Basket & Asian Option Pricing

Blair Azzopardi

September 29, 2024

Contents

1 Introduction

Follows the derivation and implementation of several papers around pricing Basket and Asian options (Borovkova et al., 2007). The derivation utilizes Sympy (Meurer et al., 2017) to remove/reduce

the tedious algebraic calculations; it's also used to generate code used for Python prototype and C++ implementations (Azzopardi, 2024). The Python implementation uses numpy (Harris et al., 2020) and Scipy (Virtanen et al., 2020) and its integration with minpack. The C++ implementation uses Eigen (Guennebaud et al., 2010) and Ceres (Agarwal et al., 2023).

2 Derivation

In what follows, the key definitions such as the Linear Log Normal (LLN) and Black 76 formula have been coded as Sympy expressions and mostly everything else has been derived programmatically then checked manually. Some extensions to the Sympy platform are described in the appendices.

2.1 Approximating distributions

Definitions:

If $X_N \sim N(\mu, \sigma^2)$, then

$$\begin{aligned} X_{LN} &= e^{X_N} \sim LN(\mu, \sigma^2) \\ X_{SLN} &= e^{X_N} + \tau \sim SLN(\mu, \sigma^2, \tau) \\ X_{NLN} &= -e^{X_N} \sim NLN(\mu, \sigma^2) \end{aligned}$$

Generalize with linear log normal:

$$X_{LLN} = \kappa \left(\tau + e^{X_N}\right)$$

2.1.1 Moments

$$\text{mgf}_N = \text{E}\left[e^{tX_N}\right] = e^{\mu t + \frac{\sigma^2 t^2}{2}}$$

For LLN

$$\begin{aligned} \text{E}\left[\left(-\tau + \frac{X_{LLN}}{\kappa}\right)^t\right] &= \text{E}\left[e^{tX_N}\right] = \text{mgf}_N \\ &= e^{\mu t + \frac{\sigma^2 t^2}{2}} \end{aligned}$$

1st

$$\begin{aligned}
\mathrm{E}\left[-\tau + \frac{X_{LLN}}{\kappa}\right] &= e^{\mu+\frac{\sigma^2}{2}} \\
-\tau + \frac{\mathrm{E}\left[X_{LLN}\right]}{\kappa} &= e^{\mu+\frac{\sigma^2}{2}} \\
\mathrm{E}\left[X_{LLN}\right] &= \kappa\left(\tau + e^{\mu+\frac{\sigma^2}{2}}\right)
\end{aligned}$$

2nd

$$\begin{aligned}
\mathrm{E}\left[\left(-\tau + \frac{X_{LLN}}{\kappa}\right)^2\right] &= e^{2\mu+2\sigma^2} \\
\tau^2 - \frac{2\tau\,\mathrm{E}\left[X_{LLN}\right]}{\kappa} + \frac{\mathrm{E}\left[X_{LLN}^2\right]}{\kappa^2} &= e^{2\mu+2\sigma^2} \\
\mathrm{E}\left[X_{LLN}^2\right] &= \kappa^2\left(\tau^2 + 2\tau e^{\mu+\frac{\sigma^2}{2}} + e^{2\mu+2\sigma^2}\right)
\end{aligned}$$

3rd

$$\begin{aligned}
\mathrm{E}\left[\left(-\tau + \frac{X_{LLN}}{\kappa}\right)^3\right] &= e^{3\mu+\frac{9\sigma^2}{2}} \\
-\tau^3 + \frac{3\tau^2\,\mathrm{E}\left[X_{LLN}\right]}{\kappa} - \frac{3\tau\,\mathrm{E}\left[X_{LLN}^2\right]}{\kappa^2} + \frac{\mathrm{E}\left[X_{LLN}^3\right]}{\kappa^3} &= e^{3\mu+\frac{9\sigma^2}{2}} \\
\mathrm{E}\left[X_{LLN}^3\right] &= \kappa^3\left(\tau^3 + 3\tau^2 e^{\mu+\frac{\sigma^2}{2}} + 3\tau e^{2\mu+2\sigma^2} + e^{3\mu+\frac{9\sigma^2}{2}}\right)
\end{aligned}$$

2.1.2 Solve for sigma as system

Can express above moments as system of 3 equations with several unknows.

$$\begin{aligned}
m_1 &= \kappa\left(\tau + e^{\mu+\frac{\sigma^2}{2}}\right) \\
m_2 &= \kappa^2\left(\tau^2 + 2\tau e^{\mu+\frac{\sigma^2}{2}} + e^{2\mu+2\sigma^2}\right) \\
m_3 &= \kappa^3\left(\tau^3 + 3\tau^2 e^{\mu+\frac{\sigma^2}{2}} + 3\tau e^{2\mu+2\sigma^2} + e^{3\mu+\frac{9\sigma^2}{2}}\right)
\end{aligned} \tag{SYS1}$$

Giving solutions:

$$\sigma = \sqrt{\log\left(\frac{\kappa^2\tau^2 - 2\kappa m_1\tau + m_2}{(\kappa\tau - m_1)^2}\right)}$$
$$\mu = \log\left(\left|-\frac{(-\kappa\tau + m_1)^2}{\kappa\sqrt{\kappa^2\tau^2 - 2\kappa m_1\tau + m_2}}\right|\right)$$

Where μ chooses a square root in the denominator that makes the input to logarithm positive.

2.1.3 Black Formula Re-expression

Recall relation $X_{LLN} = \kappa\left(\tau + e^{X_N}\right)$, implies

$$(-K + B(T)_{LLN})^+ = (-K + \kappa\left(\tau + B(T)_{LN}\right))^+$$

For various values of κ and τ we have:

κ	τ	X	payoff	CP	F_{Black}	K_{Black}
1	0	LN	$(B(T)_{LN} - K)^+$	C	F	K
-1	0	NLN	$(-B(T)_{LN} - K)^+$	P	$-F$	$-K$
1	non-zero	SLN	$(B(T)_{LN} + \tau - K)^+$	C	$F - \tau$	$K - \tau$
-1	non-zero	NSLN	$(-B(T)_{LN} - \tau - K)^+$	P	$-(F + \tau)$	$-(K + \tau)$

Black 76 formula for pricing call/put option is:

$$P_{cp} = \pm_{cp}\left(F\Phi(\pm_{cp}d_1) - K\Phi(\pm_{cp}d_2)\right)e^{-Tr}$$
$$d_1 = \frac{\frac{T\sigma^2}{2} + \log\left(\frac{F}{K}\right)}{\sqrt{T}\sigma}$$
$$d_2 = -\sqrt{T}\sigma + \frac{\frac{T\sigma^2}{2} + \log\left(\frac{F}{K}\right)}{\sqrt{T}\sigma}$$

For LN, corresponds with:

$$P_c = \left(F\Phi(d_1) - K\Phi(d_2)\right)e^{-Tr}$$
$$d_1 = \frac{\log\left(\frac{F}{K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma}$$
$$d_2 = \frac{\log\left(\frac{F}{K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} - \sqrt{T}\sigma$$
$$\sigma = \sqrt{\log\left(\frac{m_2}{m_1^2}\right)}$$

For NLN, corresponds with:

$$
\begin{aligned}
P_c &= -\left(-F\Phi(-d_1) + K\Phi(-d_2)\right)e^{-Tr} \\
d_1 &= \frac{\log\left(\frac{F}{K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} \\
d_2 &= \frac{\log\left(\frac{F}{K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} - \sqrt{T}\sigma \\
\sigma &= \sqrt{\log\left(\frac{m_2}{m_1^2}\right)}
\end{aligned}
$$

For SLN, corresponds with:

$$
\begin{aligned}
P_c &= \left((F-\tau)\,\Phi(d_1) - (K-\tau)\,\Phi(d_2)\right)e^{-Tr} \\
d_1 &= \frac{\log\left(\frac{-\tau+F}{-\tau+K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} \\
d_2 &= \frac{\log\left(\frac{-\tau+F}{-\tau+K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} - \sqrt{T}\sigma \\
\sigma &= \sqrt{\log\left(\frac{\tau^2 + m_2 - 2m_1\tau}{(\tau - m_1)^2}\right)}
\end{aligned}
$$

For NSLN, corresponds with:

$$
\begin{aligned}
P_c &= -\left((-F-\tau)\,\Phi(-d_1) - (-K-\tau)\,\Phi(-d_2)\right)e^{-Tr} \\
d_1 &= \frac{\log\left(\frac{-\tau-F}{-\tau-K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} \\
d_2 &= \frac{\log\left(\frac{-\tau-F}{-\tau-K}\right) + \frac{T\sigma^2}{2}}{\sqrt{T}\sigma} - \sqrt{T}\sigma \\
\sigma &= \sqrt{\log\left(\frac{\tau^2 + m_2 + 2m_1\tau}{(-\tau - m_1)^2}\right)}
\end{aligned}
$$

2.1.4 Correlated expectation of product

Mgf for normal

$$
E(e^{X_N}) = e^{\mu_X + \frac{\sigma_X^2}{2}} = \frac{E(X_{LNN}) - \tau}{\kappa}
$$

Lemma for $E(e^{\sum X_{i,N}})$

$$\begin{aligned}
\sum X_{i,N} &\sim N(\sum \mu_i, \sum \sigma_i^2 + 2\sum_{i<j} \rho_{ij}\sigma_i\sigma_j) \\
\implies Z = e^{\sum X_{i,N}} &\sim LN(\sum \mu_i, \sum \sigma_i^2 + 2\sum_{i<j} \rho_{ij}\sigma_i\sigma_j) \\
E(Z) = E(e^{\sum X_{i,N}}) &= e^{\mu_Z + \sigma_Z^2/2} = e^{\sum \mu_i + (\sum \sigma_i^2 + 2\sum_{i<j}\rho_{ij}\sigma_i\sigma_j)/2} \\
&= e^{\sum_{i<j}\rho_{ij}\sigma_i\sigma_j} \prod e^{\mu_i + \sigma_i^2/2} \\
&= e^{\sum_{i<j}\rho_{ij}\sigma_i\sigma_j} \prod E(e^{X_i,N})
\end{aligned}$$

2.2 Basket distribution

Note $E(F(t)) = F(0)$ when process follows $dF = \sigma F dz$ in risk neutral world. See p369 (Hull, 2012).

Basket defined as:

Basket assumed to have LN distribution so we can apply BS logic.

$$B(t) = \sum_{i=1}^{N} F(t)_i a_i$$

2.2.1 Moments

1st

$$\mathrm{E}\left[B(t)\right] = \sum_{i=1}^{N} F(0)_i a_i$$

2nd

$$\mathrm{E}\left[B(t)^2\right] = \sum_{\substack{1\le i\le N \\ 1\le j\le N}} e^{T\rho_{i,j}\sigma_i\sigma_j} F(0)_i F(0)_j a_i a_j$$

3rd

$$\mathrm{E}\left[B(t)^3\right] = \sum_{\substack{1\le i\le N \\ 1\le j\le N \\ 1\le k\le N}} e^{T\rho_{i,j}\sigma_i\sigma_j} e^{T\rho_{i,k}\sigma_i\sigma_k} e^{T\rho_{j,k}\sigma_j\sigma_k} F(0)_i F(0)_j F(0)_k a_i a_j a_k$$

3 Greeks

3.1 Derivation of formulae

For LN, corresponds with:

$$\begin{aligned}
\frac{\partial c}{\partial r} &= -Tc(T) \\
\frac{\partial c}{\partial T} &= -rc(T) + K\phi(d_2)e^{-Tr}\frac{\partial V}{\partial T} \\
\frac{\partial c}{\partial F} &= \left(\Phi(d_1)\frac{\partial M_1}{\partial F} + K\phi(d_2)\frac{\partial V}{\partial F}\right)e^{-Tr} \\
\frac{\partial c}{\partial \sigma} &= K\phi(d_2)e^{-Tr}\frac{\partial V}{\partial \sigma}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial V}{\partial T} &= \frac{\frac{\partial M_2}{\partial T}}{2M_2V} \\
\frac{\partial V}{\partial F} &= -\frac{2M_2\frac{\partial M_1}{\partial F} - M_1\frac{\partial M_2}{\partial F}}{2M_1M_2V} \\
\frac{\partial V}{\partial \sigma} &= \frac{\frac{\partial M_2}{\partial \sigma}}{2M_2V}
\end{aligned}$$

For SLN, corresponds with:

$$\begin{aligned}
\frac{\partial c}{\partial r} &= -Tc(T) \\
\frac{\partial c}{\partial T} &= \left(-\left(-\Phi(d_2) + \Phi(d_1)\right)\frac{\partial \tau}{\partial T} + \left(-\tau + K\right)\phi(d_2)\frac{\partial V}{\partial T} - rc(T)e^{Tr}\right)e^{-Tr} \\
\frac{\partial c}{\partial F} &= \left(\Phi(d_2)\frac{\partial \tau}{\partial F} + \left(-\frac{\partial \tau}{\partial F} + \frac{\partial M_1}{\partial F}\right)\Phi(d_1) + \left(-\tau + K\right)\phi(d_2)\frac{\partial V}{\partial F}\right)e^{-Tr} \\
\frac{\partial c}{\partial \sigma} &= \left(\Phi(d_2)\frac{\partial \tau}{\partial \sigma} - \Phi(d_1)\frac{\partial \tau}{\partial \sigma} + \left(-\tau + K\right)\phi(d_2)\frac{\partial V}{\partial \sigma}\right)e^{-Tr}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial V}{\partial T} &= -\frac{-\tau\frac{\partial M_2}{\partial T} + 2M_2\frac{\partial \tau}{\partial T} + M_1\frac{\partial M_2}{\partial T} - 2M_1\frac{\partial \tau}{\partial T}}{2\left(-\tau + M_1\right)\left(-\tau - M_2 + 2M_1\tau\right)V} \\
\frac{\partial V}{\partial F} &= -\frac{-\tau\frac{\partial M_2}{\partial F} + 2M_2\frac{\partial \tau}{\partial F} - 2M_2\frac{\partial M_1}{\partial F} + M_1\frac{\partial M_2}{\partial F} + 2M_1\tau\frac{\partial M_1}{\partial F} - 2M_1\frac{\partial \tau}{\partial F}}{2\left(-\tau + M_1\right)\left(-\tau - M_2 + 2M_1\tau\right)V} \\
\frac{\partial V}{\partial \sigma} &= -\frac{-\tau\frac{\partial M_2}{\partial \sigma} + 2M_2\frac{\partial \tau}{\partial \sigma} + M_1\frac{\partial M_2}{\partial \sigma} - 2M_1\frac{\partial \tau}{\partial \sigma}}{2\left(-\tau + M_1\right)\left(-\tau - M_2 + 2M_1\tau\right)V}
\end{aligned}$$

For NLN, corresponds with:

$$\begin{aligned}
\frac{\partial c}{\partial r} &= -Tc(T) \\
\frac{\partial c}{\partial T} &= -rc(T) - K\phi(-d_2)e^{-Tr}\frac{\partial V}{\partial T} \\
\frac{\partial c}{\partial F} &= \left(\Phi(-d_1)\frac{\partial M_1}{\partial F} - K\phi(-d_2)\frac{\partial V}{\partial F}\right)e^{-Tr} \\
\frac{\partial c}{\partial \sigma} &= -K\phi(-d_2)e^{-Tr}\frac{\partial V}{\partial \sigma}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial V}{\partial T} &= \frac{\frac{\partial M_2}{\partial T}}{2M_2V} \\
\frac{\partial V}{\partial F} &= -\frac{2M_2\frac{\partial M_1}{\partial F} - M_1\frac{\partial M_2}{\partial F}}{2M_1M_2V} \\
\frac{\partial V}{\partial \sigma} &= \frac{\frac{\partial M_2}{\partial \sigma}}{2M_2V}
\end{aligned}$$

For NSLN, corresponds with:

$$\begin{aligned}
\frac{\partial c}{\partial r} &= -Tc(T) \\
\frac{\partial c}{\partial T} &= \left(\left(-\Phi(-d_2) + \Phi(-d_1)\right)\frac{\partial \tau}{\partial T} - (\tau + K)\,\phi(-d_2)\frac{\partial V}{\partial T} - rc(T)e^{Tr}\right)e^{-Tr} \\
\frac{\partial c}{\partial F} &= \left(-\Phi(-d_2)\frac{\partial \tau}{\partial F} + \left(\frac{\partial \tau}{\partial F} + \frac{\partial M_1}{\partial F}\right)\Phi(-d_1) - (\tau + K)\,\phi(-d_2)\frac{\partial V}{\partial F}\right)e^{-Tr} \\
\frac{\partial c}{\partial \sigma} &= \left(-\Phi(-d_2)\frac{\partial \tau}{\partial \sigma} + \Phi(-d_1)\frac{\partial \tau}{\partial \sigma} - (\tau + K)\,\phi(-d_2)\frac{\partial V}{\partial \sigma}\right)e^{-Tr}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial V}{\partial T} &= \frac{\tau\frac{\partial M_2}{\partial T} - 2M_2\frac{\partial \tau}{\partial T} + M_1\frac{\partial M_2}{\partial T} + 2M_1\frac{\partial \tau}{\partial T}}{2\left(\tau + M_1\right)\left(\tau + M_2 + 2M_1\tau\right)V} \\
\frac{\partial V}{\partial F} &= -\frac{-\tau\frac{\partial M_2}{\partial F} + 2M_2\frac{\partial \tau}{\partial F} + 2M_2\frac{\partial M_1}{\partial F} - M_1\frac{\partial M_2}{\partial F} + 2M_1\tau\frac{\partial M_1}{\partial F} - 2M_1\frac{\partial \tau}{\partial F}}{2\left(\tau + M_1\right)\left(\tau + M_2 + 2M_1\tau\right)V} \\
\frac{\partial V}{\partial \sigma} &= \frac{\tau\frac{\partial M_2}{\partial \sigma} - 2M_2\frac{\partial \tau}{\partial \sigma} + M_1\frac{\partial M_2}{\partial \sigma} + 2M_1\frac{\partial \tau}{\partial \sigma}}{2\left(\tau + M_1\right)\left(\tau + M_2 + 2M_1\tau\right)V}
\end{aligned}$$

3.2 Partial derivatives of top moments

Define parameters and moments of top level process as:

$$p_{top} = \begin{bmatrix} \tau \\ \mu \\ \sigma \end{bmatrix}$$

$$m_{top} = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} \kappa\left(\tau + e^{\mu+\frac{\sigma^2}{2}}\right) \\ \kappa^2\left(\tau^2 + 2\tau e^{\mu+\frac{\sigma^2}{2}} + e^{2\mu+2\sigma^2}\right) \\ \kappa^3\left(\tau^3 + 3\tau^2 e^{\mu+\frac{\sigma^2}{2}} + 3\tau e^{2\mu+2\sigma^2} + e^{3\mu+\frac{9\sigma^2}{2}}\right) \end{bmatrix}$$

The corresponding Jacobian, J_{top}, is as follows:

$$J_{top} = \begin{bmatrix} \frac{\partial M_1}{\partial \tau} & \frac{\partial M_1}{\partial \mu} & \frac{\partial M_1}{\partial \sigma} \\ \frac{\partial M_2}{\partial \tau} & \frac{\partial M_2}{\partial \mu} & \frac{\partial M_2}{\partial \sigma} \\ \frac{\partial M_3}{\partial \tau} & \frac{\partial M_3}{\partial \mu} & \frac{\partial M_3}{\partial \sigma} \end{bmatrix}$$

$$= \begin{bmatrix} \kappa & \kappa e^{\mu+\frac{\sigma^2}{2}} & \kappa\sigma e^{\mu+\frac{\sigma^2}{2}} \\ \kappa^2\cdot\left(2\tau+2e^{\mu+\frac{\sigma^2}{2}}\right) & \kappa^2\cdot\left(2\tau e^{\mu+\frac{\sigma^2}{2}}+2e^{2\mu+2\sigma^2}\right) & \kappa^2\cdot\left(2\sigma\tau e^{\mu+\frac{\sigma^2}{2}}+4\sigma e^{2\mu+2\sigma^2}\right) \\ \kappa^3\cdot\left(3\tau^2+6\tau e^{\mu+\frac{\sigma^2}{2}}+3e^{2\mu+2\sigma^2}\right) & \kappa^3\cdot\left(3\tau^2 e^{\mu+\frac{\sigma^2}{2}}+6\tau e^{2\mu+2\sigma^2}+3e^{3\mu+\frac{9\sigma^2}{2}}\right) & \kappa^3\cdot\left(3\sigma\tau^2 e^{\mu+\frac{\sigma^2}{2}}+12\sigma\tau e^{2\mu+2\sigma^2}+9\sigma e^{3\mu+\frac{9\sigma^2}{2}}\right) \end{bmatrix}$$

The Jacobian is very close to (Borovkova et al., 2007). $\frac{\partial m_1}{\partial \sigma}$ is different here but corresponds with later paper (Borovkova & Permana, 2007).

3.3 Partial derivatives of component moments

Define parameters and moments at component level processes as:

$$p_{comp} = \begin{bmatrix} F(0)_l \\ \sigma_l \\ T \end{bmatrix}$$

$$m_{comp} = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} F(0)_i a_i \\ \sum_{\substack{1\le i\le N \\ 1\le j\le N}} e^{T\rho_{i,j}\sigma_i\sigma_j} F(0)_i F(0)_j a_i a_j \\ \sum_{\substack{1\le i\le N \\ 1\le j\le N \\ 1\le k\le N}} e^{T\rho_{i,j}\sigma_i\sigma_j} e^{T\rho_{i,k}\sigma_i\sigma_k} e^{T\rho_{j,k}\sigma_j\sigma_k} F(0)_i F(0)_j F(0)_k a_i a_j a_k \end{bmatrix}$$

The corresponding Jacobian is as follows:

$$
\begin{aligned}
J_{comp} &= \begin{bmatrix} \frac{\partial M_1}{\partial F(0)_l} & \frac{\partial M_1}{\partial \sigma_l} & \frac{\partial M_1}{\partial T} \\ \frac{\partial M_2}{\partial F(0)_l} & \frac{\partial M_2}{\partial \sigma_l} & \frac{\partial M_2}{\partial T} \\ \frac{\partial M_3}{\partial F(0)_l} & \frac{\partial M_3}{\partial \sigma_l} & \frac{\partial M_3}{\partial T} \end{bmatrix} \\
&= \begin{bmatrix} a_l & 0 & 0 \\ 2a_l \sum_{i=1}^{N} e^{T\rho_{l,i}\sigma_i\sigma_l} F(0)_i a_i & 2TF(0)_l a_l \sum_{i=1}^{N} e^{T\rho_{l,i}\sigma_i\sigma_l} F(0)_i a_i \rho_{l,i}\sigma_i & \sum_{\substack{1\le i\le N \\ 1\le j\le N}} e^{T\rho_{i,j}\sigma_i\sigma_j} F(0)_i F(0)_j a_i a_j \rho_{i,j}\sigma_i\sigma_j \\ 3a_l \sum_{\substack{1\le j\le N \\ 1\le i\le N}} e^{T\rho_{j,i}\sigma_i\sigma_j} e^{T\rho_{l,i}\sigma_i\sigma_l} e^{T\rho_{l,j}\sigma_j\sigma_l} F(0)_i F(0)_j a_i a_j & 6TF(0)_l a_l \sum_{\substack{1\le j\le N \\ 1\le i\le N}} e^{T\rho_{j,i}\sigma_i\sigma_j} e^{T\rho_{l,i}\sigma_i\sigma_l} e^{T\rho_{l,j}\sigma_j\sigma_l} F(0)_i F(0)_j a_i a_j \rho_{l,j}\sigma_j & \sum_{\substack{1\le i\le N \\ 1\le j\le N \\ 1\le k\le N}} \left(\rho_{i,j}\sigma_i\sigma_j + \rho_{i,k}\sigma_i\sigma_k + \rho_{j,k}\sigma_j\sigma_k\right) e^{T\rho_{i,j}\sigma_i\sigma_j} e^{T\rho_{i,k}\sigma_i\sigma_k} e^{T\rho_{j,k}\sigma_j\sigma_k} F(0)_i F(0)_j F(0)_k a_i a_j a_k \end{bmatrix}
\end{aligned}
$$

Again, the Jacobian is very close to (Borovkova et al., 2007) but with a slight difference in term $\frac{\partial M_3}{\partial \sigma_l}$.

The derivative in terms for rho (for cega greek) is as follows:

$$
\frac{d}{d\rho_{l,m}} m_{comp} = \begin{bmatrix} 0 \\ Te^{T\rho_{l,m}\sigma_l\sigma_m} F(0)_l F(0)_m a_l a_m \sigma_l \sigma_m \\ 3Te^{T\rho_{l,m}\sigma_l\sigma_m} F(0)_l F(0)_m a_l a_m \sigma_l \sigma_m \sum_{i=1}^{N} e^{T\rho_{l,i}\sigma_i\sigma_l} e^{T\rho_{m,i}\sigma_i\sigma_m} F(0)_i a_i \end{bmatrix}
$$

Notable that here the terms are different to (Borovkova et al., 2007) by a factor of 2.

3.4 Partial derivatives of parameters

The forumulae above have several cross partial derivatives that can be calculated as follows:

$$
\begin{aligned}
J_{top} \times J_{params} &= \begin{bmatrix} \frac{\partial M_1}{\partial \tau} & \frac{\partial M_1}{\partial \mu} & \frac{\partial M_1}{\partial \sigma} \\ \frac{\partial M_2}{\partial \tau} & \frac{\partial M_2}{\partial \mu} & \frac{\partial M_2}{\partial \sigma} \\ \frac{\partial M_3}{\partial \tau} & \frac{\partial M_3}{\partial \mu} & \frac{\partial M_3}{\partial \sigma} \end{bmatrix} \times \begin{bmatrix} \frac{\partial \tau}{\partial F(0)_l} & \frac{\partial \tau}{\partial \sigma_l} & \frac{\partial \tau}{\partial T} \\ \frac{\partial \mu}{\partial F(0)_l} & \frac{\partial \mu}{\partial \sigma_l} & \frac{\partial \mu}{\partial T} \\ \frac{\partial \sigma}{\partial F(0)_l} & \frac{\partial \sigma}{\partial \sigma_l} & \frac{\partial \sigma}{\partial T} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial M_1}{\partial \mu}\frac{\partial \mu}{\partial F(0)_l} + \frac{\partial M_1}{\partial \sigma}\frac{\partial \sigma}{\partial F(0)_l} + \frac{\partial M_1}{\partial \tau}\frac{\partial \tau}{\partial F(0)_l} & \frac{\partial M_1}{\partial \mu}\frac{\partial \mu}{\partial \sigma_l} + \frac{\partial M_1}{\partial \sigma}\frac{\partial \sigma}{\partial \sigma_l} + \frac{\partial M_1}{\partial \tau}\frac{\partial \tau}{\partial \sigma_l} & \frac{\partial M_1}{\partial \mu}\frac{\partial \mu}{\partial T} + \frac{\partial M_1}{\partial \sigma}\frac{\partial \sigma}{\partial T} + \frac{\partial M_1}{\partial \tau}\frac{\partial \tau}{\partial T} \\ \frac{\partial M_2}{\partial \mu}\frac{\partial \mu}{\partial F(0)_l} + \frac{\partial M_2}{\partial \sigma}\frac{\partial \sigma}{\partial F(0)_l} + \frac{\partial M_2}{\partial \tau}\frac{\partial \tau}{\partial F(0)_l} & \frac{\partial M_2}{\partial \mu}\frac{\partial \mu}{\partial \sigma_l} + \frac{\partial M_2}{\partial \sigma}\frac{\partial \sigma}{\partial \sigma_l} + \frac{\partial M_2}{\partial \tau}\frac{\partial \tau}{\partial \sigma_l} & \frac{\partial M_2}{\partial \mu}\frac{\partial \mu}{\partial T} + \frac{\partial M_2}{\partial \sigma}\frac{\partial \sigma}{\partial T} + \frac{\partial M_2}{\partial \tau}\frac{\partial \tau}{\partial T} \\ \frac{\partial M_3}{\partial \mu}\frac{\partial \mu}{\partial F(0)_l} + \frac{\partial M_3}{\partial \sigma}\frac{\partial \sigma}{\partial F(0)_l} + \frac{\partial M_3}{\partial \tau}\frac{\partial \tau}{\partial F(0)_l} & \frac{\partial M_3}{\partial \mu}\frac{\partial \mu}{\partial \sigma_l} + \frac{\partial M_3}{\partial \sigma}\frac{\partial \sigma}{\partial \sigma_l} + \frac{\partial M_3}{\partial \tau}\frac{\partial \tau}{\partial \sigma_l} & \frac{\partial M_3}{\partial \mu}\frac{\partial \mu}{\partial T} + \frac{\partial M_3}{\partial \sigma}\frac{\partial \sigma}{\partial T} + \frac{\partial M_3}{\partial \tau}\frac{\partial \tau}{\partial T} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial M_1}{\partial F(0)_l} & \frac{\partial M_1}{\partial \sigma_l} & \frac{\partial M_1}{\partial T} \\ \frac{\partial M_2}{\partial F(0)_l} & \frac{\partial M_2}{\partial \sigma_l} & \frac{\partial M_2}{\partial T} \\ \frac{\partial M_3}{\partial F(0)_l} & \frac{\partial M_3}{\partial \sigma_l} & \frac{\partial M_3}{\partial T} \end{bmatrix} = J_{comp}
\end{aligned}
$$

So we can just do:

$$J_{params} = J_{top}^{-1} \times J_{comp}$$

4 Pricer

Build the prototype code in Python. Test against published papers.

4.1 Comparison against published results

The results below use the Python prototype and C++ implementation to check against published papers.

4.1.1 Closed Form Approach to Valuation and Hedging of Basket Options

We compare results against (Borovkova et al., 2007).

Table 2: Comparison to (Borovkova et al., 2007)

basket	skew	tau	bdist	odist	btp	otp	diff
1	3.7804	-35.9532	SLN	SLN	7.7510	7.7514	0.0004
2	-4.3192	-121.5736	NSLN	NSLN	16.9100	16.9105	0.0005
3	10.3048	3.5552	LN	LN	10.8440	10.8439	0.0001
4	-52.2058	-0.5640	NLN	NLN	1.9580	1.9576	0.0004
5	-5.9440	-32.0423	NSLN	NSLN	7.7590	7.7587	0.0003
6	5.5957	-30.3085	SLN	SLN	9.0260	9.0214	0.0046

4.1.2 Asian basket options and implied correlations in energy markets

We compare results against (Borovkova & Permana, 2007).

Table 3: Comparison to (Borovkova & Permana, 2007)

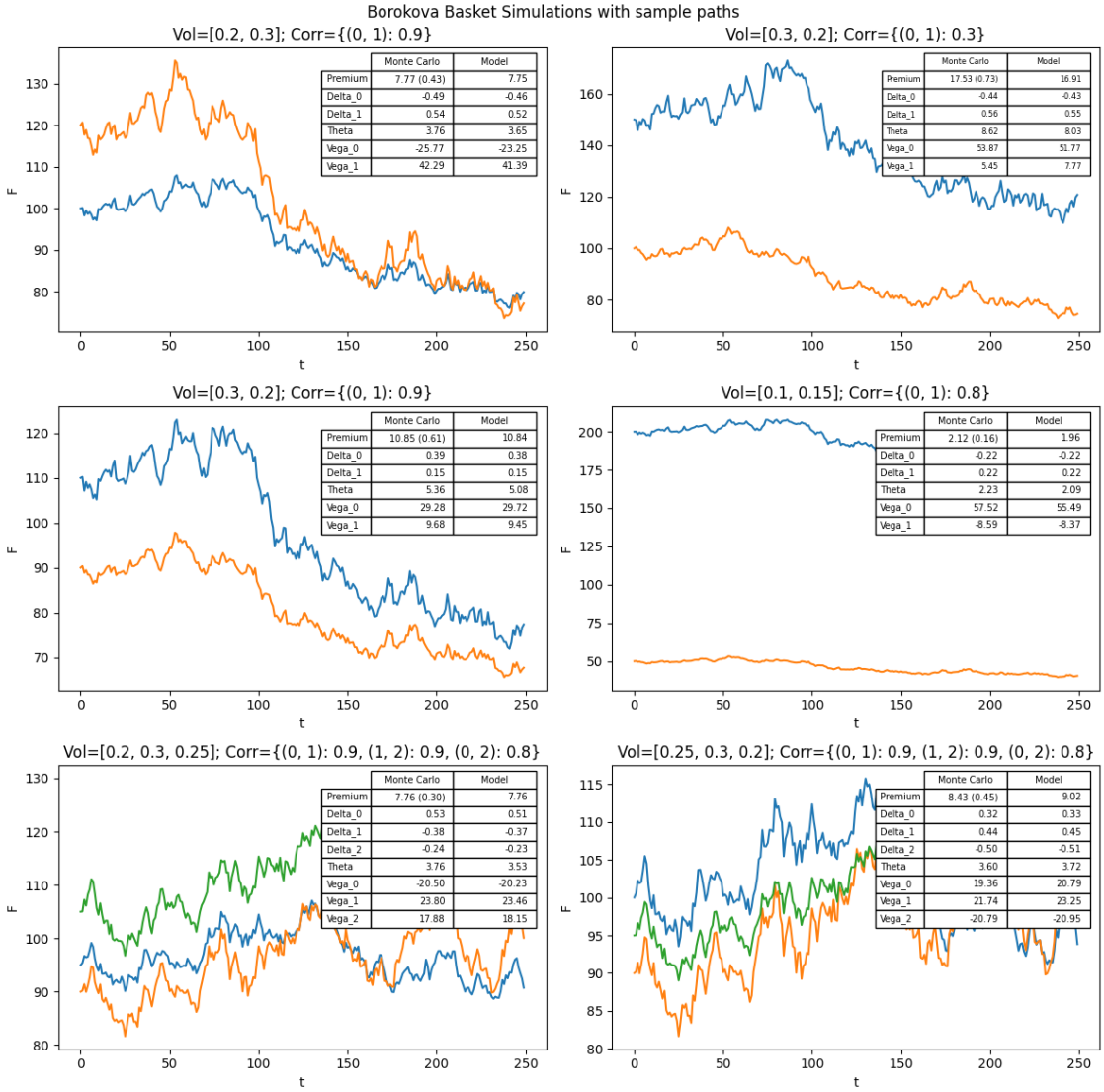
basket	skew	tau	bdist	odist	btp	otp	diff
1	4.1794	-35.7603	SLN	SLN	6.0178	6.0178	0.0000
2	-5.2638	-122.6474	NSLN	NSLN	13.1015	13.1015	0.0000
3	11.6859	3.4778	SLN	LN	8.4178	8.4253	0.0075
4	-12.6131	-12.6894	NSLN	NSLN	14.8376	14.8376	0.0000
5	-7.5125	-31.9781	NSLN	NSLN	6.0771	6.0771	0.0000
6	6.3886	-30.2169	SLN	SLN	7.2401	7.2401	0.0000

5 Simulation

Run some simulations in particular to check Greek calculations which aren't included in reference papers.

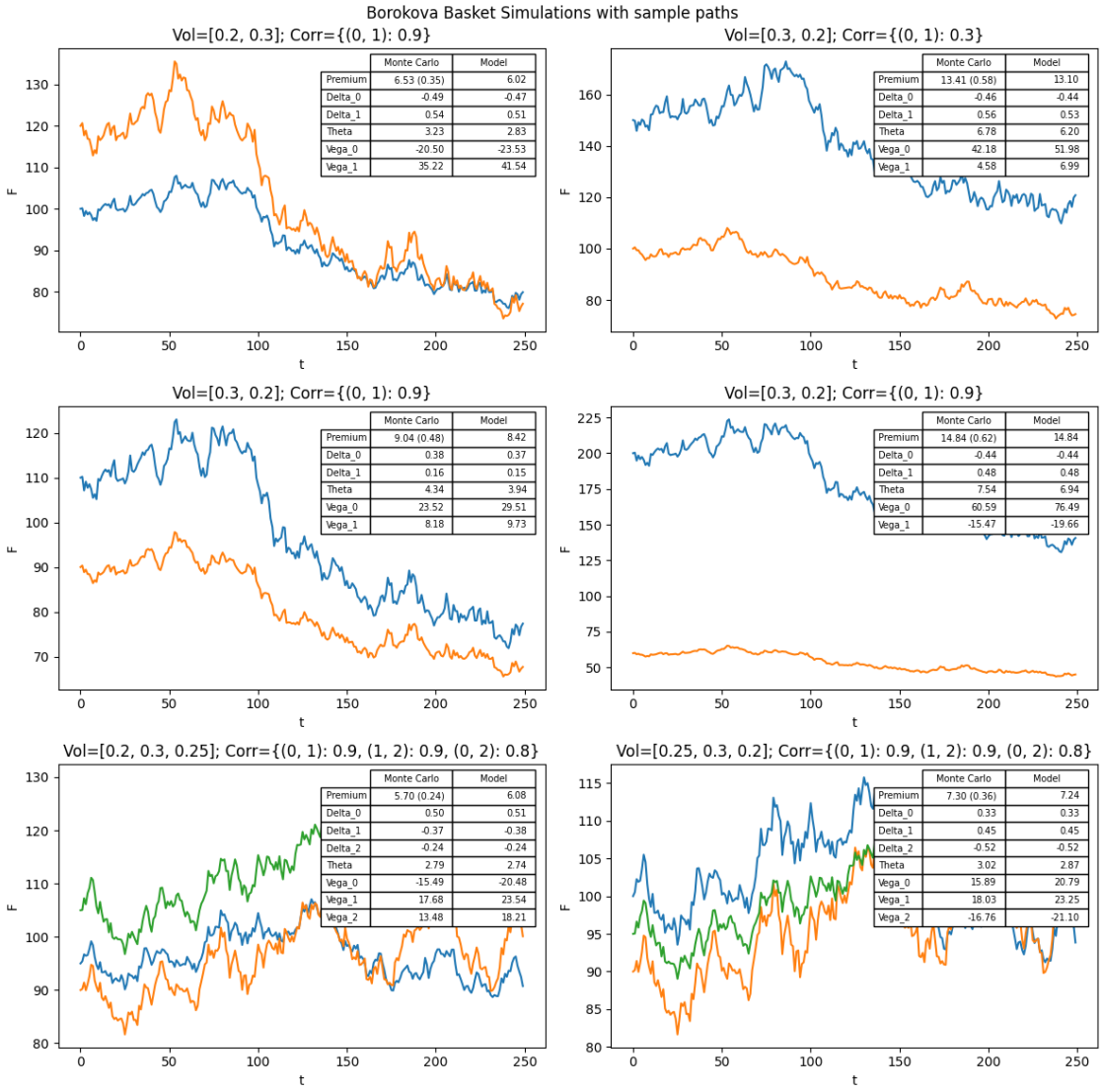
5.0.1 Closed Form Approach to Valuation and Hedging of Basket Options

We compare results against (Borovkova et al., 2007). The results show a close correspondence between the above model and Monte Carlo simulation.



5.0.2 Asian basket options and implied correlations in energy markets

We compare results against (Borovkova & Permana, 2007). The results here use the Python prototype which only adjusts the sigma component. So the premiums match Monte Carlo closely but the greeks don't since they haven't been fully implemented.



6 Appendix

6.1 Sympy Equation Helper

Sympy doesn't support simple balanced equation manipulation. Some routines were added to sympy to support basic manipulation. A `DynamicModule` wraps around the Sympy top level package injecting calls around the Eq operator.

The following is then possible:

```
>>> from ...equation_helper import sp
>>> a, b, c, x, y = sp.symbols("a b c x y", real=True)
>>> eqn = sp.Eq(0, a*x**2 + b*x + c)
>>> eqn
```

$$0 = ax^2 + bx + c$$

```
>>> sp.sqrt((((eqn/a).rexpand() - c/a) + b**2/4/a/a).rfactor()).simplify() * 2
```

$$\left|2x + \frac{b}{a}\right| = \sqrt{\frac{-4ac + b^2}{a^2}}$$

6.2 Sympy simplication

Sympy has many simplification algorithms but as yet doesn't support simplfying sums of sums with permutated bound symbols. The following routine was used to break a sum of sums into simple summations, then attempt to permutate each component summation's bound symbols until operation counts (`count_ops`) was a minimum.

```
def simplify_add_sum(self, pre_simp_func=None, ...):
    ...
    exprs = self.args

    orig_op_count = sp.count_ops(self)

    # find all permutations of bound symbols
    bvarss = [get_bound_symbols(expr) for expr in exprs]
    bvarss_lens = {len(bvars) for bvars in bvarss}
    if len(bvarss_lens) != 1:
        return self
    N = next(iter(bvarss_lens))

    # create temp bind symbols
    bsym = sp.symbols(f"p0:{N}")

    bvarss_perms = product(*[permutations(bvars, N) for bvars in bvarss])

    candidate_exprs = []
    for perm in bvarss_perms:
        # swap bound vars to common symbols
        sdcts = [{bv: bs for bv, bs in zip(bvars, bsym)} for bvars in perm]
        new_exprs = [
            sp.factor_terms(expr.xreplace(sdct)) for expr, sdct in zip(exprs, sdcts)
        ]
        new_expr = sp.factor_terms(sp.Add(*new_exprs))

        if pre_simp_func is not None:
            new_expr = pre_simp_func(new_expr)

        candidate_exprs.append((new_expr, sp.count_ops(new_expr), perm))

    candidate_exprs_ranked = sorted(
```

```
        candidate_exprs, key=lambda r: (r[1], r[0].sort_key())
    )
    best_expr, best_op_count, perm = candidate_exprs_ranked[0]
    if best_op_count < orig_op_count:
        # swap temp bind symbols with origs
        bdct = {
            _old: _new
            for _old, _new in zip(
                get_bound_symbols(best_expr),
                sorted(perm[0], key=lambda x: x.sort_key()),
            )
        }
        return best_expr.xreplace(bdct)
    ...
```

An example of a simplification is as follows:

$$
\begin{aligned}
\frac{\partial M_3}{\partial \sigma_l} = &\sum_{k=1}^{N} T e^{T\rho_{l,k}\sigma_k\sigma_l} F(0)_k F(0)_l a_k a_l \rho_{l,k}\sigma_k \sum_{j=1}^{N} e^{T\rho_{j,k}\sigma_j\sigma_k} e^{T\rho_{l,j}\sigma_j\sigma_l} F(0)_j a_j + \\
&\sum_{k=1}^{N} T e^{T\rho_{l,k}\sigma_k\sigma_l} F(0)_k F(0)_l a_k a_l \rho_{l,k}\sigma_k \sum_{i=1}^{N} e^{T\rho_{i,k}\sigma_i\sigma_k} e^{T\rho_{i,l}\sigma_i\sigma_l} F(0)_i a_i + \\
&\sum_{k=1}^{N} T e^{T\rho_{l,k}\sigma_k\sigma_l} F(0)_k F(0)_l a_k a_l \sum_{j=1}^{N} e^{T\rho_{j,k}\sigma_j\sigma_k} e^{T\rho_{l,j}\sigma_j\sigma_l} F(0)_j a_j \rho_{l,j}\sigma_j + \\
&\sum_{k=1}^{N} T e^{T\rho_{l,k}\sigma_k\sigma_l} F(0)_k F(0)_l a_k a_l \sum_{i=1}^{N} e^{T\rho_{i,k}\sigma_i\sigma_k} e^{T\rho_{i,l}\sigma_i\sigma_l} F(0)_i a_i \rho_{i,l}\sigma_i + \\
&T F(0)_l a_l \sum_{j=1}^{N} e^{T\rho_{j,l}\sigma_j\sigma_l} F(0)_j a_j \rho_{j,l}\sigma_j \sum_{i=1}^{N} e^{T\rho_{i,j}\sigma_i\sigma_j} e^{T\rho_{i,l}\sigma_i\sigma_l} F(0)_i a_i + \\
&T F(0)_l a_l \sum_{j=1}^{N} e^{T\rho_{j,l}\sigma_j\sigma_l} F(0)_j a_j \sum_{i=1}^{N} e^{T\rho_{i,j}\sigma_i\sigma_j} e^{T\rho_{i,l}\sigma_i\sigma_l} F(0)_i a_i \rho_{i,l}\sigma_i \\
= &\, 6 T F(0)_l a_l \sum_{\substack{1\le j\le N \\ 1\le i\le N}} e^{T\rho_{j,i}\sigma_i\sigma_j} e^{T\rho_{l,i}\sigma_i\sigma_l} e^{T\rho_{l,j}\sigma_j\sigma_l} F(0)_i F(0)_j a_i a_j \rho_{l,j}\sigma_j
\end{aligned}
$$

6.3 Sympy C++ Printer

Sympy has basic C++ printer support but is unable to handle summation out of the box. Furthermore, Sympy printer support takes a top down approach to converting an expression to C++. I wrote a custom bottom-up printer based on an earlier project for cloning Sympy expressions (Azzopardi, 2020) without evaluation using Python AST. This approach is also able to support C++ package Eigen for matrix and vector manipulation.

The sympy expression is traverse and a list of nodes generated along the lines of:

```
@dataclass
class Node:
    expr: sp.Basic
    pos: tuple[int, ...]
    parent_node: Node
    child_nodes: list[None]
    cpp: str | None
    cpp_type: str | None

def traverse_expression(e, depth, nodes, pos=(), parent_node=None):
    n = Node(e, pos, parent_node)
    if parent_node:
        parent_node.child_nodes.append(n)
    nodes.append(n)
    for i, arg in enumerate(e.args):
        if isinstance(arg, sp.Basic):
            traverse_expression(arg, depth + 1, nodes, pos + (i,), n)
```

One the nodes were captured in a list, all leaf nodes are processed generating C++ text for basic variables and literals. Then several iterations loop back over the node list generating the C++ text for each node if all child nodes have been previously generated. Special care is taken for converting sympy n-tuple Sums to C++ nested loops and for operations on Indexed expressions to be converted into equivalent Eigen calls.

7 References

References

[1] Agarwal, S., Mierle, K., & The Ceres Solver Team. (2023). *Ceres Solver* (2.2) [Computer software]. https://github.com/ceres-solver/ceres-solver

[2] Azzopardi, B. (2020). *Sympy Clone Expression*. https://github.com/bsdz/sympy_clone_expression

[3] Azzopardi, B. (2024). *Basket Option Pricer*. https://github.com/bsdz/basket_option_pricer

[4] Borovkova, S., & Permana, F. J. (2007). *Asian basket options and implied correlations in energy markets*.

[5] Borovkova, S., Permana, F. J., & Weide, H. V. D. (2007). A Closed Form Approach to the Valuation and Hedging of Basket and Spread Option. *The Journal of Derivatives*, *14*(4), 8–24. https://doi.org/10.3905/jod.2007.686420

[6] Guennebaud, G., Jacob, B., & others. (2010). *Eigen v3*. http://eigen.tuxfamily.org

[7] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[8] Hull, J. (2012). *Options, futures, and other derivatives* (8th ed). Prentice Hall.

[9] Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, Am., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., ... Scopatz, A. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, *3*, e103. https://doi.org/10.7717/peerj-cs.103

[10] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2