2021

Using Semialgebraic Parametric Analysis by Metaprogramming in Portfolio Optimization

Philip Blaine Meersman
University of Kentucky, philip@themeersmans.com

Using Semialgebraic Parametric Analysis by Metaprogramming in Portfolio Optimization

Submitted as the Culmination of
Undergraduate Research in Mathematics

Philip Meersman

Abstract

One classic problem in quantitative finance is portfolio optimization, which consists of assigning weights to assets in a portfolio to maximize one's expected return while keeping the level of risk at a desired level. This problem can be modeled as a linear program (LP), using a risk aversion parameter mu. For a given single value of mu, the LP can be solved using any standard LP solver. In this work, however, the problem is considered parametrically: the optimal solution is sought for every possible value of mu. This describes how weights to the portfolio assets would be assigned from the timid investor to the bold. This is accomplished by applying the novel technique of semialgebraic parametric analysis by metaprogramming (SPAM). Demonstrated in this talk is the method of applying SPAM to a textbook example of portfolio optimization. Generated in this way are numerical and symbolic representations of the solution set as well as a graphical representation of these results.

University of Kentucky
Lewis Honors College
Bachelor of Science in Mathematics
Under the supervision of Dr. Yuan Zhou
May 2021

Contents

1 Introduction

Computer assistance in solving economic problems has recently become commonplace. In undergraduate economics courses students now learn to use Microsoft Excel, JMP, Stata, and other statistical software packages to expedite the once-tedious process of estimating regressions and computing variance, among other applications. One area of research benefiting from the propagation of computer programming is that of computer-assisted proofs. One such method of generating computer-assisted proofs is termed semialgebraic parametric analysis by metaprogramming (SPAM).

This paper first introduces a few toy examples demonstrating and describing the functionality of SPAM before illustrating its utility in two trivial applications from economics. Finally, the majority of the paper is spent discussing a more interesting economic application of SPAM: portfolio optimization. Note that we consider portfolio optimization in the context of Vanderbei's *Linear Programming* [11].

While the topics in this paper are somewhat interesting and do demonstrate the capability of SPAM, they in no way attempt to capture its complete power. SageMath terminal and notebook output are provided in the Appendix, while certain useful figures are included throughout the paper.

2 Traditional Portfolio Optimization

One common undergraduate problem in financial economics is that of choosing weights to assign to assets in a portfolio to maximize one's expected return. In our experience this problem is first presented while involving two assets to choose from: one safe and one risky. The problem setup often specifies that the risky asset has a higher expected return than its safe counterpart, but with some non-zero probability may manifest a negative return. In solving problems of portfolio optimization it is important for one to keep in mind the type of problem they are solving. Depending on the formulation and desired format of solution, these problems can involve a variety of different variables, constraints, parameters, types of optimization, and conventions of notation. In this paper we use the framework detailed in Vanderbei [9], and specifically his section on the parametric simplex method.

2.1 Vanderbei's Formulation

This simple formulation of the portfolio optimization problem naturally extends to the slightly more complicated problem of assigning weights to assets in a portfolio when each asset has its own, unique, expected value and unique level of risk. We consider a problem proposed by Vanderbei in which nine risky assets are considered as possible portfolio pieces and their associated levels of risk are computed using historical data from 24 time periods. Historical data can be found in the Appendix.

Adopting Vanderbei's notation, let the potential assets be indexed from 1 to n and let R_j denote the return on investment j, with $j = 1, ..., n$. Note that this R_j is a random variable. A portfolio is comprised of a series of fractions of one's wealth that are assigned to some or all of the assets available. Let x_j with $j = 1, ..., n$ denote the fraction of one's wealth invested in asset j. Importantly, we must have that $\sum_{j=1}^{n} x_j = 1$. Then let the total return garnered by any given portfolio be:

$$R = \sum_{j=1}^{n} x_j R_j$$

While the reward associated with this portfolio is defined in terms of the expected return:

$$\mathbf{E}(R) = \sum_{j=1}^{n} x_j \mathbf{E}(R_j)$$

It is important to recognize the context of these portfolio optimization problems. For one, if the investor cares not about risk, then she will simply put all of her money into the asset with the highest expected return. This problem, while perhaps sometimes descriptive, is not interesting for use with SPAM. Instead, we consider a risk-averse decision maker. While there are several different interpretations and associated implications for what it means to measure risk aversion, we will adopt Vanderbei's basic metric: any number. Let $\mu \in [0, \infty)$ be one's level of risk aversion (their importance of risk relative to reward), with lower μ corresponding to a less risk-averse decision maker and higher μ corresponding to a more risk-loving decision maker. Intuitively, as μ approaches ∞ this problem becomes more and more similar to the problem involving someone who does not care at all about risk. In the case of extremely large μ the decision maker will choose the asset with the highest expected return.

Just as there are many definitions of measures of risk-aversion, so too are there many definitions for risk itself. Vanderbei considers for this application the *mean absolute deviation from the mean (MAD)* as a definition for risk:

$$\mathbf{E}|(R - \mathbf{E}(R))| = \mathbf{E}(\left|\sum_{j=1}^{n} x_j (R_j - \mathbf{E}(R_j)\right|)$$

Now having a measure of risk to consider, we are interested in the individual trying to maximize expected return subject to whatever constraint their level of risk aversion imposes on them. Risk and reward can be combined with the risk aversion parameter μ to produce:

$$\zeta = \mu \sum_j x_j \mathbf{E}(R_j) - \mathbf{E}(\left|\sum_j x_j (R_j - \mathbf{E}(R_j)))\right|$$

This defines the decision maker's objective function, valuing reward and disliking risk. Notice that as μ tends toward ∞ the positive reward term will dominate the negative risk term, consistent with the description above.

Note: In this chapter Vanderbei briefly describes the *hedging* process. For our purposes, hedging can simply be thought of as reducing risk without reducing expected reward, and if such an opportunity exists one would take it. This obviously has many extensions in financial economics but is not a focus of our work.

In order to evaluate ζ it appears one must know the joint distribution of the R_j's. While this could be given as part of a problem's description, generally distributions of financial data are not known with certainty, but can be estimated using historical financial data. This is what Vanderbei does. Located in the Appendix is a table of historical data used by Vanderbei in his example. This data will be used to estimate the expected returns of each asset for the purposes of portfolio optimization.

Now, let $R_j(t)$ be the return on investment j during time period t with $t \in T = \{1, 2, ..., 24\}$. Using the average of the historical returns for each asset, the expected return for each is denoted:

$$\mathbf{E}(R_j) = \tfrac{1}{T} \textstyle\sum_{t=1}^{T} R_j(t)$$

This problem can be compiled to the form:

$$\begin{array}{ll} \text{maximize} & \mu \sum_j x_j r_j - \frac{1}{T} \sum_{t=1}^{T} \left| \sum_j x_j (R_j(t) - r_j) \right| \\ \text{such that} & \sum_j x_j = 1 \\ & x_j \geq 0 \qquad j = 1, 2, ..., n \end{array}$$

with

$$r_j = \tfrac{1}{T} \textstyle\sum_{t=1}^{T} R_j(t)$$

This problem includes several absolute values in the objective function, which are inconvenient. In order to work around this, Vanderbei introduces a new variable, y_t, to replace each absolute value. By imposing new inequality constraints on y_t, the absolute values can be accurately represented in the linear program. As stated, replace $\left| \sum_j x_j (R_j(t) - r_j) \right|$ with y_t to obtain our problem as

$$\begin{aligned}
\text{maximize} \qquad & \mu \sum_j x_j r_j - \frac{1}{T}\sum_{t=1}^{T} y_t \\
\text{such that} \qquad & \sum_j x_j = 1 \\
& -y_t \le \sum_j x_j (R_j(t) - r_j) \le y_t \qquad t = 1, 2, ..., T \\
& y_t \ge 0 \qquad j = 1, 2, ..., n \\
& x_j \ge 0 \qquad j = 1, 2, ..., n
\end{aligned}$$

This problem can be solved for any desired value of μ. However, we have tools to systematically solve this parametric linear program for *every* value of μ using the parametric simplex method. This is the manual process that SPAM expedites and automates.

Note: In this context, a problem of portfolio optimization is defined by the number of assets available, their historical data, and μ. This is all that is necessary to define such a problem.

2.2 Parametric Simplex Method

To use the parametric simplex method, one must choose a starting value of the parameter. It is desirable to choose a value for μ that results in an "obvious" basic optimal solution. If this is possible we can then begin experimenting with changes in the parameter value. In the case of a portfolio optimization problem, it is natural to choose a value for μ that will result in $x_j^* = 1$ (if possible) where j^* is the asset with highest expected return. To perform the parametric simplex method Vanderbei introduces slack variables w_t^+ and w_t^- and rewrites the problem in dictionary form:

$$\begin{array}{ll}
\text{maximize} & \mu \sum_j x_j r_j - \frac{1}{T}\sum_{t=1}^{T} y_t \\
\text{such that} & -y_t - \sum_j x_j(R_j(t) - r_j) + w_t^- = 0 \\
& -y_t + \sum_j x_j(R_j(t) - r_j) + w_t^+ = 0 \\
& \sum_j x_j = 1 \\
& y_t \geq 0 \qquad j = 1, 2, ..., n \\
& x_j \geq 0 \qquad j = 1, 2, ..., n
\end{array}$$

Still assuming that we have chosen μ such that $x_j^* = 1$, the other x_j's disappear. Remaining variables that are positive are considered basic and the rest are considered nonbasic. So, x_j^* is basic, while the rest of the x_j's are nonbasic. Also, y_t's are nonzero and thus basic. For each t, one of w_t^+ or w_t^- is basic while its counterpart is nonbasic. To state which is basic, Vanderbei introduces some more notation. Let

$$D_{tj} = R_j(t) - r_j$$

Then if $D_{tj}^* > 0$, w_t^- is basic. If $D_{tj}^* < 0$, w_t^+ is basic. If it occurs that $D_{tj}^* = 0$, one can arbitrarily decide while of the w_t is basic. Let

$$T^+ = \Bigg\{ t : D_{tj}^* > 0\} \text{ and } T^- = \{t : D_{tj}^* < 0\}$$

and

$$\epsilon_t = \{1 \text{ for } t \in T^+$$
$$\epsilon_t = \{-1 \text{ for } t \in T^-$$

Then after solving this via parametric simplex method, the optimal dictionary is:

$\zeta = \frac{1}{T}\sum_{t=1}^{T} \epsilon_t D_{tj}^* - \frac{1}{T}\sum_{j\neq j^*}\sum_{t=1}^{T} \epsilon_t (D_{tj} - D_{tj}^*) x_j$

$- \frac{1}{T}\sum_{t\in T^-} w_t^- - \frac{1}{T}\sum_{t\in T^+} w_t^+ + \mu r_j^* + \mu \sum_{j\neq j^*}(r_j - r_j^*) x_j$

---

with

$$\begin{array}{llrcllll}
y_t & = & -D_{tj}^* & - & \sum_{j\neq j^*}(D_{tj}-D_{tj}*)x_j & +w_t^- & & t\in T^- \\
w_t^- & = & 2D_{tj}^* & + & 2\sum_{j\neq j^*}(D_{tj}-D_{tj}*)x_j & & +w_t^+ & t\in T^- \\
y_t & = & D_{tj}^* & + & \sum_{j\neq j^*}(D_{tj}-D_{tj}*)x_j & & +w_t^+ & t\in T^- \\
w_t^+ & = & -2D_{tj}^* & - & 2\sum_{j\neq j^*}(D_{tj}-D_{tj}*)x_j & +w_t^- & & t\in T^- \\
x_j^* & = & 1 & - & \sum_{j\neq j*}x_j & & &
\end{array}$$

For large values of μ, it is easy to check that this dictionary is optimal. The objective coefficients on w_t^- and w_t^+ in the objective function are both negative. Coefficients on x_j's in the first row of the objective function can be positive or negative depending on the differences $D_{tj} - D_{tj}^*$, but for a μ large enough, the coefficients on x_j's in the second row will grow dominant over the first row coefficients. This forces the coefficients on x_j's negative. From Vanderbei's definition of T^- and T^+, all basic variables here are positive.

Notice that as μ is allowed to vary (decrease, in this case) the coefficients on some x_j's will eventually become zero and then positive. As this happens, the parametric simplex method sends the x_j corresponding to the now-positive coefficient into the basis. As usual, the ratio test is conducted on coefficients containing μ in the old dictionary to determine which variable will leave the basis. Once this pivot has been performed, we are left with a new dictionary corresponding to a new value of μ. The μ value for which one coefficient on the x_j's changes from negative to positive is a possible boundary value for μ at which the objective value (and hence optimal solution to the linear program) changes. This process can be repeated as many times as necessary until μ becomes 0. In general, the parametric simplex method is not bounded by a parameter value of 0, but in this case of portfolio optimization $\mu < 0$ does not make sense.

After performing the parametric simplex method to completion, one has a series of *μ-intervals*. Each μ-interval is bounded by values of μ for which the objective function value changes. Within any given μ-interval, any value of μ will result in the same objective value and optimal solution. This is the result of the parametric simplex method. In the (still generalized) case of portfolio optimization, one also obtains the objective value and associated optimal solution for each μ-interval. If thought of as percentages, the x_j's can be plotted against the list of μ-intervals to produce a graphical representation of the solution set for any given problem. This, along with the so-called *efficient frontier*, is what is rediscovered using SPAM.

3 SageMath

Now that Vanderbei's method is understood, we will introduce our work with SPAM and the problem of portfolio optimization. Just as any problem must be translated from plain text to function in a programming language, so too must our problem. Personally, I began this project with little knowledge of Python and no knowledge of SageMath, so there was much time spent familiarizing ourselves with the two.

3.1 SageMath Setup

For our project we used SageMath from source (various versions between 8.9 and 9.3.beta6) via Ubuntu on the Windows Subsystem for Linux. We used SageMath both in terminal and in Jupyter notebook. Most of our work in the terminal was done by loading .sage files from our local machine to the terminal each session. We saved output often as .txt files and stored them locally. Work in notebooks was also done using .ipynb files saved locally. In most cases we pushed/pulled changes using GitHub Desktop, although we did also use git version control from the Linux terminal (this was necessary to pull ticket changes from SageTrac before they were published in a stable release of SageMath). While I did attempt to work on SageMath development, I found this less interesting and more alien than using SageMath, so most of our time was spent working with the software.

3.2 SPAM

Originally developed to automate the discovery of new cut-generating functions in linear programming [7], SPAM is an algorithm applied to an existing program, P, which exhibits certain nice properties. Without a standalone program to feed as input into SPAM, it does nothing on its own.

3.3 Requisite Properties of P

First, the mathematical problem being solved by P must be parameterized. One or more parameters of interest must appear in P, otherwise P simply solves one concrete problem and there is nothing interesting to be gained from SPAM. The parameters are often grouped in a fixed tuple, σ. These parameters are explicitly declared as elements of a Parametric Real Field,

K. Creating such a field indicates to SPAM which parameters to recognize throughout the execution of P.

Second, P must rely only on algebraic operations, branches, and comparisons. Specifically, the algebraic operations conducive for use with SPAM are elementary algebraic operations on real numbers. Branches allowed are traditional if-then-else statements and loops. These properties ensure that the set of parameters where SPAM returns any particular outcome is a semi-algebraic set. This is necessary for SPAM, and rules out programs relying on more sophisticated operations.

Also important to note, SPAM is formatted for Python, and specifically within the SageMath computer algebra software [10]. Naturally, programs we discuss and introduce are written in Python and are all executed in SageMath. This paper contains a combination of programs previously existing in Python libraries, programs written by Dr. Zhou, and programs written by myself.

In short, there are three components required for an effective use of the SPAM algorithm.

1. A program (algorithm) which acts on a parameter tuple, $P(\sigma)$
2. A proof cell generated by metaprogramming, $SPAM(P(\sigma))$
3. A *proof cell complex* combining every proof cell generated by the SPAM algorithm, $SPAM^*(P)$

In short, this constitutes the process of using SPAM by applying it to an existing Python program. The result can vary from problem to problem, but is usually depicted in graphical form in some capacity. Since SPAM records comparisons in the form of inequalities, it is natural to plot those inequalities to separate proof cells by output value.

Note: In order to use SPAM to generate precise inequality comparisons, we use rational-valued numbers in computation. Because of the approximation involved in computing with floating point numbers, rationals are more appropriate for SPAM. We refer to numbers used within our programs in fractional form.

3.4 Example: Positive Definite Matrix

Consider a program P which takes a matrix as input and yields as output a Boolean True if the matrix is positive definite and False otherwise. In Python this program is actually the **is_positive_definite** method from class *Matrix*. This is a very elementary program but does well to illustrate the use of SPAM without having to digest mathematics simultaneously. Consider the matrix

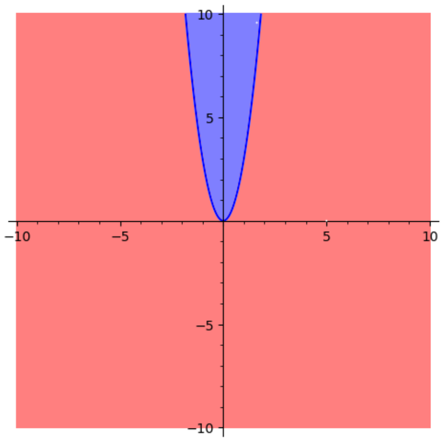
$$A = \begin{bmatrix} \frac{1}{3} & a \\ a & b \end{bmatrix}$$

One may ask the question "For which values (a, b) is this matrix positive definite?" In this example one can compute the principal minor of A and generate an inequality solution to this question by hand. Since this is not a complicated question, one might suggest an answer in the form of a theorem. Theorem 1: A is positive definite if and only if the following holds

$$\tfrac{b}{3} - a^2 > 0$$

We now hope that SPAM can provide some sort of proof of this theorem. To accomplish this, we can use **is_positive_definite**, as it relies only on algebraic operations and is parameterized here by (a, b). In handling each iteration of **is_positive_definite**, recall that SPAM records both numerical comparisons (which are generally unseen by the user) as well as the associated symbolic representation of each inequality along the way. These will become the basis for our proof cell complex.

A complete proof cell will contain, at minimum, a list of symbolic expressions corresponding to the inequalities recorded during the execution of P and an optimal solution value corresponding to every parameter value within the proof cell. In Figure 1 each proof cell is distinguished in color by the output of P when given parameter tuple (a, b) with a concrete value. Red cells correspond to matrices that are not positive definite (P output False) while blue cells correspond to matrices that are positive definite (P output True).



The proof cell complex associated with matrix A and **is_positive_definite**.

In this case, the proof cell complex shows all combinations of parameter values that result in a positive definite matrix. This "proves" Theorem 1. So, the usefulness of SPAM here is in creating a comprehensive solution to the parametric problem characterized by the matrix A. The two small white dots (one in the blue region and one in the red) indicate the test points that were used by SPAM to generate their cells. Instead of having to solve this problem for the infinitely many values of (a, b), one can employ SPAM to generate this proof cell complex describing the solution for *every* tuple of parameter values. This is the value SPAM adds, an algorithm that provides complete solutions to parameterized problems.

3.5 Example: Reducing System of Inequalities

Once we understood the basic principle of SPAM, we looked for possible applications in economics. My first attempt in applying SPAM to an economic problem was in reducing a redundant system of inequalities generated by a game of Cournot competition.

The classical two-player game modeling the actions taken between members of a duopoly is solved by obtaining a Nash-Cournot equilibrium. Specifically, this analysis is focused on the old problem concerned with finding optimal levels of research and development in a two-firm setting. This is achieved with restrictions on the five parameters (a, A, β, b, γ) within the d'Aspremont-Jacquemin duopoly model [3]. In their paper three different scenarios are presented, with three different sets of inequalities bounding the parameters. We will focus on their first example here. Six inequalities constitute these restrictions, and are summarized here for reference:

1. $0 < A < a$
2. $0 < \beta < 1$
3. $\frac{(a-A)(2-\beta)}{4.5b\gamma-(2-\beta)(1+\beta)} > 0$
4. $\frac{2(a-A)(4.5b\gamma)}{3b(4.5b\gamma-(2-\beta)(1+\beta))} > 0$
5. $\frac{(a-A)(2-\beta)(1+\beta)}{4.5b\gamma-(2-\beta)(1+\beta)} < A$
6. $\frac{2}{9}(2-\beta)^2 < b\gamma$

If this system is satisfied by values of all five parameters, then the optimal level of research and development as well as the optimal level of profit have meaningful values in this problem. These are enumerated explicitly by d'Aspremont and Jacquemin.

In order to utilize SPAM, one must create (or obtain) a "point-wise prover" program. This program must only function using semialgebraic operations in order to remain suitable for SPAM. For this application, we created a point-wise prover of our own, dubbed "ineq2" [3]. Ineq2 is a simple program designed to test values for each of the five parameters of interest and then return True if the system of inequalities is satisfied and return False if they are not. This simple operation of checking inequalities preserves the desired properties of a semialgebraic solution set.

Once verifying this program's validity, we used it as an input for SPAM. This process involves giving ineq a concrete set of arguments to pass. In our case, we chose to use $(a, b, A, \beta, \gamma) = (2, 1, 1, 1/2, 10)$ as the starting point for SPAM. All of our analysis takes place in the five-dimensional parameter space. We chose these parameter values because they yield a result of True when passed through ineq and thus satisfy the original set of inequalities from above.

Once SPAM has this concrete-valued tuple it begins its work. We used the breadth-first search (BFS) method from SPAM. With this method, the BFS first executes ineq with the given tuple and records both the Boolean result from ineq as well as the inequalities that are checked. This data is stored in a proof cell by SPAM and eventually constitutes one cell of the complex. Once BFS completion is achieved, analysis of the proof cell complex can begin. In this paper, the description of a proof cell is identical to the set of inequalities which define it, and vice versa.

Upon BFS completion a proof cell complex is yielded. This complex of cells defined by sets of inequalities lives in the parameter space, and each cell is either defined as "True" or "False", depending on the return value from ineq when components of those cells are passed through. In this case five proof cells are generated. One of these cells is "True". This is the cell we are interested in, we will refer to it as "Cell 0".

It is simple enough, once SPAM has finished its work, to access the data stored for each cell in the complex. Cell 0 is described by a system of inequalities, which are listed here:

1. $0 < \gamma$
2. $0 < \beta < 1$
3. $0 < A < a$
4. $0 < b$
5. $-2\beta^2 - 9b\gamma + 2\beta + 4 < 0$
6. $2\beta^2 - 9b\gamma - 8\beta + 8 < 0$
7. $-2a\beta^2 - 9bA\gamma + 2a\beta + 4a < 0$

We were cautious to accept that these two sets of inequalities were equivalent. After all, there are additional linear inequalities as well as one fewer high-degree inequality in this cell than in the original one. To test for equivalency of these two cells we first created a separate program designed to check this, called "ineq_check". As before, let our new set of inequalities be "Cell 0" and the original set be "Cell 1". Ineq_check works in the following way:

1. Generate a tuple in the parameter space with random values for each parameter.
2. Initialize a list to store truth values for each execution.
3. Check this newly generated tuple for membership in Cell 1.
4. Check again for membership in Cell 0.
5. If this tuple is a member of Cell 1 and Cell 0 add "True" to the list.
6. If this tuple is *not* in Cell 1 and also *not* in Cell 0, add "True" to the list.
7. If this tuple is a member of one cell but not the other, add "False" to the list.
8. Repeat the above steps 100,000 times.
9. Check the list for any instances of "False". Return False if so.
10. If no instances of "False" in the list, return True.

After running this program many times and getting a result of True every time, we began to suspect the two cells were identical. Every point we had randomly generated and tested was either contained in both cells or neither cell. We suspected that, with some algebra and rearranging of terms, the following inequalities are implied between the two cells. Inequalities (5), (6), and (7) from Cell 0 are the same as inequalities (4), (6), and (5) from Cell 1:

$$-2\beta^2 - 9b\gamma + 2\beta + 4 < 0 \quad \equiv \quad \frac{2(a-A)(4.5b\gamma)}{3b(4.5b\gamma - (2-\beta)(1+\beta))} > 0 \quad (1)$$

$$2\beta^2 - 9b\gamma - 8\beta + 8 < 0 \quad \equiv \quad \frac{2}{9}(2-\beta)^2 < b\gamma \quad (2)$$

$$-2a\beta^2 - 9bA\gamma + 2a\beta + 4a < 0 \quad \equiv \quad \frac{(a-A)(2-\beta)(1+\beta)}{4.5b\gamma - (2-\beta)(1+\beta)} < A \quad (3)$$

In some sense, Cell 0 is a "simpler" description of the original set of inequalities. It contains one fewer high-degree inequality. However, Cell 0 does impose new constraints on γ and b. This is the trade-off. In using this method for reducing systems of inequalities, one must decide whether they desire fewer high-degree inequalities or fewer total inequalities. In either case, this project has provided an alternative description of the set of inequalities bounding the parameters in d'Aspremont and Jacquemin's analysis of two-firm games.

4 Rediscovery of Textbook Results

4.1 Demonstration on Small Dataset

Prior to discussing the results from the full nine-asset 24-period problem that Vanderbei uses, we introduce a small three-asset two-period model for demonstration. Consider "historical data":

Time period	j=1	j=2	j=3
1	$\frac{5}{3}$	$\frac{7}{3}$	1
2	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{3}{4}$

This data can also be represented as a triplet of tuples

$$((\tfrac{5}{3},\tfrac{2}{3}),(\tfrac{7}{3},\tfrac{1}{3}),(1,\tfrac{3}{4}))$$

with each tuple representing the historical returns for its associated asset. Using dictionary notation and combining coefficients on like terms, this example can be formatted as a linear program:

$$LP = \begin{array}{ll} \text{maximize} & \frac{7}{6}\mu x_1 + \frac{4}{3}\mu x_2 + \frac{7}{8}\mu x_3 - \frac{1}{2}x_4 - \frac{1}{2}x_5 \\ \text{such that} & \frac{1}{2}x_1 + x_2 + \frac{1}{8}x_3 - x_4 \leq 0 \\ & -\frac{1}{2}x_1 - x_2 - \frac{1}{8}x_3 - x_4 \leq 0 \\ & -\frac{1}{2}x_1 - x_2 - \frac{1}{8}x_3 - x_5 \leq 0 \\ & \frac{1}{2}x_1 + x_2 + \frac{1}{8}x_3 - x_5 \leq 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_j \geq 0 \qquad j = 1,2,3 \end{array}$$

Where x_1, x_2, x_3 are weights on assets 1, 2, 3, respectively and x_4, x_5 are variables representing the absolute value terms mentioned earlier. The linear program is not explicitly set up with slack variables included, those are added by the SageMath solver when the program is solved. We wrote a program **setup_portfolio_lp** which takes, as input, a tuple containing historical data and a value for μ, which together define a portfolio optimization problem.

For example, this is the SageMath command and output when **setup_portfolio_lp** is called on this small example with $\mu = 5$:

```
sage: hist_data = ((5/3,2/3),(7/3,1/3),(1,3/4))
sage: mu = 5
sage: lp = setup_portfolio_lp(hist_data, mu)
sage: lp
Linear Program (maximization, 4 variables, 5 constraints)
sage: lp.show()
Maximization:
35/6 x_0 + 20/3 x_1 + 35/8 x_2 - 1/2 x_3 - 1/2 x_4

Constraints:
x_0 + x_1 + x_2 = 1
1/2 x_0 + x_1 + 1/8 x_2 - x_3 <= 0
-1/2 x_0 - x_1 - 1/8 x_2 - x_3 <= 0
-1/2 x_0 - x_1 - 1/8 x_2 - x_4 <= 0
1/2 x_0 + x_1 - 1/8 x_2 - x_4 <= 0
Variables:
x_0 is a continuous variable (min=0, max=+infty)
x_1 is a continuous variable (min=0, max=+infty)
x_2 is a continuous variable (min=0, max=+infty)
x_3 is a continuous variable (min=0, max=+infty)
x_4 is a continuous variable (min=0, max=+infty)
```

This problem is written with a concrete value for μ in mind, so it does not appear explicitly in the problem. Further, **setup_portfolio_lp** simplifies this type of linear program by reducing the number of variables. Since we have the constraint $\sum_j x_j = 1$, we can arbitrarily choose one of the x_j's as $x_j^* =$

$\sum_{j \neq j^*} x_j$. This new constraint is then included in the linear program and x_j^* is defined in terms of the other x_j's. This reduces the number of variables by one and thus creates a problem that is solved more efficiently. In this case x_3 is written as $x_3 = 1 - x_1 - x_2$ and the problem is rewritten accordingly. This problem is only presented to demonstrate **setup_portfolio_lp**, we will want μ as a parameter for the portfolio optimization problem of interest.

This program is first instantiated as an instance of the *MixedIntegerLinearProgram* (MILP) class in SageMath, a common class for setting up linear programs. This class allows for the use of different solvers and has a wide array of methods built in. For example, the *GNU Linear Programming Kit* (GLPK) solver is relatively fast and we use this for many of our information computations. However, this solver operates with floating point numbers, and in order to preserve the exact rational solution we desire, this solver is not adequate. Thankfully, there exists a solver within SageMath that does solve linear programs exactly, *InteractiveLP*. InteractiveLP could serve as an exact, rational solver compatible with SPAM, and at first this is what we attempted to use. However, InteractiveLP is relatively very slow, too slow to use for the full portfolio optimization problem. This is because InteractiveLP, when used as a solver with a MILP, generates and outputs LaTeX code along the way. This way, a user could copy and paste (or view in a notebook) the simplex method solving process for educational purposes. All these lines of LaTeX code being generated at the same time as InteractiveLP was solving a problem exactly contributed to the significantly longer execution time. Time comparisons between GLPK and InteractiveLP solvers are provided in the Appendix. This posed a problem.

4.2 Hybrid Backend

At this point in the project, Dr. Zhou referred me to a SageTrac ticket (Sage ticket 18735) in progress, one in which a so-called *HybridBackend* was being built. This is where our extremely short-lived stint in SageMath development began, as per Dr. Zhou's recommendation I tried to work on this ticket. The purpose with which HybridBackend was being developed was to create a faster exact solver. In order to accomplish this, HybridBackend was intended to first solve the MILP inexactly using the GLPK (or another inexact solver). Once this was done, HybridBackend would extract the basic variables from the GLPK-solved problem and create another MILP instance using InteractiveLP, injecting the necessary basic variables into the exact MILP and performing one final pivot to reach an optimal solution. This way, InteractiveLP is not solving an entire linear program from start to finish, but is instead solving a "warmstarted" MILP. In this way, HybridBackend could solve a problem to optimality while preserving an exact rational solution much faster than InteractiveLP alone could.

Having never attempted to develop a software before, I was inundated with unfamiliar terms and workflows. I spent time combing through the annals of SageMath development guides, forums, SageMath developer help groups, and Python tutorials to try and understand the process. Quickly my confusion accumulated and we were spending most of my time learning about development instead of the project itself. After a few weeks without making significant progress on finishing HybridBackend Dr. Zhou agreed that she would complete the ticket and I would return to working on the portfolio optimization problem.

This is not where our struggles with SageTrac ended, as once the HybridBackend was completed we had to pull in this resolved ticket to our local copy of SageMath. Because we used SageMath through the Windows Subsystem for Linux, we managed files with two sets of directories and system paths. One way to pull in changes from a SageTrac ticket is using the *git trac* command within the system terminal. After consulting the SageMath developer guide once again for installing and using git trac we thought we were prepared to pull these changes. However, we encountered permission errors when trying to access the resolved ticket. After trying to configure git trac using SSH keys and token authentication with no success, we reached out to the SageMath developer Google group for help. It was recommended that instead of using git trac we simply use plain git from our system terminal. This worked, and finally we had HybridBackend working on our local machine.

4.3 Proof Cell Complex for LP

Solving LP using HybridBackend was fast and correct. Moreover, HybridBackend was suitable for use with SPAM, and so the process of discovering the proof cell complex began. One method for discovering a proof cell complex is to first create an instance of class *SemialgebraicComplex.* From the parametric.sage file in the repository cutgeneratingfunctionology where SPAM code is stored, SemialgebraicComplex creates an empty proof cell complex. A SemialgebraicComplex takes multiple arguments when instantiated, including:

- A family of functions that will be executed (i.e. a program P) potentially many times
- Symbolic representations for each parameter
- find_region_type: A function defining the value of each proof cell in order to compare one to another
- default_var_bound: Default variables bounds (an interval within which SPAM will test parameter values

There are other, optional, arguments which SemialgebraicComplex can take, which we list in the Appendix. Once instantiated, this proof cell complex can be completed by using a breadth-first-search (BFS) method. This method takes multiple arguments as well:

- var_value: Variable values (numerical values for each parameter at which the bfs will begin)
- wall_crossing_method: Wall-crossing method (here a "wall" is an inequality that defines the border of one proof cell), because we did not have access to Mathematica we used the heuristic wall-crossing method
- goto_lower_dim: A Boolean informing bfs whether to check lower-dimensional cells or not (inour case we have only one parameter of interest, so all cells will be one-dimensional along the number line)

The BFS process involves several steps:

1. P is executed while SPAM records both the numerical and symbolic representation of any internal comparisons made involving the parameter(s) of interest
2. SPAM then generates one proof cell for P defined by the parameter inequalities at the given parameter values and the output of find_region_result
3. SPAM crosses one wall defining the previous proof cell, choosing a test point not already covered by any existing proof cell
4. P is executed again with the new test point values of the parameters
5. This process is repeated until (hopefully) the entire parameter space is covered by proof cells, at which point the SemialgebraicComplex is populated and SPAM stops running

This process is not exact, especially while using the heuristic wall-crossing method. There are variable execution times depending on the specific test points chosen by SPAM throughout the course of execution. If this is accomplished successfully (which is not a trivial assumption, we learned) the problem of solving P for any set of parameter values can be answered by simply inspecting the proof cell complex. We will describe instances in which SPAM finished executing without completing the proof cell complex and our remedies for this problem in a later section.

This is the basic idea behind SPAM. As stated, all of this code is found in the mkoeppe/cutgeneratingfunctionology repository on GitHub.

Here is the setup commands and SageMath output for the proof cell complex generated by LP:

This proof cell complex is comprised of three distinct cells. Each cell is defined by a μ-interval. By construction of the proof cells, we can make the following deductions:

- Within any proof cell, each value of μ within the μ-interval defining it will yield the same objective function value
- Within any proof cell, the optimal solutions to any portfolio optimization problem with a μ value in the μ-interval will be the same
- Since the union of the proof cells is the interval (0∞), we know that the proof cell complex is complete

So this is the desired outcome. SPAM produced a proof cell complex describing the different possible solutions to LP for any value of the risk-aversion parameter μ. As stated, we include a brief discussion about problems with BFS and manual steps one can take to produce a complete proof cell complex when BFS fails on its own.

This process will be replicated for the full problem of portfolio optimization that Vanderbei considers. Before that, however, we provide a description of a manual comparison between the Vanderbei method by hand and the SPAM method.

4.4 Checking Parametric Simplex Method vs SPAM

In order to reproduce the parametric simplex method results from Vanderbei, the MILP LP must first be translated. As mentioned previously, MILP has no support for dictionary investigation, so we used the Interactive Simplex Method module within SageMath. Using an InteractiveLPProblem object from this module, it is possible to display, define, and change dictionaries manually. Since this is the defining characteristic of the parametric simplex method, this module will work well here. Provided here is the construction of LP as an InteractiveLPProblem object, which we define as ILP:

Notice that in the construction of ILP one must explicitly define a *ParametricRealField*, K. K represents the parent field of the parameter μ, and is necessary to define before working with SPAM. However, earlier when LP was solved using BFS, the construction of SemialgebraicComplex takes care of setup of K for us. So it was not necessary to directly create a ParametricRealField earlier.

The construction of ILP looks different from the construction of LP because MILP and InteractiveLPProblem objects are from different modules and are associated with different methods. Notice that in creating ILP one must define the coefficient matrix A, the RHS constraint coefficient vector b, and the objective function c, whereas in constructing LP we only needed historical data and μ. ILP variables y_1 and y_2 are the variables replacing the absolute value terms from before, renamed here to match Vanderbei.

When displayed, ILP looks like this:

$$
\begin{array}{lrcrcrcrcrcl}
\max & \frac{7}{6}\mu x_1 & + & \frac{4}{3}\mu x_2 & + & \frac{7}{8}\mu x_3 & - & \frac{1}{2}y_1 & - & \frac{1}{2}y_2 & & \\
 & x_1 & + & x_2 & + & x_3 & & & & & \leq & 1 \\
- & x_1 & - & x_2 & - & x_3 & & & & & \leq & -1 \\
 & \frac{1}{2}x_1 & + & x_2 & + & \frac{1}{8}x_3 & - & y_1 & & & \leq & 0 \\
- & \frac{1}{2}x_1 & - & x_2 & - & \frac{1}{8}x_3 & - & y_1 & & & \leq & 0 \\
- & \frac{1}{2}x_1 & - & x_2 & - & \frac{1}{8}x_3 & & & - & y_2 & \leq & 0 \\
 & \frac{1}{2}x_1 & + & x_2 & + & \frac{1}{8}x_3 & & & - & y_2 & \leq & 0 \\
\multicolumn{12}{l}{x_1, x_2, x_3, y_1, y_2 \geq 0}
\end{array}
$$

This looks similar to LP, but is not exactly the same. Here the equality constraint $x_1 + x_2 + x_3 = 1$ is rewritten as the first two constraints in ILP. InteractiveLPProblem creates linear programs with $\leq$ constraints, so the equality constraint was necessarily modified. Now having ILP formulated, one can convert it to standard form using **standard_form** from InteractiveLPProblem. This is necessary because InteractiveLPProblem in general has no **run_simplex_method** method, which we need to generate a final dictionary. This simply solves the standard form of ILP to optimality using the simplex method, we are not yet comparing the manual computation of the parametric simplex method to SPAM. Once **run_simplex_method** has been called on the standard form of ILP one can look into the final dictionary. This is done via the **final_dictionary** method of InteractiveLPProblemStandardForm. We chose to begin with $\mu = 10$, since this is sufficiently large to guarantee all of the portfolio weight will be on one asset. This will also be the starting point for the full portfolio optimization example for the same reason. By doing so, the final dictionary from the solved standard form of ILP is:

$$
\begin{array}{|rcrcrcrcrcrcr|}
\hline
x_7 & = & 0 & - & x_6 & & & & & & & & \\
y_2 & = & 1 & - & x_6 & & & - & \frac{1}{2}x_1 & - & \frac{7}{8}x_3 & + & x_{11} \\
x_2 & = & 1 & - & x_6 & & & - & x_1 & - & x_3 & & \\
x_9 & = & 2 & - & 2x_6 & + & x_8 & - & x_1 & - & \frac{7}{4}x_3 & & \\
x_{10} & = & 2 & - & 2x_6 & & & - & x_1 & - & \frac{7}{4}x_3 & + & x_{11} \\
y_1 & = & 1 & - & x_6 & + & x_8 & - & \frac{1}{2}x_1 & - & \frac{7}{8}x_3 & & \\
\hline
z & = & \frac{4}{3}\mu - 1 & - & \frac{4}{3}\mu - 1x_6 & - & \frac{1}{2}x_8 & - & \frac{1}{6}\mu - \frac{1}{2}x_1 & - & \frac{11}{24}\mu - \frac{7}{8}x_3 & - & \frac{1}{2}x_{11} \\
\hline
\end{array}
$$

Now having the final dictionary we can begin the parametric simplex method. As described by Vanderbei, this is done by inspecting the sign of coefficients on non-constant terms containing μ within the objective function. Those coefficients are:

1. $(\frac{4}{3}\mu - 1)$ on variable x_6
2. $(\frac{1}{6}\mu - \frac{1}{2})$ on variable x_1
3. $(\frac{11}{24}\mu - \frac{7}{8})$ on variable x_3

As variables affected by the value of μ, one of these will become the entering variable in the first step of the parametric simplex method. We would like to find values for μ in which those coefficients become negative, per Vanderbei's technique. Doing so will provide μ values at which the basic variables change, a result of the now-negative coefficient's variable entering the basis. As usual, the ratio test is applied to determine which basic variable will leave. In other words, we are interested in the μ values defined by:

1. $\frac{4}{3}\mu - 1 = 0$
2. $\frac{1}{6}\mu - \frac{1}{2} = 0$
3. $\frac{11}{24}\mu - \frac{7}{8} = 0$

As these are the μ values that will result in a sign change from their respective coefficients. The corresponding μ values are $\mu = \frac{3}{4}, \mu = 3$, and $\mu = \frac{21}{11}$. According to Vanderbei, this process begins with a large enough μ value, in this case 10, and methodically lowers this value until we reach $\mu = 0$. In this case, of the three possible μ values that will change the optimal solution, $\mu = 3$ is the closest to 10, and so we will choose x_1 (the variable associated with this coefficient) as the entering variable for the first pivot of the parametric simplex method. By the ratio test x_2 leaves the basis in place of x_1. While not listed explicitly here, this process is repeated until all of the coefficients containing μ in the objective function are positive. We will list the objective function resulting from x_1 entering and x_2 leaving the basis:

$$\zeta = \frac{7}{6}\mu - \frac{1}{2} - (\frac{7}{6}\mu - \frac{1}{2})x_6 - \frac{1}{2}x_8 + (\frac{1}{6}\mu - \frac{1}{2})x_2 - (\frac{7}{24}\mu - \frac{3}{8})x_3 - \frac{1}{2}x_{11}$$

From this we see that now there are only two coefficients containing μ that are negative,

1. $(\frac{7}{6}\mu - \frac{1}{2})$ on variable x_2
2. $(\frac{7}{24}\mu - \frac{3}{8})$ on variable x_3

In the same way as before, we can solve these for μ to get $\mu = \frac{3}{7}$ and $\mu = \frac{9}{7}$, respectively. Since $\frac{9}{7}$ is larger, this is the next interval breakpoint and so its associated variable, x_3, enters the basis. This results in x_1 leaving the basis and a new objective function:

$$\zeta = \tfrac{7}{8}\mu - \tfrac{1}{8} - (\tfrac{7}{8}\mu - \tfrac{1}{8})x_6 - \tfrac{1}{2}x_8 + (\tfrac{11}{24}\mu - \tfrac{7}{8})x_2 + (\tfrac{7}{24}\mu - \tfrac{3}{8})x_1 - \tfrac{1}{2}x_{11}$$

Now there is only one negative coefficient containing μ, resulting in a μ value of $\mu = \frac{1}{7}$. This corresponds to variable x_6, which enters in place of x_7, which leaves. Finally, after this pivot we are left with the objective function:

$$\zeta = \tfrac{7}{8}\mu - \tfrac{1}{8} + (\tfrac{7}{8}\mu - \tfrac{1}{8})x_7 - \tfrac{1}{2}x_8 + (\tfrac{11}{24}\mu - \tfrac{7}{8})x_2 + (\tfrac{7}{24}\mu - \tfrac{3}{8})x_1 - \tfrac{1}{2}x_{11}$$

Notice that each coefficient on a non-constant containing μ is positive. By the parametric simplex method, we are done. The optimal solution will not change for any μ value lower than $\frac{1}{7}$. Hence, we have manually completed the parametric simplex method. The μ values we encountered along the way that resulted in a change in basis are recorded, and are actually endpoints for intervals in the proof cell complex. These intervals are $(0, \frac{1}{7}]$, $(\frac{1}{7}, \frac{9}{7}]$, $(\frac{9}{7}, 3]$, and $(3, \infty)$. The union of these intervals covers the entire positive number line, so we know this is complete. Note that the endpoints of intervals result in a change in sign of objective coefficients (from positive to 0) and thus change the optimal solution and are included as closed boundaries on cells with smaller μ values than them.

This may appear complete and correct, but there is another check to be done. Now one can investigate the objective value for each interval to check for consistency. When this is done, every interval appears fine except for $(0, \frac{1}{7}]$. The change in μ from $\mu > \frac{1}{7}$ to $\mu < \frac{1}{7}$ does not result in a change in objective value or optimal solution. The explanation for this is not obvious, but is known. During the pivot that resulted in the generation of $\mu = \frac{1}{7}$, the variable that entered was x_6, and x_7 left. These are both auxiliary variables generated by **run_simplex_method**, and so their movement in and out of the basis does not affect the objective value or optimal solution. In the first few pivots, variables x_1, x_3, and x_2 moved in and out of the basis, and since those are the decision variables of interest, the μ values generated from their coefficients were meaningful to the change in objective value and optimal solution.

So, the refined proof cell complex generated from manual computation of the parametric simplex method can be defined by the list here, organized as interval; optimal solution; objective value:

1. $(0, \frac{9}{7}]$; $(0, 0, 1)$; $\frac{7}{8}\mu - \frac{1}{8}$
2. $(\frac{9}{7}, 3]$; $(1, 0, 0)$; $\frac{7}{6}\mu - \frac{1}{2}$

3. $(3, \infty)$; $(0, 1, 0)$; $\frac{4}{3}\mu - 1$

In validating the proof cell complex via manual computation this suggests that SPAM will work successfully with the large portfolio optimization problem, as the format of the problem is the same. The difference between this small example and the large example is merely a difference in size of problem.

4.5 Nine Assets and 24 Time Periods

Now to demonstrate SPAM on the textbook example. In applying the same techniques as in the small toy example, we are able to replicate the results from Vanderbei. Our graphical representation of results (located in the appendix) from SPAM matches with the textbook results, as expected after confirming the validity of our method. While this example is not the most exciting or realistic, it demonstrates the ability of SPAM to match and rediscover the existing textbook method results. As SPAM is a relatively new development we expect many more potential applications with SPAM in economics.

4.6 Rediscovery of Efficient Frontier

This section contains a discussion of the rediscovery of the efficient frontier from Vanderbei's example with SPAM. An efficient frontier is the set of all risk-reward combinations that dominate every other risk-reward combinations possible from a given set of assets. Each point on the efficient frontier (in this case a curve) represents either the maximum expected return subject to a given level of risk or the minimum risk subject to a certain level of expected return. Any investor would aim to locate their portfolio on an efficient frontier, as choosing a portfolio on the interior of the frontier is irrational. They could increase their expected return with their current level of risk or reduce their risk without altering expected reward by moving to the frontier. This concept in finance is important, and is used in several applications of the field.

The historical data that Vanderbei uses can be found in the Appendix. For ease of reference and computational convenience we will refer to them as assets and by their index rather than their actual sector name. This is monthly data describing the expected return and risk associated with each asset. This, along with our parameter μ, is all we need to proceed. To discover the efficient frontier, the process is as simple as computing the risk and expected reward of every optimal portfolio (optimal portfolios discovered via SPAM). Then each portfolio is plotted on the risk-reward plane. the piecewise function joining the points represents the efficient frontier itself, and every point above the frontier is every possible portfolio. These results are listed in the appendix.

We also included the portfolios consisting of $x_j = 1$ for $j = 1, 2, ..., 9$. It is interesting to note that most of these portfolios do not lie on the efficient frontier. That is, unless you are choosing to invest for maximum reward (energy) an investor would not invest their entire portfolio into a single asset.

Note: the dotted lines extending from the efficient frontier itself indicate where the minimum risk and maximum expected return lie. At a certain point it becomes impossible to reduce risk past a given level while selecting from the same group of investments. Likewise, it becomes impossible to increase expected return past a certain point without searching for other assets.

4.7 Real World Data

Beyond the textbook we collected our own data, from Yahoo Finance, on historical returns from 9 real-world assets. These are Tesla, Apple, Microsoft, Bitcoin, Nike, Facebook, 5YR Corporate Bonds, Ford, and General Electric. We used the same techniques (parametric simplex method, SPAM, efficient frontier discovery) to produce results for these assets also. Results are listed in the Appendix and real historical data can be found on our GitHub repository for this project. These real-world data produce an interesting result - every optimal portfolio includes some investment in Bitcoin. This is expected, as its return over the past several years has been high.

5 Appendix

Listed here are possible extensions, links to our SageMath code, figures, and historical data we used for this project.

5.1 Possible Extensions: Mixed Strategies in Zero-Sum Games

In this possible extension we consider a two-player zero-sum simultaneous game of perfect information. This could be compatible with SPAM by formulating this game as a linear program and solving the program using SageMath methods. For reference to basic game theory terminology, we provide brief descriptions in terms of monetary payouts. A *two-player game* is an interaction between two agents in which they both must make a decision simultaneously. A *zero-sum* game is one in which Player 1's payout is exactly the negative of Player 2's payout. One can think of this game as a game in which the losing player pays the winning player. A game of *perfect information* is one in which the possible payouts are known by both players.

Note that these games are considered in strategic form [4]. A game in strategic form is described as a triplet (X, Y, A), where

1. X is a nonempty set representing the possible strategies for Player 1
2. Y is a nonempty set representing the possible strategies for Player 2
3. A is a real-valued function with $A(x, y)$ defining the payoff determined by $x \in X$, $y \in Y$

Depending on what a player would like to optimize, there are different objectives when solving that player's problem. In this case, let the optimal mixed strategy for a player be that player's *minimax* solution. In some sense this is choosing a strategy such that (without loss of generality) Player 1 maximizes her expected payout while knowing her opponent can choose from multiple actions. Her opponent faces a similar problem, minimizing the expected payout of Player 1. Let us first inspect this game intuitively. Let p be the proportion of times Player 1 chooses x. We would like to find p such that Player 1 wins the same expected amount whether Player 2 chooses x or y. So, when Player 2 chooses x Player 1's expected payout is $-2p + 3(1-p)$. Similarly, when Player 2 chooses y Player 1's expected payout is $3p - 4(1-p)$. Player 1, wanting to earn the same expected payout in either case, sets these equal and solves the problem

$$-2p + 3(1-p) = 3p - 4(1-p)$$

$$\text{with solution } p = \tfrac{7}{12}$$

That is, if choosing a minimax strategy, Player 1 would choose x with probability $\frac{7}{12}$ and y with probability $1-\frac{7}{12}=\frac{5}{12}$. This results in an expected payout in either case of

$$-2(\tfrac{7}{12})+3(\tfrac{5}{12})=\tfrac{1}{12}$$

So, if Player 1 chooses this minimax strategy, she has a positive expected payout. This is the *value* of this game. As denoted by its name, **primal_solver** solves a primal linear program, arbitrarily chosen here as Player 1's problem. In this case of a two-player game, one player will face the dual problem of the other. Let Player 2's problem be the dual of Player 1's. For the above game, Player 1's optimal strategy is defined as the solution to the linear program

$$(P) = \begin{array}{lrrrrrrrl}
\max & & & & & & x_3 & & \\
 & & x_1 & + & x_2 & & & \leq & 1 \\
 & - & x_1 & - & x_2 & & & \leq & \text{-1} \\
 & - & x_1 & & & & & \leq & 0 \\
 & & & - & x_2 & & & \leq & 0 \\
 & & 2x_1 & - & 3x_2 & + & x_3 & \leq & 0 \\
 & - & 3x_1 & + & 4x_2 & + & x_3 & \leq & 0 \\
x_1, x_2, x_3 \geq 0 & & & & & & & &
\end{array}$$

Where x_1 and x_2 are probabilities that Player 1 chooses x and y, respectively. If either x_1 or x_2 is 1, this strategy is called *pure*. A pure strategy is one in which Player 1 chooses the same action with certainty. If both $x_1, x_2 \neq 0$ then this defines a mixed strategy, one in which Player 1 assigns probabilities to each action and then chooses with those probabilities. In this case, the solution to (P) is $(\frac{7}{12}, \frac{5}{12})$. Expectedly, this solution is the same as the solution by inspection earlier. That is, with probability $\frac{7}{12}$ Player 1 would choose action x and with probability $\frac{5}{12}$ Player 1 would choose y. This will maximize his minimum payout.

This game, parameterized by the probabilities assigned to each action, could theoretically function within the SPAM framework. Unfortunately we were unsuccessful in doing so, and our attempts are documented in the Appendix. In the future this is something that could be investigated further, although we do not know how interesting of a question discovering optimal mixed strategies is, at least on this scale.

5.2 SageMath code

`Our SageMath programs/code and test examples can be found at:`

`https://github.com/yuan-zhou/portfolio-optimization`

5.3 Figures

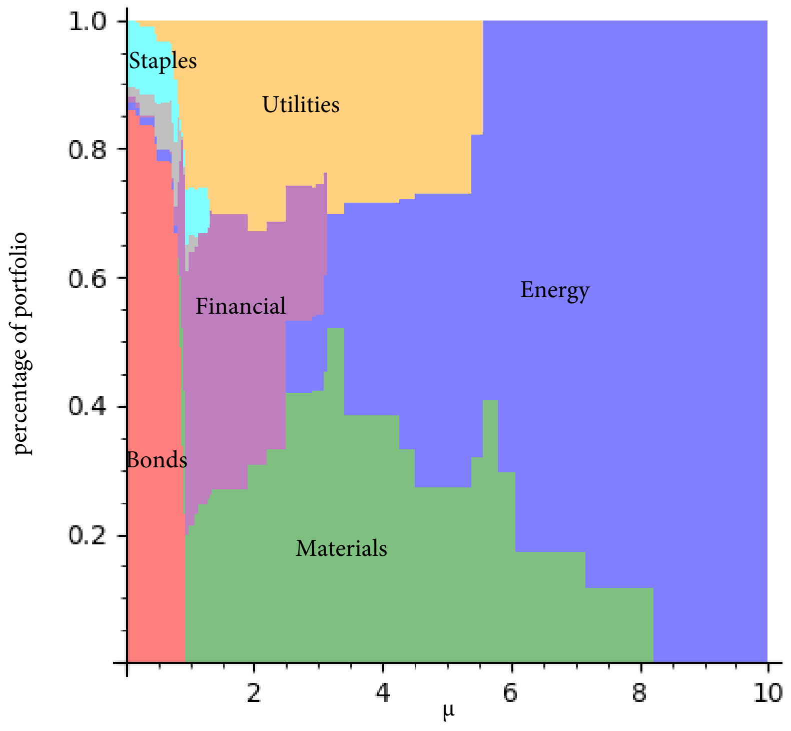


Figure 1: Textbook example portfolio weights plotted against μ.

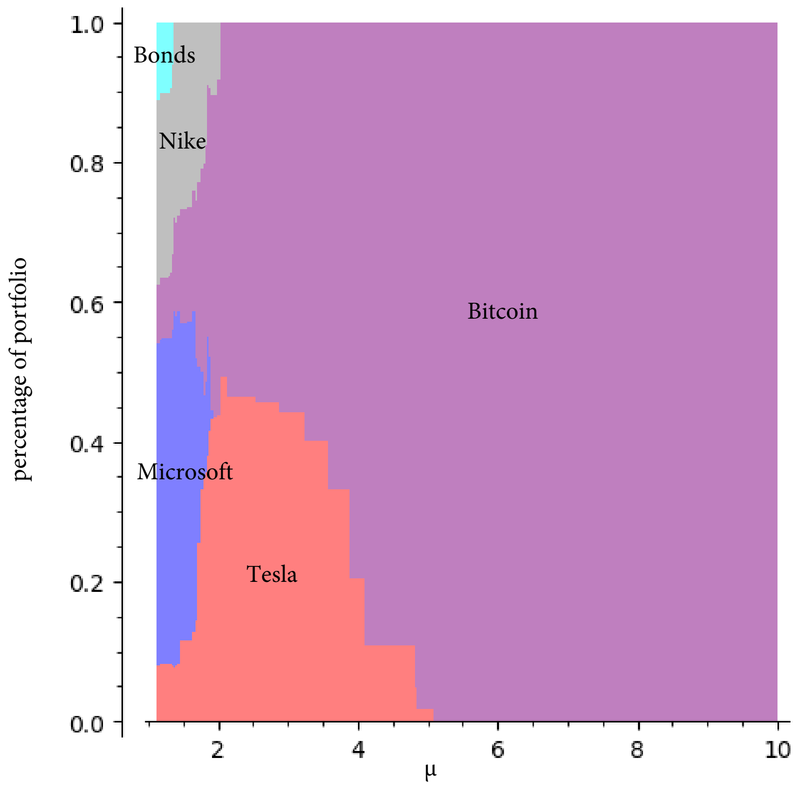


Figure 2: Real-world data portfolio weights plotted against μ.

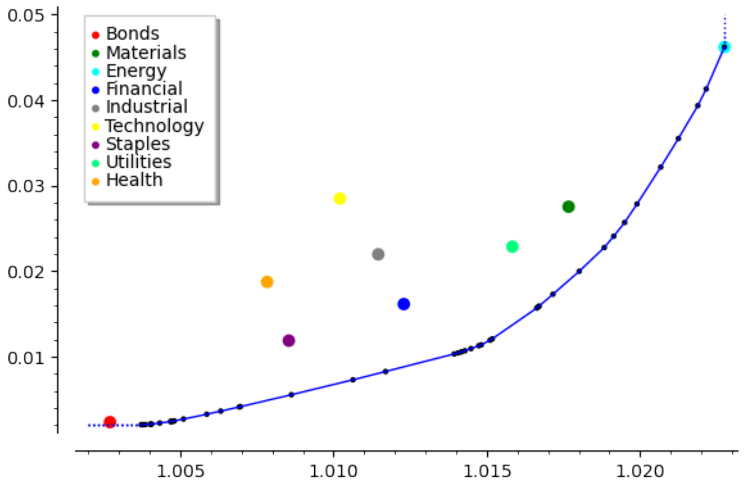


Figure 3: Efficient frontier for textbook data.

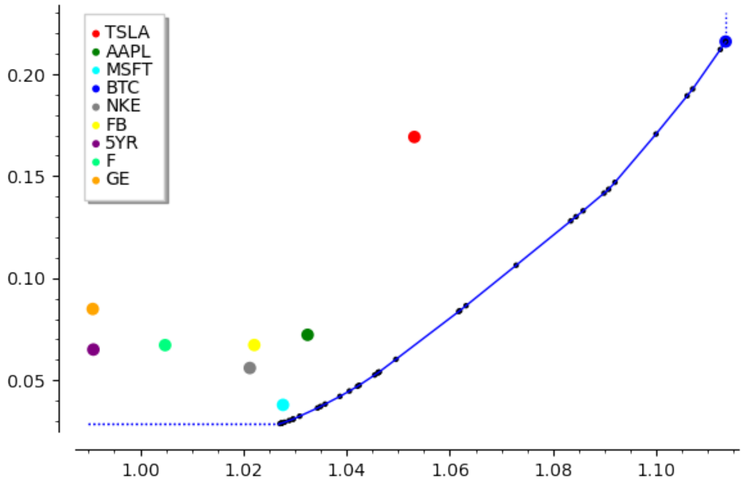


Figure 4: Efficient frontier for real-world data.

5.4 Historical Data

Time period	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
2007-04	1.000	1.044	1.068	1.016	1.035	1.032	1.004	0.987	1.014
2007-03	1.003	1.015	1.051	1.039	1.046	1.047	1.028	1.049	1.073
2007-02	1.005	1.024	1.062	0.994	1.008	1.010	1.021	1.036	1.002
2007-01	1.007	1.027	0.980	0.971	0.989	0.973	0.985	1.053	0.977
2006-12	1.002	1.040	0.991	1.009	1.021	1.020	1.020	0.996	1.030
2006-11	1.001	0.995	0.969	1.030	0.997	0.989	1.020	0.999	1.007
2006-10	1.005	1.044	1.086	1.007	1.024	1.028	0.991	1.026	0.999
2006-09	1.004	1.060	1.043	1.023	1.028	1.040	1.018	1.053	1.003
2006-08	1.004	1.000	0.963	1.040	1.038	1.040	0.999	0.985	1.015
2006-07	1.008	1.030	0.949	1.012	1.011	1.070	1.039	1.028	1.029
2006-06	1.007	0.963	1.034	1.023	0.943	0.974	1.016	1.048	1.055
2006-05	1.002	1.005	1.022	0.995	0.999	0.995	1.018	1.023	1.000
2006-04	1.002	0.960	0.972	0.962	0.983	0.935	1.002	1.016	0.979
2006-03	1.002	1.035	1.050	1.043	1.021	0.987	1.010	1.016	0.969
2006-02	1.002	1.047	1.042	1.003	1.044	1.023	1.008	0.954	0.987
2006-01	1.000	0.978	0.908	1.021	1.031	1.002	1.008	1.013	1.012
2005-12	1.002	1.048	1.146	1.009	1.003	1.034	1.002	1.024	1.013
2005-11	1.004	1.029	1.018	1.000	1.005	0.969	1.001	1.009	1.035
2005-10	1.004	1.076	1.015	1.048	1.058	1.063	1.009	0.999	1.012
2005-09	0.999	1.002	0.909	1.030	0.986	0.977	0.996	0.936	0.969
2005-08	0.997	1.008	1.063	1.009	1.017	1.002	1.014	1.042	0.995
2005-07	1.007	0.958	1.064	0.983	0.976	0.991	0.983	1.006	0.996
2005-06	0.996	1.056	1.071	1.016	1.038	1.057	1.032	1.023	1.023
2005-05	1.002	0.980	1.070	1.012	0.974	0.987	0.981	1.059	0.994

Textbook example of 24 monthly returns per dollar for nine different assets.

References

[1] Bard, G.: Sage for Undergraduates. *Mathematics Subject Classification.* (2010)

[2] Cornuéjols, G.: Valid inequalities for mixed integer linear programs. *Mathematical programming.* 112, 3-34 (2008)

[3] d'Aspremont, C., Jacquemin, A.: Cooperative and Noncooperative R&D in Duopoly with Spillovers. *The American Economic Review.* 78, 1133–1137 (1988)

[4] Ferguson, T.: *A Course in Game Theory.* (2020)

[5] Flesch, J., Predtetchinkski, A.: Parameterized games of perfect information. *Annals of Operations Research.* 287, 683-699 (2018)

[6] Goemans, M.: Linear Programming and Polyhedral Combinatorics. *Combinatorial Optimization.* (2017)

[7] Köeppe, M., Zhou, Y., Hong, C.Y., and Wang, J.: cutgenerating-functionology. *https://github.com/mkoeppe/cutgeneratingfunctionology.* (2021)

[8] Pak, I.: *Lectures on Discrete and Polyhedral Geometry.* (2010)

[9] Prisner, E.: Game Theory through Examples. *Mathematical Association of America.* (2014)

[10] SageMath, the Sage Mathematics Software System (Version 9.3.beta6), The Sage Developers, 2021, https://www.sagemath.org.

[11] Vanderbei, R.: Linear Programming, 4e. *International Series in Operations Research and Management Science.* 196, (2014)

[12] Zhou, Y.: portfolio-optimization. *https://github.com/yuan-zhou/portfolio-optimization.* (2021)