7.3 Greeks

Definition 91 (Delta): Partial Derivative w.r.t. the value of the underlying asset. [Black-Scholes: $\frac{\partial V}{\partial S} = \Phi(d_+) =: \Delta$]

Definition 92 (Gamma): Second derivative w.r.t. the underlying. [Black-Scholes: $\frac{\partial^2 V}{\partial S^2} = \frac{\Phi'(d_+)}{S_0 \sigma \sqrt{T}} =: \Gamma$]

Definition 93 (Vega): The partial derivative to "an option price" [Black-Scholes: $\frac{\partial V}{\partial \sigma} = S_0 \sqrt{T}\, \Phi'(d_+)$]

$\rightsquigarrow$ see Script for Theta $\left(\frac{\partial V}{\partial T}\right)$, Rho $\left(\frac{\partial V}{\partial r}\right)$.

7.4. Hedging in Discrete Time: Delta and Delta-Gamma Hedging.

Recall the situation in continuous time

$$dV(t) = \frac{\partial V(t)}{\partial t} dt + \sum_{i=0}^{n} \frac{\partial V}{\partial S_i} \cdot dS_i + \frac{1}{2} \sum_{i,j=0}^{n} \frac{\partial^2 V(t)}{\partial S_i \partial S_j} dS_i dS_j$$

$$= \sum_{i=0}^{n} \frac{\partial V}{\partial S_i} dS_i + \underbrace{\left(\frac{\partial V(t)}{\partial t} dt + \frac{1}{2} \sum_{i,j=0}^{n} \frac{\partial^2 V}{\partial S_i \partial S_j} \overbrace{dS_i dS_j}^{\gamma_{i,j}\, dt} \right)}_{=0}$$

In a time discrete setup we have for $\Delta V(t) = V(t+\Delta t) - V(t)$

$$\Delta V(t) = \frac{\partial V(t)}{\partial t} \Delta t + \sum_{i=0}^{n} \frac{\partial V}{\partial S_i} \Delta S_i + \frac{1}{2} \sum_{i,j=0}^{n} \frac{\partial^2 V}{\partial S_i \partial S_j} \Delta S_i \Delta S_j + \text{h.o.t.}$$

$$= \underbrace{\boxed{\sum_{i=0}^{n} \frac{\partial V}{\partial S_i} \cdot \Delta S_i}}_{\Delta \Pi} + \frac{1}{2} \sum_{i,j=0}^{n} \frac{\partial^2 V}{\partial S_i \partial S_j} \Big(\underbrace{\Delta S_i \Delta S_j - \gamma_{ij} \Delta t}_{\neq 0} \Big) \underbrace{\text{+h.o.t.}}_{\approx}$$

Compare this to the replication portfolio:

$$\Delta \Pi(t) = \sum_{i=0}^{n} \phi_i(t)\, \Delta S_i(t)$$

To ensure that the replication portfolio remains self-financing we have to reformulate the cond. in discrete time:

$$\sum_{i=0}^{n} \Big(\phi_i(t_k) - \phi_i(t_{k-1}) \Big) S_i(t_k) = 0$$

For example, we may ensure ~~the~~ this via

$$(*) \quad \phi_0(t_k) = \phi_0(t_{k-1}) - \frac{1}{S_0(t_k)} \sum_{i=1}^{n} \Big(\phi_i(t_k) - \phi_i(t_{k-1}) \Big) S_i(t_k)$$

7.4.1 Delta Hedging:

We choose $\phi_1, \ldots, \phi_n$ according to the delta hedge, i.e.

$$\phi_i = \frac{\partial V}{\partial S_i} \quad \text{for } i = 1, \ldots, n$$

and then choose ϕ_0 according to $(*)$.

7.4.2.1 Example: Time-Discrete Delta Hedge under a Black-Scholes Model

Given a time discretization $0 = t_0 < t_1 < \ldots$:

For the replication portfolio $\phi_0(t) N(t) + \phi_1(t) \cdot S(t)$

we have

$$\phi_1(t_k) = \frac{\partial V(t_k)}{\partial S(t_k)} = \Phi\left(\frac{1}{\sigma\sqrt{T-t_k}} \left[\log \frac{S(t_k)}{K} + r(T-t_k) + \frac{\sigma^2}{2}(T-t_k) \right] \right)$$

$(k = 0, \ldots)$

$$\phi_0(t_k) = \frac{1}{N(t_k)} \left[\phi_0(t_{k-1}) \cdot N(t_k) + \phi_1(t_{k-1}) S(t_k) - \phi_1(t_k) S(t_k) \right]$$

$(k = 1, \ldots)$

At time $t_0 = 0$ the portfolio is set up according to the option value

$$\phi_0(t_0) = \frac{1}{N(t_0)} \left(V(t_0) - \phi_1(t_0) S(t_0) \right)$$

Java - finmath lib/src/main/java6/net/finmath/montecarlo/assetderivativevaluation/AssetModelMonteCarloSimulationInterface.java - Eclipse - /Users/fries/Documents/Development

AssetModelMonteCarloSimulationInterface.java | EuropeanOption.java | BlackScholesDeltaHedgedPortfolio.java

```java
public interface AssetModelMonteCarloSimulationInterface extends MonteCarloSimulationInterface {

    /**
     * Returns the number of asset price processes.
     *
     * @return The number of asset price processes
     */
    int                 getNumberOfAssets();

    /**
     * Returns the random variable representing the asset's value at a given time for a given asset.
     *
     * @param timeIndex     Index of simulation time
     * @param assetIndex    Index of the asset (0 for a single asset model)
     * @return              The asset process as seen on simulation time
     * @throws net.finmath.exception.CalculationException Thrown if the valuation fails, specific cause may be available via the <code>cause()
     */
    RandomVariableInterface     getAssetValue(int timeIndex, int assetIndex) throws CalculationException;

    /**
     * Returns the random variable representing the asset's value at a given time for a given asset.
     *
     * @param time          Simulation time
     * @param assetIndex    Index of the asset (0 for a single asset model)
     * @return              The asset process as seen on simulation time
     * @throws net.finmath.exception.CalculationException Thrown if the valuation fails, specific cause may be available via the <code>cause()
     */
    RandomVariableInterface     getAssetValue(double time, int assetIndex) throws CalculationException;

    /**
     * Returns the numeraire associated with the valuation measure used by this model.
     *
     * @param timeIndex The time index (associated with this models time discretization).
     * @return The numeraire associated with the valuation measure used by this model.
     * @throws CalculationException Thrown if calculation of numeraire fails.
     */
    RandomVariableInterface     getNumeraire(int timeIndex) throws CalculationException;

    /**
     * Returns the numeraire associated with the valuation measure used by this model.
     *
     * @param time The time for which the numeraire is requested.
     * @return The numeraire associated with the valuation measure used by this model.
     * @throws CalculationException Thrown if calculation of numeraire fails.
```

Absraction of the model and the way it is numerically simulated.

$\omega_j \mapsto S_i(t_k, \omega_j)$

$N(t_k)$

Writable | Smart Insert | 16 : 4 | 280M of 716M

Java - finmath lib/src/main/java6/net/finmath/montecarlo/assetderivativevaluation/products/EuropeanOption.java - Eclipse - /Users/fries/Documents/Development

AssetModelMonteCarloSimulationInterface.java | EuropeanOption.java | BlackScholesDeltaHedgedPortfolio.java

```java
public class EuropeanOption extends AbstractAssetMonteCarloProduct {

    private final double maturity;          <- T
    private final double strike;            <- K
    private final Integer underlyingIndex;
    private final String nameOfUnderliyng;

    /**
     * Construct a product representing an European option on an asset S (where S the asset with index 0 from the model - single asset case).
     * @param maturity The maturity T in the option payoff max(S(T)-K,0)
     * @param strike The strike K in the option payoff max(S(T)-K,0).
     * @param underlyingIndex The index of the underlying to be fetched from the model.
     */
    public EuropeanOption(double maturity, double strike, int underlyingIndex) {
        super();
        this.maturity           = maturity;
        this.strike             = strike;
        this.underlyingIndex    = underlyingIndex;
        this.nameOfUnderliyng   = null;     // Use underlyingIndex
    }

    /**
     * Construct a product representing an European option on an asset S (where S the asset with index 0 from the model - single asset case).
     * @param maturity The maturity T in the option payoff max(S(T)-K,0)
     * @param strike The strike K in the option payoff max(S(T)-K,0).
     */
    public EuropeanOption(double maturity, double strike) {
        this(maturity, strike, 0);
    }

    /**
     * This method returns the value random variable of the product within the specified model, evaluated at a given evalutationTime.
     * Note: For a lattice this is often the value conditional to evalutationTime, for a Monte-Carlo simulation this is the (sum of) value dis
     * Cashflows prior evaluationTime are not considered.
     *
     * @param evaluationTime The time on which this products value should be observed.
     * @param model The model used to price the product.
     * @return The random variable representing the value of the product discounted to evaluation time
     * @throws net.finmath.exception.CalculationException Thrown if the valuation fails, specific cause may be available via the <code>cause()
     */
    @Override
    public RandomVariableInterface getValue(double evaluationTime, AssetModelMonteCarloSimulationInterface model) throws CalculationException
        // Get underlying and numeraire
```

Writable | Smart Insert | 27 : 16 | 309M of 716M

Java - finmath lib/src/main/java6/net/finmath/montecarlo/assetderivativevaluation/products/EuropeanOption.java - Eclipse - /Users/fries/Documents/Development

AssetModelMonteCarloSimulationInterface.java | EuropeanOption.java | BlackScholesDeltaHedgedPortfolio.java

```java
    /**
     * Construct a product representing an European option on an asset S (where S the asset with index 0 from the model - single asset case).
     * @param maturity The maturity T in the option payoff max(S(T)-K,0)
     * @param strike The strike K in the option payoff max(S(T)-K,0).
     */
    public EuropeanOption(double maturity, double strike) {
        this(maturity, strike, 0);
    }

    /**
     * This method returns the value random variable of the product within the specified model, evaluated at a given evalutationTime.
     * Note: For a lattice this is often the value conditional to evalutationTime, for a Monte-Carlo simulation this is the (sum of) value dis
     * Cashflows prior evaluationTime are not considered.
     *
     * @param evaluationTime The time on which this products value should be observed.
     * @param model The model used to price the product.
     * @return The random variable representing the value of the product discounted to evaluation time
     * @throws net.finmath.exception.CalculationException Thrown if the valuation fails, specific cause may be available via the <code>cause()
     */
    @Override
    public RandomVariableInterface getValue(double evaluationTime, AssetModelMonteCarloSimulationInterface model) throws CalculationException
        // Get underlying and numeraire
                                                                                       T
        // Get S(T)
        RandomVariableInterface underlyingAtMaturity    = model.getAssetValue(maturity, underlyingIndex);    <- S(T)

        // The payoff: values = max(underlying - strike, 0) = V(T) = max(S(T)-K,0)
        RandomVariableInterface values = underlyingAtMaturity.sub(strike).floor(0.0);    <- V(T) = max(S(T)-K,0)
                                  V(T)                            K
        // Discounting...
        RandomVariableInterface numeraireAtMaturity     = model.getNumeraire(maturity);
        RandomVariableInterface monteCarloWeights       = model.getMonteCarloWeights(maturity);
        values = values.div(numeraireAtMaturity).mult(monteCarloWeights);    <- V(T)/N(T)

        // ...to evaluation time.
        RandomVariableInterface numeraireAtEvalTime                 = model.getNumeraire(evaluationTime);
        RandomVariableInterface monteCarloProbabilitiesAtEvalTime   = model.getMonteCarloWeights(evaluationTime);
        values = values.mult(numeraireAtEvalTime).div(monteCarloProbabilitiesAtEvalTime);

        return values;    <- V(T)/N(T) · N(t)
    }
}
```

Writable | Smart Insert | 27 : 16 | 330M of 716M

Java - finmath lib/src/main/java6/net/finmath/montecarlo/assetderivativevaluation/products/BlackScholesDeltaHedgedPortfolio.java - Eclipse - /Users/fries/Documents/Development

AssetModelMonteCarloSimulationInterface.java | EuropeanOption.java | BlackScholesDeltaHedgedPortfolio.java

```java
 */
public class BlackScholesDeltaHedgedPortfolio extends AbstractAssetMonteCarloProduct {

    // Properties of the European option we wish to replicate
    private final double maturity;
    private final double strike;

    // Model assumptions for the hedge
    private final double riskFreeRate;    // Actually the same as the drift (which is not stochastic)
    private final double volatility;

    /**
     * Construction of a delta hedge portfolio assuming a Black-Scholes model.
     *
     * @param maturity        Maturity of the option we wish to replicate.
     * @param strike          Strike of the option we wish to replicate.
     * @param riskFreeRate    Model riskFreeRate assumption for our delta hedge.
     * @param volatility      Model volatility assumption for our delta hedge.
     */
    public BlackScholesDeltaHedgedPortfolio(double maturity, double strike, double riskFreeRate, double volatility) {
        super();
        this.maturity = maturity;
        this.strike = strike;
        this.riskFreeRate = riskFreeRate;
        this.volatility = volatility;
    }

    @Override
    public RandomVariableInterface getValue(double evaluationTime, AssetModelMonteCarloSimulationInterface model) throws CalculationException

        // Ask the model for its discretization
        int timeIndexEvaluationTime = model.getTimeIndex(evaluationTime);

        /*
         *  Going forward in time we monitor the hedge portfolio on each path.
         */

        // Initialize the portfolio to zero stocks and as much cash as the Black-Scholes Model predicts we need.
        RandomVariableInterface underlyingToday = model.getAssetValue(0.0,0);
        RandomVariableInterface numeraireToday  = model.getNumeraire(0.0);

        RandomVariableInterface valueOfOptionAccordingBlackScholes =    AnalyticFormulas.blackScholesGeneralizedOptionValue(
                underlyingToday.mult(Math.exp(riskFreeRate * (maturity - 0.0))),
                model.getRandomVariableForConstant(volatility),
```

T, K } the spec. of the product we like to hedge. r, σ } the model assumptions used to calculate the hedge (i.e. ϕ_1 ($\rightsquigarrow \phi_0$)). Does not need to correspond to the model which finally generate N, S

Writable | Smart Insert | 13 : 4 | 371M of 716M

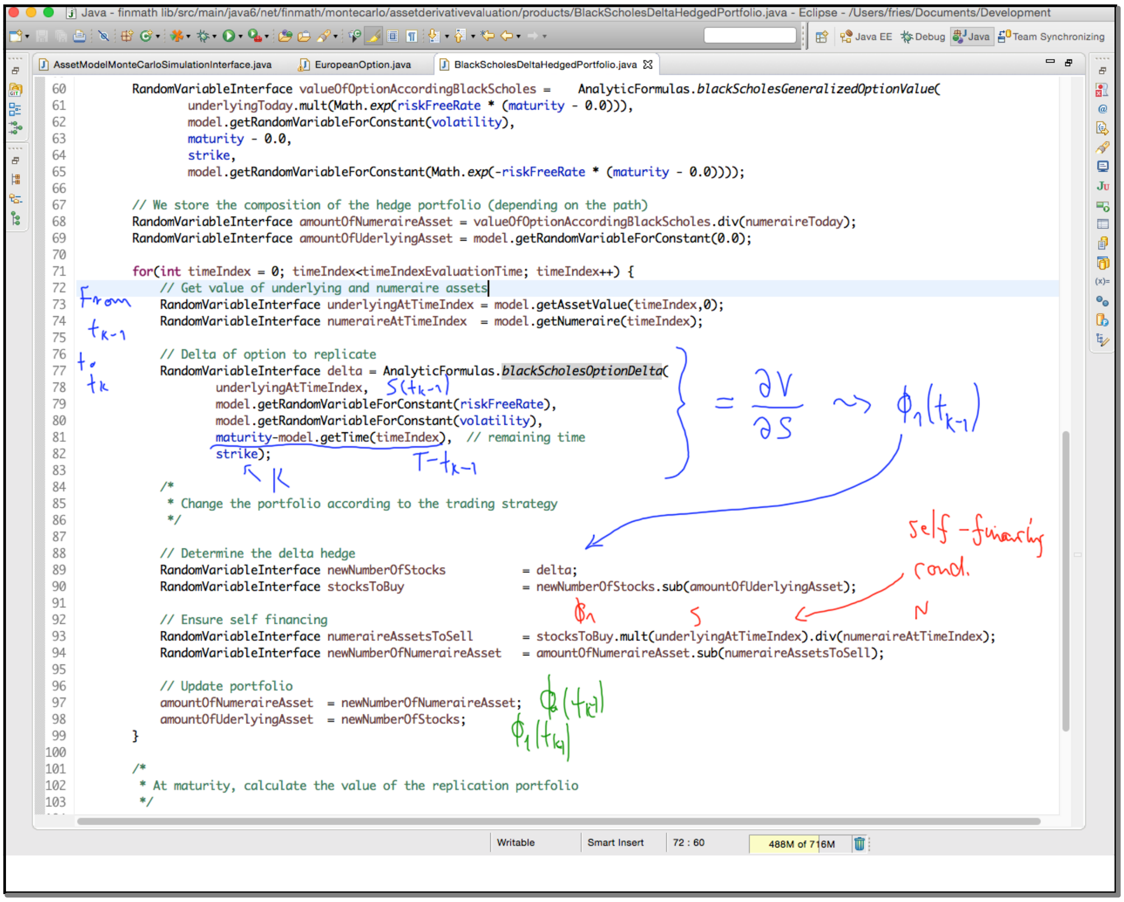
Java - finmath lib/src/main/java6/net/finmath/montecarlo/assetderivativevaluation/products/BlackScholesDeltaHedgedPortfolio.java - Eclipse - /Users/fries/Documents/Development

AssetModelMonteCarloSimulationInterface.java | EuropeanOption.java | BlackScholesDeltaHedgedPortfolio.java

```java
    public RandomVariableInterface getValue(double evaluationTime, AssetModelMonteCarloSimulationInterface model) throws CalculationException

        // Ask the model for its discretization
        int timeIndexEvaluationTime = model.getTimeIndex(evaluationTime);

        /*
         *  Going forward in time we monitor the hedge portfolio on each path.
         */

        // Initialize the portfolio to zero stocks and as much cash as the Black-Scholes Model predicts we need.
        RandomVariableInterface underlyingToday = model.getAssetValue(0.0,0);
        RandomVariableInterface numeraireToday  = model.getNumeraire(0.0);

        RandomVariableInterface valueOfOptionAccordingBlackScholes =    AnalyticFormulas.blackScholesGeneralizedOptionValue(
                underlyingToday.mult(Math.exp(riskFreeRate * (maturity - 0.0))),
                model.getRandomVariableForConstant(volatility),
                maturity - 0.0,
                strike,
                model.getRandomVariableForConstant(Math.exp(-riskFreeRate * (maturity - 0.0))));

        // We store the composition of the hedge portfolio (depending on the path)
        RandomVariableInterface amountOfNumeraireAsset = valueOfOptionAccordingBlackScholes.div(numeraireToday);
        RandomVariableInterface amountOfUderlyingAsset = model.getRandomVariableForConstant(0.0);

        for(int timeIndex = 0; timeIndex<timeIndexEvaluationTime; timeIndex++) {
            // Get value of underlying and numeraire assets
            RandomVariableInterface underlyingAtTimeIndex = model.getAssetValue(timeIndex,0);
            RandomVariableInterface numeraireAtTimeIndex  = model.getNumeraire(timeIndex);

            // Delta of option to replicate
            RandomVariableInterface delta = AnalyticFormulas.blackScholesOptionDelta(
                    underlyingAtTimeIndex,
                    model.getRandomVariableForConstant(riskFreeRate),
                    model.getRandomVariableForConstant(volatility),
                    maturity-model.getTime(timeIndex),  // remaining time
                    strike);

            /*
             * Change the portfolio according to the trading strategy
             */

            // Determine the delta hedge
            RandomVariableInterface newNumberOfStocks         = delta;
            RandomVariableInterface stocksToBuy               = newNumberOfStocks.sub(amountOfUderlyingAsset);
```

Set up $\Pi(0)$. Initial value of V (and Π) according to B.S. $\phi_0(0) = V(0)/N(0)$. $\phi_1(0) = 0$.

Writable | Smart Insert | 13 : 4 | 394M of 716M



```java
        RandomVariableInterface valueOfOptionAccordingBlackScholes =    AnalyticFormulas.blackScholesGeneralizedOptionValue(
                underlyingToday.mult(Math.exp(riskFreeRate * (maturity - 0.0))),
                model.getRandomVariableForConstant(volatility),
                maturity - 0.0,
                strike,
                model.getRandomVariableForConstant(Math.exp(-riskFreeRate * (maturity - 0.0))));

        // We store the composition of the hedge portfolio (depending on the path)
        RandomVariableInterface amountOfNumeraireAsset = valueOfOptionAccordingBlackScholes.div(numeraireToday);
        RandomVariableInterface amountOfUderlyingAsset = model.getRandomVariableForConstant(0.0);

        for(int timeIndex = 0; timeIndex<timeIndexEvaluationTime; timeIndex++) {
            // Get value of underlying and numeraire assets
            RandomVariableInterface underlyingAtTimeIndex = model.getAssetValue(timeIndex,0);
            RandomVariableInterface numeraireAtTimeIndex  = model.getNumeraire(timeIndex);

            // Delta of option to replicate
            RandomVariableInterface delta = AnalyticFormulas.blackScholesOptionDelta(
                    underlyingAtTimeIndex,
                    model.getRandomVariableForConstant(riskFreeRate),
                    model.getRandomVariableForConstant(volatility),
                    maturity-model.getTime(timeIndex),  // remaining time
                    strike);

            /*
             * Change the portfolio according to the trading strategy
             */

            // Determine the delta hedge
            RandomVariableInterface newNumberOfStocks          = delta;
            RandomVariableInterface stocksToBuy                = newNumberOfStocks.sub(amountOfUderlyingAsset);

            // Ensure self financing
            RandomVariableInterface numeraireAssetsToSell      = stocksToBuy.mult(underlyingAtTimeIndex).div(numeraireAtTimeIndex);
            RandomVariableInterface newNumberOfNumeraireAsset  = amountOfNumeraireAsset.sub(numeraireAssetsToSell);

            // Update portfolio
            amountOfNumeraireAsset  = newNumberOfNumeraireAsset;
            amountOfUderlyingAsset  = newNumberOfStocks;
        }

        /*
         * At maturity, calculate the value of the replication portfolio
         */
```

From t_{k-1} to t_k

$S(t_{k-1})$

$= \frac{\partial V}{\partial S} \rightsquigarrow \phi_1(t_{k-1})$

$T - t_{k-1}$

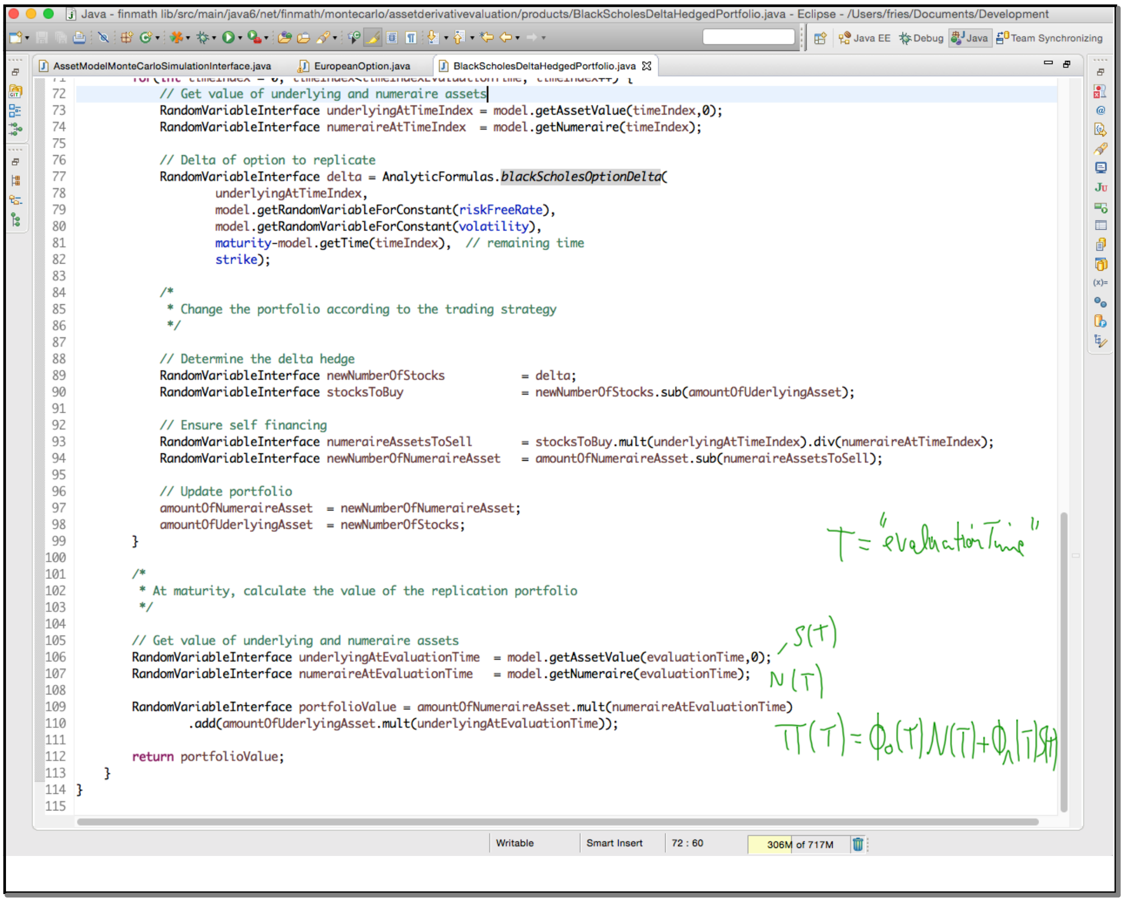
K

self-financing cond.

ϕ_1 S N

$\phi_0(t_k)$

$\phi_1(t_k)$



```java
            // Get value of underlying and numeraire assets
            RandomVariableInterface underlyingAtTimeIndex = model.getAssetValue(timeIndex,0);
            RandomVariableInterface numeraireAtTimeIndex  = model.getNumeraire(timeIndex);

            // Delta of option to replicate
            RandomVariableInterface delta = AnalyticFormulas.blackScholesOptionDelta(
                    underlyingAtTimeIndex,
                    model.getRandomVariableForConstant(riskFreeRate),
                    model.getRandomVariableForConstant(volatility),
                    maturity-model.getTime(timeIndex),  // remaining time
                    strike);

            /*
             * Change the portfolio according to the trading strategy
             */

            // Determine the delta hedge
            RandomVariableInterface newNumberOfStocks          = delta;
            RandomVariableInterface stocksToBuy                = newNumberOfStocks.sub(amountOfUderlyingAsset);

            // Ensure self financing
            RandomVariableInterface numeraireAssetsToSell      = stocksToBuy.mult(underlyingAtTimeIndex).div(numeraireAtTimeIndex);
            RandomVariableInterface newNumberOfNumeraireAsset  = amountOfNumeraireAsset.sub(numeraireAssetsToSell);

            // Update portfolio
            amountOfNumeraireAsset  = newNumberOfNumeraireAsset;
            amountOfUderlyingAsset  = newNumberOfStocks;
        }

        /*
         * At maturity, calculate the value of the replication portfolio
         */

        // Get value of underlying and numeraire assets
        RandomVariableInterface underlyingAtEvaluationTime  = model.getAssetValue(evaluationTime,0);
        RandomVariableInterface numeraireAtEvaluationTime   = model.getNumeraire(evaluationTime);

        RandomVariableInterface portfolioValue = amountOfNumeraireAsset.mult(numeraireAtEvaluationTime)
                .add(amountOfUderlyingAsset.mult(underlyingAtEvaluationTime));

        return portfolioValue;
    }
}
```

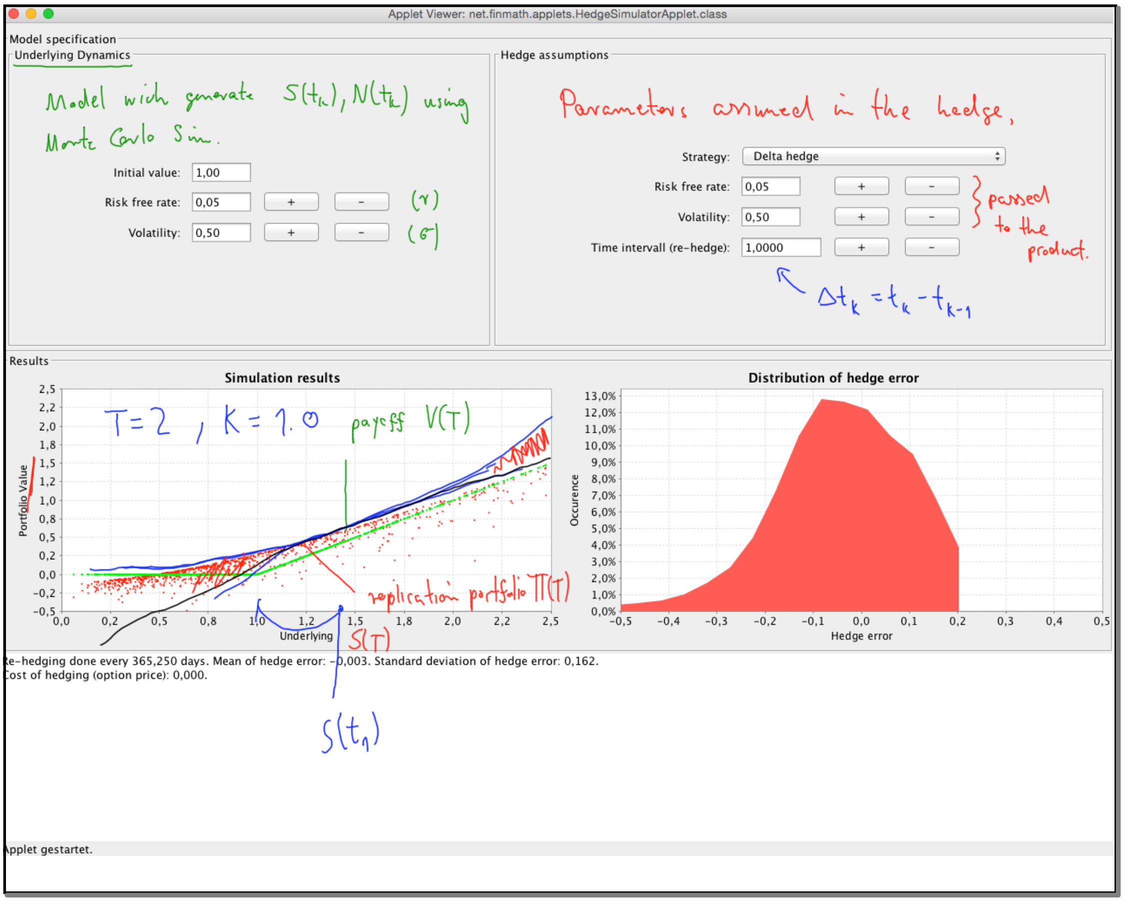
$T =$ "evaluationTime"

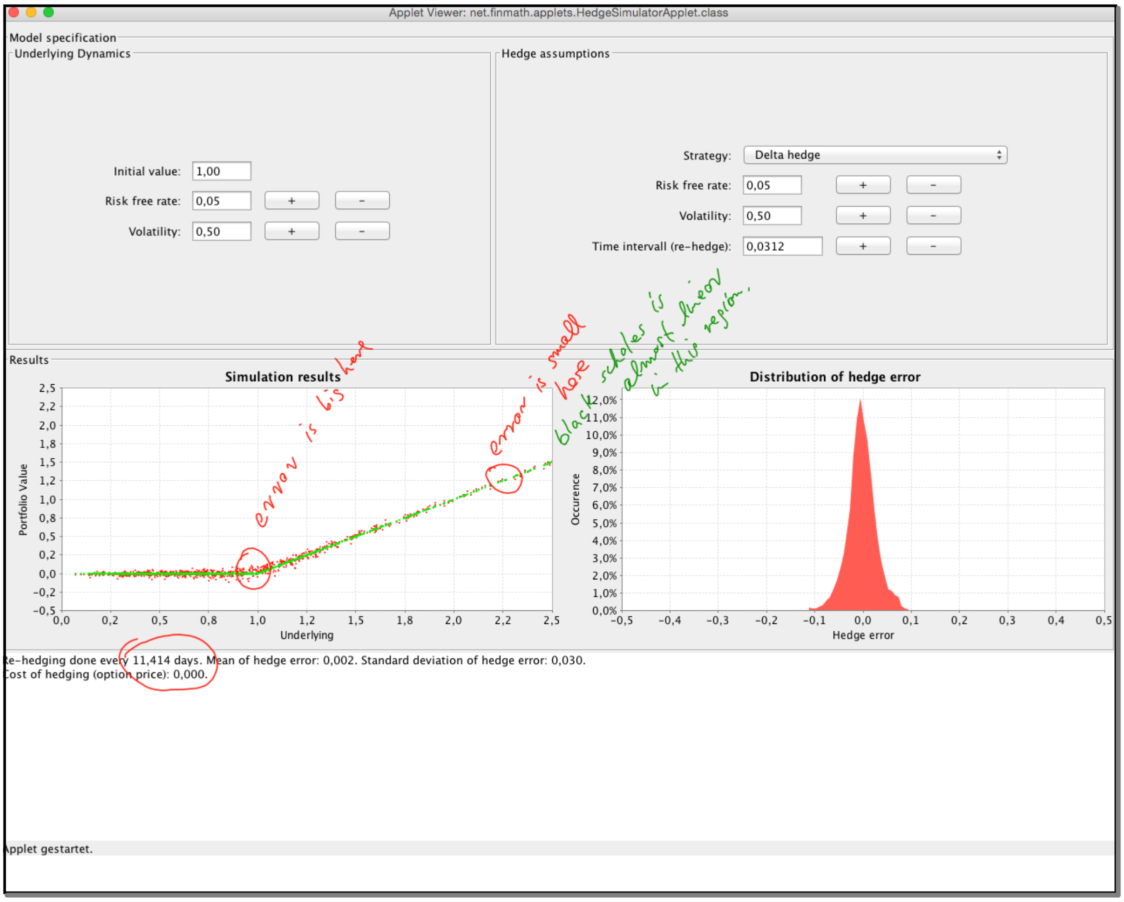
$S(T)$

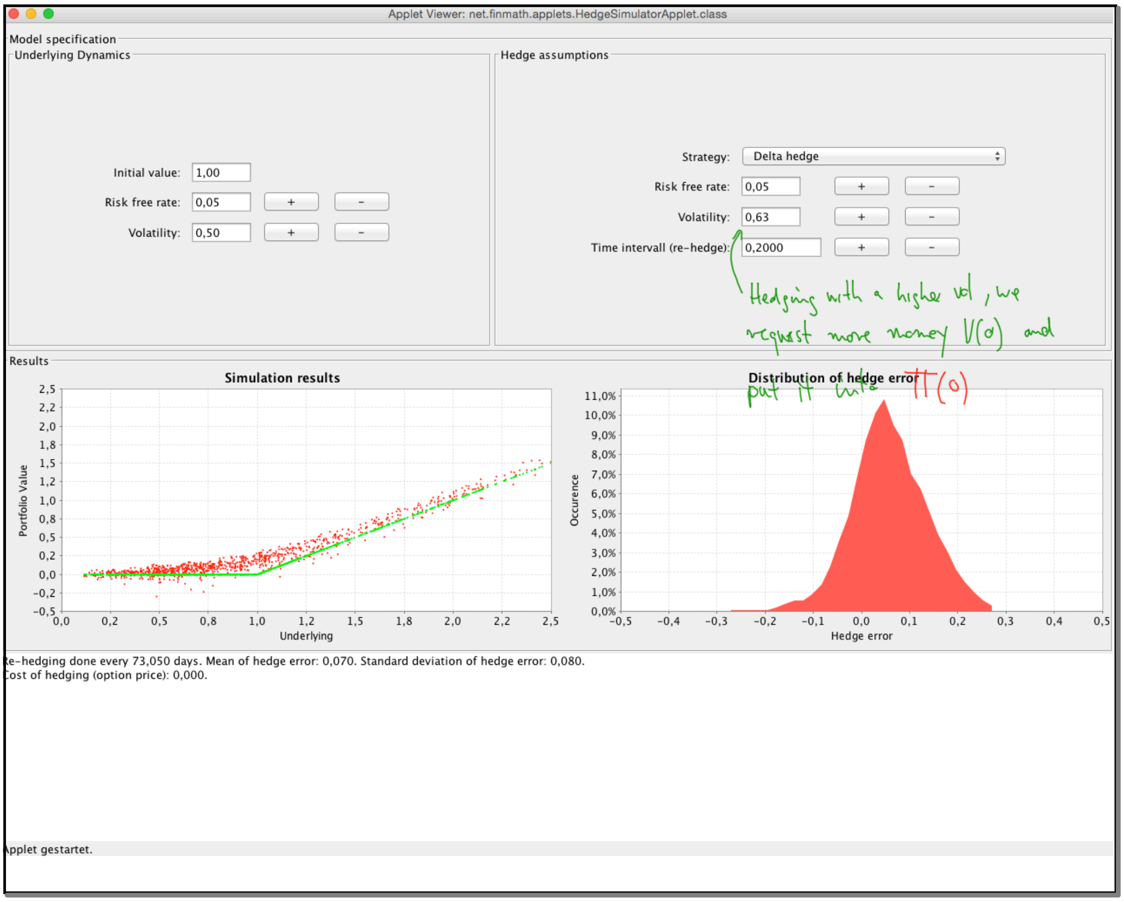
$N(T)$

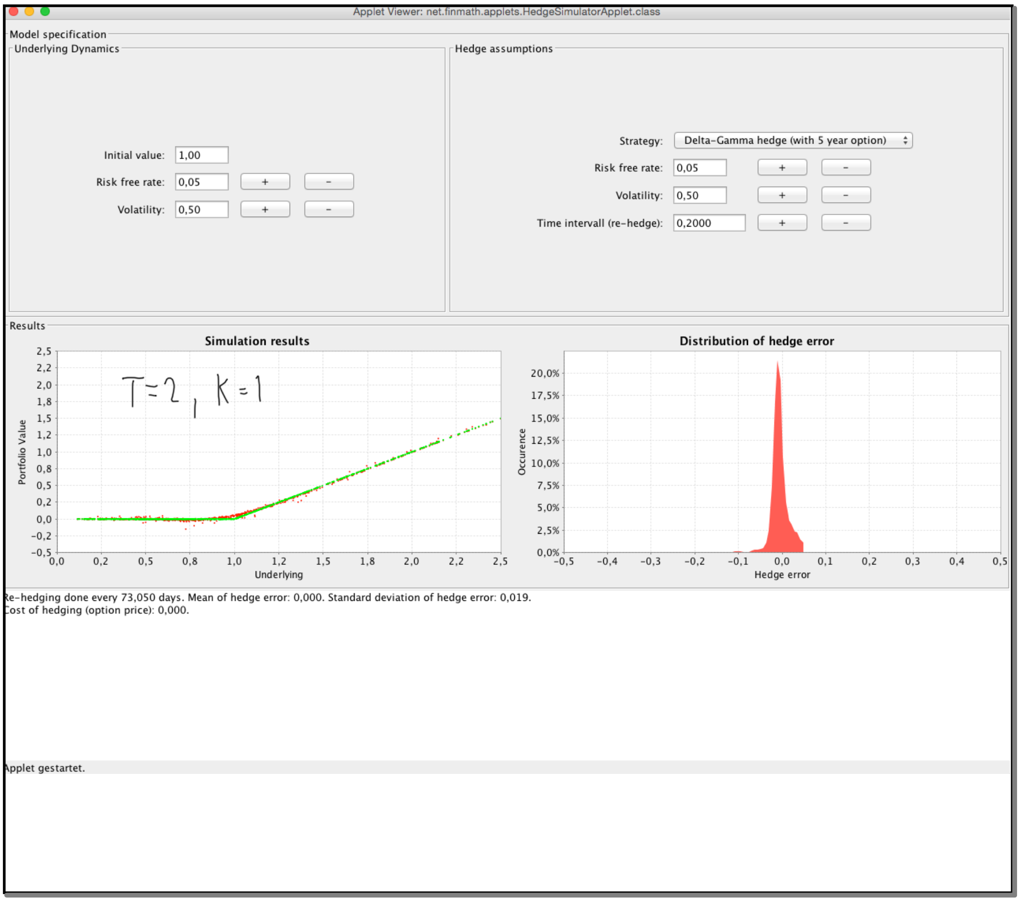
$\Pi(T) = \phi_0(T) N(T) + \phi_1(T) S(T)$


Applet Viewer: net.finmath.applets.HedgeSimulatorApplet.class
Model specification
Underlying Dynamics
Model will generate S(t_k), N(t_k) using Monte Carlo Sim.
Initial value: 1,00
Risk free rate: 0,05 (r)
Volatility: 0,50 (σ)
Hedge assumptions
Parameters assumed in the hedge,
Strategy: Delta hedge
Risk free rate: 0,05
Volatility: 0,50
passed to the product.
Time intervall (re-hedge): 2,0000
Δt_k = t_k - t_{k-1}
Results
Simulation results
T=2, K=1.0 payoff V(T)
replication portfolio Π(T)
Portfolio Value
Underlying
S(T)
Distribution of hedge error
Occurence
Hedge error
Re-hedging done every 730,500 days. Mean of hedge error: -0,004. Standard deviation of hedge error: 0,248.
Cost of hedging (option price): 0,000.
Applet gestartet.


Applet Viewer: net.finmath.applets.HedgeSimulatorApplet.class
Model specification
Underlying Dynamics
Model will generate S(t_k), N(t_k) using Monte Carlo Sim.
Initial value: 1,00
Risk free rate: 0,05 (r)
Volatility: 0,50 (σ)
Hedge assumptions
Parameters assumed in the hedge,
Strategy: Delta hedge
Risk free rate: 0,05
Volatility: 0,50
passed to the product.
Time intervall (re-hedge): 1,0000
Δt_k = t_k - t_{k-1}
Results
Simulation results
T=2, K=1.0 payoff V(T)
replication portfolio Π(T)
Portfolio Value
Underlying
S(T)
S(t_1)
Distribution of hedge error
Occurence
Hedge error
Re-hedging done every 365,250 days. Mean of hedge error: -0,003. Standard deviation of hedge error: 0,162.
Cost of hedging (option price): 0,000.
Applet gestartet.


Applet Viewer: net.finmath.applets.HedgeSimulatorApplet.class
Model specification
Underlying Dynamics
Initial value: 1,00
Risk free rate: 0,05
Volatility: 0,50
Hedge assumptions
Strategy: Delta hedge
Risk free rate: 0,05
Volatility: 0,50
Time intervall (re-hedge): 0,0312
Results
Simulation results
Portfolio Value
Underlying
Distribution of hedge error
Occurence
Hedge error
error is big here
error is small here
black scholes is almost linear in this region.
Re-hedging done every 11,414 days. Mean of hedge error: 0,002. Standard deviation of hedge error: 0,030.
Cost of hedging (option price): 0,000.
Applet gestartet.


Applet Viewer: net.finmath.applets.HedgeSimulatorApplet.class
Model specification
Underlying Dynamics
Initial value: 1,00
Risk free rate: 0,05
Volatility: 0,50
Hedge assumptions
Strategy: Delta hedge
Risk free rate: 0,05
Volatility: 0,63
Time intervall (re-hedge): 0,2000
Hedging with a higher vol, we request more money V(0) and put it into Π(0)
Results
Simulation results
Portfolio Value
Underlying
Distribution of hedge error
Occurence
Hedge error
Re-hedging done every 73,050 days. Mean of hedge error: 0,070. Standard deviation of hedge error: 0,080.
Cost of hedging (option price): 0,000.
Applet gestartet.


Applet Viewer: net.finmath.applets.HedgeSimulatorApplet.class
Model specification
Underlying Dynamics
Initial value: 1,00
Risk free rate: 0,05
Volatility: 0,50
Hedge assumptions
Strategy: Delta-Gamma hedge (with 5 year option)
Risk free rate: 0,05
Volatility: 0,50
Time intervall (re-hedge): 0,2000
Results
Simulation results
T=2, K=1
Portfolio Value
Underlying
Distribution of hedge error
Occurence
Hedge error
Re-hedging done every 73,050 days. Mean of hedge error: 0,000. Standard deviation of hedge error: 0,019.
Cost of hedging (option price): 0,000.
Applet gestartet.