Gentle Introduction to Linear Algebra, with Spectacular Applications

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com

Version 2.2, June 2022

Abstract

This simple introduction to matrix theory offers a refreshing perspective on the subject. Using a basic concept that leads to a simple formula for the power of a matrix, I show how it can solve time series, Markov chains, linear regression, linear recurrence equations, pseudo-inverse and square root of a matrix, data compression, principal components analysis (PCA) or dimension reduction, and other machine learning problems. These problems are usually solved with more advanced matrix algebra, including eigenvalues, diagonalization, generalized inverse matrices, and other types of matrix normalization. My approach is more intuitive and thus appealing to professionals who do not have a strong mathematical background, or who have forgotten what they learned in math textbooks. It will also appeal to physicists and engineers, and to professionals more familiar or interested in calculus, than in matrix algebra. Finally, it leads to simple algorithms, for instance for matrix inversion. The classical statistician or data scientist will find my approach somewhat intriguing. The core of the methodology is the characteristic polynomial of a matrix, and in particular, the roots with lowest or largest moduli. It leads to a numerically stable method to solve Vandermonde systems, and thus, many linear algebra problems. Simulations include a curious fractal time series that looks incredibly smooth.

Contents

1 Power of a matrix

This is not a traditional tutorial on linear algebra. The material presented here, in a compact style, is rarely taught in college classes. It covers a wide range of topics, while avoiding excessive use of jargon or advanced math. The fundamental tool is the power of a matrix, and its byproduct, the characteristic polynomial. It can solve countless problems, as discussed later in this article, with illustrations. It has more to do with calculus, than matrix algebra.

For simplicity, in this section, I illustrate the methodology for a 2×2 matrix denoted as A. The generalization is straightforward. I provide a simple formula for the n-th power of A, where n is a positive integer. I then extend the formula to $n = -1$ (the most useful case) and to non-integer values of n. Using the notation

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \text{with } A^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} a_{n-1} & b_{n-1} \\ c_{n-1} & d_{n-1} \end{pmatrix},$$

we obtain

$$\begin{cases} a_n = a \cdot a_{n-1} + b \cdot c_{n-1} \\ b_n = a \cdot b_{n-1} + b \cdot d_{n-1} \\ c_n = c \cdot a_{n-1} + d \cdot c_{n-1} \\ d_n = c \cdot b_{n-1} + d \cdot d_{n-1} \end{cases}$$

Using elementary substitutions, this leads to the following system:

$$\begin{cases} a_n = (a+d) \cdot a_{n-1} - (ad-bc) \cdot a_{n-2} \\ b_n = (a+d) \cdot b_{n-1} - (ad-bc) \cdot b_{n-2} \\ c_n = (a+d) \cdot c_{n-1} - (ad-bc) \cdot c_{n-2} \\ d_n = (a+d) \cdot d_{n-1} - (ad-bc) \cdot d_{n-2} \end{cases}$$

We are dealing with identical linear homogeneous recurrence relations. Only the initial conditions corresponding to $n = 0$ and $n = 1$, are different for these four equations. The solution to such equations is obtained as follows [Wiki]. First, solve the quadratic equation

$$x^2 - (a+d)x + (ad-bc) = 0. \tag{1}$$

The two solutions r_1, r_2 are

$$r_1 = \frac{1}{2}\Big[a + d + \sqrt{(a+d)^2 - 4(ad-bc)}\Big],$$
$$r_2 = \frac{1}{2}\Big[a + d - \sqrt{(a+d)^2 - 4(ad-bc)}\Big].$$

If the quantity under the square root is negative, then the roots are complex numbers. The final solution depends on whether the roots are distinct or not:

$$A^n = \begin{cases} r_1^n Q_1 + r_2^n Q_2 & \text{if } r_1 \neq r_2, \\ r_1^n Q_1 + n r_1^{n-1} Q_2 & \text{if } r_1 = r_2, \end{cases} \tag{2}$$

with

$$\begin{cases} Q_1 = (A - r_2 I)/(r_1 - r_2) & \text{if } r_1 \neq r_2, \\ Q_2 = (A - r_1 I)/(r_2 - r_1) & \text{if } r_1 \neq r_2, \\ Q_1 = I & \text{if } r_1 = r_2, \\ Q_2 = A - r_1 I & \text{if } r_1 = r_2. \end{cases} \tag{3}$$

Here the symbol I represents the 2×2 identity matrix. The last four relationships were obtained by applying formula (2) to A^n, with $n = 0$ and $n = 1$. It is easy to prove (by recursion on n) that 2, together with 3, is the correct solution. If none of the roots is zero, then the formula are still valid for $n = -1$, and thus it can be used to compute the inverse of A.

2 Examples, Generalization, and Matrix Inversion

For a $p \times p$ matrix, the methodology generalizes as follows. The quadratic polynomial becomes a polynomial of degree p, known as the characteristic polynomial, and linked to the Cayley-Hamilton theorem [Wiki]. If its roots are distinct, we have

$$A^n = \sum_{k=1}^{p} r_k^n Q_k, \quad \text{with} \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_p \end{pmatrix} = V^{-1} \begin{pmatrix} I \\ A \\ \vdots \\ A^{p-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ r_1 & r_2 & \cdots & r_p \\ \vdots & \vdots & & \vdots \\ r_1^{p-1} & r_2^{p-1} & \cdots & r_p^{p-1} \end{pmatrix}^{-1} \begin{pmatrix} I \\ A \\ \vdots \\ A^{p-1} \end{pmatrix} \tag{4}$$

The matrix V is a Vandermonde matrix [Wiki], so there is an explicit formula to compute its inverse, see [Wiki]. For a fast algorithm for the computation of its inverse, see [11]. However, inverting V should be avoided as it is a numerically unstable procedure. Instead, approximations methods based on selected roots of the characteristic polynomial – those with highest or lowest moduli – are preferred. They are discussed in section 4.2.

The determinants of A and V are respectively equal to

$$|A| = (-1)^p r_1 r_2 \cdots r_p$$
$$|V| = \prod_{1\leq k<l\leq p} (r_k - r_l)$$

Note that the roots can be real or complex numbers, simple or multiple, or equal to zero. Usually the roots are ordered by decreasing modulus, that is

$$|r_1| \geq |r_2| \geq |r_3| \geq \cdots \geq |r_p|.$$

That way, a good approximation for A^n is obtained by using the first three or four roots if $n > 0$, and the last three or four roots if $n < 0$. In the context of linear regression, one of the main problems consists of inverting a matrix, that is, using $n = -1$ in formula (4). Working with first three roots only is equivalent to performing a principal component analysis (PCA) [Wiki] as well as PCA-induced dimension reduction. This technique can be used for data compression.

If some roots have a multiplicity higher than one, the formulas must be adjusted. The solution can be found by looking at how to solve an homogeneous linear recurrence equation. See theorem 4 in section 8.2 of "Math 55 Lecture Notes" [3], taught at Berkeley University.

2.1 Example with a non-invertible matrix

Even if A is non-invertible, some useful quantities can still be computed when $n = -1$, not unlike using a pseudo-inverse matrix [Wiki] in the generalized linear model [Wiki] in regression analysis. Let's look at the following example, using the methodology previously discussed:

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \Rightarrow A^n = 5^n \cdot \frac{1}{5}\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} + 0^n \cdot \frac{1}{5}\begin{pmatrix} 4 & -2 \\ -2 & 1 \end{pmatrix}.$$

Note that $0^n = 1$ if $n = 0$. The rightmost matrix attached to the second root 0 is of particular interest, and plays the role of a pseudo-inverse matrix for A. If that second root was very close to zero rather than exactly zero, then the term involving the rightmost matrix would largely dominate in the expression of A^n, when $n = -1$. At the limit, some ratios involving the (non-existent!) inverse of A still make sense. For instance:

- The sum of the elements of the inverse of A, divided by its trace, is $(4-2-2+1)/(4+1) = 1/5$.
- The arithmetic mean divided by the geometric mean of its elements, is $1/2$.

2.2 Fast computations

If n is large, one way to efficiently compute A^n is as follows. Let's say that $n = 100$. Do the following computations:

$$A^2 = A\cdot A, A^4 = A^2\cdot A^2, A^8 = A^4\cdot A^4, A^{16} = A^8\cdot A^8,$$
$$A^{32} = A^{16}\cdot A^{16}, A^{64} = A^{32}\cdot A^{32}, A^{100} = A^{64}\cdot A^{32}\cdot A^4.$$

This can be useful to quickly get an approximation of the largest root of the characteristic polynomial, by eliminating all but the first (largest) root r_1 in formula (4), and using $n = 100$. Once the first root has been found, it is easy to also get an approximation for the second one, and then for the third one. If instead, you are interested in approximating the smallest roots, you can proceed the other way around, by using the formula for A^n, with $n = -100$ this time. Note that $A^{-100} = (A^{-1})^{100}$.

2.3 Square root of a matrix

Formula (4) works not only for integer values of n, but can be extended to fractional arguments, such as $n = 1/2$. Here, I illustrate how it works on an example. Thus, I show how to compute the square root of a matrix [Wiki], using the methodology developed so far. This problem has many applications, especially when the matrix A is symmetric positive semidefinite [Wiki]: then it has only one symmetric positive semidefinite square root, denoted as $A^{1/2}$. See for instance my article on linear regression and synthetic data [6], especially section 2.1 entitled "correlation structure". The square root is needed to generate synthetic data with a pre-specified correlation matrix.

Now, using formula (4) with $n = 1/2$, let us compute the square root of the 2×2 matrix A defined as

$$A = \begin{pmatrix} 3 & 1 \\ 1 & 1 \end{pmatrix}.$$

Its characteristic polynomial, according to formula (1), is $x^2 - 4x + 2$. The roots are $r_1 = 1 + \sqrt{2}$ and $r_2 = 1 - \sqrt{2}$ and are both real and positive. According to formula (2), the matrices Q_1, Q_2 are respectively equal to

$$Q_1 = \frac{A - r_2 I}{r_1 - r_2} = \frac{1}{4} \cdot \begin{pmatrix} 2+\sqrt{2} & \sqrt{2} \\ \sqrt{2} & 2-\sqrt{2} \end{pmatrix}, \quad Q_2 = \frac{A - r_1 I}{r_2 - r_1} = \frac{1}{4} \cdot \begin{pmatrix} 2-\sqrt{2} & -\sqrt{2} \\ -\sqrt{2} & 2+\sqrt{2} \end{pmatrix}.$$

The matrix A has one positive definite square root, namely

$$A^{1/2} = \sqrt{r_1} \cdot Q_1 + \sqrt{r_2} \cdot Q_2 = \frac{1}{2} \begin{pmatrix} \sqrt{10+\sqrt{2}} & \sqrt{2-\sqrt{2}} \\ \sqrt{2-\sqrt{2}} & \sqrt{2+\sqrt{2}} \end{pmatrix}.$$

It is easy to show that $A^{1/2} \cdot A^{1/2} = A$. In higher dimensions, the Vandermonde system (4) is solved using the method described in section 4.2.

3 Application to Machine Learning problems

I discussed principal component analysis (PCA), data compression via PCA, and pseudo-inverse matrices in section 2. Here I focus on applications to time series, Markov chains, and linear regression.

3.1 Markov chains

A Markov chain [Wiki] is a particular type of time series or stochastic process. At iteration or time n, a system is in a particular state i with probability $P_n(i)$. The probability to move from state i at time n, to state j at time $n+1$ is called a transition probability and denoted as p_{ij}. It does not depend on n, but only on i and j. The Markov chain is governed by its initial conditions $P_0(i), 1 \leq i \leq p$, and the transition probability matrix denoted as A, containing the elements p_{ij}. The size of the transition matrix is $p \times p$, where p is the number of potential states in the system. Thus $1 \leq i, j \leq p$. As n tends to infinity, A^n and the whole system reaches an equilibrium distribution. This is because

- The characteristic polynomial attached to A has a root equal to 1.
- The absolute value of any root is less than or equal to 1.

3.2 Time series: auto-regressive processes

Auto-regressive processes (AR) [Wiki] represent another basic type of time series. Unlike Markov chains, the number of potential states is infinite, and a state can any real value, not just an integer. Yet the time is still discrete. Time-continuous AR processes such as Gaussian processes [Wiki], are not included in this discussion. An AR(p) process is defined as follows:

$$X_n = a_1 X_{n-1} + \cdots + a_p X_{n-p} + e_n.$$

Its characteristic polynomial is

$$x^p = a_1 x^{p-1} + a_2 x^{p-2} + \cdots + a_{p-1} x + a_p. \tag{5}$$

Here $\{e_n\}$ is a white noise process (typically uncorrelated Gaussian variables with same variance) [Wiki]. We assume that all expectations are zero. We are dealing here with a non-homogeneous linear (stochastic) recurrence relation. The most interesting case is when all the roots of the characteristic polynomial have absolute value less than 1. Processes satisfying this condition are called stationary. In that case, the auto-correlations are decaying exponentially fast.

The lag-k covariances satisfy the relation

$$\gamma_k = \text{Cov}[X_n, X_{n-k}] = \begin{cases} a_1\gamma(k-1) + \cdots + a_p\gamma(k-p) & \text{if } k \neq 0 \\ a_1\gamma(k-1) + \cdots + a_p\gamma(k-p) + \sigma^2 & \text{if } k = 0 \end{cases}$$

with

$$\sigma^2 = \text{Var}[e_n], \quad \text{Var}[X_n] = \gamma(0), \quad \rho(k) = \text{Corr}[X_n, X_{n-k}] = \gamma(k)/\gamma(0).$$

Thus the auto-correlations can be explicitly computed, and are also related to the characteristic polynomial. This fact can be used for model fitting, as the auto-correlation structure uniquely characterizes the (stationary) time series. Note that if the white noise is Gaussian, then the X_n's are also Gaussian.

More about the auto-correlation structure can be found in lecture notes from Marc-Andreas Muendler [9], who teaches economics at UCSD. His material is very similar to what I discuss here, but more comprehensive. See also Barbara Bogacka's lecture notes on time series [1], especially chapter 6. Finally, section 4 in this article explores the mathematical aspects in more details.

3.3 Linear regression

Linear regression problems can be solved using the ordinary least squares (OLS) method [Wiki]. The framework involves a response y, a data set X consisting of p features or variables and m observations, and p regression coefficients (to be determined) stored in a vector b. In matrix notation, the problem consists of finding b that minimizes the distance $||y - Xb||$ between y and Xb. Here X is an $m \times p$ matrix, and y, b are column vectors. The solution is

$$b = A^{-1}X^Ty, \quad \text{with } A = X^TX.$$

The techniques discussed in section 4.2 can be used to compute the inverse of A using formula (4) with $n = -1$, either exactly using all the roots of its characteristic polynomial, or approximately using the 2–3 roots with the lowest moduli. You need to use the recursion (8) backward when implementing the methodology in section 4.2, that is, for $n = -1, -2$ and so on, rather than for $n = 1, 2$ and so on.

If A is not invertible, the methodology described in section 2.1 can be useful: it amounts to working with a pseudo inverse of A. Note that A is a $p \times p$ matrix as in section 2. Questions regarding confidence intervals can be addressed using model-free techniques [5].

4 Mathematics of auto-regressive time series

Here I connect the dots between the auto-regressive time series described in section 3.2, and the material in section 2. For the AR(p) process in section 3.2, we have

$$X_n = g(e_p, e_{p+1}, \dots, e_n) + \sum_{k=1}^{p} r_k^n q_k, \quad \text{with} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_p \end{pmatrix} = V^{-1} \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{p-1} \end{pmatrix}$$

where V is the same matrix as in formula (4), the r_k's are the roots (assumed distinct here) of the characteristic polynomial defined by (5), and g is a linear function of $e_p, e_{p+1}, ..., e_n$. For instance, if $p = 1$, we have

$$g(e_p, e_{p+1}, \dots, e_n) = \sum_{k=0}^{n-p} a_1^k e_{n-k}.$$

This allows you to compute $\text{Var}[X_n]$ and $\text{Cov}[X_n, X_{n-k}]$, conditionally to $X_0, ..., X_{p-1}$. The limit, when n tends to infinity, allows you to compute the unconditional variance and auto-correlations attached to the process, in the stationary case. For instance, if $p = 1$, we have

$$\text{Var}[X_\infty] = \lim_{n\to\infty} \sum_{k=0}^{n-p} a_1^{2k} \text{Var}[e_{n-k}] = \sigma^2 \sum_{k=0}^{\infty} a_1^{2k} = \frac{\sigma^2}{1 - a_1^2}$$

where $\sigma^2 = \text{Var}[e_0]$ is the variance of the white noise $\{e_n\}$, and $|a_1| < 1$ because we assumed stationarity. For the general case (any p) the formula, if n is larger than or equal to p, is

$$g(e_p, e_{p+1}, \dots, e_n) = \sum_{k=0}^{n-p} \alpha_k e_{n-k}, \text{ with } \alpha_0 = 1, \ \alpha_k = \sum_{t=1}^{p} a_t \alpha_{k-t} \text{ if } k > 0, \ \alpha_k = 0 \text{ if } k < 0.$$

In this case, we have

$$\text{Var}[X_\infty] = \sigma^2 \sum_{k=0}^{\infty} \alpha_k^2.$$

The initial conditions for the coefficients α_k correspond to $k = 0, -1, -2, ..., -(p-1)$. Thus $\alpha_0 = 1$, and the remaining α_k's ($k < 0$) are zero. Thus, the recurrence relation for α_n can be solved using the same roots $r_1, \dots, r_p$ solution of equation (5). Assuming the roots are distinct, we have

$$\alpha_n = \sum_{k=1}^{p} r_k^n q_k', \quad \text{with} \begin{pmatrix} q_1' \\ q_2' \\ \vdots \\ q_p' \end{pmatrix} = W^{-1} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \text{and } W = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ r_1^{-1} & r_2^{-1} & \cdots & r_p^{-1} \\ \vdots & \vdots & & \vdots \\ r_1^{-(p-1)} & r_2^{-(p-1)} & \cdots & r_p^{-(p-1)} \end{pmatrix}. \tag{6}$$

Again, a direct computation of W^{-1} is numerically unstable, except in some special cases. Typically, a few of the r_k's are dominant. The other ones can be ignored, leading to approximations and increased stability. Finally, if two time series models, say an ARMA and an AR models, have the same variance and covariance structure, they are actually identical.

4.1 Simulations: curious fractal time series

To simulate the auto-regressive time series described in section 3.2, I proceed backwards: first, pick up the roots $r_1, \dots, r_p$ of the characteristic polynomial, then compute the coefficients $a_1, \dots, a_p$. This leads to interesting discoveries. I want at least one of the roots to have its modulus equal to one. The other roots must have a modulus strictly smaller than one. Roots can have a multiplicity greater than one. Depending on the roots, whether they are all real, whether some are complex, or whether some have a multiplicity greater than one, the pattern is very different. It falls into three categories:

- Type 1: A very smooth time series if the root(s) with largest modulus has a mutiplicity greater than one.
- Type 2: A Brownian-like appearance if all the roots are real and distinct.
- Type 3: An highly oscillating time series that fills a dense area when complex roots with modulus equal to one, are present.

In Figure 1, I have re-scaled the horizontal and vertical axes so that the time series look time-continuous. With this transformation, Type 2 actually corresponds to a 1-D Brownian motion [Wiki]. Type 3 has a curve that exhibits a fractal dimension [Wiki] strictly between 1 and 2. Type 1 looks very smooth. It seems like it has derivatives of any order, everywhere. Thus it can not be a Brownian motion. Yet if you zoom in or out, the same statistical properties (smoothness, expected numbers of bumps and so on) repeat themselves. This means that we are dealing with a strange mathematical function: one that can not be approximated by a Taylor series, yet very smooth and chaotic with self-replicating features, at the same time. Also, we need to keep in mind that it is a stochastic function.

The smoothness of a time series is typically measured using its Hurst exponent [Wiki]. The extreme, most chaotic case is a white noise, and the "middle case" is a Brownian motion. Here Type 3 seems more extreme than a white noise, and Type 1 is unusually smooth. Except for Type 3, the discrete version of these time series all have long-range auto-correlations.

4.1.1 White noise: Fréchet, Weibull and exponential cases

I use one of the simplest distributions to sample from, to generate the white noise $\{e_n\}$. Using independent uniform deviates $U_1, U_2, \dots$ on $[0, 1]$, I first generate the deviates

$$v_n = \tau\Big(-\log(1-U_n)\Big)^{\gamma}, \quad n = 1, 2, \dots \tag{7}$$

Then the noise e_n is produced using

$$e_n = v_n - \text{E}[v_n], \quad \text{with } \text{E}[v_n] = \tau\Gamma(1+\gamma) \text{ and } \text{Var}[e_n] = \text{Var}[v_n] = \tau^2\Big[\Gamma(1+2\gamma) - \Gamma^2(1+\gamma)\Big].$$

Here τ, γ are parameters, with $\gamma > -\frac{1}{2}$, and Γ is the Gamma function [Wiki]. We have the following cases:

- If $\gamma = 1$, then v_n has an exponential distribution.
- If $-1 < \gamma < 0$, then v_n has a Fréchet distribution. If in addition, $\gamma > -\frac{1}{2}$, then its variance is finite.
- If $\gamma > 0$, then v_n has a Weibull distribution, with finite variance.

It is surprising that this distribution has different names, depending on whether $\gamma < 0$ or $\gamma > 0$. This distribution is discussed in my book on stochastic processes [7], pages 41–42.

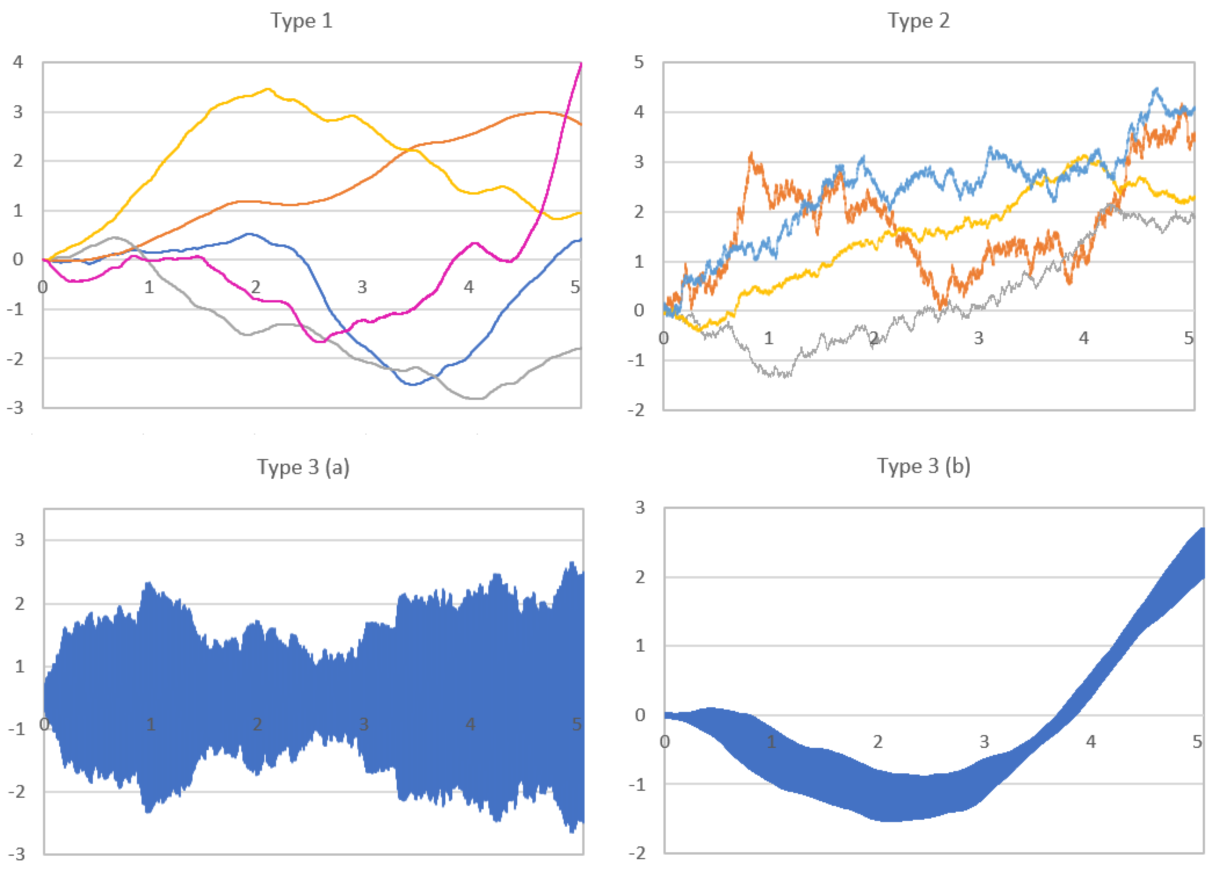

Figure 1: AR models, classified based on the types of roots of the characteristic polynomial

Category	a_1	a_2	a_3	a_4	Factored version
Type 1 (a)	$\frac{11}{4}$	$-\frac{21}{8}$	1	$-\frac{1}{8}$	$(x-1)^2(x-\frac{1}{2})(x-\frac{1}{4})$
Type 1 (b)	2	-2	2	-1	$(x-1)^2(x^2+1)$
Type 2	$\frac{15}{8}$	$-\frac{35}{32}$	$\frac{15}{64}$	$-\frac{1}{64}$	$(x-1)(x-\frac{1}{2})(x-\frac{1}{4})(x-\frac{1}{8})$
Type 3 (a)	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	$(x^2+1)(x^2-\frac{1}{2})$
Type 3 (b)	0	2	0	-1	$(x-1)^2(x-\frac{1}{2})^2$

Table 1: Characteristic polynomials used in the simulations

4.1.2 Illustration

The simulation results shown in Figure 1 come from my spreadsheet `linear2-small.xlsx`, available on my GitHub repository, here. The cells highlighted in light yellow correspond to the model parameters, such as $\tau, \gamma, a_1, \ldots, a_4$: you can modify them, and it will automatically update the picture. A much larger spreadsheet `linear2.xlsx` (about 30MB), containing many interesting simulations, each with 50,000 observations, is available here.

The simulations use $\tau = \gamma = 0.5$, and the characteristic polynomials $x^4 - (a_1x^3 + a_2x^2 + a_3x + a_4)$ listed in Table 1. The "type 2" plot is a classic, while "type 3 (a)" looks like an ordinary audio signal. In types 3 (a) and 3 (b), the frequency of oscillations is extremely high: this explains while the curve seems to completely fill an area. The "type 3 (b)" time series is rather original. Its characteristic polynomial has two distinct real roots, each with multiplicity 2. Thus, you are unlikely to encounter it in college classes or textbooks.

But the truly spectacular plot, surprisingly, is "type 1". These curves are very smooth, but no matter how much you zoom in, it never becomes a flat line, unlike polynomials or well-behaved math functions. The "type 1" curves in Figure 1 have this property: when differentiated, they become a "type 2" curve. In short, these curves are the integral of Brownian motions, and called Itô integrals [Wiki]. The "type 2" curves are continuous, and their derivatives are white noises. If the multiplicity of the root with largest modulus is 3, then the corresponding curve is even smoother and can be differentiated three times. It's first derivative is a "type 1" curve, and its second derivative is a “type 2” curve.

Other unusual Brownian motions, this time in two dimensions, are found in my book on stochastic processes [7]. See pages 41–43 for a family of Brownian motions, exhibiting an extraordinary strong clustering structure, depending on model parameters. Some Gaussian processes can have a behavior similar to “type 1” depending on parameters, see here. For Gaussian processes, see also [10], especially chapter 4.

4.2 Solving Vandermonde systems: a numerically stable method

Vandermonde systems [Wiki] are notoriously ill-conditioned [Wiki]. This explains why nobody really solve them, and instead, use other techniques. Yet they could potentially be used to solve many problems, including linear regression and eigenvalues [Wiki] via the characteristic polynomial, Lagrange interpolation [Wiki], Markov chains (computation of the stationary distribution), linear recurrence equations, principal component analysis (PCA), auto-regressive time series, square root of a matrix, and more.

Using the methodology presented in this article, I propose a numerically stable algorithm to solve such systems, in a very indirect way. I illustrate my method on a particular example. Let’s consider the auto-regressive process

$$X_n = \frac{3}{4}X_{n-1} + \frac{7}{8}X_{n-2} - \frac{3}{4}X_{n-3} + \frac{1}{8}X_{n-4}, \tag{8}$$

with no white noise. In other words, let $\tau = 0$ in formula (7). The characteristic polynomials has roots $r_1 = 1, r_2 = -1, r_3 = \frac{1}{2}, r_4 = \frac{1}{4}$. Let the initial conditions be $X_0 = 1, X_{-1} = 0, X_{-2} = 0, X_{-3} = 0$ as in formula (6). Thus, I will be solving the Vandermonde system

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 2 & 4 \\ 1 & 1 & 4 & 16 \\ 1 & -1 & 8 & 64 \end{pmatrix} \begin{pmatrix} q'_1 \\ q'_2 \\ q'_3 \\ q'_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{9}$$

We know that

$$X_n = r_1^n q'_1 + r_2^n q'_2 + r_3^n q'_3 + r_4^n q'_4. \tag{10}$$

Computing X_n iteratively using formula (8), we find that $X_{30} \approx 1.6000$ and $X_{31} \approx 1.0667$. The influence of r_3, r_4 is negligible in formula (10), and we can reasonably ignore these two roots if n is large enough. Thus, if $n = 30$ we have $1.6000 \approx q'_1 + q'_2$, according to (10). And $n = 31$ yields $1.0667 \approx q'_1 - q'_2$. These two equations allow us to get a very good approximation both for q'_1 and q'_2. We don’t care about q'_3 and q'_4 as the associated roots r_3, r_4 have almost no impact on X_n when n is large.

However, we can still compute them if we want to. Consider the sub-recursion $\{Y_n\}$ with characteristic polynomial $(x - r_3)(x - r_4)$, that is $Y_n = \frac{3}{4}Y_{n-1} - \frac{1}{8}Y_{n-2}$. Let the initial conditions be

$$Y_n = X_n - (r_1^n q'_1 + r_2^n q'_2), \quad \text{for } n = 0, -1. \tag{11}$$

Use the approximated values $q'_1 = 1.333$ and $q'_2 = 0.2667$ computed in the previous step. Clearly, by design,

$$Y_n = r_3^n q'_3 + r_4^n q'_4 \tag{12}$$

with $r_3 = \frac{1}{2}, r_4 = \frac{1}{4}$. Also, since r_3 dominates over r_4, for n large enough, we have $Y_n \approx r_3^n q'3$, that is, $q'_3 \approx r_3^{-n} Y_n \approx 2^n Y_n$. Using formulas (11) and (12) combined with the approximated values of q'_1, q'_2 and $n = 15$, one eventually obtains $q'_3 \approx 2^{15} Y_{15} \approx -0.6667$. Now we are left with finding the last coefficient q'_4. It can be done using the linear recurrence with characteristic polynomial $x - r_4$. The details are left as an exercise. The exact values are

$$q'_1 = \frac{4}{3} = 1.3333\ldots, q'_2 = \frac{4}{15} = 0.2666\ldots, q'_3 = -\frac{2}{3} = -0.6666\ldots, q'_4 = \frac{1}{15} = 0.0666\ldots.$$

If we have more than $p = 4$ unknowns, we can proceed iteratively in the same manner, obtaining approximate values successively for q'_1, q'_2 and so on, assuming the roots are ordered by modulus, with r_1 having the largest modulus. We accumulate inaccuracies at each new iteration, but the loss of accuracy is controlled: the biggest losses are on the coefficients with the lowest impact. Roots with same moduli must be treated jointly, as I did here for r_1 and r_2. If some roots have a multiplicity greater than 1, formula (10) must be adapted: see [3].

5 Math for Machine Learning: Must-Read Books

In general, my articles are not traditional. I try to offer original content to the reader, presenting innovative methods explained in simple English, in a style that is pleasant to read. If you are looking for standard textbooks to learn the math of machine learning, I recommend “Mathematics for Machine Learning” [2] and “Introduction to Mathematical Statistics” [8]. The book “Deep Learning” [4] also covers the math of matrix algebra. Lectures notes from Zico Kolter at Stanford University (2015), covering linear algebra, are available here. The book “Linear Algebra for Data Science” by Shaina Bennett (2021) is available online here, and features examples in R. See also “Introduction to Probability for Data Science” by Stanley Chan (2021) available here, with Matlab and Python code, especially the last chapter on random processes.

For hands-on references with Python code, StatsModels.org is a vast GitHub repository covering a lot of linear algebra and time series. Another one is Statistics and Machine Learning in Python, here. See also the Python Scikit-learn.org guide about linear models, and Interpretable Machine Learning. You won’t find the math or type of examples discussed here, in any of these books. However, they are useful, classic yet modern references.

References

[1] Barbara Bogacka. *Lecture Notes on Time Series*. 2008. Queen Mary University of London [Link]. 5

[2] Marc Deisenroth, A. Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. [Link]. 9

[3] Arash Farahmand. *Math 55 Lecture Notes*. 2021. University of Berkeley [Link]. 3, 8

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. [Link]. 9

[5] Vincent Granville. Interpretable machine learning: Multipurpose, model-free, math-free fuzzy regression. *Preprint*, pages 1–11, 2022. MLTechniques.com [Link]. 5

[6] Vincent Granville. Little known secrets about interpretable machine learning on synthetic data. *Preprint*, pages 1–14, 2022. MLTechniques.com [Link]. 3

[7] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLTechniques.com, 2022. [Link]. 6, 8

[8] Robert V. Hogg, Joseph W. McKean, and Allen T. Craig. *Introduction to Mathematical Statistics*. Pearson, eighth edition, 2016. [Link]. 9

[9] Marc-Andreas Muendler. Linear difference equations and autoregressive processes. 2000. University of Berkeley [Link]. 5

[10] Carl Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. [Link]. 8

[11] Shaohong Yan, Aimin Yang, et al. Explicit algorithm to the inverse of Vandermonde matrix. In *2009 International Conference on Test and Measurement*, 2009. IEEE [Link]. 2