1 Numerical implementation details

This section covers detail regarding the solution algorithm and numerical implementation. Each method is presented in a separate subsection, and the final solution algorithm is presented in the last subsection, which combines each of the methods. These span points sampling, numerical integration techniques, function approximation methods and solution techniques specific to this class of problemns.

1.1 Numerical integration

Consider the basic problem with proportional transaction costs, basic risky assets and a risk-free asset and no stochastic parameters. We need to evaluate the expectation of the value function: $\mathbb{E}\left[v_{t+\Delta t}(\mathbf{x}_{t+\Delta t})\right]$. In order to compue this expectation, we need to evaluate the integral:

$$\mathbb{E}_t\left[\pi_{t+1}^{1-\gamma} v_{t+1}(x_{t+1})\right] = \int \pi_{t+1}^{1-\gamma} v_{t+1}(x_{t+1}) f(R_{t+1}), dR_{t+1} \tag{1}$$

where $f(R_{t+1})$ is the probability density function of the risky asset returns. If we look at the case of stochastic parameters, would need to evaluate the conditional expectation with regard to these aswell, given some distributional assumption on the parameters. The integral can be computed using Monte-carlo methods or by using quadrature rules.

1.1.1 Gauss-Hermite quadrature

Gaussian quadrature is a numerical integration method based on approximation and interpolation theory. Gaussian quadrature can be used to approximate integrals using the following form, Judd (1998):

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^{n} \omega_i f(x_i), \tag{2}$$

Where ω_i are quadrature weights, x_i are quadrature nodes and $w(x)$ is a weighting function. This approximation is exact when $f(x)$ is a polynomial of degree $2n-1$ or less. Then we can approximate the integral using n points x_i and n weights ω_i. There are many different Gaussian quadrature schemes, with differering intervals $[a, b]$ and weighting functions $w(x)$. We consider the use of a Gauss-Hermite quadrature rule, for a comprehensive review on Gaussian quadrature rules, see Judd (1998). Gauss-Hermite quadrature is used to approximate integrals of the form:

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}dx \approx \sum_{i=1}^{n} \omega_i f(x_i) + \frac{n!\sqrt{\pi}}{2^n} \cdot \frac{f^{(2n)}(\zeta)}{(2n)!}, \tag{3}$$

Where $\zeta \in (-\infty, \infty)$. If a random variable X is normally distributed, i.e $X \sim \mathcal{N}(\mu, \sigma^2)$, then we can compute the expectation, $\mathbb{E}[f(X)]$, which is given by:

$$\mathbb{E}[f(X)] = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} f(x) e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \tag{4}$$

Using a change of variables $y = \frac{x-\mu}{\sqrt{2}\sigma}$, then we can rewrite the expectation on the form of the Gauss-Hermite quadrature rule:

$$\begin{aligned}
\mathbb{E}[f(X)] &= \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} f(\sqrt{2}\sigma y + \mu) e^{-y^2} \sqrt{2}\sigma dy & (5)\\
&= \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-y^2} f(\sqrt{2}\sigma y + \mu) dy & (6)\\
&\approx \frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} \omega_i f(\sqrt{2}\sigma x_i + \mu) & (7)
\end{aligned}$$

Where ω_i are the quadrature weights, x_i are the quadrature nodes over the interval $(-\infty, \infty)$.
When X is log-normal, i.e $\log X \sim \mathcal{N}(\mu, \sigma^2)$, then we can use a variable change once again: $X = e^Y$ and $Y \sim \mathcal{N}(\mu, \sigma^2)$. Then we can rewrite the expectation as:

$$\mathbb{E}[f(X)] = \mathbb{E}[f(e^Y)] \approx \pi^{-\frac{1}{2}} \sum n_{i=1} \omega_i f\left(e^{\sqrt{2}\sigma x_i + \mu}\right) \tag{8}$$

If we want to extend this framework to multiple dimensions we can use product rules as noted by Cai, Judd and Xu (2013). Consider Y which is multivariate normal, i.e $Y \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where μ is the drift vector and Σ is the covariance matrix. Let L be a lower-triangular matrix such that $LL^\top = \Sigma$ (Cholesky factorisation). Then we have that:

$$\begin{aligned}
\mathbb{E}\{f(Y)\} &= \left((2\pi)^d \det(\Sigma)\right)^{-\frac{1}{2}} \int_{\mathbb{R}^d} f(y)\, e^{-\frac{1}{2}(y-\mu)^\top \Sigma^{-1}(y-\mu)}\, dy & (9)\\
&= \left((2\pi)^d \det(L)^2\right)^{-\frac{1}{2}} \int_{\mathbb{R}^d} f\left(\sqrt{2}Ly + \mu\right)\, e^{-\frac{1}{2}y^\top y}\, dy & (10)\\
&\approx \pi^{-\frac{d}{2}} \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} \omega_{i_1} \cdots \omega_{i_d}\, f\Bigg(\sqrt{2}L_{1,1}y_{i_1} + \mu_1, \\
&\qquad \sqrt{2}(L_{2,1}y_{i_1} + L_{2,2}y_{i_2}) + \mu_2,\ \ldots,\ \sqrt{2}\Big(\sum_{j=1}^{d} L_{d,j}y_{i_j}\Big) + \mu_d\Bigg) & (11)
\end{aligned}$$

Where d refers to the number of dimensions, n is the number of quadrature points, ω_i are the quadrature weights and y_i are the quadrature nodes. $L_{i,j}$ is the ith row and jth column of the Cholesky factorisation matrix L. det is the matrix determinant. We note that the use of product rules suffers from the curse of dimensionality, as the complexity scales exponentionally with the number of dimensions. This is because the quadrature points with the product rule, normally use a tensor product grid, which is constructed using the Cartesian product of the quadrature points in each dimension. We can use sparse grid methods to partially tackle this. One common method is the Smolyak method, Smolyak (1963). Smolyaks sparse grid method approximates multidimensional integrals, over dimesion d while limiting the amount of points used. The method is composed of the following:

1. **Univariate Quadrature Rules**: Each dimension of the integration domain is assigned a univariate quadrature rule, which provides both nodes (quadrature points) and weights for numerical integration in that dimension. The accuracy of each rule is determined by its *level*, denoted by i_d for each dimension d. The level determines the number of quadrature points in that dimension, which improves the accuracy of the quadrature rule.

2. **Approximation Level (μ)**: The accuracy of the Smolyak sparse grid is controlled by the *approximation level* μ. This parameter sets a limit on the sum of levels across all dimensions, controlling the total number of grid points. Higher values of μ result in more accurate approximations but increase computational complexity.

3. **Multi-Index and Combination of Levels**: In a d-dimensional integral, the Smolyak method uses a *multi-index* $i = (i_1, i_2, \ldots, i_d)$ to represent the level of the quadrature rule in each dimension. The multi-index specifies a unique combination of quadrature levels for each dimension, where i_d denotes the level for dimension d. To construct a sparse grid, Smolyak's method restricts the sum of these levels using the following condition:

$$d \leq i_1 + i_2 + \cdots + i_d \leq d + \mu$$

   This constraint on the sum of levels, reduces the number of tensor products. We denote the sum of multi indicies: $|i| = i_1 + i_2 + \ldots i_d$.

4. **Tensor Product of Univariate Rules**: The Smolyak grid is formed by taking the *tensor product* of univariate quadrature rules that satisfy the multi-index constraint. Each univariate quadrature rule, represented by Q_{i_d} at level i_d in dimension d, is combined across dimensions according to the set of multi-indices i. This combination is given by:

$$A(\mu, d) = \sum_{d \leq |i| \leq d+\mu} (-1)^{\mu+d-|i|} \binom{d-1}{\mu+d-1-|i|} \bigotimes_{d=1}^{d} Q_{i_d}$$

   where:

   - Q_{i_d} is the univariate quadrature rule at level i_d in dimension d,

- $\bigotimes$ denotes the tensor product, and
- $\binom{d-1}{\mu+d-1-|i|}$ is a combinatorial coefficient that assigns weights to each tensor product, for accurate integration up to the specified approximation level μ.

By resticting the multi indicies i with the approximation level μ, the Smolyak method reduces the number of points needed for numerical integration in higher dimensions. Tensor grid methods grows exponentially with the number of dimensions d, the Smolyak grid grows polynomially, Judd et al. (2014), hence it directly combats the curse of dimensionality. For more on this see Smolyak (1963), Judd et al. (2014) and Horneff, Maurer and Schober (2016).

1.1.2 Monte Carlo integration (MC)

Monte Carlo integration is a numerical integration method based on *sampling*, as opposed to quadrature rules which are based on interpolation.
The convergence of Monte Carlo integration is generally slower than some quadrature methods; however, its convergence rate is independent of the dimensionality of the integral, making it well-suited for high-dimensional problems. Monte Carlo integration breaks the curse of dimensionality. Monte Carlo (MC) integration is based on random sampling[1] over the domain of the integral, and then computing the sample average of the function to be integrated. Assume we wish to approximate the d-dimensional integral:

$$I = \int_{\Omega} f(\mathbf{x})g(\mathbf{x})d\mathbf{x} = \mathbb{E}[f(\mathbf{x})], \tag{12}$$

where $g(\mathbf{x})$ is the probability density function of the random variable $\mathbf{x}$ over its support Ω, we approximate I as:

$$Q_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{X}_i), \tag{13}$$

where $\mathbf{X}_i$ are independent samples drawn from $g(\mathbf{x})$. The procedure is then:

1. Sample N points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ from $g(\mathbf{x})$.
2. Approximate the expectation $\mathbb{E}[f(\mathbf{x})]$ by the sample average:

$$I \approx Q_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i).$$

The Law of Large Numbers ensures that the sample average converges to the mean as $N \to \infty$:

$$\lim_{N\to\infty} Q_N = \mathbb{E}[f(\mathbf{x})] = I.$$

[1]Strictly speaking the samples are not random, but pseudo-random, meaning that deterministic samples are used, which appear random. For more in this see Judd (1998) or Glasserman (2004)

And by the Central Limit Theorem, we have:

$$\sqrt{N}\left(Q_N - I\right) \xrightarrow{d} N\left(0, \sigma^2\right),$$

where $\sigma^2 = \mathrm{Var}[f(\mathbf{x})]$ does not depend on N or d. The standard error of Q_N is:

$$\sigma_{Q_N} = \frac{\sigma}{\sqrt{N}}.$$

The convergence rate of $1/\sqrt{N}$ is independent of the dimension.

1.1.3 Quasi-Monte Carlo integration (QMC)

Quasi-Monte Carlo integration substitutes the 'random' samples in Monte Carlo integration with specific deterministic sequences such as equidistributred sequences, low-discrepancy sequences (LDS) or Lattice point rules etc. We will focus on the use of low discrepancy sequences. For a comprehesive review of sequences and rules see Judd (1998). LDS are deterministic sequences which cover the domain of the integral more evenly than random samples. Discrepancy is in this case a measure of deviation from perfect uniformity over the domain of the integral. Thus to go from MC in (13) to QMC, we replace the random samples $\mathbf{X}_i$ with LDS samples. We note that the sampling of the QMC is now dependent on the dimensionality of the integral, as opposed to MC, as the LDS samples have to be drawn with respect to the dimensionality of the integral.

We consider two different types of LDS sequences, the Halton sequence and the Sobol sequence. Both sequences are popular LDS sequences, which are used in quasi-Monte Carlo (MPT) applications, (Glasserman 2004).

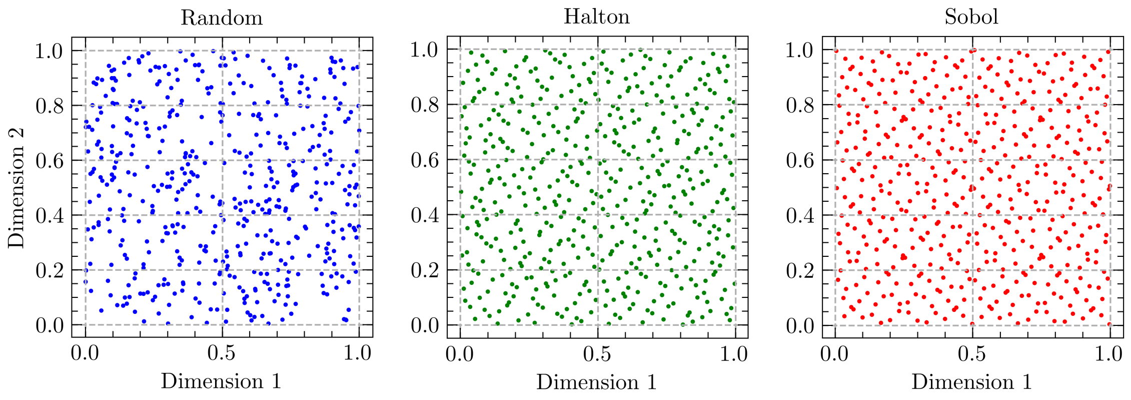
The convergence rate of MPT is:

$$\frac{(\log N)^d}{N} \tag{14}$$

Hence QMC is generally faster than MC, e.g $\frac{(\log N)^d}{N} < \frac{1}{\sqrt{N}}$ for large N and small d. We note that as dimensionality d increases, the quality of the Halton sequence decreases, as the dimensions become more correlated, Glasserman (2004). Specifically the Halton seuqnce will produce diagonal points when projected onto a 2D plane. This is displayed in figure 1.1. We therefore prefer the Sobol sequence when the dimensionality is sufficiently high, and as not to complicate matters, also use the Sobol sequence in lower dimensions, when MPT schemes are used. Figures below shows Random samples, Halton samples and Sobol samples in 2d. Second figure shows the same in 18 dimensions. Halton shows that dimension 17 and 18 are correlated.

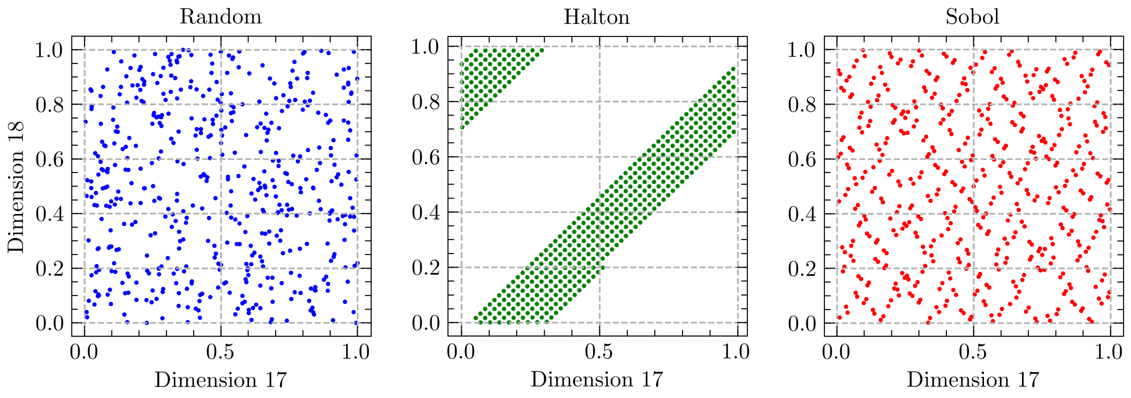
QMC is generally found to be more efficient than MC, as noted by Glasserman (2004), Judd (1998), and notably Glasserman find that dimensionality has to be quite large before

Figure 1.1: Comparison of sample generation for Monte Carlo and Quasi-Monte Carlo



Note: Each sequence was generated using $N = 500$ samples and $d = 2$ dimensions.

Figure 1.2: Comparison of sample generation for Monte Carlo and Quasi-Monte Carlo with increased dimensionality



Note: Each sequence was generated using $N = 500$ samples and $d = 18$ dimensions.

the Monte Carlo method is favorable to the quasi Monte Carlo method. Furthermore Glasserman find that while we generally might assume that N must by increase a lot when d is increased, this is not always the case in classic financial applications, as the integrals employed in these examples can often be approximated by integrals of much lower dimension. QMC therefore performs better than to be expected.

However we note that MPT lacks a straightforward variance estimator, a feature recovered through *randomized QMC*, which will be discussed in the next section.

1.1.4 Randomized Quasi-Monte carlo integration (RQMC)

Randomized quasi-Monte Carlo integration (RQMC) is a combination of MPT and MC integration. We consider the the QMC integral, i.e the equaiton of (13), using an LDS sequence. The point of randomized quasi-Monte Carlo (MPT) is then to introduce randomness to the sequence: $P_n = \{x_1, \ldots, x_n\}$. We will cover the most simple case, *Random shift* and *Scrambling* methods, however for a comprehensive review of randomization methods see Glasserman (2004). The most simple method of randomizing P_n is to add a *random shift* to each point in the sequence, using random numbers drawn from a uniform distribution of the same dimensionality as the sequence, wrapped to the interval of P_n. Hence if $x_i \in [0,1)^d$ then we add a random shift $u_i \bmod 1$, where $\bmod 1$ keeps the shift within the interval $[0,1)$. A major disadvantage of the random shift is that is changes the discrepancy properties of the sequence, and hence the quality of the sequence is lost. Scrambled nets is a method of randomization which can be applied to LDS sequences specifically. Scrambling works by applying a sequence of random permutations to the digits in the base-b representation of each coordinate in the LDS. Each digit is permuted based on the values of the digits that came before it. This structure retains the low-discrepancy properties while introducing a controlled level of randomness, which enables the calculation of variance for RQMC estimates. In multi-dimensional settings, this scrambling is applied independently to each coordinate of the sequence, allowing us to estimate variance across the entire space. Scrambling the Sobol sequence has been found to be particularily effective in financial applications, as noted by Hok and Kucherenko (2023). QMC is generally more efficient than MC, and RQMC increases the rate of converence of QMC and allows for the estimation of variance.

1.2 Value function approximation

This section covers the necessary function approximation methods used in the solution algorithm. We will cover the use of Gaussian process regression (GPR) and Baysian optimization, in order to maximize the value function of the dynamic portfolio allocation problem.

1.2.1 Gaussian process regressions (GPR)

A Gaussian process (GP) is a probabilistic model that defines a distribution over functions used to make predictions based on available data. It is specified by two functions: the mean function and the covariance function, also called the kernel. The mean function, $m(\mathbf{x})$, represents the expected value of the function at a given input $\mathbf{x}$, and the covariance function, $k(\mathbf{x}, \mathbf{x}')$, captures the covariance between function values at different input points $\mathbf{x}$ and $\mathbf{x}'$. In a GP, any finite set of input points $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ within the domain $\mathbb{R}^d$ results in the function values $\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))$ having a joint multivariate Gaussian distribution. This property enables a GP to provide a prior distribution over functions based on the defined mean and covariance.

We use GPR to estimate the value function in the dynamic portfolio allocation problem, when we are not at the terminal period, i.e., $t < T$, following Gaegauf, Scheidegger and Trojani (2023). The GP is formulated by the previously mentioned mean and covariance functions:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{15}$$

The covariance kernel function $k(\mathbf{x}, \mathbf{x}')$ can be any Mercer kernel, i.e., positive definite (Murphy 2023). Common kernel choices include the Radial Basis Function (RBF) kernel, the Matern kernel, and the Exponential kernel. We employ a Matern kernel, which, depending on the parameter ν, can be a generalization of the RBF kernel or the Exponential kernel. This choice follows Gaegauf, Scheidegger and Trojani (2023). The Matern kernel is given by:

$$k_{\text{Matern}}(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right) K_\nu \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right), \tag{16}$$

where $\|\cdot\|_2$ is the Euclidean norm, Γ is the gamma function, and K_ν is the modified Bessel function. The length scale ℓ and smoothness parameter ν are both positive. As $\nu \to \infty$, the Matern kernel converges to the RBF kernel (Gonzalvez et al. 2019). Functions from this class are k-times differentiable when $\nu > k$. When $\nu = 1/2$, the Matern kernel corresponds to the Ornstein-Uhlenbeck process (Murphy 2023), which is commonly used in financial applications, such as models of interest rates (Glasserman 2004).

Consider a training dataset $\{\mathbf{X}, \mathbf{y}\}$ with N states $\mathbf{x}_i$ and observed values $\mathbf{y}$. We assume that the observations $\mathbf{y}$ are generated by an unknown function f, such that

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2),$$

where σ_ε^2 represents the observational noise[2]. The goal is to train a GP on this dataset and then use it to predict the value function at a new state $\mathbf{x}_*$, yielding a new predicted output f_*.

The training observations $\mathbf{y}$ and the predicted noise-free function f_* have a joint Gaussian distribution:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N} \left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) + \sigma_\varepsilon^2 \mathbf{I} & k(\mathbf{X}, \mathbf{x}_*) \\ k(\mathbf{x}_*, \mathbf{X}) & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \right) \tag{17}$$

[2]The noise assumption implies that the GP model does not interpolate the data but rather fits a smooth function. This results in computational costs of $O(N)$ for the mean prediction and $O(N^2)$ for the variance prediction. For more details, see (Murphy 2023).

Here i have assumed a zero mean function[3], and the kernel function is the Matern kernel. The posterior distribution of the predicted value function f_* given the training data is then a multivariate normal (Murphy 2023), with mean:

$$\tilde{\mu}(\mathbf{x}) = k(\mathbf{x}_*, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma_\varepsilon^2 \mathbf{I}]^{-1} \mathbf{y}, \tag{18}$$

And covariance:

$$\tilde{k}(\mathbf{x}_*, \mathbf{x}_*^{'}) = k(\mathbf{x}_*, \mathbf{x}_*^{'}) - k(\mathbf{x}_*, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma_\varepsilon^2 \mathbf{I}]^{-1} k(\mathbf{X}, \mathbf{x}_*^{'}) \tag{19}$$

Therefore in order to predict the value function at a new state $\mathbf{x}_*$, we need to compute the mean and covariance. This step is computationally burdensome as we have to compute the four covariance matrices in the joint distribution (17). Afterwards we can compute predictions using the mean function (18) and the covariance function (19) can be used to compute error bands on our predictions.

As noted, training and predicting with a GP is computationally expensive. I will therefore introduce the methods employed to reduce the computational burden of the GP.

We use automatic relevance detection (ARD) which is a modification to the Matern kernel to use a length scale for each dimension, ℓ_i. Dimensions with low impact has a high length scale, and are effectively ignored. Note that this is not the same as Lasso, as these coefficients are not set to 0. We use Structured Kernel Interpolation for Products (SKIP) to reduce the computational burden of computen the matrices in the joint distribution (17).

Use LancZos Variance Estimate (Love) and SKIP to reduce computational burden.

Here is the documentation in my package `https://docs.gpytorch.ai/en/stable/examples/02_Scalable_Exact_GPs/index.html`

And here is a paper on the subject `https://arxiv.org/pdf/1803.06058`.

1.3 Approximating the No trade region

Since i now have introduced methods to approximate the next-period value function v_{t+1}, and methods for evaluating the expectation $\mathbb{E}[\cdot]$ over known distributions, we can now approximate the no-trade-region (NTR) using a dynamic programming (DP) scheme. In order to do this some assummptions regarding the unknown NTR are formed, these are drawn directly from (Gaegauf, Scheidegger and Trojani 2023)

Assumption 1. *The NTR is a D-dimensional convex polytope.*

A polytope is a generalization of a polyhedron (polytope in 2D), which is a geometric object

[3]Zero mean ... XXXX

with flat sides and straight edges. The convex polytope is a polytope which bounds a convex set, and can therefore be defined by a convex hull. Therefore any linear combination of points in the NTR or on the boundary of the NTR is also in the NTR. ***Wikipedia reference her?***

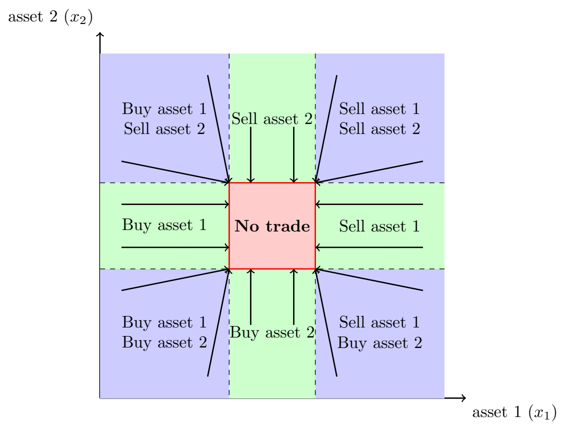
Assumption 2. *The NTR has 2^D vertices.*
This assumption is regarding the shape of the NTR. Note that if the actual NTR has less than 2^D vertices, the approximation will be close to the actual shape, as the approximated vertices will be on top of each other. However if the NTR as more then 2^D vertices, then the approximation will be a simplification of the actual shape.

With these two assumptions in place, a strategy for approximating the NTR can be formed with few initial points. Given assumption 2 we can approximate the NTR by using 2^D points, which are the vertices of the NTR, and by assumption 1 we can approximate the NTR by using the convex hull of these points, i.e connecting the vertices to form the outer hull.

I can leverage the following intution from (Gaegauf, Scheidegger and Trojani 2023), and from **??**: For any point outside the NTR, the optimal policy is to trade towards the boundary of the NTR. Since each point on the boundary of the NTR is optimal, the optimal trading route minimizes the distance, and hence the optimal trading route is a straight line to the boundary of the NTR. If the points ahre chosen correctly, the optimal trading route will be to a vertex of the NTR. This is seen in the figure below:

If one considers the example in figure 1.3, i can effectively approximate the NTR, by

Figure 1.3: Illustration of the no-trade region (NTR) and the optimal policies outside this.



This is a schematic NTR. Blue regions are regions where optimal policy $\boldsymbol{\delta}$ is to buy in one asset and sell in another. Green regions are regions where the optimal policy is to hold in one asset and adjust the other. This figure is a recreation of Figure 1. in Gaegauf, Scheidegger and Trojani (2023).

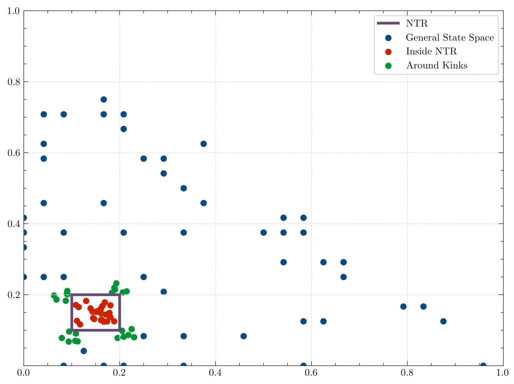
sampling a point in each of the blue regions, and then solving the optimization problem to find the vertices. When the NTR is unknown, sampling from the blue regions seem difficult at a first glance. However, i can sample bounds are the vertices of the simplex that cover the feasible space, and the midpoint between these. This sample scheme leads to the following points in the 2-dimensional case:

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Extensions of this sampling scheme to higher dimensions is trivial. This should effectively cover the feasible space, and allow for approximating the NTR. Note that this sampling scheme only covers NTR with no borrowing, and no short-selling as noted in (Gaegauf, Scheidegger and Trojani 2023). If borrowing and short-selling were introduced, we would have to set some bounds on the borrowing and short-selling, and then sample from these bounds. Effectively creating a square (cube / hypercube) around the feasible space, and then sample the vertices of this space.

1.3.1 Strategic point sampling

Figure 1.4: A designed sampling strategy

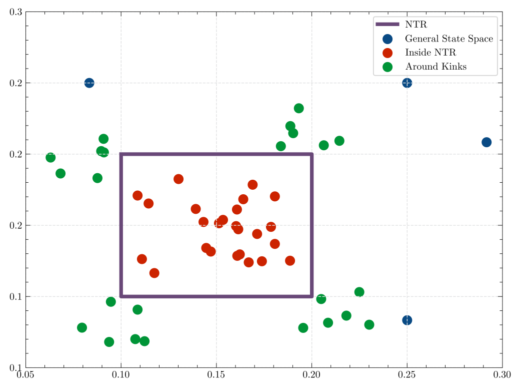


Note: Sample consists of $N = 100$ points, with 51 points in the general state space, 25 points inside the NTR and 24 points around the NTR kinks.

1.4 Leveraging the no trade region

1.5 Final solution algorithm

Figure 1.5: Zoom in on the NTR



Note: Sample consists of $N = 100$ points, with 51 points in the general state space, 25 points inside the NTR and 24 points around the NTR kinks.