition/coefficient). The simplest polynomials have one variable. Polynomials can exist in factored form or written out in full. For example:

The value of the largest exponent is called the degree of the polynomial.

If a set of data contains *n* known points, then there exists exactly one polynomial of degree *n*-1 or smaller that passes through all of those points. The polynomial's graph can be thought of as "filling in the curve" to account for data between the known points. This methodology, known as polynomial interpolation, often (but not always) provides more accurate results than linear interpolation.

**Applications of Polynomial Interpolation**

Polynomials can be used to approximate complicated curves, for example, the shapes of letters in [typography](https://en.wikipedia.org/wiki/Typography) (typography is the art of arranging letters and test in a way that makes the copy legible, clear, and visually appealing to the reader), given a few points. A relevant application is the evaluation of the [natural logarithm](https://en.wikipedia.org/wiki/Natural_logarithm) and [trigonometric functions](https://en.wikipedia.org/wiki/Trigonometric_function): pick a few known data points, create a [lookup table](https://en.wikipedia.org/wiki/Lookup_table), and interpolate between those data points. This results in significantly faster computations. Polynomial interpolation also forms the basis for algorithms in [numerical quadrature](https://en.wikipedia.org/wiki/Numerical_quadrature) and [numerical ordinary differential equations](https://en.wikipedia.org/wiki/Numerical_ordinary_differential_equations) and [Secure Multi Party Computation](https://en.wikipedia.org/wiki/Secure_multi-party_computation), [Secret Sharing](https://en.wikipedia.org/wiki/Secret_sharing) schemes.

Polynomial interpolation is also essential to perform sub-quadratic multiplication and squaring such as [Karatsuba multiplication](https://en.wikipedia.org/wiki/Karatsuba_multiplication) and [Toom–Cook multiplication](https://en.wikipedia.org/wiki/Toom%E2%80%93Cook_multiplication), where an interpolation through points on a polynomial which defines the product yields the product itself. For example, given

and

,

The product  is equivalent to . Finding points along  by substituting  for small values in and  yields points on the curve. Interpolation based on those points will yield the terms of  and subsequently the product . In the case of Karatsuba multiplication this technique is substantially faster than quadratic multiplication, even for modest-sized inputs. This is especially true when implemented in parallel hardware.

**Karatsuba’s Algorithm**

Multiplying two polynomials efficiently is an important issue in a variety of applications, including signal processing, cryptography and coding theory.

The **Karatsuba algorithm** is a fast multiplication algorithm that uses a [divide and conquer](https://brilliant.org/wiki/divide-and-conquer/) approach to [multiply](https://brilliant.org/wiki/multiplication/) two numbers. The standard procedure for multiplication of two n-digit numbers requires a number of elementary operations proportional to , or in big-O notation. Andrey Kolmogorov conjectured that the traditional algorithm was asymptotically optimal, meaning that any algorithm for that task would require Ω elementary operations.

While this algorithm, the Karatsuba’s Algorithm has a running time of Ɵ ~ Ɵ. Being able to multiply numbers quickly is very important. Computer scientists often consider multiplication to be a constant time operation, and this is a reasonable simplification for smaller numbers; but for larger numbers, the actual running times need to be factored in, which is . The point of the Karatsuba algorithm is to break large numbers down into smaller numbers so that any multiplications that occur happen on smaller numbers. Karatsuba can be used to multiply numbers in all base systems (base-10, base-2, etc.).

**Algorithm**

The basic step of Karatsuba's algorithm is a formula that allows one to compute the product of two large numbers {\displaystyle x}and {\displaystyle y} n ,mcxnjusing three multiplications of smaller numbers, each with about half as many digits as {\displaystyle x} or {\displaystyle y}, plus some additions and digit shifts.

Let and be represented as -digit strings in some base. For any positive integer less than , one can write two given numbers as

,

Where and are less than . The product is then

,

Where

,

,

These formulae require four multiplications and were known to [Charles Babbage](https://en.wikipedia.org/wiki/Charles_Babbage).

Karatsuba observed that can be computed in only three multiplications, at the cost of a few extra additions. With {\displaystyle z\_{0}}as before one can observe that

.

An issue that occurs, however, when computing is that the above computation of  and may result in overflow, which require a multiplier having one extra bit. This can be avoided by noting that

.

 This method may produce negative numbers, which require one extra bit to encode signedness, and would still require one extra bit for the multiplier. However, one way to avoid this is to record the sign and then use the absolute value of   
 and  to perform an unsigned multiplication, after which the result may be negated when both signs originally differed. Another advantage is that even though may be negative, the final computation of only involves additions.

If  is four or more, the three multiplications in Karatsuba's basic step involve operands with fewer than *n* digits. Therefore, those products can be computed by [recursive](https://en.wikipedia.org/wiki/Recursion) calls of the Karatsuba algorithm. The recursion can be applied until the numbers are so small that they can (or must) be computed directly.

**Efficiency analysis of Karatsuba’s Algorithm**

Karatsuba's basic step works for any base  and any, but the recursive algorithm is most efficient when  is equal to, rounded up. In particular, if  is, for some integer , and the recursion stops only when  is , then the number of single-digit multiplications is , which is  where  = .

Since one can extend any inputs with zero digits until their length is a power of two, it follows that the number of elementary multiplications, for any , is at most

.

Since the additions, subtractions, and digit shifts (multiplications by powers of ) in Karatsuba's basic step take time proportional to , their cost becomes negligible as  increases. More precisely, if denotes the total number of elementary operations that the algorithm performs when multiplying two -digit numbers, then

For some constants and . For this recurrence relation, the master theorem for divide and conquer references gives the asymptotic bound .

It follows that, for sufficiently large , Karatsuba's algorithm will perform fewer shifts and single-digit additions than longhand multiplication, even though its basic step uses more additions and shifts than the straightforward formula. For small values of *n*, however, the extra shift and add operations may make it run slower than the longhand method. The point of positive return depends on the [computer platform](https://en.wikipedia.org/wiki/Computer_platform) and context. As a rule of thumb, Karatsuba's method is usually faster when the multiplicands are longer than 320–640 bits.

**Karatsuba’s Algorithm and polynomial Interpolation**

**Input:** Two univariate polynomials:

+

**Output:**

On putting and , and evaluating at an appropriate power of 10, the model simplifies to Karatsuba’s algorithm.

We can also represent an - degree univariate polynomial as a list of evaluations:

Where all’s are distinct.

Note: There is a one-to-one correspondence between the two representations of, namely, as a list of coefficients and as a list of evaluations.

We identify by evaluation the process of getting the list of evaluations from the coefficient vector, and we identify by interpolation the other direction. We first detail the procedure of interpolation.

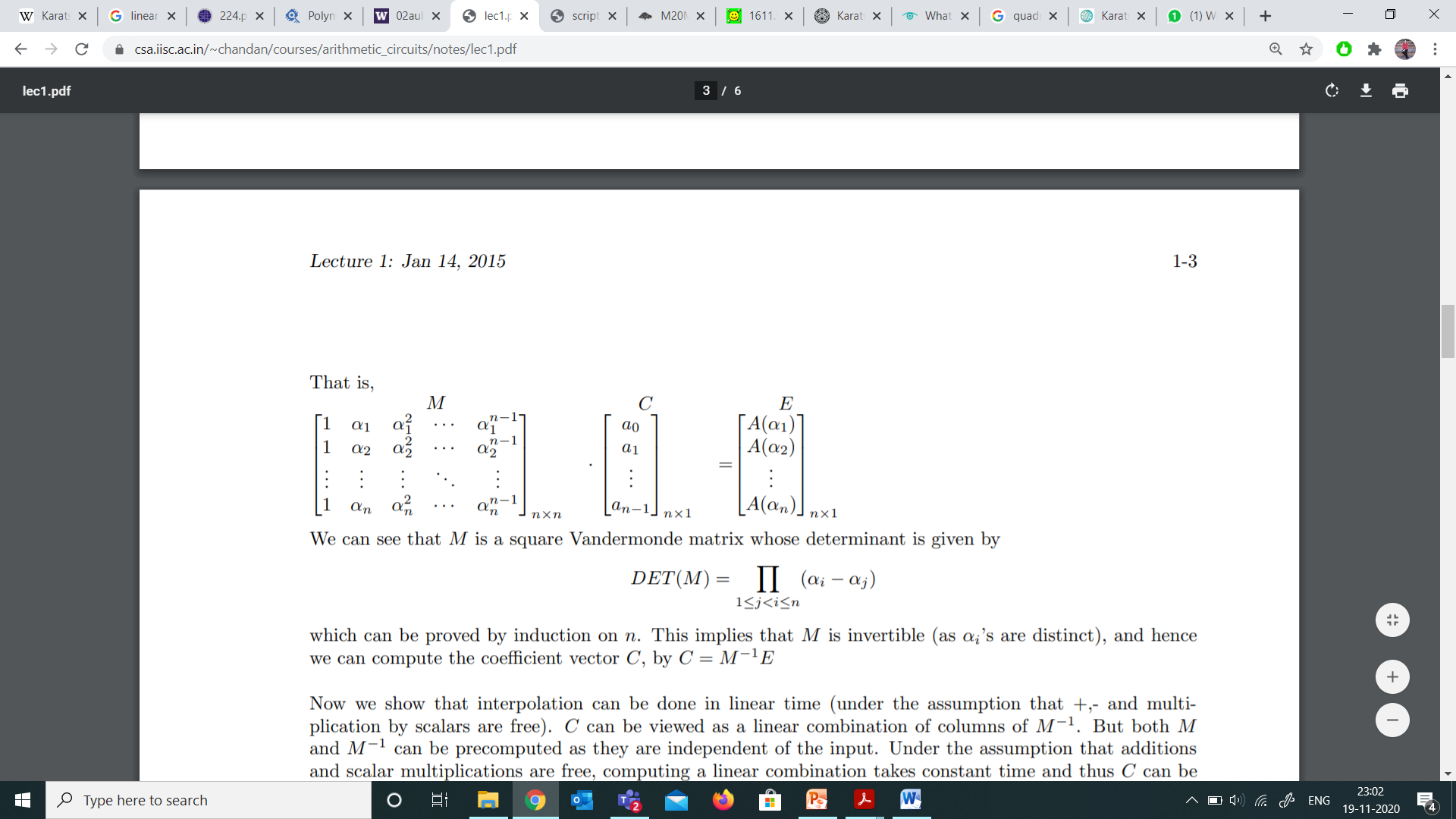
Given a polynomial evaluated at points , . . . , , suppose we need to find out its coefficients , . . . , . We just need to solve the following system of equations (where are unknowns).

.

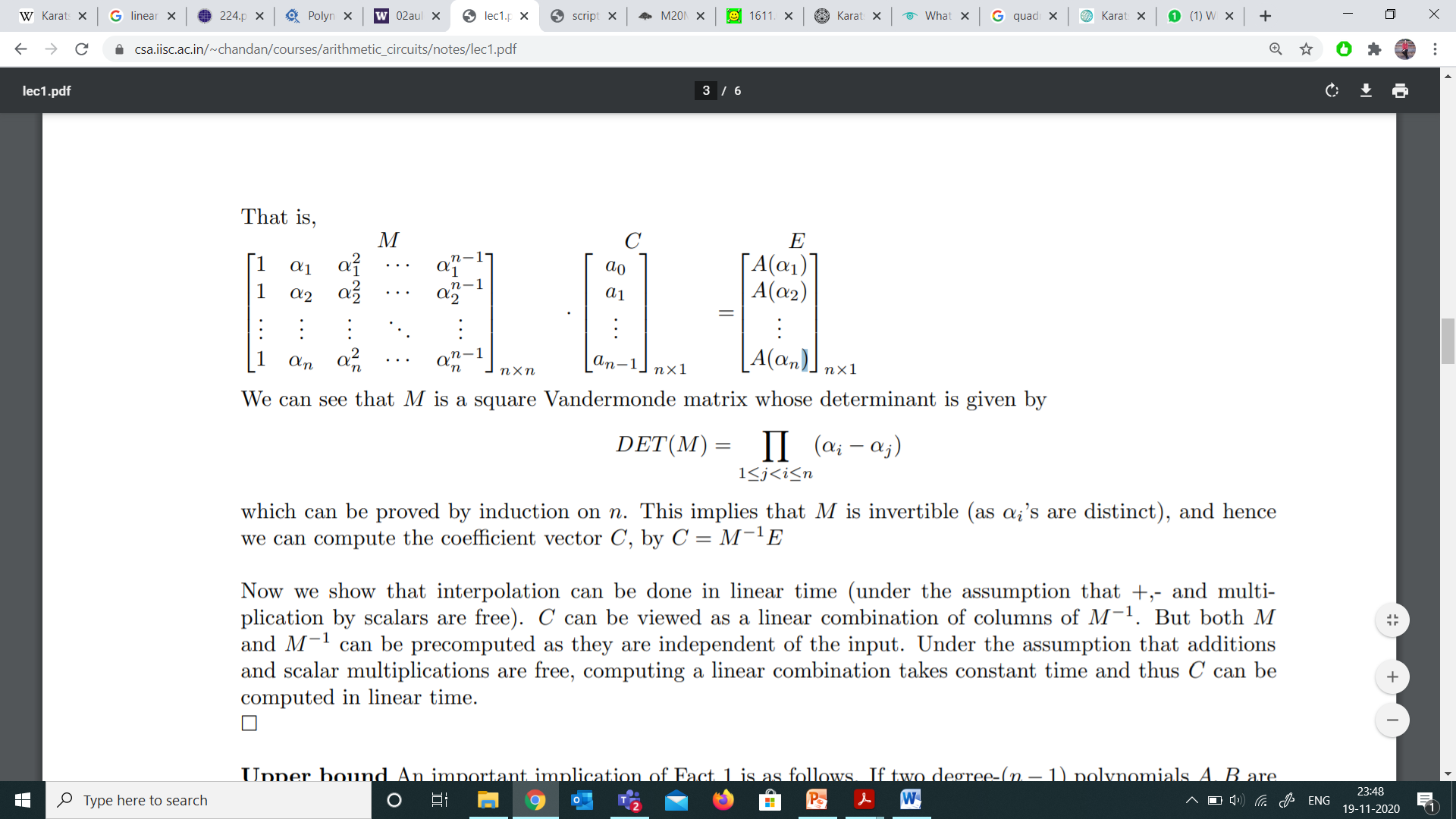
.

.

That is,



We can see that M is a square Vandermonde matrix whose determinant is given by



which can be proved by induction on . This implies that is invertible (as ’s are distinct), and hence we can compute the coefficient vector , by .

can be viewed as a linear combination of columns of . But both and can be precomputed as they are independent of the input. Under the assumption that additions and scalar multiplications are free, computing a linear combination takes constant time and thus can be computed in linear time.

Since there is a one-to-one correspondence between the two representations of, namely, as a list of coefficients and as a list of evaluations, if two degree- polynomials are given as evaluations, i.e.

Then the product polynomial can be given by the pointwise product of the evaluations, i.e.

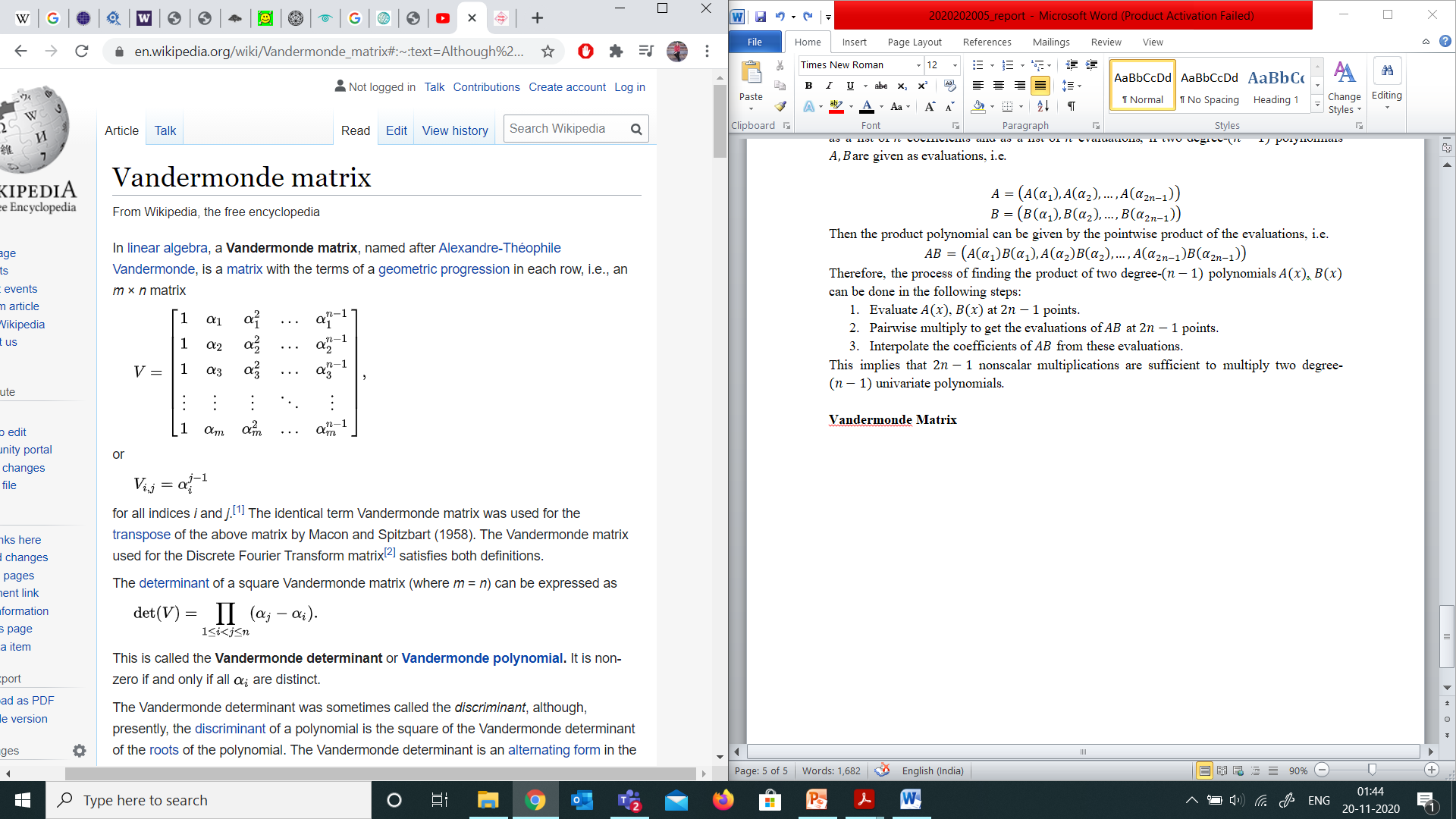
Therefore, the process of finding the product of two degree- polynomials, can be done in the following steps:

1. Evaluate, at points.
2. Pairwise multiply to get the evaluations of at points.
3. Interpolate the coefficients of from these evaluations.

This implies that nonscalar multiplications are sufficient to multiply two degree- univariate polynomials.

**Vandermonde Matrix**

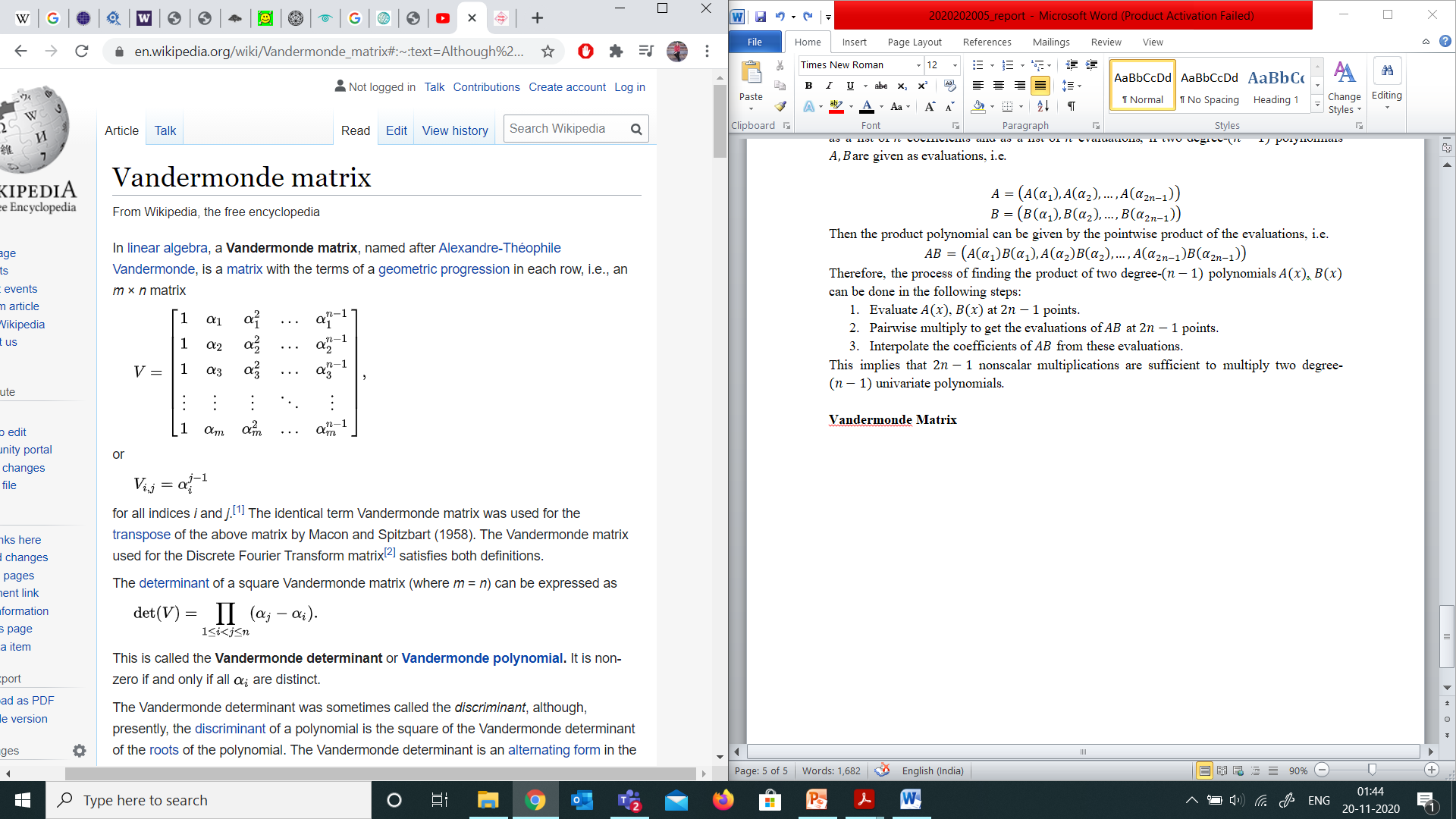
In [linear algebra](https://en.wikipedia.org/wiki/Linear_algebra), a **Vandermonde matrix**, named after [Alexandre-Théophile Vandermonde](https://en.wikipedia.org/wiki/Alexandre-Th%C3%A9ophile_Vandermonde), is a [matrix](https://en.wikipedia.org/wiki/Matrix_(math)) with the terms of a [geometric progression](https://en.wikipedia.org/wiki/Geometric_progression) in each row, i.e., an *m* × *n* matrix



Or

For all indices i and j.

The [determinant](https://en.wikipedia.org/wiki/Determinant) of a square Vandermonde matrix (where *m* = *n*) can be expressed as



This is called the **Vandermonde determinant** or [**Vandermonde polynomial**](https://en.wikipedia.org/wiki/Vandermonde_polynomial)**.** It is non-zero if and only if all {\displaystyle \alpha \_{i}} are distinct.

**Proof using linear maps**

Let be a field containing all and the vector space of the polynomials of degree less than with coefficients in . Let

Φ :

Be the linear map defined by

.

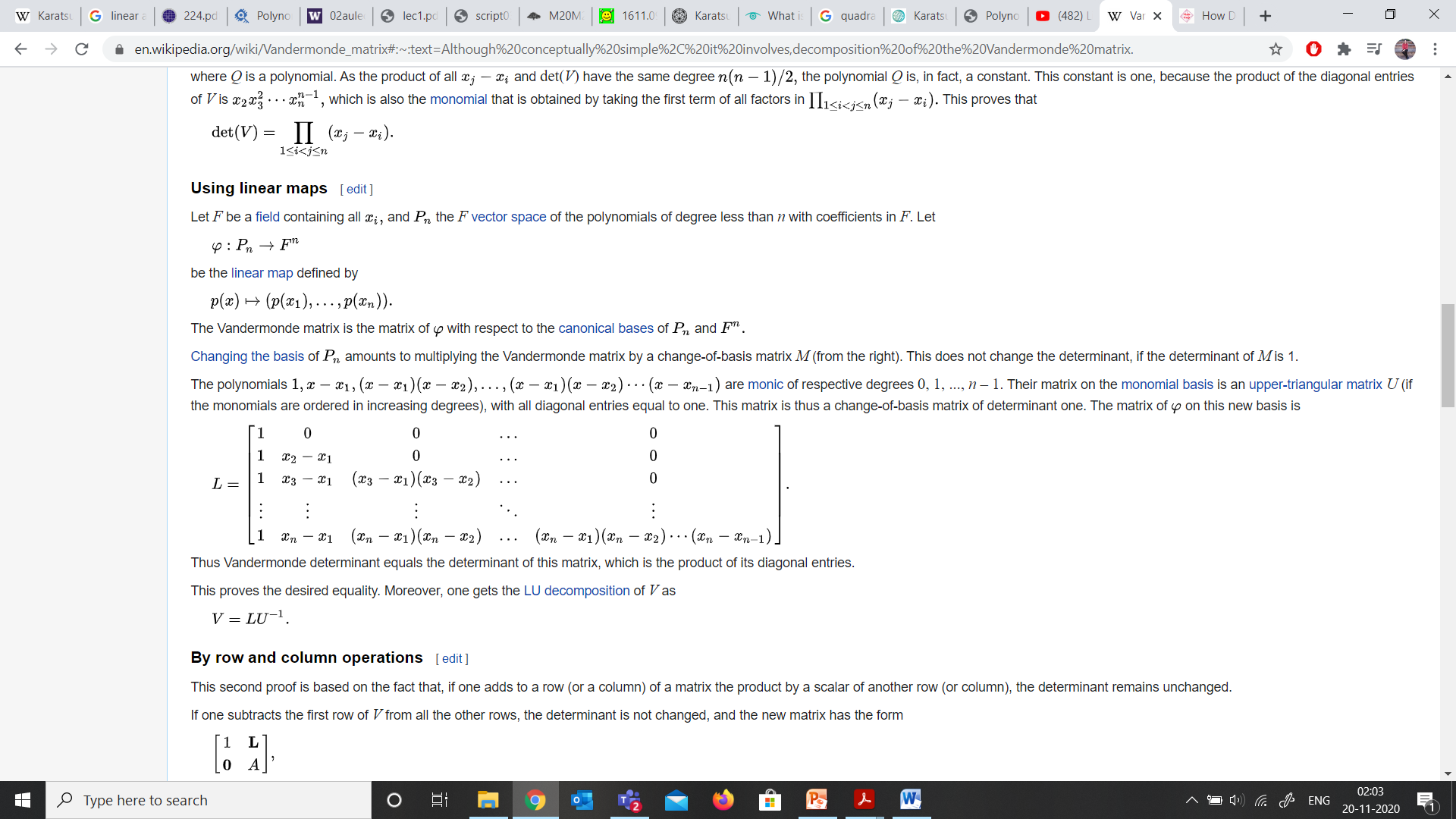
The Vandermonde matrix is the matrix of Φ {\displaystyle \varphi } with respect to the [canonical bases](https://en.wikipedia.org/wiki/Canonical_basis) of {\displaystyle P\_{n}} and {\displaystyle F^{n}.}.

[Changing the basis](https://en.wikipedia.org/wiki/Change_of_basis) of {\displaystyle P\_{n}} amounts to multiplying the Vandermonde matrix by a change-of-basis matrix (from the right). This does not change the determinant, if the determinant of  is 1.

The polynomials

 {\displaystyle 1,x-x\_{1},(x-x\_{1})(x-x\_{2}),\ldots ,(x-x\_{1})(x-x\_{2})\cdots (x-x\_{n-1})}

are [monic](https://en.wikipedia.org/wiki/Monic_polynomial) of respective degrees 0, 1, ..., *n* – 1. Their matrix on the [monomial basis](https://en.wikipedia.org/wiki/Monomial_basis) is an [upper-triangular matrix](https://en.wikipedia.org/wiki/Upper-triangular_matrix) (if the monomials are ordered in increasing degrees), with all diagonal entries equal to one. This matrix is thus a change-of-basis matrix of determinant one. The matrix of Φ {\displaystyle \varphi } on this new basis is



Thus Vandermonde determinant equals the determinant of this matrix, which is the product of its diagonal entries.

This proves the desired equality. Moreover, one gets the [LU decomposition](https://en.wikipedia.org/wiki/LU_decomposition) of  as

<https://en.wikipedia.org/wiki/Lagrange_polynomial>

<https://www.isical.ac.in/~arnabc/numana/interpol1.html>

<https://mathworld.wolfram.com/KaratsubaMultiplication.html>

<https://whatis.techtarget.com/definition/polynomial-interpolation>

<https://brilliant.org/wiki/karatsuba-algorithm/>

<https://arxiv.org/pdf/1611.09347.pdf>

<https://student.cs.uwaterloo.ca/~cs487/handouts/script03.pdf>

<https://www.csa.iisc.ac.in/~chandan/courses/arithmetic_circuits/notes/lec1.pdf>

<https://courses.cs.washington.edu/courses/cse417/02au/02aulecture10.pdf>

<http://cryptocode.net/docs/c44.pdf>

<https://eprint.iacr.org/2006/224.pdf>

https://en.wikipedia.org/wiki/Karatsuba\_algorithm