Class Design

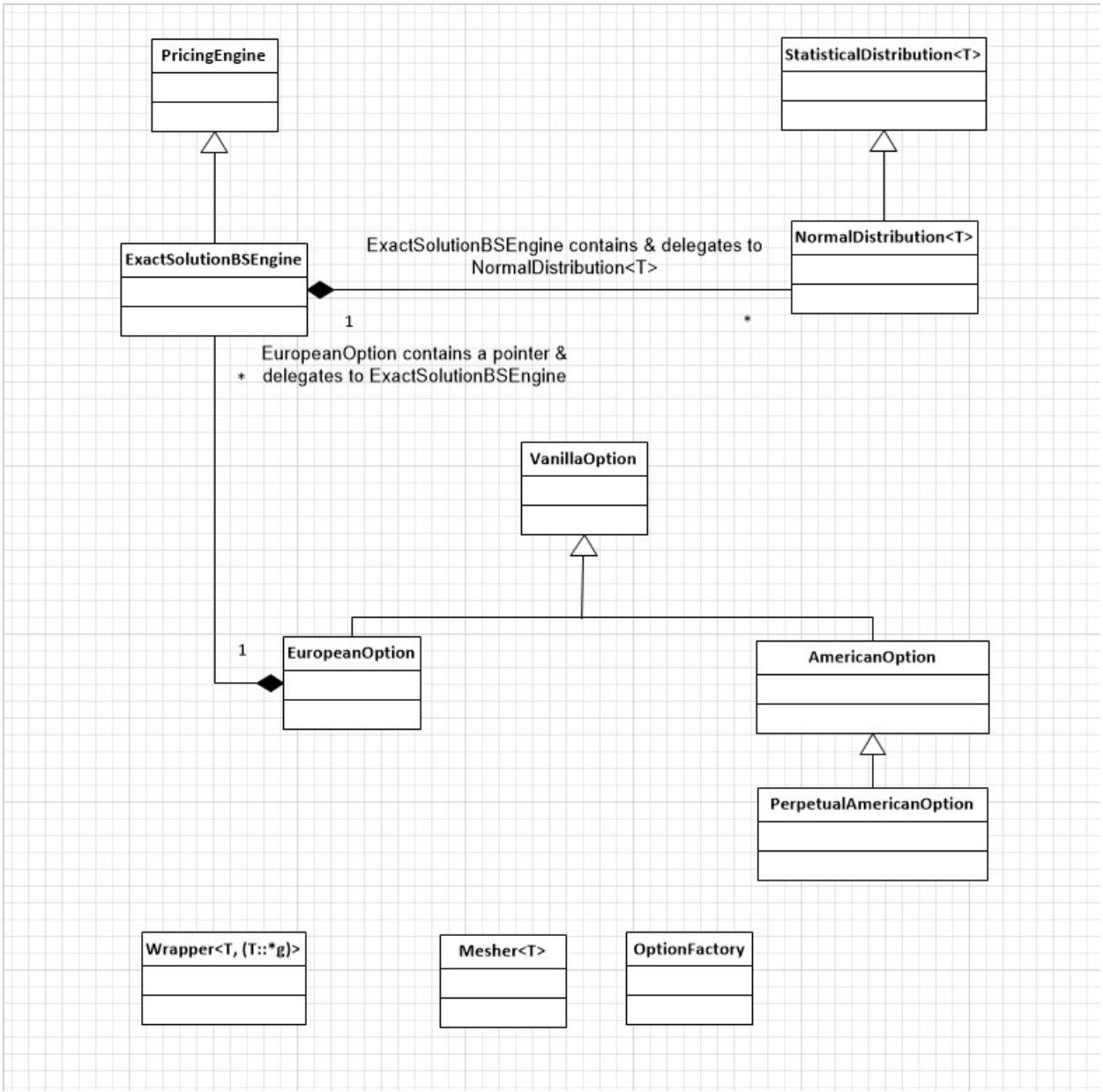


Figure 1 Class Design

My approach towards designing the classes per the TA requirements is shown above.

- *PricingEngine:* In an effort to accommodate other pricing methods (i.e., Monte Carlo, FDM, Lattice, etc.) apart from closed-form Black Scholes, I created this as a base class. Although this class currently doesn't serve much purpose, with future enhancements, I intend to transform it into an API that can, among other things, indicate what input arguments are required by a consumer to run a calculation for any pricing methodology (i.e., Black Scholes, Monte Carlo, etc.) that derives itself from this base class.

- *ExactSolutionBSEngine:* This class contains all functionality to price a vanilla European option, including Greeks. This class does NOT cater to the PerpetualAmericanOption class, since that class has its own particular calculation requirements.

  This class can receive either a vector or matrix of option parameters as an input, and return to the caller a vector of option prices:

  - *CalculateVector():* Returns a vector of option prices as a function of a monotonically increasing range of S values. This method only accommodates a monotonically increasing range of **S values**. It does not accommodate permutations to the other option parameters such as K, T, etc. I had a difficult time deciphering the requirements of the Group A&B write-up associated with this. I chose therefore, to limit this method only to a permutating S parameter, and defer complete permutation flexibility to the *CalculateMesh()* function (described below).

  - *CalculateMesh():*This function can calculate an option price as a function of ANY option parameter permutation. It parses a "matrix" of option parameters (PxN), and returns a (1xN matrix) of option prices, one for each row.

- *NormalDistribution<T>:* This class contains the Probability Density and Cumulative Distribution functions associated with a Normal (aka Gaussian Distribution). This class serves as a wrapper around the boost library - Normal Gaussian Distribution. This class provides flexibility associated with T, which represents 'class RealType' in the boost library. You can also specify the mean and standard deviation. If left to the default settings, the normal distribution will initialize with a mean of 0 and standard deviation one is 1, and therefore, would be considered as a *Standard Normal Distribution*.

- *StatisticalDistribution<T>:* This class serves as an abstract base class for NormalDistribution<T>, and potentially future distributions provided in the boost/math/distributions libraries. Per boost documentation, all boost/math/distributions implement the Probability Density and Cumulative Distribution functions. Therefore, these methods have been declared as pure virtual in this class, given that they will vary depending on the distribution being implemented. I did, however, implement Mean and Standard Deviation in this class.

- *EuropeanOption*: This class is derived from base class VanillaOption. It contains methods to price itself, check put-call parity, and mesh. In an effort to decouple an option class from its calculation, this class delegates the Black Scholes calculations to class ExactSolutionBSEngine. As part of a future enhancement, the intent would be to abstract this even further such that *any pricing engine* could be passed through EuropeanOption, or any of the "option" classes.

- *AmericanOption*: Since the exact solution Black Scholes equation does not apply to this class, there's very little functionality contained in this class. But as mentioned in the EuropeanOption section, future design considerations would call for passing any calculation methodology through this class, such as binomial model.

- *PerpetualAmericanOption*: I created a separate class for the perpetual american option, since this particular option differentiates itself from an American Option such that the Perperpetual American Option does not have an expiry date. Pricing calculations to price itself are also contained in this class.

  I wanted to abstract mesh functions (i.e., CalculateVector(), CalculateMesh(), etc.) contained both in this class and ExactSolutionBSEngine, and put them into the Mesher class. But doing so proved to be too complex for me. Specifically, I already pass a function pointer (see screenshot on following page) in an effort to avoid code duplication, and use std::invoke() to achieve this. But trying to also abstract the *call back class name* as a pointer proved too difficult at the present moment.

```cpp
// Returns a vector of option prices as a function of i) expiry time, ii) volatility, or iii) any of the option pricing parameters.
// Mesher object is passed into the CalculateMesh method, along with a function pointer based on whether the object is a call or put.
// Struct MeshParamData is passed in with all necessary mesh parameters needed to instatiate Mesher object.
vector<double> EuropeanOption::MeshPriceMatrix(const MeshParamData& _mesh_param_data) const
{
    Mesher<double> m_mesh_(this->option_vector_data(), _mesh_param_data._start, _mesh_param_data._end, _mesh_param_data._step, _mesh_param_data._pr

    if (IsCall())  // Call
        return bs->CalculateMesh(m_mesh_, &DevonKaberna::Engine::ExactSolutionBSEngine::CalculateCallPrice);   // Call CalculateMesh function in pr
    else
        return bs->CalculateMesh(m_mesh_, &DevonKaberna::Engine::ExactSolutionBSEngine::CalculatePutPrice);   // Call CalculateMesh function in pri
}
```

Figure 2 Caller function

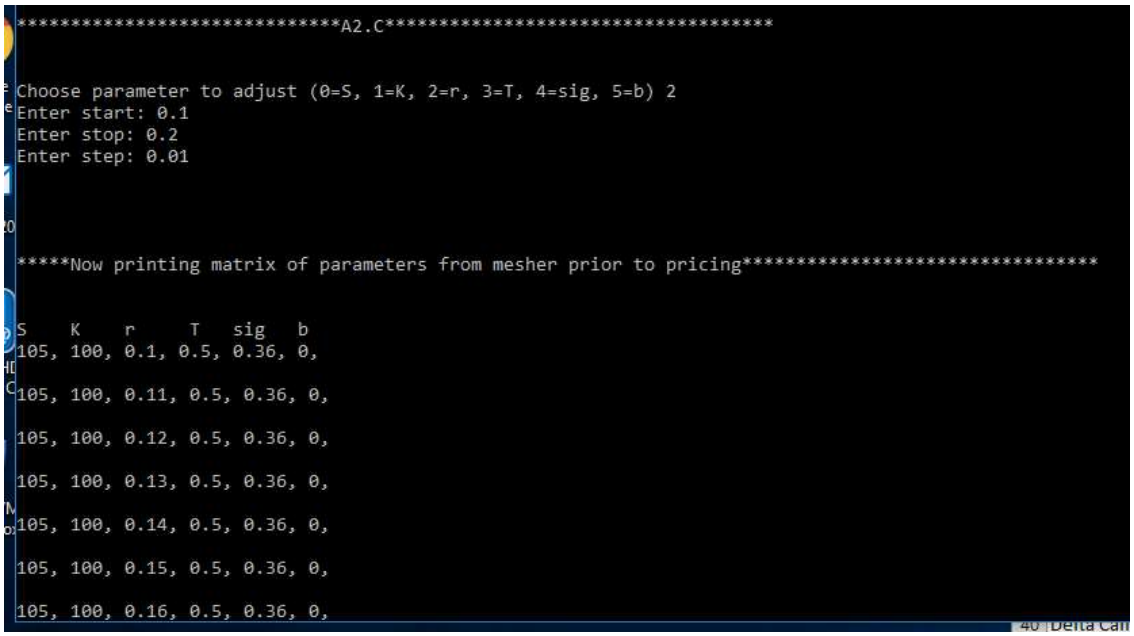
```cpp
// Calculcates option price as a function of i) expiry time, ii) volatility, or iii) any of the option pricing parameters.
// Mesher object provides a "matrix" (i.e., vector of vectors) of option parameters to this function.
// CallPrice/PutPrice/CallDelta/PutDelta/Gamma passed in as a function pointer in argument vcm.
vector<double> ExactSolutionBSEngine::CalculateMesh(const Mesher<double>& mesh, MeshModel mm) const
{
    std::size_t numberOfRows = mesh.MeshVector().size();  // Size is driven from user input of mesh size
    int start = 0;
    int row_arr = 0;

    vector<vector<double> > _OptionParamMatrix = mesh.MeshParamMatrix();  // Mesh object returns a "matrix" (i.e., vector of vectors) of

    std::vector<double> result(numberOfRows, start);  // Will store vector of option prices
    vector<vector<double> >::const_iterator vvi_iterator;   // STL iterator that iterates through each row of the matrix - each row cont

    for (vvi_iterator = _OptionParamMatrix.begin(); vvi_iterator != _OptionParamMatrix.end(); ++vvi_iterator)  // Loop through each row
    {
        result[row_arr] = std::invoke(mm, this, *vvi_iterator);  // CallPrice/PutPrice/CallDelta/PutDelta/Gamma functions is passed in v
        row_arr++;
    }
    return result;   // Return vector of option prices
}
```

Figure 3 CalculateMesh() function in ExactSolutionBSEngine

- *Mesher<T>*: This class provides functionality required by this assignment for creating meshes.
    - *MeshVector():* Outputs a mesh array of double separated by mesh size h. A consumer class such as EuropeanOption would then pass this vector onto *ExactSolutionBSEngine* (as an example), which would in turn, provide back a vector of option prices.

```
**********Now entering B1.C**********

Choose parameter to adjust (0=S) 0

Enter start: 110

Enter stop: 120

Enter step: 1

Mesh input for call

110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
```

- *MeshParamMatrix()*: Outputs a matrix (vector of vectors) of option parameters. End user can choose any parameter to permutate, such as what's shown on the following page. A consumer class such as EuropeanOption would then pass this matrix of option parameters onto *ExactSolutionBSEngine* (as an example), which would in turn, provide back a vector of option prices (1xN).



- *OptionFactory*: Although not required as part of this project, I created this class more out of curiosity, and wanted to experiment with the Factory Method design pattern. There's not much to it currently, other than to defer instantiation of any class derived from VanillaOption until run-time.
- *Wrapper<T, (T::*g)>:* This was also created out of pure curiosity and experimentation. I use it in ExactSolutionBSEngine when retrieving cdf and pdf from the boost libraries.