Constrained

Multiple

Polynomial

Ordinary

Least

Squares

Efficient, overpowered data fitting for applications in engineering, physical modelling, statistical inference, and more

By J.A. Ferrand B.Sc M.Sc
Embry-Riddle Aeronautical University – Daytona Beach FL

Symbols and Acronyms:

- "N" is the number of observations.
- "n" is the order of the polynomial fit.
- "i" is a counter that enumerates the positive powers of the polynomial (goes from *1* to n)
- "I" is also a counter that runs from *1* to n, this one however, denotes the power of a coefficient during interim partial derivatives that make up the minimization criteria of *SSR*.
- "o" is a counter that enumerates the observations (goes from *1* to N)
- "y_o" is an observation of the dependent variable.
- "x_o" is an observation of the independent variable.
- "y_P" is a prediction of y that is a function of x_o.
- "P_n" is a polynomial fit of order n.
- "$P_{n,m}$" is a Laurent polynomial of orders n and m.
- "m" is the lowest negative power in a Laurent Polynomial.
- "ε" is called a residual, defined as the difference between y_o and its corresponding y_P.
- "K" is the number of constraints imposed on the system.
- "k" is a counter that enumerates the constraints (goes from *1* to K)
- "R" is the number of rows of the Design matrix, which equals the number of unknown coefficients.
- "R^2" is the coefficient of determination.
- "*SSR*" is an acronym for 'Sum of Squares of the Residuals'
- "a_i" is a coefficient that corresponds to the i^{th} power.
- "$[X^TX]$" is the Design matrix.
- "X" is the Vandermonde matrix.
- "*RHS*" is an acronym for 'Right Hand Side'
- "$B_\#$" is the band number counter (goes from 1 to 2*R - 1).
- "B_P" is the pivotal band number.
- "P" is the linear index of the pivotal coefficient (its location in a MATLAB® array).
- "Z" is the standardized variable space.

Contents

Chapter 0: Abstract

Objective (i.e., What?):
To develop an Object-Oriented Programming (OOP) script capable of deploying highly versatile Ordinary Least Squares (OLS) regression analyses on user-input data sets. Basic features include, multiple variable regression capabilities, ability to impose constraints, data visualization capabilities, and output metrics that asses the model's performance. The script must allow the user to parametrically define a predictive model from common algebraic functions, such as polynomials or exponentials. Due to the inherent computational cost of building OLS models using brute force methods, the script should reflect meaningful attempts at computationally optimizing the OLS analysis.

Importance (i.e., Why?)
In virtually all fields of industry and business, data plays an essential role as it reveals trends and enables quantification of performance (amongst other broad applications). Data is quite simply information about something, knowledge about it if you will. And as the saying goes: "knowledge is power." Data is powerful because its possession distinguishes *uninformed* from *informed* decision makers. Those informed benefit from advantages that the uninformed do not. Would you rather be some ordinary stock investor, or the *Wolf of Wallstreet* (who has insider information)? Obviously, don't break the law unless you can single-handedly take on the federal government, but a similar point stands: One would not want to be the car manufacturer who advertises to bike owners (or, broadly speaking, people who don't own cars). One, therefore, must strive to be informed.

Data has long established its importance, however, simply having it is a required but not always sufficient condition to provide insight. Alone, data can be cryptic, purely numeric, and otherwise overwhelming such that its raw form is unusable to the manager, commander, or chief executive officer (i.e., the decision makers). Data almost always must be made more digestible by means of a statistical model. This can be achieved by presenting a formula with explainable features to the decision makers.

Implementation (i.e., How?)
Two (2) OOP software are considered: MATLAB®[1] and Python® 3[2]. The algorithm, product of this work, is to feature scripts in the syntaxes of both OOP software. The MATLAB® script can be written in said program's native installation (i.e., no libraries or add-ons are required). The Python® 3 script will be heavily based on the Numpy™ and Matplotlib™ libraries as the native Python® 3 software lacks innate features that facilitate OLS computations and data visualization. The structure of the OOP scripts will be founded upon the literature review and theory developed in this work. The work of writing is composed using Microsoft Word.

Intended Users (i.e., Who?)
The layman (i.e., the average Joe and Jane). This is specifically aimed at people who would rather not pay for software packages that just "handle it" (like Microsoft Excel, or certain OOP libraries). This work is intended for the people who wish to take control of their personal data fitting. The greatest utility will be extracted by those who practice a discipline that involves numeric data.

[1] MATLAB is short for "Matrix Laboratory," and it is an OOP software actively developed by MathWorks®
[2] Python 3 is a free, open-source OOP software that is actively developed by the Python Software Foundation®

Chapter 1: Introduction

Ordinary Least-Squares (OLS):

OLS is a calculus-based technique used to curve-fit discrete data and predict the response of some dependent (predictive) variable to its independent (predictor) variables. The discrete data consist of so-called *observations*, which are values of the dependent variable *y* coupled with values of its dependencies *x*. The product of an OLS analysis is an algebraically writeable parametric model[3] (denoted y_P) that for its type will feature the smallest possible Sum of Squares of Residuals (SSR) on a given data set. The SSR are quite literally the sum of the so-called "residuals" which are the differences between the observed dependent data y_o and y_P at the corresponding x_o. This is expressed as:

$$\varepsilon_o = y_o - y_P \tag{1}$$

$$SSR = \sum_{o=1}^{N} \varepsilon_o^2 = \sum_{o=1}^{N} [y_o - y_P]^2 \tag{2}$$

Where:

- "ε_o" is a residual.
- "y_o" is a particular observation of the dependent variable *y*.
- "x_o" is a particular observation of the independent variable *x*.
- "y_P" is the OLS model's prediction of *y* at x_o. NOTE: $y_P = f(x_o)$
- "*N*" is the number of observations (i.e., the size of the data set).
- "*o*" is now the enumerator of the observations (i.e., a counter that goes from *1* to *N*)
- "*SSR*" is once again, the Sum of Squares of Residuals.

Minimization of SSR:

OLS is a minimization problem, meaning (in layman's terms) that an algebraic expression for the "badness" is formulated and then made least. In the case of OLS the SSR is the badness because it grows when the model predicts y_p that is ever farther away from y_o. If so, then y_P will presumably diverge from *y*. If the "badness" is minimized, then the model is said to be best because it features the least possible badness. Algebraically this is accomplished using differential vector calculus, specifically, with the gradient operator:

$$\nabla f(x_1, x_2, \dots, x_N) = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \dots \quad \frac{\partial f}{\partial x_3}\right]^T = \vec{0} \tag{3}$$

Where:

- "∇" is the gradient operator.
- "f" is a scalar-valued function of many variables.
- "x_1" through "x_n" are the independent variables of *f*.

The gradient operator, when applied to a scalar-valued function *f*, yields a vector ∇f that contains the rates of change of *f w.r.t* all its dependencies. **When ∇f equals the zero vector (i.e., there is no rate of change *w.r.t* any dependence) then *f* is either at a global or local maximum or minimum value**.

[3] OLS models are commonly referred to as "best fit."

Some clarity must be shed on y_P and what is meant by a "parametric model" before illustrating the need for the gradient operator. To perform an OLS regression a user must compose y_P from functions (called regressors) of the variables on which y depends. Said regressors define the features of y_P and must be chosen appropriately to efficiently encapsulate the observed trends between y and its predictors. The idea is to make y_P mimic y_o (and thus presumably imitate y). The observations y_o, however, may experience multiple behaviors of different intensities. To determine representative relative intensities between the behaviors, *parameters* are included with each regressor of the independent variable that makes y_P, that is:

$$y_P = g(a_i, x_o) = \sum_{i=1}^{n} a_i f_i(x_o) = a_1 f_1(x_o) + a_2 f_2(x_o) + \cdots + a_n f_n(x_o) \tag{4}$$

Where:

- "a_i" are the parameters.
- "f_i" are the regressors (functions meant to encapsulate the behavior between y and x).

These a_i can be thought by the layman as weights that give y_P say; "a little bit of this behavior, a lot more of this other behavior, and some more of that next behavior, etc." Since the $SSR = f(y_P)$, and $y_P = g(a_i)$, then $SSR = f(g(a_i))$. Therefore, the *SSR* is a scalar-valued function in a_i, thus, the solution to the equation:

$$\nabla(\mathrm{SSR}) = \left[\frac{\partial(\mathrm{SSR})}{\partial \mathrm{a}_0} \quad \frac{\partial(\mathrm{SSR})}{\partial \mathrm{a}_1} \quad \ldots \quad \frac{\partial(\mathrm{SSR})}{\partial \mathrm{a}_\mathrm{n}}\right]^{\mathrm{T}} = \vec{0} \tag{5}$$

Produces the optimal a_i and culminates an OLS analysis.

Wait, you said maximum OR minimum!

The reader can rest assured that Equation 5 always corresponds to a minimum point. In fact, there is no global maximum for the *SSR* because egregiously bad choices for a_i could produce arbitrarily large values of the former. This fact is a direct consequence of the SSR's definition (Equation 2). Because the *SSR* is a sum of squared differences it is, by nature, always positive as it is a quadratic in all a_i. This also means that the *SSR* has a unique minimum point.

Linear OLS

The formulation of y_P as shown in Equation 4 implies that the setup of Equation 5 will inevitably produce a *linear system* in which a_i are the unknown coefficients. This is possible only because the parameters a_i appear to be "constant" multipliers. This keeps Equation 5 *linear* in terms of the fine-tunable parameters. However, if any of the f_i were to be a *nonlinear* function and the user decides to include a parameter inside the argument of said f_i then said user will enter the realm of *nonlinear OLS*, which requires numerical methods to create predictive models. The scope of this work is currently limited to *Linear OLS* only.

Chapter 2: Regressor Choices

This chapter introduces the reader to several functions with which one may build y_P. So far, no practical way of evaluation Equation 5 has been discussed as that requires one to first define their own y_P. Therefore, one should get exposure to the functions that can essentially be the building blocks of an OLS model. Note that the ensuing discussion is limited to functions that result in a *linear* system when setting up Equation 5. Functions that do not meet this criterion are omitted. Any selection of *algebraic* regressors will guarantee that the OLS is linear whereas *transcendental* regressors that are *parametrized* result in nonlinear OLS. By the term *parametrized* it is meant that a parameter a_i is inside the argument of the transcendental regressor. *Nonparametric* transcendental regressors may, however, be incorporated to an OLS analysis without turning the problem nonlinear.

Algebraic Functions

These are functions that are exactly described by the operations of addition/subtraction, multiplication/division, and exponentiation with any algebraic number.

Polynomials

These are functions made up of so-called *indeterminates*, that involve only the operations of addition, multiplication, and exponentiation with positive integer powers only. The term *indeterminate* is the formal name given to the variables that make up the polynomial. Consider the following sample polynomials:

Quadratic Equation:

$$P_2(x) = 0 = c + bx + ax^2$$

3-D Ellipsoid[4]:

$$P_2(x, y, z) = 1 = \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2}$$

General Univariate Polynomial (degree *n*)

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = \sum_{i=0}^{n} a_i x^i$$

Bivariate Polynomial (degree *3*)

$$P_3(x, y) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4y + a_5y^2 + a_6y^3 + a_7xy + a_8x^2y + a_9xy^2$$

Trivariate Polynomial (degree *2*)

$$P_2(x, y, z) = a_0 + a_1x + a_2x^2 + a_3y + a_4y^2 + a_5z^2 + a_6xy + a_7xz + a_8yz$$

General Multivariate Polynomial (no mixed terms):

[4] All other quadric surfaces (cones, paraboloids, hyperboloids, and cylinders) are written as polynomials.

$$P_n(x_1, x_2, \dots, x_k) = \sum_{j=1}^{k} \sum_{i=0}^{n} a_{j,i} x_j^i$$

Where:

- "*n*" is called the *degree* of the polynomial.
- "*x*, *y*, and *z*" are examples of *indeterminates*.

As evidenced above, polynomials are expressed as sums of terms (i.e., they are superpositions of products of the *indeterminates*). Said terms are called *monomials* (i.e., xy, x^2y, x^3) and each of them can be "weighed" by some parameter a_i and thus be turned into a regressor. Polynomials are further characterized by their *degree*, which is the highest sum of the powers of all *indeterminates* that make up a monomial term. The *degree* of a polynomial defines the number of "*humps*" that its curve will feature. These so-called humps are exhibitions of a polynomial's ability to turn, and thus, more closely fit data. Polynomials of high degree, therefore, feature remarkable flexibility and can thus be used to fit data with many changes in behavior.

Laurent Polynomials

A *Laurent Polynomial* is one which includes negative integer powers of the independent variable (in addition to or exclusion of the positive powers). To describe a Laurent polynomial, the highest order of the positive powers *n* is needed just like before, but additionally, the "largest negative" power "*m*" is needed. A univariate Laurent Polynomial of orders *m* and *n* is defined below:

$$P_{n,m}(x) = \frac{a_{-m}}{x^m} + \cdots + \frac{a_{-2}}{x^2} + \frac{a_{-1}}{x} + a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$P_{n,m}(x) = \sum_{i=m}^{n} a_i x^i \; \forall \, (m,n) \in \mathbb{Z} \mid m \leq 0 \; \wedge \; 0 < n$$

OLS with Laurent polynomials is rarely needed, however, there may be data sets that exhibit abrupt behavioral changes, such as a singularity. Since all monomials of negative powers in a Laurent Polynomial will feature asymptotic behavior at *x=0*, then they can prove to be efficient regressors for particularly stiff trends near the origin. Nonsingular trends may be still be efficiently modeled by Laurent polynomials if they feature a decaying trend.

Transcendental Functions

These are functions that cannot be exactly described with the addition, subtraction, multiplication, division, or algebraic power exponentiation operations. These functions, therefore, lack an exact algebraic definition and are exclusively available as infinite sums of algebraic terms. These sums, when finitely evaluated to an acceptable accuracy produce acceptable estimates.

Exponentials

Functions whose rate of change is proportional to themselves are termed *exponential functions* and are of the form:

$$f(x) = ab^x$$

Where "*b*" is called the *base* and *a* is some constant coefficient. Exponentials feature the following property:

$$\frac{df}{dx} = cb^x$$

Where "*c*" is a *proportionality* constant. There is a special value for *b* such that *c* is equal to *1*, and that is the constant *e* which goes by the name of *Euler's number*[5].

Trigonometric functions

Functions that relate the sides of *right* triangles to its angles are termed *trigonometric*. Examples of said functions are:

$\sin(x)$	$\tan(x)$	$\sec(x)$
$\cos(x)$	$\cot(x)$	$\csc(x)$

Trigonometric functions are useful in modeling periodic data. Should one have to work with data that exhibits oscillations, then trigonometrics will be useful. The trigonometric functions have four (4) possible parameters that could aid in a regression problem. Those are the amplitude *A*, the vertical shift *C*, the phase angle ϕ, and the frequency ω. Let "*trig*" denote any of the trigonometric functions listed above, that is:

$$\mathrm{f(x)} = \mathrm{trig(x)}, \qquad \mathrm{trig(x)} \in \{\sin(\mathrm{x}), \cos(\mathrm{x}), \tan(\mathrm{x}), \sec(\mathrm{x}), \csc(\mathrm{x}), \cot(\mathrm{x})\}$$

Then algebraically, the parameters modify the functions as:

$$\mathrm{y_P} = (\mathrm{A})\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) + \mathrm{C},$$

Thinking about the chain rule of differentiation, it is evident that linear OLS is only possible on *A* and *C*. Optimization of ϕ or ω requires the setup of a nonlinear system. Per the scope of this work, no optimization on ϕ or ω is exemplified here, however, said parameters can still be preset as non-zero constants. Note that *C* is essentially the same as a_0 in the polynomial functions, a constant term.

Other Transcendentals:

Rather than giving a summary of every transcendental know, this section shall finish with a brief mention of other transcendentals that may prove useful:

- Hyperbolic functions.
- Logarithms
- Sigmoids
- Inverse trigonometrics

[5] *e* goes by many names. Another common name is "the natural base."

Chapter 3: Minimization of SSR

With a repertoire of regressors to choose from, it is time demonstrate sample setups of Equation 5. In doing so, the formulation of linear systems will be exemplified. Concepts from linear algebra will be discussed as appropriate. The gist of this chapter is essentially a multitude of ways to write Equation 5 as a linear equation of the form:

$$[X^T X]\overrightarrow{a_i} = \overrightarrow{RHS}$$

Where $[X^T X]$ is called the Design matrix, a_i are the parameters, and *RHS* is an acronym that stands for Right Hand Side.

Univariate Polynomials

$$\mathrm{y_P} = \mathrm{P_n(x)} = \sum_{\mathrm{i=0}}^{\mathrm{n}} \mathrm{a_i x^i}$$

A univariate polynomial of degree *n* has "*n + 1*" parameters, namely a_0 through a_n. Therefore, per Equation 3:

$$\mathrm{SSR} = \sum_{\mathrm{o=1}}^{\mathrm{N}} \left(\mathrm{y_o} - \sum_{\mathrm{i=0}}^{\mathrm{n}} \mathrm{a_i x_o^i} \right)^2$$

Using the chain rule on Equation 5 produces *n+1* equations of the form:

$$\frac{\partial(\mathrm{SSR})}{\partial \mathrm{a_I}} = -2 \sum_{\mathrm{o=1}}^{\mathrm{N}} \left(\mathrm{y_o} - \sum_{\mathrm{i=0}}^{\mathrm{n}} \mathrm{a_i x_o^i} \right) \mathrm{x_o^I} = 0$$

Where "*I*" is a counter that goes from *0* to *n* that denotes the parameter *w.r.t* which the partial derivative of *SSR* is being taken. The regressors with the subscript and superscript "*i*" are present in the same equation as that of the I^{th} regressor.Rearranging the minimization condition yields the expression:

$$\sum_{\mathrm{o=1}}^{\mathrm{N}} \left(\sum_{\mathrm{i=0}}^{\mathrm{n}} \mathrm{a_i x_o^{i+I}} \right) = \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{y_o x_o^I} \quad \forall\, (I, n) \in \mathbb{Z} |\, 0 \leq I \leq n$$

Which when evaluated for all coefficients can be represented using matrix notation:

$$\begin{bmatrix} \mathrm{N} & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o} & \cdots & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^n} \\ \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o} & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^2} & \cdots & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^{n+1}} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^n} & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^{n+1}} & \cdots & \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{x_o^{2n}} \end{bmatrix} \begin{bmatrix} \mathrm{a_0} \\ \mathrm{a_1} \\ \vdots \\ \mathrm{a_n} \end{bmatrix} = \begin{bmatrix} \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{y_o} \\ \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{y_o x_o} \\ \vdots \\ \sum_{\mathrm{o=1}}^{\mathrm{N}} \mathrm{y_o x_o^n} \end{bmatrix}$$

Univariate Laurent Polynomials:

OLS with a univariate Laurent polynomial will feature *m+n+1* parameters, therefore, the size of $[X^T X]$ will be $m+n+1 \times m+n+1$. Since the power rule for taking derivatives of polynomials works the same way

for negative powers as it does for positive powers, no major change to Equation X is needed but to change the range of the counter *I* as:

$$\sum_{o=1}^{N} y_o x_o^I = \sum_{o=1}^{N} \left(\sum_{i=m}^{n} a_i x_o^{i+I} \right) \quad \forall\, (I, m, n) \in \mathbb{Z} \mid m < 0 \wedge 0 < n \ \wedge \ m \leq I \leq n$$

The change on $[X^T X]$ is reflected below:

$$\begin{bmatrix} \sum_{o=1}^{N} \frac{1}{x_o^{2m}} & \cdots & \sum_{o=1}^{N} \frac{1}{x_o^{m+1}} & \sum_{o=1}^{N} \frac{1}{x_o^{m}} & \cdots & \sum_{o=1}^{N} \frac{1}{x_o} & N \\ \vdots & \ddots & \vdots & \vdots & \iddots & \iddots & \sum_{o=1}^{N} x_o \\ \sum_{o=1}^{N} \frac{1}{x_o^{m+1}} & \cdots & \sum_{o=1}^{N} \frac{1}{x_o^{2}} & \sum_{o=1}^{N} \frac{1}{x_o} & N & \iddots & \vdots \\ \sum_{o=1}^{N} \frac{1}{x_o^{m}} & \cdots & \sum_{o=1}^{N} \frac{1}{x_o} & N & \sum_{o=1}^{N} x_o & \cdots & \sum_{o=1}^{N} x_o^{n} \\ \vdots & \iddots & N & \sum_{o=1}^{N} x_o & \sum_{o=1}^{N} x_o^{2} & \cdots & \sum_{o=1}^{N} x_o^{n+1} \\ \sum_{o=1}^{N} \frac{1}{x_o} & \iddots & \iddots & \vdots & \vdots & \ddots & \vdots \\ N & \sum_{o=1}^{N} x_o & \cdots & \sum_{o=1}^{N} x_o^{n} & \sum_{o=1}^{N} x_o^{n+1} & \cdots & \sum_{o=1}^{N} x_o^{2n} \end{bmatrix} \begin{bmatrix} a_{-m} \\ \vdots \\ a_{-1} \\ a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{o=1}^{N} \frac{y_o}{x_o^{m}} \\ \vdots \\ \sum_{o=1}^{N} \frac{y_o}{x_o} \\ \sum_{o=1}^{N} y_o \\ \sum_{o=1}^{N} y_o x_o \\ \vdots \\ \sum_{o=1}^{N} y_o x_o^{n} \end{bmatrix}$$

Unmixed, Multivariate, Laurent Polynomials

OLS can be extended to model the trends between a dependent variable and more than one independent variable. Suppose that discrete data is to be fit using the superposition of a constant term and polynomials in arbitrarily many variables of arbitrary order (with no mixed terms):

$$y_P = a_0 + P_{n_1}(z_1) + P_{n_2}(z_2) + \cdots + P_{n_Q}(z_Q)$$

$$y_P = a_0 + \sum_{q=1}^{Q} \sum_{i=m_q}^{n_q} a_{i_q} z_q^i \ \forall\, m_q \leq 0 \leq n_q$$

Where:

- "*Q*" is the number of independent variables.
- "*q*" is a counter that enumerates the number of independent variables (runs from *1* to "*Q*")
- "z" is a generic independent variable which is enumerated by *q*.

Note that the multiple polynomial fit will feature the Polynomial and Laurent orders specified to each of independent variables.

$$\mathrm{SSR} = \sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right)^2$$

Where:

$$n_q = \{n_1, n_2, \ldots, n_Q\}, \qquad m_q = \{m_1, m_2, \ldots, m_Q\}$$

The reader should "know the drill" by now, so one proceeds straight with the generic partial derivative w.r.t the curve-fit parameters:

$$\frac{\partial(\mathrm{SSR})}{\partial a_{I_q}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) z_{o\ q}^{I_q} = 0 \;\forall\; q \mid 1 \le q \le Q \mid m_q \le I_q \le n_q \mid i \ne 0$$

This expression, when evaluated for all permissible values of *I* and *q* results in a linear system of size "*R*", which is given by:

$$R = 1 + \sum_{q=1}^{Q}(n_q + m_q)$$

Complete unravelling of the expression results in an $R \times R$ linear system which is summarized in Table 1.

Table 1: Multivariate Laurent polynomial OLS minimization equations.

	Partial Derivatives	Minimization Conditions
Laurent Orders	$\frac{\partial(\mathrm{SSR})}{\partial a_{-m_{Q_{z_Q}}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right)\frac{1}{z_{Q_o}^{m_Q}} = 0$	$\sum_{o=1}^{N}\frac{1}{z_{Q_o}^{m_Q}}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N}\frac{y_o}{z_{Q_o}^{m_Q}}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{-1_{z_Q}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right)\frac{1}{z_{Q_o}} = 0$	$\sum_{o=1}^{N}\frac{1}{z_{Q_o}}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N}\frac{y_o}{z_{Q_o}}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{-m_{1_{z_1}}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right)\frac{1}{z_{1_o}^{m_1}} = 0$	$\sum_{o=1}^{N}\frac{1}{z_{1_o}^{m_1}}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N}\frac{y_o}{z_{1_o}^{m_1}}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{-1_{z_1}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right)\frac{1}{z_{1_o}} = 0$	$\sum_{o=1}^{N}\frac{1}{z_{1_o}}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N}\frac{y_o}{z_{1_o}}$
a_0	$\frac{\partial(\mathrm{SSR})}{\partial a_0} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) = 0$	$Na_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i} = \sum_{o=1}^{N} y_o$

Polynomial Orders		
	$\frac{\partial(\mathrm{SSR})}{\partial a_{1_{z_1}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) z_{1_o} = 0$	$\sum_{o=1}^{N} z_{1_o}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N} y_o z_{1_o}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{n_{q_{z_1}}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) z_{1_o}^{n_q} = 0$	$\sum_{o=1}^{N} z_{1_o}^{n_q}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N} y_o z_{1_o}^{n_q}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{n_{Q_{z_Q}}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) z_{Q_o} = 0$	$\sum_{o=1}^{N} z_{Q_o}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N} y_o z_{Q_o}$
	$\vdots$	$\vdots$
	$\frac{\partial(\mathrm{SSR})}{\partial a_{n_{Q_{z_Q}}}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) z_{Q_o}^{n_Q} = 0$	$\sum_{o=1}^{N} z_{Q_o}^{n_Q}\left[a_0 + \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right] = \sum_{o=1}^{N} y_o z_{Q_o}^{n_Q}$

An added complexity of multivariate regression over univariate regression is that $[X^T X]$ will feature sums of mixed terms as coefficients. To illustrate this, consider the following multivariate example involving two independent variables (call them "x" and "z") with Polynomial orders of 2 and 3 and Laurent orders of 1 and 0 respectively. Therefore, $Q = 2$, $n_x = 2$, $m_x = 1$, $n_z = 3$, $m_z = 0$, and y_P is given by:

$$y_P = f(x,z) = a_0 + \frac{a_{-1_x}}{x} + a_{1_x}x + a_{2_x}x^2 + a_{1_z}z + a_{2_z}z^2 + a_{3_z}z^3$$

Minimization of SSR produces (i is never 0):

$$\frac{\partial(\mathrm{SSR})}{\partial a_{I_x}} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_x}^{n_x} a_{i_x} z_{o_x}^{i}\right) z_{o_x}^{I_q}$$

$$\frac{\partial(\mathrm{SSR})}{\partial a_0} = -2\sum_{o=1}^{N}\left(y_o - a_0 - \sum_{q=1}^{Q}\sum_{i=m_q}^{n_q} a_{i_q} z_{o_q}^{i}\right) = 0$$

$$\begin{bmatrix} \frac{1}{x_o^2} & \frac{1}{x_o} & N & x_o & \frac{z_o}{x_o} & \frac{z_o^2}{x_o} & \frac{z_o^3}{x_o} \\ \frac{1}{x_o} & N & x_o & x_o^2 & z_o & z_o^2 & z_o^3 \\ N & x_o & x_o^2 & x_o^3 & x_o z_o & x_o z_o^2 & x_o z_o^3 \\ x_o & x_o^2 & x_o^3 & x_o^4 & x_o^2 z_o & x_o^2 z_o^2 & x_o^2 z_o^3 \\ \frac{z_o}{x_o} & z_o & x_o z_o & x_o^2 z_o & z_o^2 & z_o^3 & z_o^4 \\ \frac{z_o^2}{x_o} & z_o^2 & x_o z_o^2 & x_o^2 z_o^2 & z_o^3 & z_o^4 & z_o^5 \\ \frac{z_o^3}{x_o} & z_o^3 & x_o z_o^3 & x_o^2 z_o^3 & z_o^4 & z_o^5 & z_o^6 \end{bmatrix} \begin{bmatrix} a_{-1_x} \\ a_0 \\ a_{1_x} \\ a_{2_x} \\ a_{1_z} \\ a_{2_z} \\ a_{3_z} \end{bmatrix} = \begin{bmatrix} \frac{y_o}{x_o} \\ y_o \\ y_o x_o \\ y_o x_o^2 \\ y_o z_o \\ y_o z_o^2 \\ y_o z_o^3 \end{bmatrix}$$

Trigonometric functions:

This is exemplified below:

$$\frac{\partial(\mathrm{SSR})}{\partial \mathrm{A}} = -2\sum_{\mathrm{o}=1}^{\mathrm{N}}[\mathrm{y_o} - \mathrm{A}\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) - \mathrm{C}]\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) = 0$$

$$\sum_{\mathrm{o}=1}^{\mathrm{N}} \mathrm{y_o}\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) = \mathrm{A}\sum_{\mathrm{o}=1}^{\mathrm{N}} \mathrm{trig}^2(\omega \mathrm{x_o} + \phi) + \mathrm{C}\sum_{\mathrm{o}=1}^{\mathrm{N}} \mathrm{trig}(\omega \mathrm{x_o} + \phi)$$

$$\frac{\partial(\mathrm{SSR})}{\partial \omega} = -2\sum_{\mathrm{o}=1}^{\mathrm{N}}[\mathrm{y_o} - \mathrm{A}\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) - \mathrm{C}]\,\mathrm{Ax_o}\frac{\partial\big(\mathrm{trig}(\omega \mathrm{x_o} + \phi)\big)}{\partial \omega} = 0$$

$$\frac{\partial(\mathrm{SSR})}{\partial \phi} = -2\sum_{\mathrm{o}=1}^{\mathrm{N}}[\mathrm{y_o} - \mathrm{A}\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) - \mathrm{C}]\,\mathrm{A}\frac{\partial\big(\mathrm{trig}(\omega \mathrm{x_o} + \phi)\big)}{\partial \phi} = 0$$

$$\frac{\partial(\mathrm{SSR})}{\partial \mathrm{C}} = -2\sum_{\mathrm{o}=1}^{\mathrm{N}}[\mathrm{y_o} - \mathrm{A}\,\mathrm{trig}(\omega \mathrm{x_o} + \phi) - \mathrm{C}] = 0$$

$$\sum_{\mathrm{o}=1}^{\mathrm{N}} \mathrm{y_o} = CN + A\sum_{\mathrm{o}=1}^{\mathrm{N}} \mathrm{trig}(\omega \mathrm{x_o} + \phi)$$

Chapter 4: Generating the Design Matrix

Now that the minimization equations of the *SSR* have been developed, it is time to wonder how to evaluate $[X^TX]$ and the *RHS* vectors so that one may produce numeric values for the a_i parameters. Once the numeric values are available, direct, or numeric linear algebra techniques such as Gaussian elimination or Gauss-Seidel iterations may be used to numerically compute a_i.

Vandermonde decomposition

A prominent feature of $[X^TX]$ is that it is always a *symmetric* matrix, meaning that it is mirrorable about its main diagonal. Specifically, the strictly lower and strictly upper triangular coefficients are their own mirror images about the main diagonal. Algebraically, this has the consequence that $[X^TX]$ can be written as:

$$[X^TX] = X^T \otimes X^T = X^TX$$

Where "*X*" is a *Vandermonde* matrix, "⊗" is the outer-product operator, and "^{T}" is the transpose operator. The Vandermonde matrix is an array of powers of the observations of the predictor variables. The rows of *X* equal the number of observations *N*, and its number of columns equals *one* plus the number of parameters a_i being optimized. The *RHS* vector can be conveniently written in terms of X^T as:

$$\overrightarrow{RHS} = X^TY$$

Where the matrix *Y* is analogue to the *X* matrix, just that it is comprised of the dependent variable observations y_o. The rectangular matrices *Y* and *X* can be used to generate the linear system of an OLS problem as:

$$(\mathrm{X^TX})\vec{\mathrm{a}} = \mathrm{X^TY}$$

Examples of Vandermonde decompositions for the different regressors discussed are listed next:

Univariate Polynomials:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \qquad X = \begin{bmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \dots & x_N^n \end{bmatrix}, \qquad X^T = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \dots & x_N^n \end{bmatrix},$$

Univariate Laurent Polynomials:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \qquad X = \begin{bmatrix} x_1^{-m} & \dots & x_1^{-1} & 1 & x_1 & \dots & x_1^n \\ x_2^{-m} & \dots & x_2^{-1} & 1 & x_2 & \dots & x_2^n \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_N^{-m} & \dots & x_N^{-1} & 1 & x_N & \dots & x_N^n \end{bmatrix}, \qquad X^T = \begin{bmatrix} x_1^{-m} & x_2^{-m} & \dots & x_N^{-m} \\ \vdots & \vdots & \iddots & \vdots \\ x_1^{-1} & x_2^{-1} & \dots & x_N^{-1} \\ 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \dots & x_N^n \end{bmatrix}$$

Multivariate Laurent Polynomials:

$$X = \begin{bmatrix} \dots & z_1^{-m_z} & \dots & z_1^{-1} & x_1^{-n_x} & \dots & x_1^{-1} & 1 & x_1 & \dots & x_1^{n_x} & z_1 & \dots & z_1^{n_z} & \dots \\ \dots & z_2^{-m_z} & \dots & z_2^{-1} & x_2^{-n_x} & \dots & x_2^{-1} & 1 & x_2 & \dots & x_2^{n_x} & z_2 & \dots & z_2^{n_z} & \dots \\ \dots & \vdots & \iddots & \vdots & \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \dots \\ \dots & z_N^{-m_z} & \dots & z_N^{-1} & x_N^{-n_x} & \dots & x_N^{-1} & 1 & x_N & \dots & x_N^{n_x} & z_N & \dots & z_N^{n_z} & \dots \end{bmatrix}$$

Single Sinusoid:

Simple, but wasteful

TheVandermonde decomposition is straightforward: simply evaluate each observation of an independent variable across all chosen regressors and arrange them as shown in the previous examples. The advantage of this method is that it is the simplest way of generating $[X^TX]$ and *RHS*. No further sophistication is needed. There is one major problem, however, that is related to the fact that $[X^TX]$ is a symmetric matrix and that the Vandermonde decomposition entails a matrix multiplication. The symmetry of $[X^TX]$ implies that it features multiple coefficients with the same numeric value. In this context, a matrix multiplication would imply many computations of the same number. Essentially, computational power is wasted crunching algebra for the same number over and over. Take for example, $[X^TX]$ in the case of OLS with a univariate Laurent polynomial. Evidently, all counter diagonals have identical elements. If done in a computer, evaluation of $[X^TX]$ using the Vandermonde decomposition would result in $m+n+1$ computations of the same value N, which is known prior to an OLS analysis anyway as it is simply the number of observations. Similarly, the first sub-counter diagonal and first super-counter diagonal crunch the same summation of x_o and the reciprocal of x_o respectively a total of $m+n$ times.

Custom Algorithms

To remediate the computational shortcomings of the Vandermonde decomposition, custom algorithms that leverage the symmetry of the design matrix X^TX can be developed to improve the evaluation of the latter. A custom algorithm for the evaluation of multivariate Laurent polynomials with unmixed terms is developed in this work. The major shortcoming of this approach is that the ensuing algorithm is only suitable for a particular choice of regressors. To augment the algorithm so that it may efficiently evaluate X^TX with many different regressors now becomes a coding challenge. There are many combinations of regressors, therefore, those obsessed with efficiency are doomed to spend countless hours developing custom algorithms for all the combinations they conceive. For this reason, the Vandermonde decomposition has merit as it can truly generate X^TX for any combination of regressors (albeit at a cost).

Regardless, overtly extravagant regressor choices are unwarranted, so most of the time the data-fitting practitioner ought to work with simple regressor choices. By far, the simplest models consist of polynomials, so the first custom algorithm developed here is termed "UMLP-OLS," which is an acronym of Unconstrained Multivariate Laurent Polynomial Ordinary Least Squares. Steppingstone algorithms developed prior are discussed first, as they are built upon to arrive to the final product.

Leverageable features of X^TX:

1) **Symmetry:**
   Recall that the design matrix is symmetric, meaning that at most, one should only have to compute the sums at and below (or above) the main diagonal. The sums below the main diagonal can then be mirrored about the main diagonal to obtain the full X^TX matrix.
2) **Pivot:**
   Every conceivable X^TX has an element that is strictly equal to the number of observations N. This element is referred to as the *pivot*. Because of the "default" order in which the a_i coefficients have been laid out, for OLS involving positive powers only (i.e., $m=0$ and no trigonometric reciprocals are chosen as regressors) the pivot will always be the top-left corner. In contrast, if an OLS were to consist of strictly negative powers, the pivot would be the bottom-right corner. For OLS with both positive and negative powers, however, the pivot's location predictably depends on the number of regressors. The *pivot* strictly belongs to the main diagonal of X^TX.
3) **Primordial Sums:**
   There is a bare minimum of lists that can be stored from which all of X^TX can be produced. These lists consist of the powers of the independent variable observations and are referred to as *primordial lists*. Summing any primordial list will produce an element that belongs to the pivotal row or column. Primordial lists can be combined (two at a time) as elementwise products of multiplication to produce a temporary list. Individual summation of all possible temporary lists produces all elements that do not belong to the pivotal row or column of X^TX .
4) **Bands:**
   The matrix X^TX features counter diagonal bands that all have the same coefficients.

A note on OOP arrays

Since the objective is to implement OLS capability through OOP software now is a good time to discuss how the software of choice (MATLAB® and Python® 3) implement matrix algebra. MATLAB's default object variable is termed the *double*[6] *array*, whereas the staple Numpy object is the *ndarray*. Both objects are suitable coding constructs that can represent matrices and vectors such as X^TX, and *RHS*. The background computer nuances are irrelevant to the discussion except for the nomenclature used to refer to the specific matrix elements. The chosen programs give their users the ability to reference array elements using either an integer *linear index* or a pair of integer *subscripts*. The latter consists of specifying a row number and a column number (akin to an XY-coordinate pair). The former consists of specifying a single number ranging from some starting value (corresponding to the first entry of the array) and a final value (that corresponds to the final array element). Subscripts are more human readable; however, they are not native to the way software store elements in memory. Both the *double array* and the *ndarray* store data as a long list that is referenced with a *linear index*. The subscript functionality incurs a penalty because it must be transformed to a linear index before returning array elements. To avoid this penalty, the ensuing algorithms are developed directly as formulas that produce linear indices.

[6] Double (computer science): A numeric variable stored as double-precision, meaning that it is encoded in 64 bits.

Linear Indices

In OOP, the two most prominent linear index notations are the so-called *C-order* and *F-order*. The "*C*" in the former is a reference to the C Programming language, and the "*F*" in the latter is a reference to the Fortran Programming language. The C-order enumerates elements top to bottom in a column for all columns from left to right. The F-order enumerates elements left to right in a row for all rows from top to bottom. MATLAB's *double array* uses the C-order whereas Numpy's *ndarray* uses the F-order. Knowing how to traverse the arrays is necessary so that custom algorithms can paste the repeated coefficient of X^TX in the correct locations.

C-order

For some array *A*, let *i* denote a linear index representing some element in *A* and let *R* denote the column length of *A*, i.e., the number of rows of *A*. Then the elements above and below *A(i)* are given by *A(i-1)* and *A(i+1)* respectively. The the elements to the left and right of *A(i)* are given by *A(i-R)* and *A(i+R)* respectively. To move diagonally from *A(i)* in either a down-right and up-left direction use *A(i+R+1)* and *A(i-R-1)* respectively. Finally, to move diagonally from *A(i)* in either a down-left and up-right direction use *A(i -R+1)* and *A(i+R-1)* respectively. C-ordering is exemplified by Figure 1.

F-order

For the same array *A*, indexed by a different *i*, let *C* denote the row length, i.e., the number of columns of *A*. The elements to the left and right of *A(i)* are given by *A(i-1)* and *A(i+1)* respectively. Elements above and below *A(i)* are given by *A(i-C)* and *A(i+C)* respectively. The top-right and bottom-left adjacent diagonal elements of *A(i)* are given by *A(i-C+1)* and *A(i+C-1)* respectively. The bottom-right and top-left adjacent diagonal elements of *A(i)* are given by *A(i+C+1)* and *A(i-C-1)*. F-ordering is exemplified by Figure 2.

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Figure 1: 4×4, one-indexed, C-ordered array

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figure 2: 4×4, zero-indexed, F-ordered array

Counter diagonal indices

A special technique is developed to generate the linear indices of an array's counter diagonal bands. This is done with the objective of exploiting the fourth leverageable feature of X^TX. Say that the linear index for the starting and ending points of a counter diagonal are known; denote the former and latter by k_1 and k_2 respectively; then in the column-major order:

$$\{\mathrm{R}, \mathrm{C}, \mathrm{k}_1, \mathrm{k}_2, \mathrm{n}_\mathrm{B}, \mathrm{N}_\mathrm{B}, \mathrm{B}, \mathrm{B}_\#\} \in \mathbb{Z}$$

$$\mathrm{B} = \mathrm{R} + \mathrm{C} - 1$$

$$1 \leq \mathrm{B}_\# \leq \mathrm{B}$$

$$\mathrm{k}_1 = \mathrm{B}_\# + [\,\mathrm{R}(\mathrm{B}_\# - \mathrm{R} + 1) - \mathrm{B}_\#] \left[\mathrm{floor}\left(\frac{\mathrm{B}_\#}{\mathrm{R}}\right)\right]$$

$$\mathrm{k}_2 = (\mathrm{R}(\mathrm{B}_\# - 1) + 1) + [\mathrm{R}(\mathrm{R} - 2 - \mathrm{B}_\#) + \mathrm{B}_\# + 1] \left[\mathrm{floor}\left(\frac{\mathrm{B}_\#}{\mathrm{C}}\right)\right]$$

$$\mathrm{N}_\mathrm{B} = \frac{\mathrm{k}_2 - \mathrm{k}_1}{\mathrm{R} - 1}$$

$$\mathrm{i} = \mathrm{k}_1 + \mathrm{n}_\mathrm{B}(\mathrm{R} - 1) \;\; \forall \;\; 0 \leq \mathrm{n}_\mathrm{B} \leq \mathrm{N}_\mathrm{B}$$

Where:

- "N_B" is the number of elements in the counter diagonal band *minus 1*.
- "n_B" is a counter that goes from *0* to N_B.
- "$B_{\#}$" is the band number. On any rectangular array there are *R+C-1* bands *B*.

Unconstrained Univariate Laurent Polynomial Ordinary Least Squares

1) Compute the size of X^TX:

$$R = 1 + n + m = C$$

2) Pre-allocate memory for an R×R array (X^TX) and an R×1 vector (*RHS*).
3) Compute the pivot's index:

$$P = 1 + R(m + 1)$$

$$[X^TX](P) = N$$

4) Compute the pivotal band number
5) Compute positive power primordial lists.
6) Compute negative power primordial lists.

The pivotal diagonal is given by

Unconstrained Multivariate Laurent Polynomial Ordinary Least Squares

Let *R* replace I_q denote the number of rows of the design matrix X^TX, then *R* is given by:

$$R = 1 +$$

Chapter 5: Weaknesses of OLS

The technique is not perfect and is subject errors that arise from its linear algebra implementation and the inherent biases in the data that it is modelling, not to mention also that OLS cares about one thing only: minimizing the SSR. The issues from the linear algebra stem from the fact that the entries of X^TX are sums of potentially large powers over potentially long lists (i.e., huge quantities). This leads to an issue in numerical linear algebra known as *ill-conditioning*. The issues stemming from the data itself include the influence of outliers and potential *skewness* present in the data set. Statistical inference drawn from OLS models hinge upon the assumption that the regressed data follow a *Gaussian distribution*. OLS may be deployed on a *skewed* data set and thus hinder the quality of the statistical inference. Finally, since OLS only cares about SSR, then it may miserably fail to model physical relations. In 'wanting' to minimize the SSR, OLS may produce a curve fit with features unsuitable for the problem at hand. There are a multitude of other phenomena that hinder OLS, however, in this paper only the issues of ill-conditioning, skewness, and physical modelling will be addressed. This section will elaborate upon these issues and the next section will address how to solve or at least mitigate them.

Ill-conditioning (of X^TX)

In numerical linear algebra this term refers to the sensitivity of the solution vector (x) to changes in the either coefficient matrix (A) or the *RHS* vector (b). To contextualize this to OLS, this would refer to the sensitivity of the values of the parameters a_i to the independent data that makes up X^TX and the regressor choices or the *RHS* vector X^Ty. These "small" changes in b or A are introduced by the finite precision representation of numbers in computers and any ensuing round-off errors. A linear system, if sufficiently ill-conditioned, may be solved inaccurately. The degree of ill-conditioning is measured by a quantity called the *condition number* which is defined as:

$$\kappa(A) = \|A^{-1}\|\|A\| \geq 1$$

Where the "$\|\cdot\|$" operator denotes the L^p norm, and for a linear system the sensitivity of x to changes in b and the sensitivity of x to changes in A are given by:

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|}$$

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leq \kappa(A)\frac{\|\delta A\|}{\|A\|}$$

For matrices norms can be defined in many ways:

$$\|A\|_1 = \max_{1\leq j\leq n}\sum_{i=1}^{n}|a_{ij}|$$

$$\|A\|_\infty = \max_{1\leq j\leq n}\sum_{j=1}^{n}|a_{ij}|$$

$$\|A\|_2 = \min\lambda_i$$

$$\|A\|_e = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}^2}$$

All the above are valid choices for a norm that will characterize a condition number. $\kappa(A)$ close to *1* corresponds to linear systems $Ax = b$ that are well-conditioned, whereas $\kappa(A)$ that is arbitrarily large corresponds to ill-conditioned systems. Well-conditioned systems are characterized by coefficient matrices *A* whose entries are relatively equal in magnitude whereas ill-conditioned systems feature *A* with entries vastly different in magnitude. The latter is the case for OLS with polynomial regression where the polynomial degree is high. Addition of Laurent polynomials may increase the condition number of X^TX if the data is not sufficiently close to the origin. Regardless, the higher $\kappa(X^TX)$, the greater the error in a_i when computed from OLS. "A rule of thumb is to expect loss of *k* digits of accuracy if $\kappa \approx 10^k$." (Cheney and Kincaid)

Skewness

This is a measure of how asymmetric the distribution of a data set is and said measures is taken about the set's arithmetic mean. A quantity used to measure skewness is the so-called 3rd standardized moment defined as:

$$\overline{\mu_3} = E\left[\left(\frac{X - \mu}{\sigma}\right)^3\right]$$

Where *E* is the expectation operator, *X* is a random variable, μ is the set's (sample) arithmetic mean, σ is the set's (sample) standard deviation, and μ_3 is the 3rd standardized moment (hereto referred to as "skewness"). A perfectly symmetric data set has no skewness ($\mu_3 = 0$). When the data set appears to favor the left side of an axis the ensuing skew is positive and conversely, when the bias is towards the right the skewness is negative.

The reason that skewness is being mentioned here is because one of the most common objectives of an OLS analysis is to draw inference or the so-called statistically significant results. These conclusions rely heavily on the assumption that one is working with samples drawn from populations that follow a *Gaussian distribution* (also called *normal* distribution). By that assumption, the sample should also resemble said distribution. A prime feature of the Gaussian distribution is that it is symmetric. Therefore, the skewness is a metric that helps the data scientist assess whether the samples that are being worked with follow the Gaussian distribution. Samples with relatively no skewness may be safely processed using OLS without any treatment (leading up to the statistical inference). Skewed samples, however, should be "treated" prior to the OLS phase. At no point in the formulation of OLS was the skewness considered, therefore blind use of OLS may lead the data scientist to make fundamental mistakes.

SSR "tunnel vision"

OLS is a systematic method designed to minimize the *SSR*, which is ultimately a quantity. OLS is not inherently designed to encapsulate the behavior between *y* and *x*; its usefulness as a predictive model is a collateral of the fact that the true behavior between *x* and *y* would indeed have an *SSR* of 0. Therefore, the smaller the *SSR*, the closer the model should be to the truth. Nevertheless, the point stands: OLS cares only about *SSR* regardless of context, and nothing else. This built-in tunnel vision can lead the OLS to produce curve fits which may be nonsensical once brought into context. This is of particular importance when the curve fit is used for physical modelling.

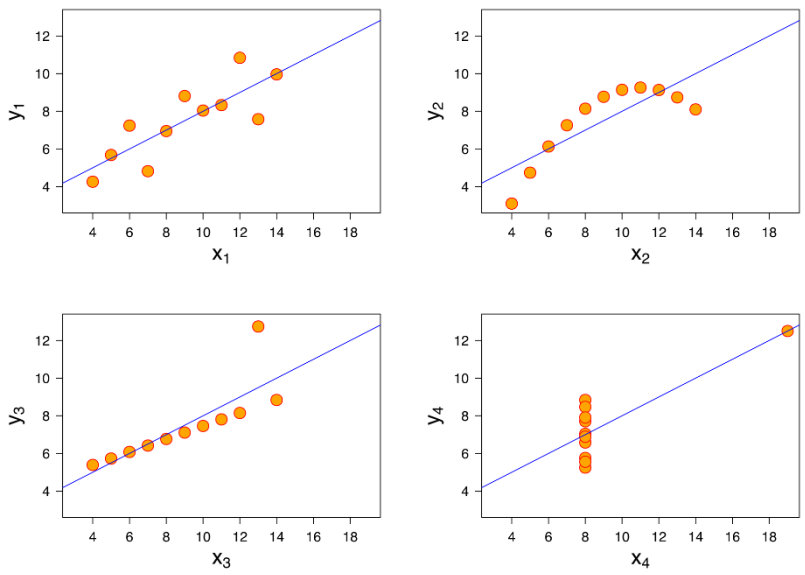

Figure 3: Anscombe's quartet

Sometimes, however, the results of an OLS analysis will produce curve fits, which if applied to engineering and science problems, could be used to make predictions that are an egregious neglection of physics. To illustrate this point, the following aerodynamics example is used: Boundary layer data consisting of height and velocity measurements was produced in an experiment. An intermediate goal is to curve-fit the data to determine if the boundary layer is laminar or turbulent. OLS so far has been presented in its unconstrained form, the next section develops its constrained form.

Overfitting

Recall that a measure of "goodness of fit" is the coefficient of determination R^2. The closer R^2 is to *1* to smaller the *SSR*. In practice, simple OLS models may struggle to achieve such R^2 values outside of perhaps trying to approximate an analytic inverse function. Real data sets feature enough noise that simple regressors cannot perfectly handle. To nevertheless improve the R^2 one may increase the number of regressors (within existing predictor variables or as part of new predictors). This is said to increase the "power" of the curve as it is now able to flex more into shaping the data set. Presumably the curve runs closer to more data points, therefore, its R^2 will increase. This may seem good at first and may as well be, but there is a caveat: Why not keep adding regressors until $R^2 = 1$? The answer is that one shouldn't because at some point additional regressors stop improving the OLS model and start hindering it, thus, leading up to the issue of *overfitting*. This is a term used to describe statistical models that are extremely good at modelling the so-called training data set but are poor at modelling test or validation data sets. In practice data scientists will only perform an OLS analysis on a subset of their available data (called a training data set) to produce their curve fits, leaving a remainder of data which they may further subset to make so-called test data sets.

To help quantify the degree of overfitting the quantity of *adjusted* R^2 was devised. It is defined as:

$$\bar{R}^2 = 1 - (1 - R^2)\frac{N-1}{df}$$

$$df = N - \#regressors$$

Where *"df"* are the *degrees of freedom* defined as the number of observations *N* minus the total number of regressors across all predictor variables (including the constant regressor a_i). The ratio of *N – 1* to *df* is bounded between *1* and *0* without inclusion of those limits.

Splitting the data this way enables the use of *bootstrapping*, a technique whereby many permutations of training data sets (from the same pool of data) are run through an OLS analysis to produce a range of possible curve fits.

Chapter 6: Improving OLS

This section discusses methods that mitigate the issues discussed in the previous sections. The measures discussed in this section do not eliminate the respective problems that they are meant to tackle, rather, they suppress them to an acceptable extent.

Lowering the condition number: Standardized variables

In statistical analysis one can treat the random variables being worked with by means of *standardization*. This consists of transforming the observations of data from a "raw" state *X* to a "standard" state *Z* through the transformation:

$$Z = \frac{X - \mu}{\sigma}$$

Where *Z* is called a standardized variable. The effect that this transformation has is to effectively reduce the magnitude of the observations by a factor of σ, the standard deviation of the sample. Standardization is essentially a translation followed by a scaling, both of which do not impact the symmetry of the sample. The effect of standardization is to produce data points that are closer together (their differences in magnitude shrink). The idea is then to run the OLS analysis on *Z* as opposed to *X*. The only caveat is that the inputs to the ensuing OLS model cannot be raw, rather, standardized.

Symmetry through redistribution of data: Logarithms

When the raw data of a sample exhibits unacceptable skew, incorrect inferences from an OLS may be drawn. If the raw data itself is not symmetric, then the data scientist should look for some aspect of it that is. Positively skewed data can be (somewhat) evenly unraveled by taking its logarithm (may be the natural one or with any base greater than 1). Negatively skewed data may be evenly unraveled by exponentiating a base to a power equivalent to the raw data. These are two options to treat different kinds of skew. The OLS should then be run on the logarithm or exponents of the raw data that exhibits the skew. The data need not be in its raw state, skew can be treated on the standardized state of the data. These "treatments" are permitted as they are in accordance with the formulation of OLS. Although they essentially instruct the data scientist to use transcendental regressors, these treatments don't prescribe *parameters* within the transcendental operations. Therefore, the OLS remains linear and the skew treatments are as simple as swapping the raw (or standard) data with its treated version at the input stage.

Bringing Context into play: Lagrange Multipliers

Say that the curve fit must satisfy constraints that force it to go through specified points given as coordinate pairs like x_k and y_k. The subscript "*k*" is the constraint number, which will range from 0 to an arbitrary value "*K"*. The constraints on a univariate Laurent polynomial may be written as:

$$y_k = P_{n,m}(x_k) = \sum_{i=m}^{n} a_i x_k^i$$

Or, in expanded univariate form:

$$y_k = \frac{a_{-m}}{x_k^m} + \frac{a_{1-m}}{x_k^{m-1}} + \cdots + \frac{a_{-2}}{x_k^2} + \frac{a_{-1}}{x_k} + a_0 + a_1 x_k + a_2 x_k^2 + \cdots + a_{n-1} x_k^{n-1} + a_n x_k^n$$

The enforcement of these constraints along with the optimization of SSR introduces complexity because the linear system used to determine the coefficients becomes over-constrained: There are more equations than there are unknowns. It is possible to avoid this issue by developing a linear system whereby the curve is first constrained and then have the gradient of SSR be reformulated and set equal to zero. Said approach is a nightmare though because it involves gargantuan algebraic substitutions. Instead, the more "digestible" method of Lagrange multipliers is employed to proceed:

Development of CUP-OLS (Constrained Univariate Polynomial OLS):

Define the Lagrangian as:

$$\mathcal{L} = f_{obj}(x) + \sum_{k=1}^{K} \lambda_k g_k$$

Where:

- "g_k" is a constraint in the form [$y_k - f_{obj}(x_k) = 0$]
- "f_{obj}" is an objective function that is to be minimized subject to all g_k.
- "λ_k" is a Lagrange multiplier.
- "K" is the number of constraints imposed on the problem.
- "k" is the constraint number, a counter that goes from *1* to *K*.

In constrained OLS the objective function "f_{obj}" is the SSR, whilst "g_k" represents the equality constraints. The OLS Lagrangian is a function, therefore, of the curve-fit coefficients "a_i" and the Lagrange multipliers "λ_k:"

$$\mathcal{L} = \sum_{o=1}^{N} \varepsilon^2 + \sum_{k=1}^{K} \lambda_k g_k = f(x_0, a_1, a_2, \dots, a_n, \lambda_1, \lambda_2, \dots, \lambda_K)$$

Rewriting:

$$\mathcal{L} = \sum_{o=1}^{N} \left(y_o - P_{m,n}(x_o)\right)^2 + \sum_{k=1}^{K} \lambda_k \left(P_{m,n}(x_k) - y_k\right)$$

Rewriting further:

$$\mathcal{L} = \sum_{o=1}^{N} \left(y_o - \sum_{i=m}^{n} a_i x_o^i\right)^2 + \sum_{k=1}^{K} \lambda_k \left(\sum_{i=m}^{n} a_i x_k^i - y_k\right)$$

The Lagrange multipliers are collaterally additional unknowns that are added to address the issue of over-constraining the linear system. Instead of minimizing the SSR directly, one now minimizes the Lagrangian by setting its gradient equal to the zero vector:

$$\nabla\mathcal{L} = \left[\frac{\partial\mathcal{L}}{\partial a_{-m}} \quad \cdots \quad \frac{\partial\mathcal{L}}{\partial a_0} \quad \cdots \quad \frac{\partial\mathcal{L}}{\partial a_n} \quad \frac{\partial\mathcal{L}}{\partial\lambda_1} \quad \frac{\partial\mathcal{L}}{\partial\lambda_2} \quad \cdots \quad \frac{\partial\mathcal{L}}{\lambda_K}\right] = \vec{0}$$

Minimization of the Lagrangian with strictly equality constraints requires two different types of partial derivatives: Those with respect to the curve fit coefficients (a_i), and those with respect to the Lagrange multipliers (λ_k).

$$\frac{\partial\mathcal{L}}{\partial a_i} = -2\sum_{o=1}^{N}\left[\left(y_o - \sum_{i=m}^{n} a_i x_o^i\right)x_o^I\right] + \sum_{k=1}^{K}\lambda_k x_k^I = 0$$

$$\frac{\partial\mathcal{L}}{\partial\lambda_k} = \sum_{i=m}^{n} a_i x_k^i - y_k = 0$$

From the above two (2) expressions, the minimization conditions are obtained below:

$$2\sum_{o=1}^{N}\left[\left(\sum_{i=m}^{n} a_i x_o^i\right)x_o^I\right] + \sum_{k=1}^{K}\lambda_k x_k^I = 2\sum_{o=1}^{N} y_o x_o^I$$

$$\sum_{i=m}^{n} a_i x_k^i = y_k$$

These two expressions when written for all coefficients of $P_{n,m}$ and all constraints g_k produce a linear problem in "$m + n + K + 1$" unknowns. In terms of the "linear algebra intuition," the new linear system is not much more complicated for it can be built by concatenating the Design matrix and its right-hand-side vector with coefficients doubled as follows:

$$\begin{bmatrix} 2V_{n,m} & A^T \\ A & \vec{0} \end{bmatrix}\begin{bmatrix} \vec{a}_{n,m} \\ \vec{\lambda} \end{bmatrix} = \begin{bmatrix} 2\overrightarrow{RHS}_{n,m} \\ \overrightarrow{RHS}_k \end{bmatrix}$$

Where:

- "A" is a rectangular coefficient matrix defined by the constraints imposed.
- "RHS_k" and the "λ" vector are extensions of the right-hand-side and unknown coefficient vectors to grant the Lagrange multipliers their place in the equation.
- The "$\vec{0}$" is a $K \times K$ matrix of zeros.

Ill-Conditioning:

Design matrices are easily subject to ill-conditioning because the coefficients vary greatly in magnitude. This is because the coefficients are defined by sums of potentially large powers of several data points. This means that the above system, while providing an analytical solution to the coefficients, may be

challenging to evaluate numerically. When solving a least-squares problem, one must be mindful of the digital precision used. When using scripting utilities such as MATLAB® or Python, it may be necessary to deploy scaling algorithms, which, provide more accurate numeric evaluation of the curve fit coefficients at a minor computational cost. If the data sets being that one works with are large and a high-order curve fit is needed, it may even be necessary to enable Variable Precision Arithmetic (VPA) features to achieve reasonable accuracy (simply owing to the data set being formidably large). From here on out, the above system is condensed to a simple matrix equation:

$$V_n\vec{a} = \overrightarrow{RHS}_n$$

Building the UUPOLS code:

Key to the code that computes Laurent regression is an understanding of its matrix builder. To exemplify it, consider the linear equation in six (6) unknowns below which corresponds to an OLS problem for a Laurent polynomial $P_{2,3}$. Note the color-coding scheme meant to enhance the ongoing explanation.

$$\begin{bmatrix} \sum_{o=1}^{N}\frac{1}{x_o^4} & \sum_{o=1}^{N}\frac{1}{x_o^3} & \sum_{o=1}^{N}\frac{1}{x_o^2} & \sum_{o=1}^{N}\frac{1}{x_o} & \sum_{o=1}^{N}1 & \sum_{o=1}^{N}x_o \\ \sum_{o=1}^{N}\frac{1}{x_o^3} & \sum_{o=1}^{N}\frac{1}{x_o^2} & \sum_{o=1}^{N}\frac{1}{x_o} & \sum_{o=1}^{N}1 & \sum_{o=1}^{N}x_o & \sum_{o=1}^{N}x_o^2 \\ \sum_{o=1}^{N}\frac{1}{x_o^2} & \sum_{o=1}^{N}\frac{1}{x_o} & \sum_{o=1}^{N}1 & \sum_{o=1}^{N}x_o & \sum_{o=1}^{N}x_o^2 & \sum_{o=1}^{N}x_o^3 \\ \sum_{o=1}^{N}\frac{1}{x_o} & \sum_{o=1}^{N}1 & \sum_{o=1}^{N}x_o & \sum_{o=1}^{N}x_o^2 & \sum_{o=1}^{N}x_o^3 & \sum_{o=1}^{N}x_o^4 \\ \sum_{o=1}^{N}1 & \sum_{o=1}^{N}x_o & \sum_{o=1}^{N}x_o^2 & \sum_{o=1}^{N}x_o^3 & \sum_{o=1}^{N}x_o^4 & \sum_{o=1}^{N}x_o^5 \\ \sum_{o=1}^{N}x_o & \sum_{o=1}^{N}x_o^2 & \sum_{o=1}^{N}x_o^3 & \sum_{o=1}^{N}x_o^4 & \sum_{o=1}^{N}x_o^5 & \sum_{o=1}^{N}x_o^6 \end{bmatrix} \begin{bmatrix} a_{-2} \\ a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \sum_{o=1}^{N}\frac{y_o}{x_o^2} \\ \sum_{o=1}^{N}\frac{y_o}{x_o} \\ \sum_{o=1}^{N}y_o \\ \sum_{o=1}^{N}y_o x_o \\ \sum_{o=1}^{N}y_o x_o^2 \\ \sum_{o=1}^{N}y_o x_o^3 \end{bmatrix}$$

- The purple term in the Design matrix is named the "pivotal" coefficient. It corresponds to the constant coefficient a_0 and its respective RHS coefficient
- The cyan counter diagonal is called the "pivotal" counter diagonal as it includes the pivotal coefficient.
- The green terms are dubbed the "lower L" (in reference to their "L" shape). The number of coefficients that make up the "L" equals the number of counter diagonals in any matrix and one of its coefficients is shared with the pivotal counter diagonal.
- The yellow terms are dubbed the "upper L" and include the pivotal coefficient. It has a mirror image about the main pivotal diagonal. It also has corresponding coefficients in the RHS vector.

Let “R” denote the rows of the Design matrix for an unconstrained univariate Laurent regression problem. The number of counter-diagonal (and main-diagonal) bands is given by:

$$B = 2R - 1$$

“B” is important because it equals the number of unique coefficients that must be determined to build the “V” matrix. These coefficients need be computed only once, then, their linear indices are used to populate their corresponding counter-diagonal bands. This code is originally developed in MATLAB®, therefore, the linear index notation follows a “column-major ordering” convention. These indices are related to the band numbering scheme, beginning with the pivotal band number:

$$B_P = 1 + 2m$$

```
%b1 = @(Bn) Bn*(1-floor(Bn/R)) + R*(Bn-R+1)*floor(Bn/R);
b1 = @(Bn) Bn*(1-floor(Bn/m1n)) + (R*(Bn-m1n) + m1n)*floor(Bn/m1n);
b2 = @(Bn) (R*(Bn-1)+1)*(1-floor(Bn/m1n)) + ((m+n)*R + 1 + Bn - m1n)*floor(Bn/m1n);
%b2 = @(Bn) (R*(Bn-1)+1)*(1-floor(Bn/R)) + ((R-1)^2+Bn)*floor(Bn/R);
```

Works Cited

Cheney, Ward and David Kincaid. *Numerical Mathematics and Computing*. Ed. Barbara Willete. 6th. Bob Pirtle, 2008.

Gavin, Henry P. *Constrained Linear Least Squares*. Duke University Department of Civil and Environmental Engineering, 2015.

Hoffman, Joe D. *Numerical Methods for Engineers and Scientists*. 2nd. Basel: Marcel Dekker Inc., 2001.