Division of Labor (Alphabetical order):

- Fei Liu: Language guru
- Mengdi Zhang: System architect
- Taikun Liu: Verification and validation
- Jiayi Yan: Manager

1. Describe the language that you plan to implement:

Matrix-driven language: This is a language that is specialized in matrix computation and offering handy computational advantage particularly for financial applications such as option pricing, simulation, and solving Stochastic Differential Equations.

The key difference is that our language has built-in real time graphic support for the pricing mechanics.

The language should be able to shelter the user from intensive loops and manipulation of matrices, and data collection for graph or analysis. For example, the built-in operator should be able to conduct the algebra equivalent operations on matrix types automatically. At the same time, we should be able to calculate and compute individual entries of the matrix by index.

When the user apply the specific data structure tailored for option pricing, the graphing function is built-in.

The graph function can also be overloaded so that people have tweaks on their own applications.

2. Explain what sort of programs are meant to be written in your language

- Partial differential equation with Finite Difference method or Crank Nicolson method (This method is essentially dynamic programing and backward reducing. The characteristic of this approach is the dependency on matrix operation)
- Backward Pricing (Specialized pricing starting from payoff. This approach can be used to price barrier options.)
- Monte Carlo Simulations (Take advantage of sampling distribution and random generate the paths. We then evaluate the path results to understand the pricing basis.)
- Quasi-ML simulation. (A Columbia professor and his PhD student first came up with this idea and the difference is using Low Discrepancy Random generators to price financial products. This is domain specific and works well only with financial applications.)

3. Explain the parts of your language and what they do

- Basic Syntax
    - Datatypes
        - Float: The basic block of all computation is float.
        - Bool: Boolean variable
        - Matrix: matrix of float values, could be array, vector or matrix.

- String: For comments and outputs.
- Struct: Customized container of information.
- Option: A built-in object for option. The option is pre-typed to be one of “European”, “American”, “Barrier” and “Exotic”.
  - European: excercise the option exactly on the date pecified
  - American: excercise the option anywhere before the date specified
  - Exotic: The option type can have customized PayOff, Boundary conditions, Integration Method (for PDE approach), Random Number generator, and condition checking. Most notably, all options have graph() method. They will display in real time the pricing mechanism.

- Variable Declaration and Assignment

  General rules are:

  (type)(variable name);
  (variable name) = …;
  (type)(variable name) = …;

  For matrix:

  matrix A(dim1, dim2, …, dimn);
  A is by default filled with zero values.

  For struct:

  struct is implemented as map in C++.
  struct S;
  S(“name”, data)

  For option:

  Option myoption(<type>);

- Function

  returnarg1, …, returnargn. functionName (inputarg1, …, inputargn)

- Control Structure
  - if, else, elseif,
  - while,

  while(bool expression) or
  while(matrix traverse)
  - break, continue, return

- Arithmetic and relational operators:

  - = (assignment),
  - + (matrix addition),
  - - (matrix subtraction),
  - * (matrix multiplication),
  - / (matrix division is like inversion of the left matrix and multiply the right),
  - ** (pointwise matrix multiplication),
  - // (pointwise matrix division),
  - == (equality), <, <=, >, >=, !=,
  - ~ (negation), &&, ||

- Built-in functions:
  - normalcdf(x, y), the cumulative density function for normal

distribution. The result is a matrix of specified size.
- normalpdf(), the probability density function for normal distribution.
- log2(), logarithmic function base 2
- logE(), logarithmic function base e
- inverse(), matrix inversion
- transpose(), matrix transpose
- exp(): exponential value
- max(): maximum value
- abs(): absolute value
- more to be decided.

- Program Structure
  - Comments: "{** *COMMENTS* **}"
  - Price <a option typed variable>: start the pricing function process. This is a starter function.

4. Include the source code for an interesting program in your language

{**The following piece code express a LU decomposition algorithm**}

```
float s=size(A);
matrix U=A;
matrix L(s,s);
matrix PV=transpose(A(0 to s-1));
while float j=1 to s
    float ind;
    [~,ind]=max(abs(U(j to s, j)));
    ind=ind+j-1;
    matrix t=PV(j); PV(j)=PV(ind); PV(ind)=t;
    t=L(j,1 to j-1); L(j,1 to j-1)=L(ind,1 to j-1); L(ind,1 to
j-1)=t;
    t=U(j,j to end); U(j,j to end)=U(ind,j to end); U(ind,j to
end)=t;
    L(j,j)=1;
    while float i=(1+j) to size(U,1)
       float c= U(i,j)/U(j,j);
       U(i,j to s)=U(i,j to s)-U(j,j to s)*c;
       L(i,j)=c;
matirx P(s,s);
P(PV(:)*s+transpose(1 to s))=1;
```