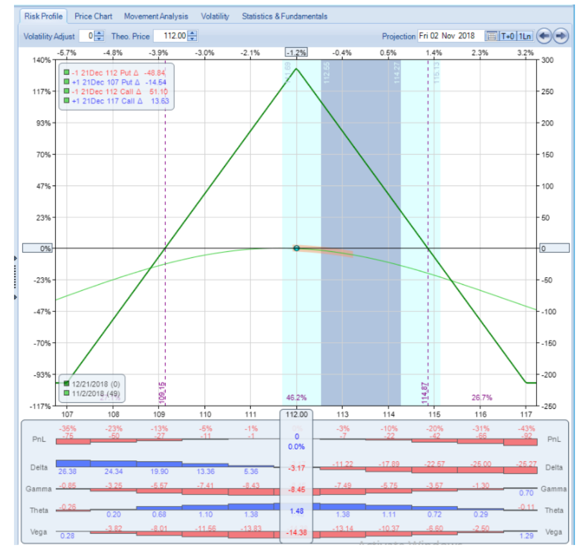
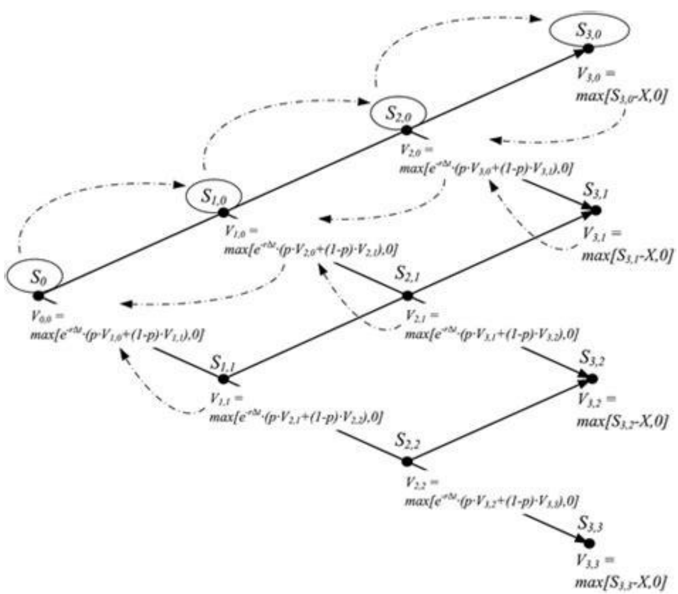
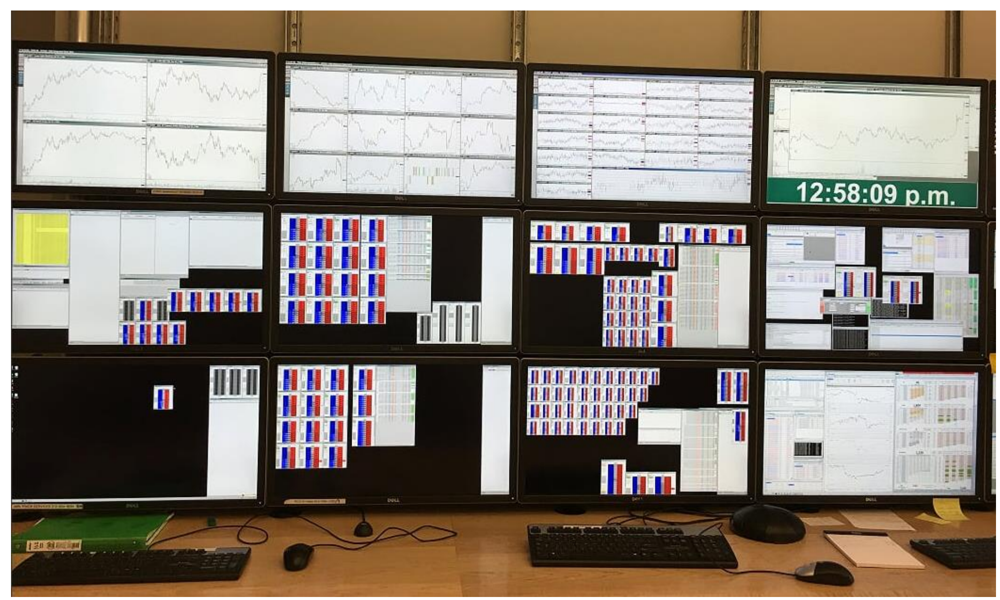
Pricing Equity Options with `mdarray`

NWCPP

17 May 2023







- Coming this year to C++23: **`std::mdspan`** (P0009)
  - Can impose a multidimensional array structure on a reference to a container, such as an STL vector (without tears)
  - View of existing data (ie, non-owning)
  - Example: 2-D “matrix” view of vector data:

```
101   102
103   104
105   106
```

```
vector<int> v{ 101, 102, 103, 104, 105, 106 };
auto mds1 = std::mdspan{ v.data(), 3, 2 };        // using CTAD
```

- Terminology:
  - rows and columns are referred to as *extents*
  - the number of rows and columns are accessed by the index of each extent, 0 for rows and 1 for columns
  - The total number of extents is referred to as the *rank*

```
int n_rows{ mds1.extent(0) };                     // 3 rows
int n_cols{ mds1.extent(1) };                     // 2 columns
int n_extents{ mds1.rank() };                     // rank = 2
```

Note: The term *rank* as applied to **`mdspan`** is not the same as the mathematical definition of the *rank of a matrix*. This naming might unfortunately seem confusing, but it’s something one should be aware of.

  - For higher-order multidimensional arrays, the rank would be greater than two

- Extent sizes for **`mdspan`** (and **`mdarray`**) can be dynamically allocated

Introduction

- What if we don’t know the data a priori?
- What if we want to generate values and place them in a two-dimensional array format, for example?
- Enter **`mdarray`** (P1684)
  - Expected in C++26
  - But available on GitHub now (https://github.com/kokkos/mdspan)
  - In namespace **`std::experimental`**
  - Will use the alias

```
namespace stdex = std::experimental;
```

- Number of rows and columns can be allocated dynamically:

```
stdex::mdarray<int, stdex::dextents<int, 2>> test_mdarray{ m, n };
```

  - **`stdex::dextents`** => dynamic extents
  - “Two dynamic extents of **`int`** type” (**`m, n`** are arbitrary (positive) **`int`** values)

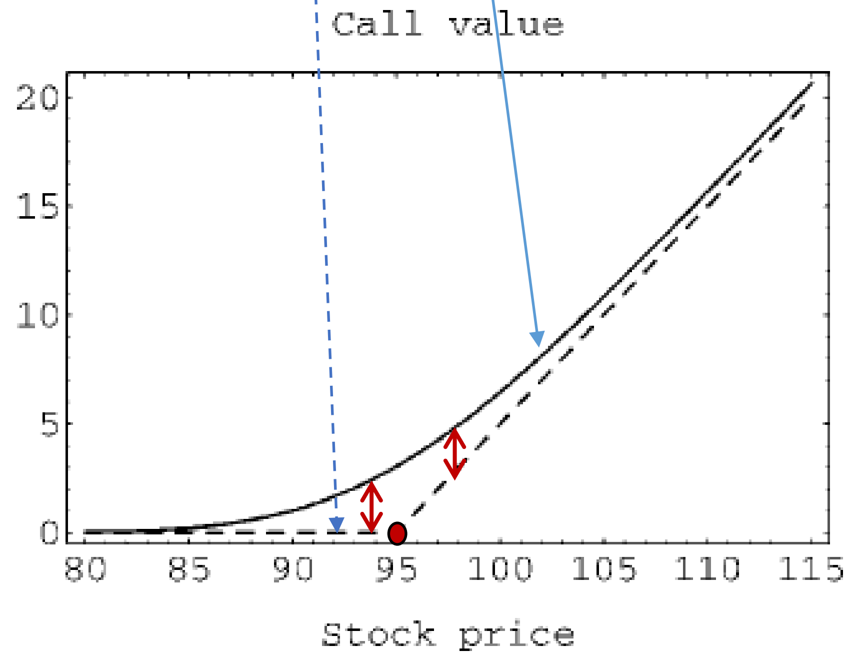
Binomial Lattice Option Pricing

Options Trading and the Model

- An equity (stock) option is a derivative that gives the holder the *right* to buy or sell the stock at set price (the exercise price) on or before the expiration date
  - Call option: option to buy
  - Put option: option to sell
- Let S = the underlying stock price
- Let X = the exercise (strike) price
- The payoff when *exercised*:
  - Call option: $max(S - X, 0)$
  - Put option: $max(X - S, 0)$
- In-the-money (ITM)
  - Call option: $(S - X) > 0$
  - Put option: $(X - S) < 0$
- At-the-money (ATM) if equal
- Out-of-the-money (OTM) otherwise

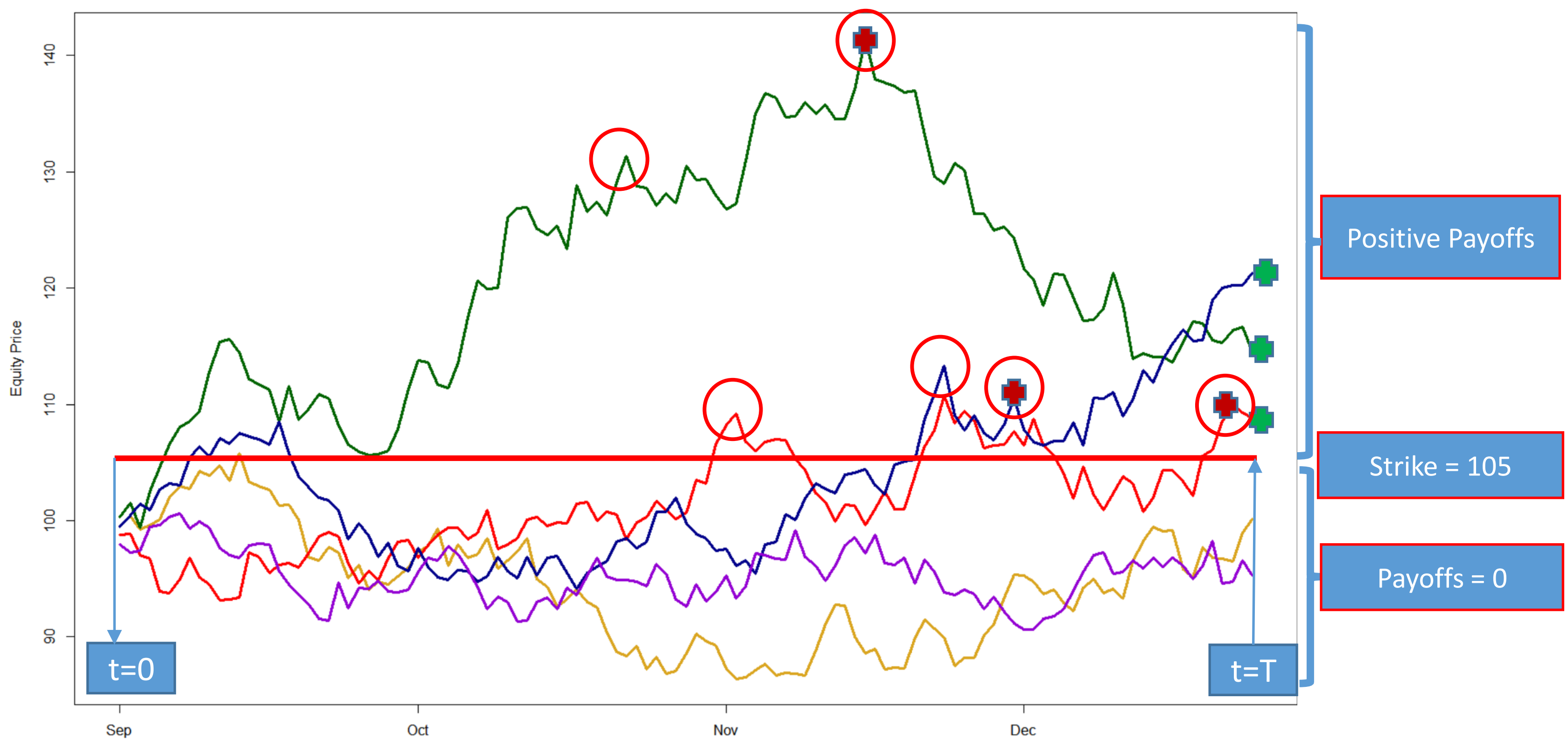
Call option time t < expiration
Strike X = 95
Solid line: Value of option (prior to exercise)
Dotted line: Intrinsic value (payoff if exercised)
Difference: Time value
S > 95 ATM, S = 95 ITM, S < 95 OTM

(Lyuu, Figure 7.3, p 77)



Binomial Lattice Option Pricing

- *When* can we exercise an options contract? (example: call option)
  - European: may only be exercised on the expiration date
  - American: may be exercised any time before or on the expiration date
- *Either* type may be bought or sold anytime before expiration (t = T)
- Goal: Calculate a fair price for a call option today (t = 0)
- Note: The price of an American option is never less than a European option

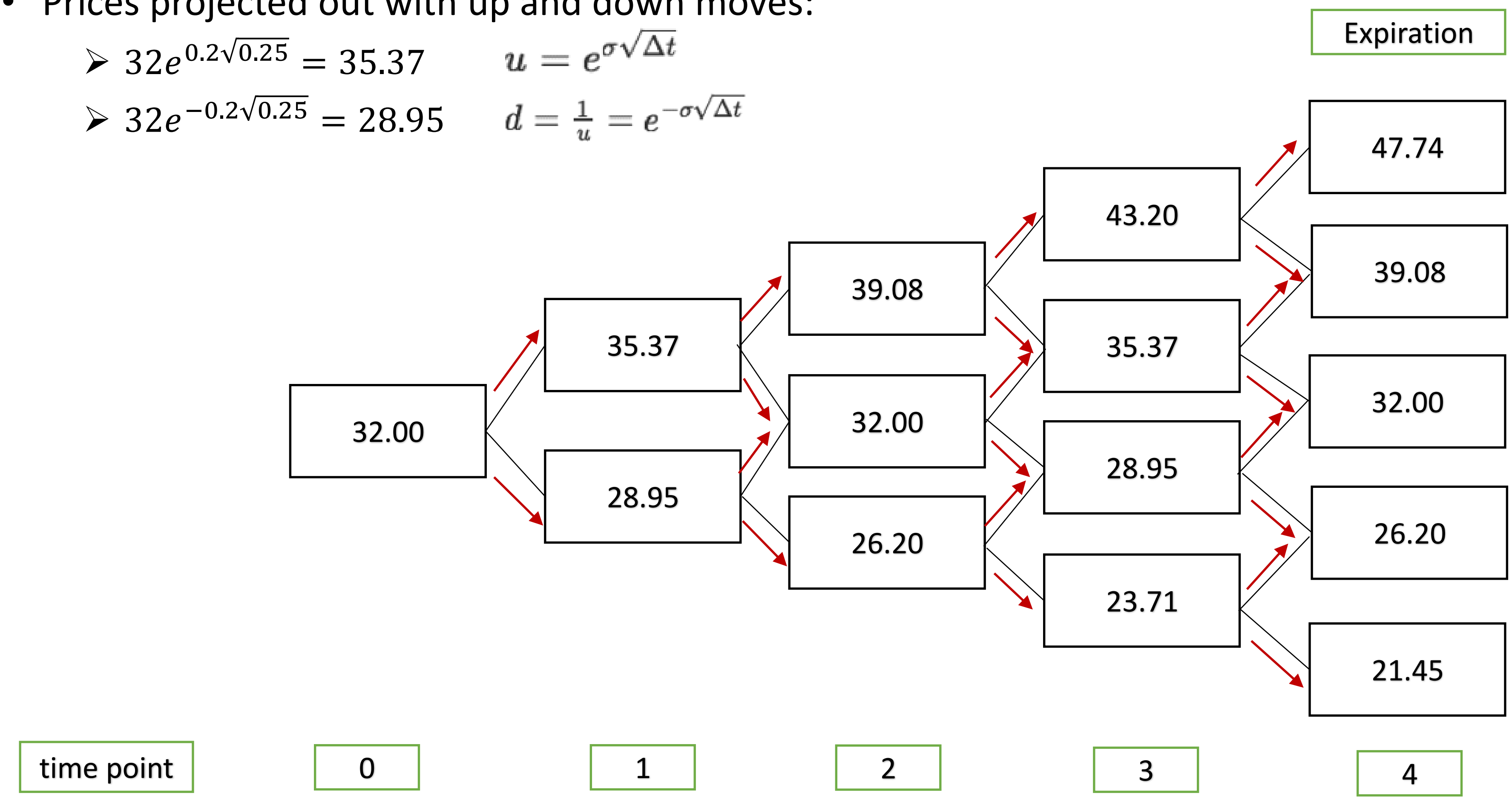


Binomial Lattice Option Pricing

- European equity options can be valued using the closed-form Black-Scholes pricing formula
- American option valuation requires numerical approximation
  - No closed form formula exists
  - Need to check for optimal early exercise
  - The Binomial Lattice pricing model is well-suited for this purpose
  - This is where **`mdarray`** can help us
  - Boost MultiArray is also useful
  - But now we will have a tool in the C++ Standard Library

Step 1: Project Prices Forward

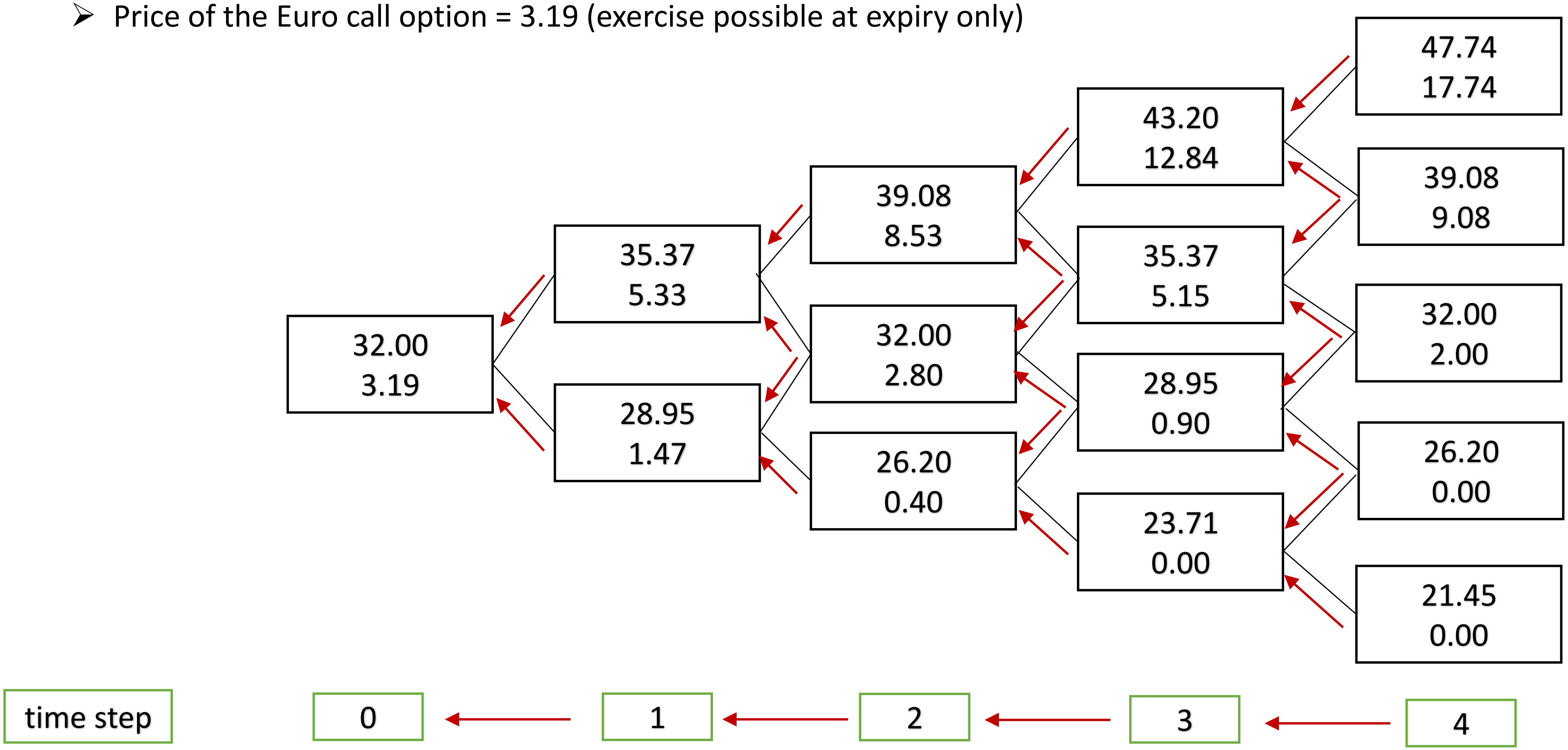
- Call option, time to expiration = 1 year
- Binomial lattice with 4 time steps (plus t = 0)
  - Time step = Δt = ¼ = 0.25
  - Initial (spot) equity price = 32
  - Volatility (σ) = 0.20, annual risk-free rate (r) = 6%, annual (continuous) dividend(q) = 7.5%
  - Prices projected out with up and down moves:
    - $32e^{0.2\sqrt{0.25}} = 35.37$ $u = e^{\sigma\sqrt{\Delta t}}$
    - $32e^{-0.2\sqrt{0.25}} = 28.95$ $d = \frac{1}{u} = e^{-\sigma\sqrt{\Delta t}}$



Step 2: Discounted Expected Payoffs (European Case)

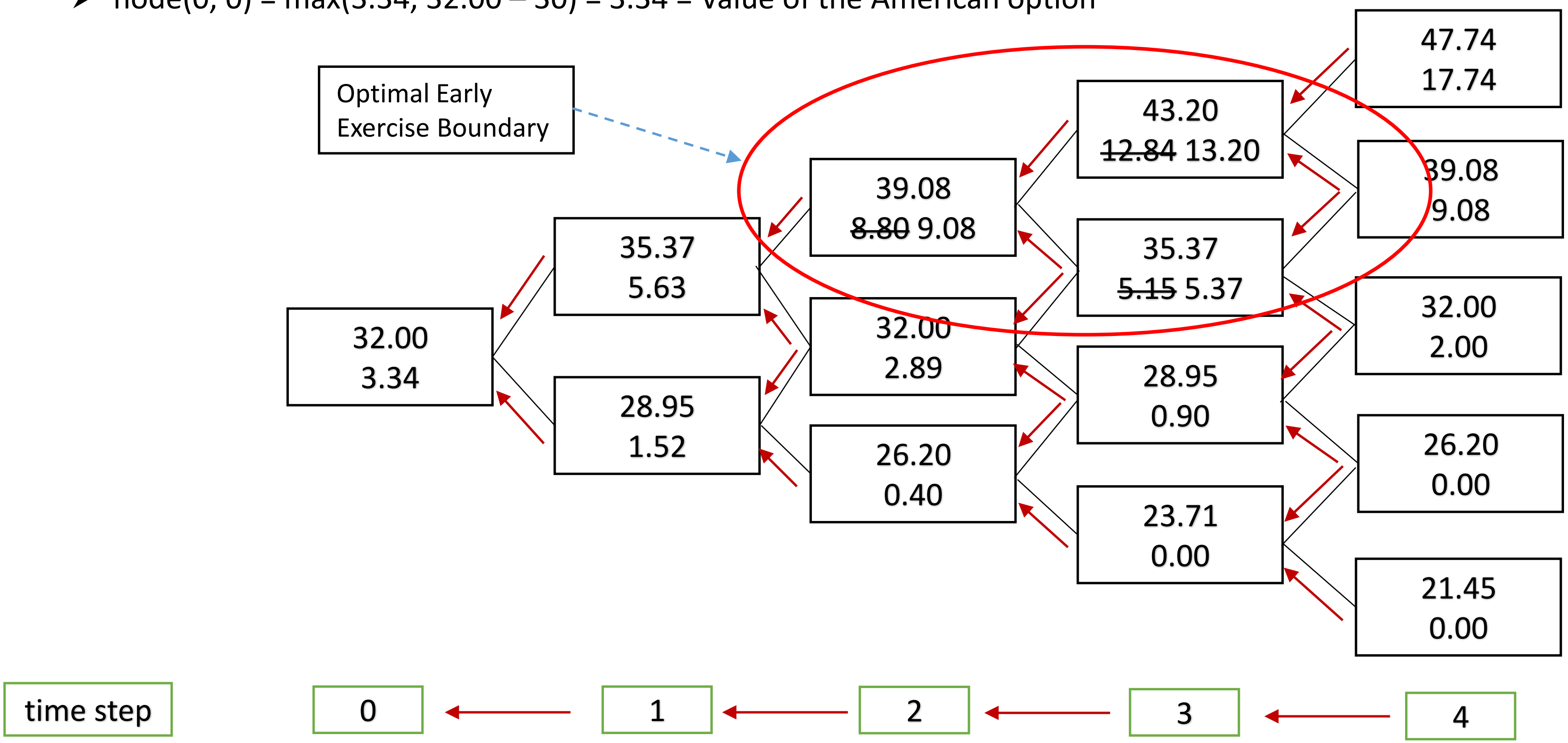
- Calculate discounted expected payoffs moving back in time
  - X = 30: Strike price (exercise price)
  - At expiration: $max(S - X, 0)$
    - ➢ node (0, 4): max(47.74 – 30.00, 0) = 17.74
    - ➢ node (3, 4): max(26.20 – 30.00, 0) = 0
  - At each preceding node, compute the expected value and discount back by Δt = 0.25, using
    - ➢ node(0, 3) = $e^{-0.06(0.25)}\big(17.74p + 9.08(1-p)\big)$ = 12.84
    - ➢ node(0, 0) = $e^{-0.06(0.25)}\big(5.33p + 1.47(1-p)\big)$ = 3.19
    - ➢ Price of the Euro call option = 3.19 (exercise possible at expiry only)

$$p = \frac{e^{(r-q)\Delta t} - e^{-\sigma\sqrt{\Delta t}}}{e^{\sigma\sqrt{\Delta t}} - e^{-\sigma\sqrt{\Delta t}}}$$



American Call Option

- American options can be exercised anytime before expiration
  - Need to check for optimal early exercise (**`mdarray`** becomes especially useful here)
  - Start with payoffs at expiration
  - Compare discounted expected payoff to actual payoff at each node moving backward in time
    - ➢ node(0, 3) = max(12.84, 43.20 - 30) = 13.20
    - ➢ node(0, 2) = max(8.80, 39.08 - 30) = 9.08
    - ➢ Each subsequent discounted expected value is based on the updated payoffs in the subsequent time period
    - ➢ It is then compared with the actual payoff if exercised
    - ➢ node(0, 0) = max(3.34, 32.00 – 30) = 3.34 = Value of the American option



Binomial Lattice Option Pricing

Implementation using **`mdarray`**

Extensions and Convergence


"All the News That's Fit to Print"

The New York Times

Late Edition

VOL.CXXXVII ... No. 47,298 — NEW YORK, TUESDAY, OCTOBER 20, 1987 — 30 CENTS

STOCKS PLUNGE 508 POINTS, A DROP OF 22.6%; 604 MILLION VOLUME NEARLY DOUBLES RECORD

U.S. Ships Shell Iran Installation In Gulf Reprisal

Offshore Target Termed a Base for Gunboats

By STEVEN V. ROBERTS

A Huge Blow to the Five-Year Bull Market

Dow's Record Fall

WORLDWIDE IMPACT

Frenzied Trading Raises Fears of Recession — Tape 2 Hours Late

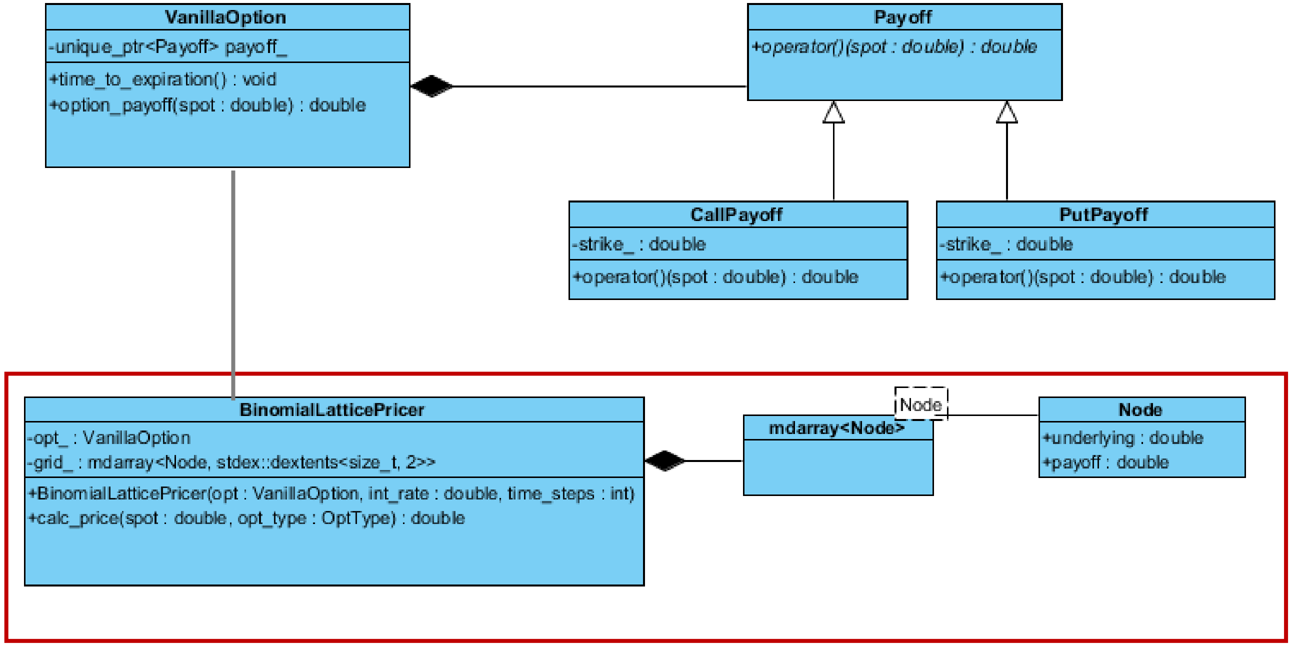
By LAWRENCE J. De MARIA

Does 1987 Equal 1929?

By ERIC GELMAN

Binomial Lattice Option Pricing Implementation

- A **VanillaOption** class
  - Does not perform valuation
  - Holds a **Payoff** resource (call or put, determined at runtime), and time to expiration
- The **Payoff** resource will represent the payoff of an option
  - Call or Put, determined dynamically
  - Holds the contractual strike price
  - Returns the payoff given the underlying market share price
- The assembled vanilla option contract
  - Is passed to **BinomialLatticePricer** for valuation
  - The generated lattice data is held on the **mdarray** member **grid_** with **Node** template parameter



Price Projection in Code

- Putting this into a two-dimensional array in C++

  - Project up movement from each element in a given column

  - The last element is also projected to a down movement

Index	0	1	2	3	4
0	32.00	35.37	39.08	43.20	47.74
1		28.95	32.00	35.37	39.08
2			26.20	28.95	32.00
3				23.71	26.20
4					21.45

Binomial Lattice Option Pricing Declaration

- **`BinomialLatticePricer`** class:
  - **`VanillaOption opt`** holds the payoff (call or put) and time to expiration
  - **`calc_price(.)`**
    - ➢ Can take in a European or American option type – **`mdarray`** itself doesn't care
    - ➢ Underlying spot price taken from current market data
  - **`Node`** struct
    - ➢ Set underlying prices projecting out the lattice
    - ➢ Set discounted expected payoffs on the return trip
  - **`mdarray<Node, stdex::dextents<size_t, 2>> grid_`** member: stores the lattice dynamically
    - ➢ 2 extents (rows and columns)
    - ➢ Will store underlying and payoff values at each lattice node

```cpp
struct Node
{
    double underlying;
    double payoff;
};
```

```cpp
enum class OptType
{
    Euro,
    American
};
```

```cpp
class BinomialLatticePricer
{
public:
    BinomialLatticePricer(VanillaOption&& opt,
        double vol, double int_rate, int time_steps,
        double div_rate = 0.0);

    double calc_price(double spot, OptType opt_type);

    void display_lattice_nodes() const;

private:
    VanillaOption opt_;
    double vol_, int_rate_;
    int time_points_;
    double div_rate_;

    double u_{ 0.0 }, d_{ 0.0 };    // up and down factors
    double p_{ 0.0 };               // probability of up move
    double disc_fctr_;              // discount factor

    stdex::mdarray<Node, stdex::dextents<int, 2>> grid_;

    double spot_{ 0.0 }, opt_price_{ 0.0 };

    void project_prices_();
    void calc_payoffs_(OptType opt_type);
};
```

Binomial Lattice Option Pricing Implementation

- r = risk-free interest rate (**`int_rate`**), q (**`div_rate`**) = continuous annual dividend rate
- σ (**`vol`**) = volatility of stock price (annualized)
- Δt (**`dt`**)
  - year fraction over one time step
  - Ex: One year divided into four time steps of length Δt = 0.25
- **`BinomialLatticePricer`** implementation – constructor
  - Initializes the 2-D mdarray
    - number of rows and columns = number of time steps + 1 (“time points”)
    - Template parameter: **Node**
  - Computes and assigns the lattice parameters **u_**, **d_**, **p_**
  - Calculates the constant discount factor (**`disc_fctr_`**) over each Δt: $e^{-r\Delta t}$

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = \frac{1}{u} = e^{-\sigma\sqrt{\Delta t}}$$

$$p = \frac{e^{(r-q)\Delta t} - e^{-\sigma\sqrt{\Delta t}}}{e^{\sigma\sqrt{\Delta t}} - e^{-\sigma\sqrt{\Delta t}}}$$

```cpp
BinomialLatticePricer::BinomialLatticePricer(VanillaOption opt,
    double vol, double int_rate, int time_steps, double div_rate) :
    opt_{ std::move(opt) }, vol_{ vol }, int_rate_{ int_rate },
    time_points_{ time_steps + 1}, div_rate_{ div_rate },
    grid_{ time_points_, time_points_ }
{
    double dt{ opt_.time_to_expiration() / time_steps };
    u_ = std::exp(vol * std::sqrt(dt));
    d_ = 1.0 / u_;
    p_ = 0.5 * (1.0 + (int_rate - div_rate - 0.5 * vol * vol) * std::sqrt(dt) / vol);
    disc_fctr_ = std::exp(-(int_rate)*dt);
}
```

Binomial Lattice Option Pricing Implementation

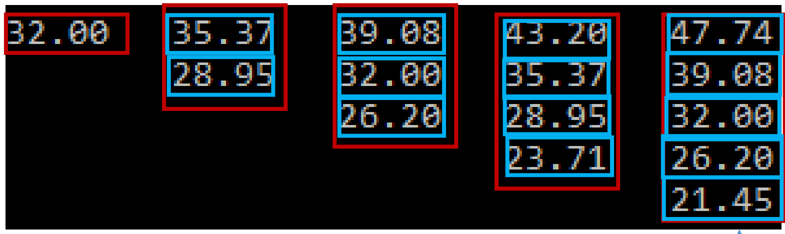
- **`BinomialLatticePricer`** implementation – valuing the option
- Divided into two steps
    - Project the generated prices forward in time: Calculate the expected payoffs
    - Traverse the lattice back in time: Check for optimal early exercise
    - **`opt_type`**: European or American

```
double BinomialLatticePricer::calc_price(double spot, OptType opt_type)
{
    spot_ = spot;
    project_prices_();
    calc_payoffs_(opt_type);

    return opt_price_;
}
```

Binomial Lattice Option Pricing Implementation

- Project the underlying share prices forward in time: **`project_prices_()`**
  - Initial (spot) share price = 32 set at terminal node
  - Iterate forward over the **`mdarray`**
    - ➢ By column
    - ➢ Then row
    - ➢ Multiply each previous price by values of u (up move return) and d (down move return)
    - ➢ Set the underlying value on the **`Node`** struct at each node

```cpp
void BinomialLatticePricer::project_prices_()
{
    grid_(0, 0).underlying = spot_;       // Initial share price at terminal node

    // j: columns, i: rows.
    // Traverse by columns, then set node in each row.
    for (int j = 1; j < time_points_; ++j)
    {
        for (int i = 0; i <= j; ++i)
        {
            if (i < j)
            {
                grid_(i, j).underlying = u_ * grid_(i, j - 1).underlying;
            }
            else    // (i == j)
            {
                grid_(i, j).underlying = d_ * grid_(i - 1, j - 1).underlying;
            }
        }
    }
}
```



Projected share prices at option expiration

- **<u>Note</u>**: in C++23, we will have the multidimensional square bracket operator overload:
  - **`grid_(i, j)`**   **`// Prior to C++23`**
  - **`grid_[i, j]`**   **`// C++23`**

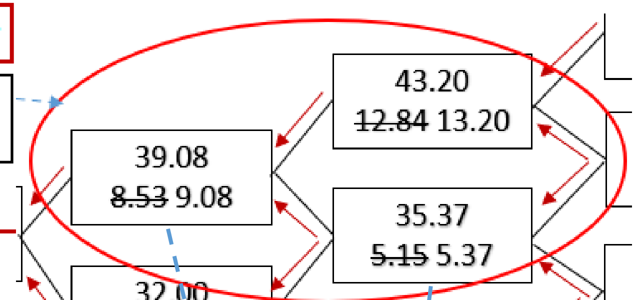
Binomial Lattice Option Pricing Implementation

- Then, iterate *backward* in time through the **`mdarray`**: **`calc_payoffs_(.)`**
  - Start at last column at expiration, calculate actual payoff and set on struct
  - Traverse backward by column first, then row top to bottom
  - Calculate the discounted expected payoffs for each node and set on the Node
  - If an American option, compare with what the actual payoff would be at each prior node and replace if greater
  - Value at final node ([0, 0] position) in the mdarray, is the estimated option value

Projected Share Prices (previous step)

32.00	35.37	39.08	43.20	47.74
	28.95	32.00	35.37	39.08
		26.20	28.95	32.00
			23.71	26.20
				21.45

```cpp
void BinomialLatticePricer::calc_payoffs_(OptType opt_type)
{
    for (int j = time_points_ - 1; j >= 0; --j)
    {
        for (int i = 0; i <= j; ++i)
        {
            if (j == time_points_ - 1)  // Actual (not expected) payoff at expiration
            {
                grid_(i, j).payoff = opt_.option_payoff(grid_(i, j).underlying);
            }
            else                        // Each node prior to exp
            {
                double expct_val = p_ * grid_(i, j + 1).payoff;
                expct_val += (1.0 - p_) * grid_(i + 1, j + 1).payoff;
                expct_val *= disc_fctr_;
                if (opt_type == OptType::American)
                {
                    grid_(i, j).payoff =        // Check if exercise is optimal
                        std::max(expct_val, opt_.option_payoff(grid_(i, j).underlying));
                }
                else    // OptType::Euro
                {
                    grid_(i, j).payoff = expct_val;
                }
            }
        }
    }

    opt_price_ = grid_(0, 0).payoff;        // Value of the option
}
```

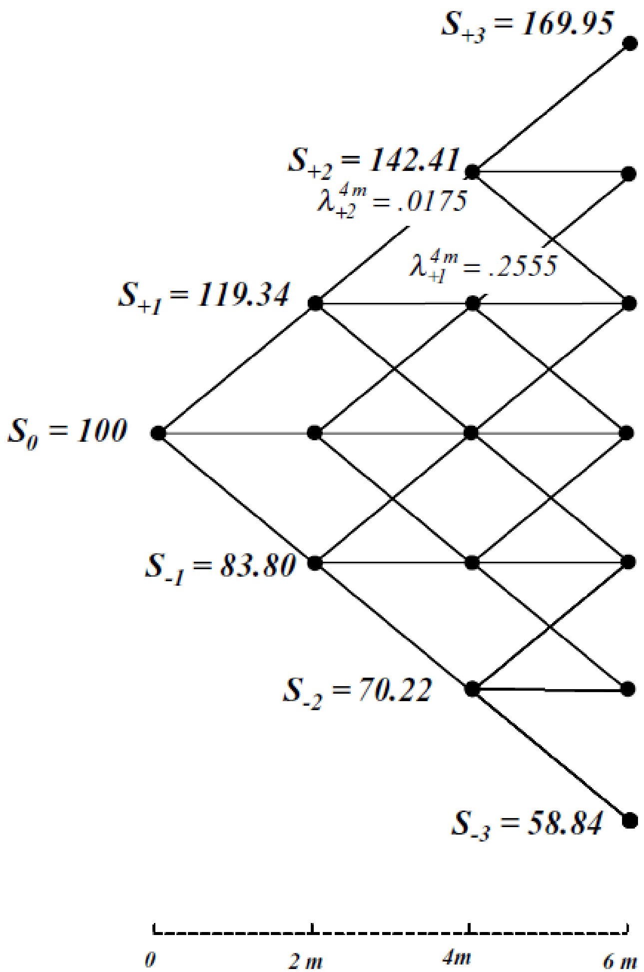


3.34	5.63	9.08	13.20	17.74
	1.52	2.89	5.37	9.08
		0.40	0.90	2.00
			0.00	0.00
				0.00

Trinomial Lattices

- Trinomial lattices are common in
  - Option models with stochastic interest rates
  - Incorporating variable volatility models

```
time_points_{ time_steps + 1 },
grid_{ time_points_ + 1, time_points_ }
```

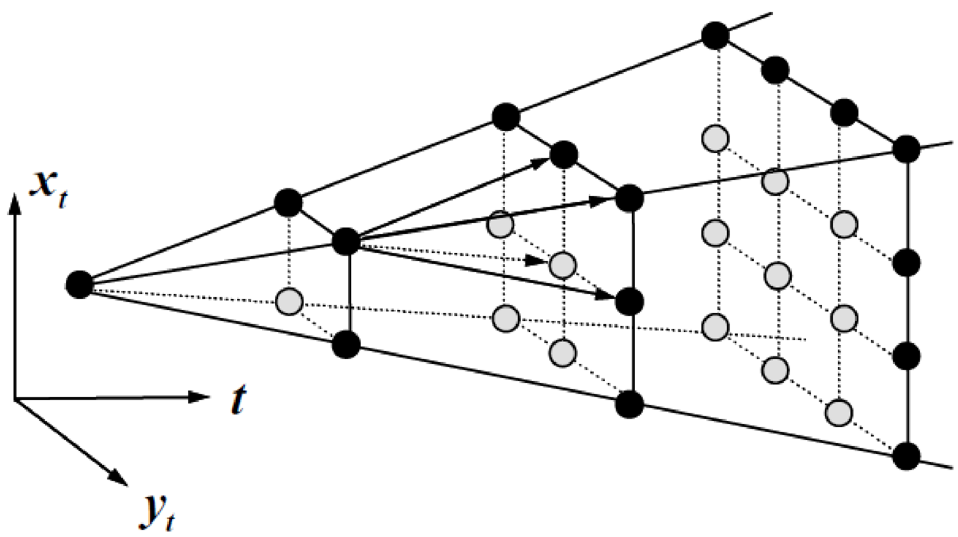


Index	0	1	2	3
0	100	119.34	142.41	169.95
1		100	119.34	142.41
2		83.80	100	119.34
3			83.80	100
4			70.22	83.80
5				70.22
6				58.84

James, Figure 9.8, p 119

Higher Dimensions

- It is also possible to have options on more than one security
- Example payoffs, S_i^T = price of i^{th} security at expiration T, strike price X
  - $\max(S_1^T, S_2^T)$ Maximum of two asset prices
  - $\max(S_1^T, S_2^T, S_3^T)$ Maximum of three asset prices
  - $\max(S_1^T, S_2^T, X) - X$ Call on the maximum of two asset prices
- Price projection of two assets:




```
stdex::mdarray<Node, stdex::dextents<int, 3>> grid_;
. . .
time_points_{ time_steps + 1 },
grid_{ time_points_, time_points_, time_points_ }
```

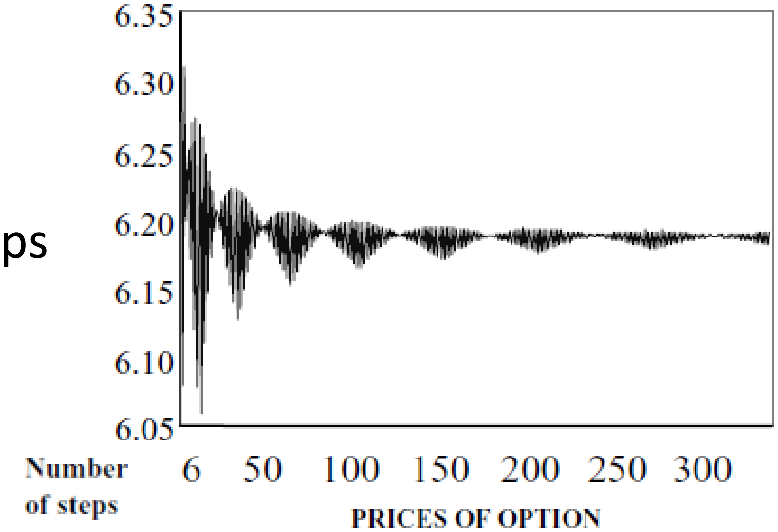
James, Figure 12.1, p 160

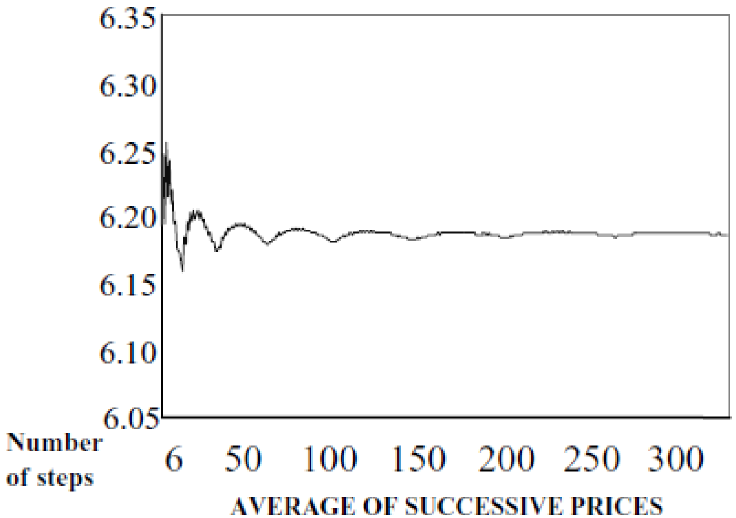
Convergence

- Need more time steps (and nodes) for the price to converge

- Results tend to oscillate

- Mitigate by taking average over n and $n+1$ steps
  - "Typical" range for n: 50-350
  - Depends upon the type and terms of the option





James, Figure 7.11, p 85

Convergence

- Previous example: American call with dividend

```
int start_iter{ 65 }, max_iterations{ 10 };

vector<double> avg_values;
double penult_val = 0.0, max_pts_val = 0.0, new_price = 0.0, last_price = 0.0;
for (int n = start_iter; n <= start_iter + max_iterations; ++n)
{
  auto cp = make_unique<CallPayoff>(strike);// cp = "call pointer"
  VanillaOption call(std::move(cp), time_to_exp);

  BinomialLatticePricer call_pricer{ std::move(call), mkt_vol, rf_rate, n, div_rate };

  new_price = call_pricer.calc_price(spot, OptType::American);
  cout << format("Time points = {}, Call option price = {}\n", n, new_price) ;

  if (last_price > 0)
  {
    avg_values.push_back((new_price + last_price) / 2.0);
  }

  last_price = new_price;
}

cout << "\nAverage values over (n, n + 1):\n";
for (double x : avg_values)
{
  cout << x << "\n";
}

cout << "\n\nOption value = last average value: " << avg_values.back() << "\n\n";
```

- Results:

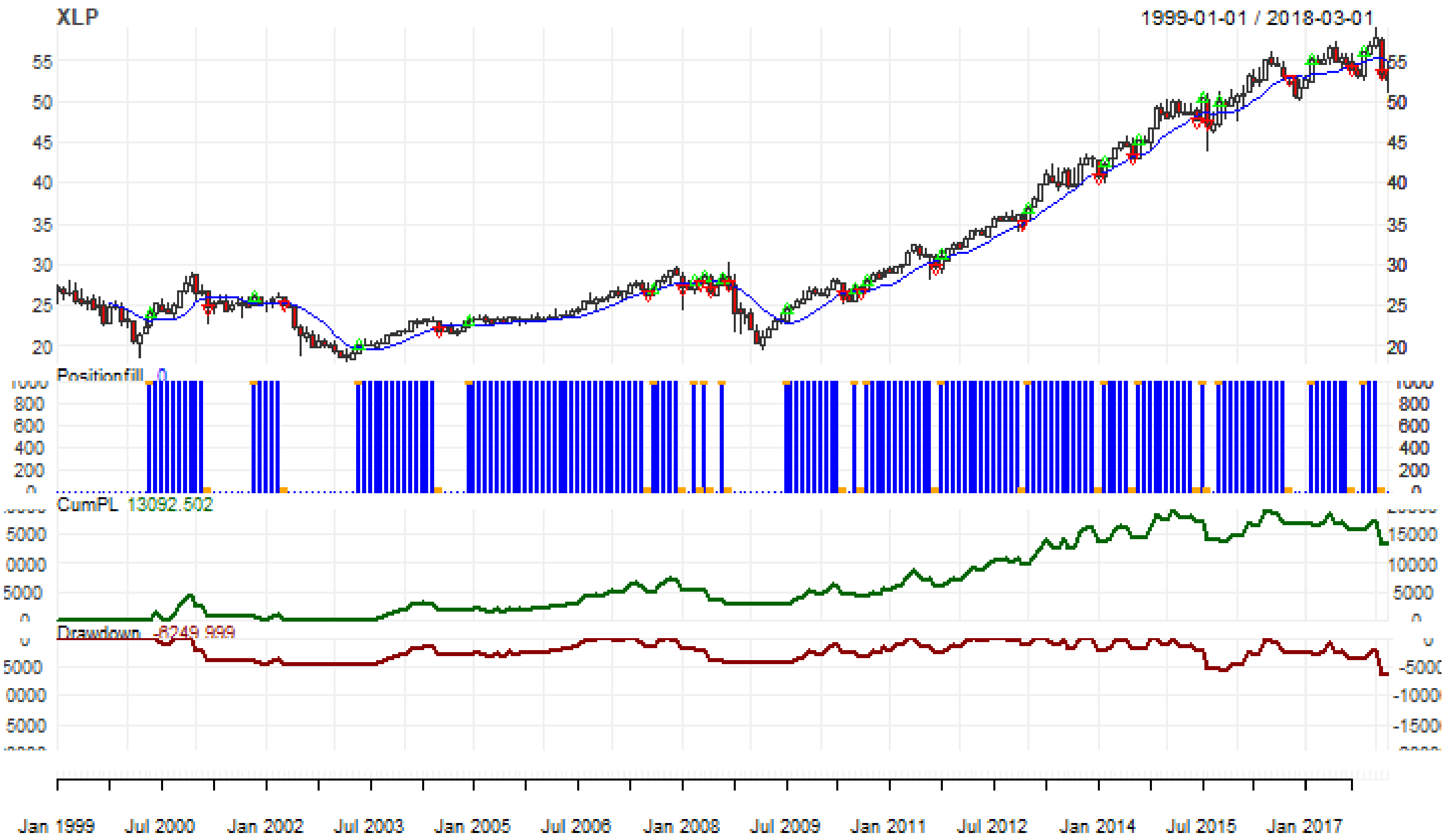
```
Price convergence of an American call option (with dividend):

American Call Option: strike = 30, rf rate = 0.06, vol = 0.2
time to exp (yr) = 1, ann div rate = 0.075, spot = 32

Time points = 65, Call option price = 3.2410748526085094
Time points = 66, Call option price = 3.2451088835792774
Time points = 67, Call option price = 3.240382233154151
Time points = 68, Call option price = 3.2452714284503634
Time points = 69, Call option price = 3.2397379581732584
Time points = 70, Call option price = 3.2453696939935845
Time points = 71, Call option price = 3.2391012017460277
Time points = 72, Call option price = 3.2454060060945635
Time points = 73, Call option price = 3.238464115494065
Time points = 74, Call option price = 3.24539330008411
Time points = 75, Call option price = 3.2378588597204407
```

```
Average values over (n, n + 1):
3.24309
3.24275
3.24283
3.2425
3.24255
3.24224
3.24225
3.24194
3.24193
3.24163

Option value = last average value: 3.24163
```

Wind-Up

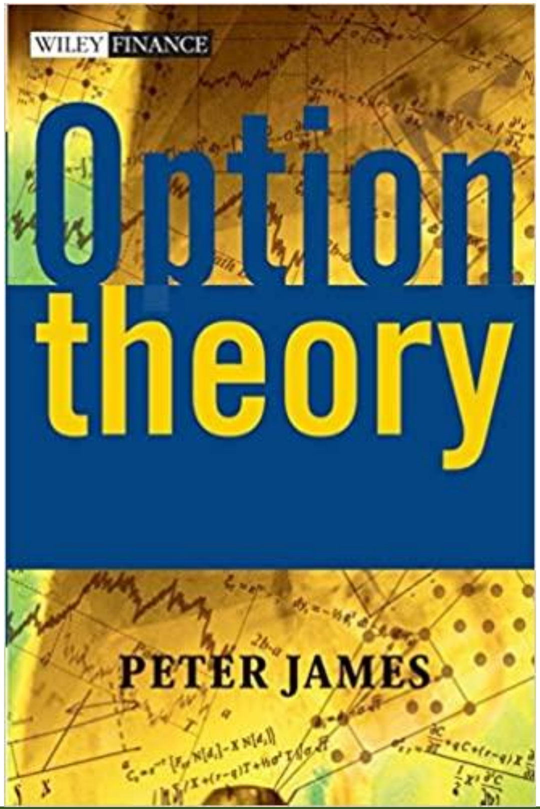
References

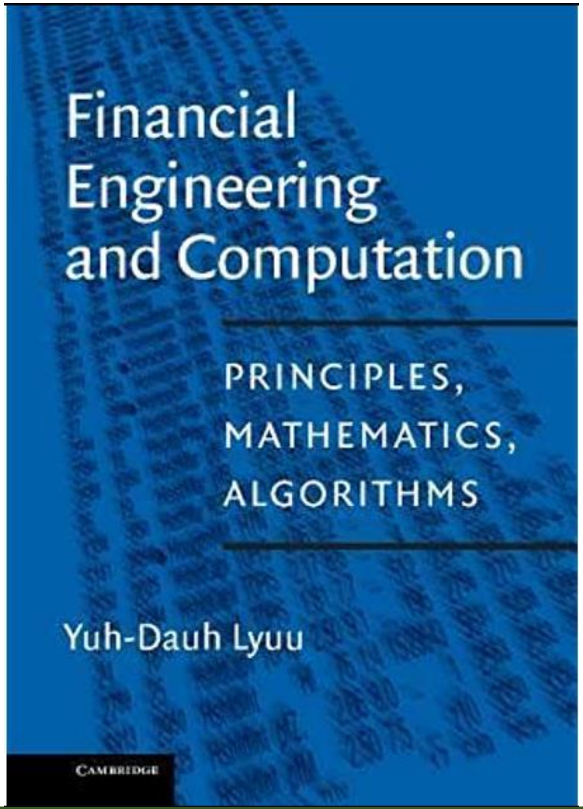
Summary

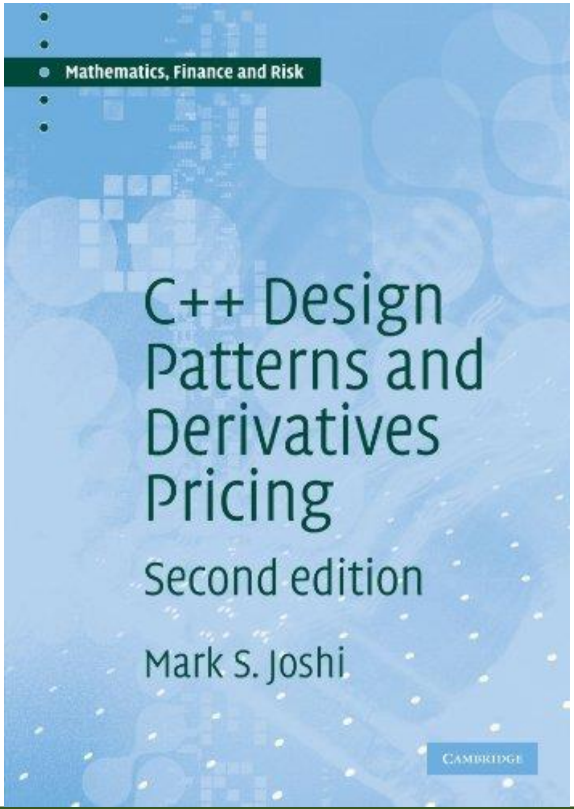


References

- Peter James, *Option Theory*, Wiley (2003)
  - Chapters 7, 9, and 12
- Yuh-Dauh Lyuu, *Financial Engineering and Computation*, Cambridge (2002)
  - Chapters 7-8
- Mark Joshi, C++ Design Patterns and Derivatives Pricing (2E), Cambridge (2008)
  - Chapter 4







References

- WG21 Proposal P1684
  - https://wg21.link/p1684

- Special thanks to Mark Hoemmen, co-author P1684

`mdarray`: An Owning Multidimensional Array Analog of `mdspan`

Document #:	P1684R4
Date:	2023-01-14
Project:	Programming Language C++ Library Evolution
Reply-to:	Christian Trott <crtrott@sandia.gov> Daisy Hollman <me@dsh.fyi> Mark Hoemmen <mark.hoemmen@gmail.com> Daniel Sunderland <dansunderland@gmail.com> Damien Lebrun-Grandie <lebrungrandt@ornl.gov>

Summary

- `mdspan` coming in C++23: Multidimensional view of 1-D container
- `mdarray` expected in C++26
  - Can use for case where data not known in advance
  - "Ready made" for binomial lattice models
    - American option pricing
    - Can evaluate optimal early exercise
- Generic multidimensional array
  - Can be extended to trinomial lattice
    - Stochastic interest rates
    - Stochastic volatility
  - Can be extended to three or more dimensions
    - Maximum of two asset prices
    - Maximum of three asset prices
    - Call/Put on the maximum of two asset prices
    - Other multi-asset exotic options…
  - **Node** object can be modified or extended for more complex payoffs

Contact/ Questions

- Contact:
  - `daniel (at) cppcon.org` (Student Program Advisor, CppCon)

  - https://www.linkedin.com/in/danielhanson/

- Thank You!

- Questions?