

[Help](#)

```

#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cstring>

using namespace std;

#include "math/ImportanceSampling_jl/src/BarrierOption.hpp"
#include "math/ImportanceSampling_jl/src/parser.hpp"

//
// true digital option (only at maturity time)
//
double DigitalFinalOption::payoff(const PnlMat *path_val)
{
    PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
    for (int j = 0 ; j < size ; j++)
    {
        if (GET(lower_barrier, j) != 0 &&
            GET(&final, j) <= GET(lower_barrier, j))
            return 0.;
        else if (GET(upper_barrier, j) != 0 &&
            GET(&final, j) >= GET(upper_barrier, j))
            return 0.;
    }

    return 1.0;
}

DigitalFinalOption::DigitalFinalOption() { }

DigitalFinalOption::DigitalFinalOption(const Param &P) :
    BaseOption(P)
{
    label = "digitalfinal";

    P.extract("upper barrier", upper_barrier, size, true);
    if (upper_barrier == NULL)
        upper_barrier = pnl_vect_create_from_zero(size);

```

```

    P.extract("lower barrier", lower_barrier, size, true);
    if (lower_barrier == NULL)
        lower_barrier = pnl_vect_create_from_zero(size);
}

void DigitalFinalOption::print() const
{
    cout << "**** Digitalfinal Option Characteristics ****" << endl;
    BaseOption::print();
    cout << " lower barrier : ";
    pnl_vect_print_asrow(lower_barrier);
    cout << " upper barrier : ";
    pnl_vect_print_asrow(upper_barrier);
}

DigitalFinalOption::~DigitalFinalOption()
{
    pnl_vect_free(&lower_barrier);
    pnl_vect_free(&upper_barrier);
}

//
// digital option at each time step
//
double DigitalOption::payoff(const PnlMat *path_val)
{
    int nTimeSteps = path_val->m - 1;
    for (int j = 0 ; j < size ; j++)
    {
        if (GET(lower_barrier, j) != 0 && GET(upper_barrier, j) != 0)
            for (int i = 0 ; i <= nTimeSteps ; i++)
            {
                if (MGET(path_val, i, j) <= GET(lower_barrier, j) ||
                    MGET(path_val, i, j) >= GET(upper_barrier, j))
                    return 0.;
            }

        else if (GET(lower_barrier, j) != 0 && GET(upper_barrier, j) == 0)
            for (int i = 0 ; i <= nTimeSteps ; i++)
            {

```

```

        if (MGET(path_val, i, j) <= GET(lower_barrier, j))
            return 0.;
    }

    else if (GET(lower_barrier, j) == 0 && GET(upper_barrier, j) != 0)
        for (int i = 0 ; i <= nTimeSteps ; i++)
            if (MGET(path_val, i, j) >= GET(upper_barrier, j))
                return 0.;

    }

    return 1.;
}

DigitalOption::DigitalOption() { }

DigitalOption::DigitalOption(const Param &P) :
    BaseOption(P)
{
    label = "digital";

    P.extract("upper barrier", upper_barrier, size, true);
    if (upper_barrier == NULL)
        upper_barrier = pnl_vect_create_from_zero(size);

    P.extract("lower barrier", lower_barrier, size, true);
    if (lower_barrier == NULL)
        lower_barrier = pnl_vect_create_from_zero(size);
}

void DigitalOption::print() const
{
    cout << "**** Digital Option Characteristics ****" << endl;
    BaseOption::print();
    cout << " lower barrier : ";
    pnl_vect_print_asrow(lower_barrier);
    cout << " upper barrier : ";
    pnl_vect_print_asrow(upper_barrier);
}

DigitalOption::~DigitalOption()

```

```

{
    pnl_vect_free(&lower_barrier);
    pnl_vect_free(&upper_barrier);
}

//
// true digital option (only at maturity time) combined with a put/call
// option
//
double DigitalBasketOption::payoff(const PnlMat *path_val)
{
    double sum = 0.;
    PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
    for (int j = 0 ; j < size ; j++)
    {
        if (GET(lower_barrier, j) != 0 &&
            GET(&final, j) <= GET(lower_barrier, j))
            return 0.;
        else if (GET(upper_barrier, j) != 0 &&
            GET(&final, j) >= GET(upper_barrier, j))
            return 0.;
    }

    sum = pnl_vect_scalar_prod(lambda, &final);
    return max(sum - K, 0.0);
}

DigitalBasketOption::DigitalBasketOption()
{
}

DigitalBasketOption::DigitalBasketOption(const Param &P)
    : BaseOption(P)
{
    label = "digitalbasket";

    P.extract("payoff coefficients", lambda, size);
    P.extract("upper barrier", upper_barrier, size, true);
    if (upper_barrier == NULL)
        upper_barrier = pnl_vect_create_from_zero(lambda->size);

    P.extract("lower barrier", lower_barrier, size, true);
}

```

```

    if (lower_barrier == NULL)
        lower_barrier = pnl_vect_create_from_zero(lambda->size);

    P.extract("strike", K);
}

void DigitalBasketOption::print() const
{
    cout << "**** Digitalbasket Option Characteristics ****" << endl;
    BaseOption::print();
    cout << " lower barrier : ";
    pnl_vect_print_asrow(lower_barrier);
    cout << " upper barrier : ";
    pnl_vect_print_asrow(upper_barrier);
    cout << " payoff coefficients : ";
    pnl_vect_print_asrow(lambda);
    cout << " strike : " << K << endl;
}

DigitalBasketOption::~DigitalBasketOption()
{
    pnl_vect_free(&lower_barrier);
    pnl_vect_free(&upper_barrier);
    pnl_vect_free(&lambda);
}

//
// Basket barrier
//

double BarrierOption::payoff(const PnlMat *path_val)
{
    int nTimeSteps = path_val->m - 1;
    for (int j = 0 ; j < size ; j++)
    {
        if (GET(lower_barrier, j) != 0 && GET(upper_barrier, j) != 0)
            for (int i = 0 ; i <= nTimeSteps ; i++)
            {
                if (MGET(path_val, i, j) <= GET(lower_barrier, j) ||
                    MGET(path_val, i, j) >= GET(upper_barrier, j))

```

```

        return 0.;
    }

    else if (GET(lower_barrier, j) != 0 && GET(upper_barrier, j) == 0)
        for (int i = 0 ; i <= nTimeSteps ; i++)
        {
            if (MGET(path_val, i, j) <= GET(lower_barrier, j))
                return 0.;
        }

    else if (GET(lower_barrier, j) == 0 && GET(upper_barrier, j) != 0)
        for (int i = 0 ; i <= nTimeSteps ; i++)
        {
            if (MGET(path_val, i, j) >= GET(upper_barrier, j))
                return 0.;
        }

    }

    PnlVect final = pnl_vect_wrap_mat_row(path_val, path_val->m - 1);
    double sum = pnl_vect_scalar_prod(lambda, &final);
    return max(sum - K, 0.0);
}

BarrierOption::BarrierOption()
{
}

BarrierOption::BarrierOption(const Param &P) :
    BaseOption(P)
{
    label = "barrier";

    P.extract("payoff coefficients", lambda, size);
    P.extract("strike", K);
    P.extract("upper barrier", upper_barrier, size, true);
    if (upper_barrier == NULL)
        upper_barrier = pnl_vect_create_from_zero(lambda->size);

    P.extract("lower barrier", lower_barrier, size, true);
    if (lower_barrier == NULL)

```

```
        lower_barrier = pnl_vect_create_from_zero(lambda->size);
    }

void BarrierOption::print() const
{
    cout << "**** Barrier Option Characteristics ****" << endl;
    BaseOption::print();
    cout << " lower barrier : ";
    pnl_vect_print_asrow(lower_barrier);
    cout << " upper barrier : ";
    pnl_vect_print_asrow(upper_barrier);
    cout << " payoff coefficients : ";
    pnl_vect_print_asrow(lambda);
    cout << " strike : " << K << endl;
}

BarrierOption::~BarrierOption()
{
    pnl_vect_free(&lower_barrier);
    pnl_vect_free(&upper_barrier);
    pnl_vect_free(&lambda);
}
```