Pricing Engine architecture

State of the current architecture and improvements

- The first draft implemented is a first good draft to start the implementation
- However, there are some challenges to be tackled to enhance the architecture
- There is an ongoing task to restructure the code while integrating the incoming logic of implementation provided by the other teammates.

Bottlenecks in the current architecture

- Classes that are currently used are too interdependent and that will in the future complexifies the extensibility of the code.

- The use of conditional structure: we should find a better way to avoid using too much conditional structure in the code.

- Enhance the responsibility of each class

Challenges to be addressed

Principle of oriented object programming :

- **Open-closed Principle (OCP) states**:

- Objects or entities should be **open for extension but closed for modification.**
- This means that a class should be extendable without modifying the class itself.

- **Separate interface and implementation**

- Interface: An interface is an abstract concept in programming that is used to describe a behavior that classes must implement.
- Implementation: Implementation is the behavior of the interface

- **Single responsibility :**

Each class should have only one job

Improvement to the existing architecture

Abstract class: An abstract class provides the interface

Children class: Implement the interface provided by the abstract class

- Design patterns: Factory and singleton
- Polymorphism: Abstract class and Inheritance

Rearranging the current code + Ongoing Implementation + Design patterns ➔ Pricing engine

Abstract class : interface

- **PayOff** of an option will be an abstract class and Payoff Call and Payoff put will implement the abstract method of the abstract class PayOff
- **PDEConvectionDiffusion:** will be an abstract class and convection-diffusion PDE will implement the abstract method of the abstract class PDE.PDE will have access to Payoff via an instance of option class
- **FiniteDifference** will be an abstract class and EulerImplicit, Euler Explicit classes will implement the abstract methods. FiniteDifference: have access to PDEConvectionDiffusion instance

Payoff abstract class /Inherited class: payoffPut and PayoffCall

```python
# ==========================================================================
class PayOff(ABC):

    @abstractmethod
    def value(self):
        raise NotImplementedError('Implementation required!')
```

```python
class PayOffCall(PayOff):

    def __init__(self,strike_):
        self.strike  = strike_

    def value(Spot):
        return max(spot - self.strike,0 )
```

```python
class PayOffPut(PayOff):

    def __init__(self,Strike_):
        self.Strike  = Strike_

    def value(Spot):
        return max(0,self.Strike - Spot )
```

More abstract methods will be added if required

Class payoffPut:
  Implement   value

Class PayoffCall:
  Implement value

Factory Payoff class: Design patterns, factory and singleton

```
class PayOffFactory:
    def __init__(self):
        self._builders = {}

    def register_builder(self, PayOffId, builder):
        self._builders[PayOffId] = builder

    def createPayoff(self, PayOffId, **kwargs):
        builder = self._builders.get(PayOffId)
        if not builder:
            raise ValueError(PayOffId)
        return builder(**kwargs)
```

Factory design pattern to register and create a new payoff object from user input.
This design pattern will help us to avoid the use of many conditional structures in the code.

Vanilla option

```
class VanillaOption():
      def __init__(self,PayOff,Expiry,parameters):
            self.PayOff   = PayOff #copy object to be checked
```

This class is not abstract and will have Payoff as an attribute and external data provided by the user

PDE abstract class and BS PDE inherited

```python
#-----------------------------------------------------------------
#convection diffusion equation - second order PDE
class PDEconvectionDiffusion(ABC):
      #attribute
      def __init__(self,):
          pass
      #pde coefficients
      @abstractmethod
      def coefficient_convection(self):
          raise NotImplementedError('Implementation required!')

      @abstractmethod
      def coefficient_diffusion(self):
          raise NotImplementedError('Implementation required!')

      @abstractmethod
      def coefficient_source(self):
          raise NotImplementedError('Implementation required!')

      @abstractmethod
      def zero_coefficient(self):
          raise NotImplementedError('Implementation required!')

     #boundary and init condition
      @abstractmethod
      def boundary_left(self):
          raise NotImplementedError('Implementation required!')

      @abstractmethod
      def boundary_right(self):
          raise NotImplementedError('Implementation required!')

      @abstractmethod
      def init_cond(self):
          raise NotImplementedError('Implementation required!')
```

```python
class PDEBlackScholes(PDEconvectionDiffusion):

      def __init__(self,..):

      def coefficient_convection(self,x,t):

          return

      def coefficient_diffusion(self,x,t):

          return

      def coefficient_source(self):

          return

      def zero_coefficient(self):

          return

      def boundary_left(self):

          return

      def boundary_right(self):

          return

      def init_cond(self):

          return
```

Class Inheritance :PDE convection diffusion

- Class PDE BS: inherits from class PDE
- Attribute: an attribute related to the option
- Method to be implemented by the class:
- coefficient_convection( )
- coefficient_diffusion()
- coefficient_source()
- boundary_left()
- boundary_right()
- init_cond()

Finite difference abstract class and Euler explicit/implicit inherited

```python
class FiniteDifference(ABC):

      def __init__(self,_x_dom,_J,_t_dom,_N,_pde):
          ##space discredization
          #spatial extent
          self.x_dom = _x_dom

          #number of special differencing points
          self.J = _J
          #temporal step size
          self.dx
          ##time discredisation

          #temporal extent
          self.t_dom = _t_dom

          #coordinates of the x dimension
          self.xvalue

          #Number of temporal differencing points
          self.N = _N

          #temporal step size (to be calculated)
          self.dt

          self.pde = _pde

          #time marching
          self.current_time ;
          self.previous_time;

          #differencing coefficient
          self.alpha
          self.beta
          self.gamma

          #storage
          #New result
          self.new_result
          #oldresult
          self.old_result
```

```python
class FDMEulerExplicit(FiniteDifference):

      def calculate_step_sizes():
      def set_initial_condition()
      def calculate_boundary_conditions()
      def calculate_inner_domain()
      def step_march()
```

FDMEulerExplicit will implement the method provided by the abstract Parent class FiniteDifference

Abstract class: Finite difference

Attribute related to the time discretization:

- Temporal extent (0,t_dom)
- Number of temporal differencing points N
- Temporal step size (calculated from above) dt;

Attribute related to the Time marching (useful to track the computation):

- Current and previous times: prev_t,cur_t

Attribute related to the Differencing coefficients:

Alpha,beta, gamma

Attribute related to the solution: Storage

- A new solution: new_result;
- Old Solution: old_result;

Abstract class: Finite difference

Attribute related to the time discretization

- Temporal extent (0,t_dom)
- Number of temporal differencing points N
- Temporal step size (calculated from above) dt;

Attribute related to the Time marching (useful to track the computation):

- Current and previous times: prev_t,cur_t

Attribute related to the Differencing coefficients:

- Alpha,beta,gamma

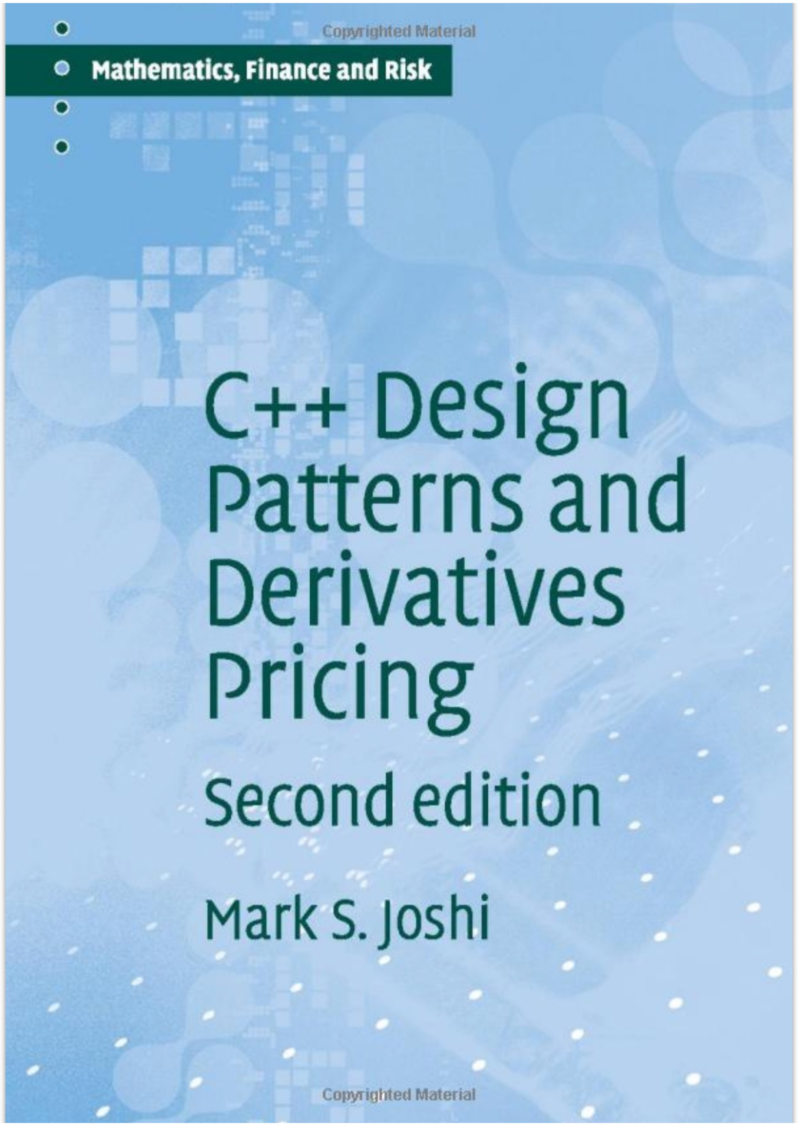
Attribute related to the solution: Storage

- A new solution: new_result;
- Old Solution: old_result;

Abstract Method

- FiniteDifference(_x_dom,_J,_t_dom,_N,_pde_convect_diff)
- Abstract Method to be implemented by the child class (FiniteDifferenceEulerExplicit, FiniteDifferenceEulerImplicit, FiniteDifferenceCrankNicolson):
- Calculate_step_sizes()
- Set_initial_conditions()
- Interpolate()
- *Calculate_boundary_conditions() : use the boundary of the PDE (boundary_left() and boundary_right())*
- Calculate_inner_domain(): solve the equation
- Step_march (): Carry out the actual time-stepping, traversing the grid from current time (cur_t) to the border (t_dom):

The end


Copyrighted Material
Mathematics, Finance and Risk
C++ Design Patterns and Derivatives Pricing
Second edition
Mark S. Joshi
Copyrighted Material