



b) It holds

$$\begin{aligned}(D^+D^+u)(x) &= D^+ \left( \frac{1}{h}(u(x+h) - u(x)) \right) = \frac{1}{h}((D^+u)(x+h) - (D^+u)(x)) \\ &= \frac{1}{h^2}(u(x+2h) - 2u(x+h) + u(x)).\end{aligned}$$

Taylor's formula yields

$$\begin{aligned}u(x+h) &= u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x) + O(h^4), \\ u(x+2h) &= u(x) + 2hu'(x) + \frac{4h^2}{2}u''(x) + \frac{8h^3}{6}u'''(x) + O(h^4).\end{aligned}$$

Hence

$$(D^+D^+u)(x) = \frac{1}{h^2}(h^2u''(x) - h^3u'''(x) + O(h^4)) = u''(x) + O(h),$$

so that the approximation order is 1 in contrast to the order 2 for  $(D^-D^+u)(x)$  (see Lemma 4.3.1).

## Exercise 2: Recursion Method

Solve the one-dimensional Poisson problem

$$\begin{cases} -u''(x) = f(x) & , x \in (0, 1), \\ u(0) = u(1) = 0. \end{cases}$$

using Finite Differences for the following right hand sides. Plot the convergence rates and compare them with the theoretical results for the convergence order.

a)  $f(x) = 9\pi^2 \cos(3\pi(x + \frac{1}{2}))$ . The exact solution is  $u(x) = \cos(3\pi(x + \frac{1}{2}))$ .

b)  $f(x) = \begin{cases} 1, & x \in (0, \frac{1}{2}], \\ -1, & x \in (\frac{1}{2}, 1). \end{cases}$  The exact solution is  $u(x) = \begin{cases} -\frac{1}{2}x^2 + \frac{1}{4}x, & x \in (0, \frac{1}{2}], \\ \frac{1}{2}x^2 - \frac{3}{4}x + \frac{1}{4}, & x \in (\frac{1}{2}, 1). \end{cases}$

### Solution:

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

double rhsA(double x){
    return 9.*M_PI*M_PI*cos(3.*M_PI*(x+0.5));
}

double solA(double x){
    return cos(3.*M_PI*(x+0.5));
}

double rhsB1(double x){
    return 1.;
}

double solB1(double x){
    return -(0.5*x*x - 0.25*x);
}
```

```

}

double rhsB2(double x){
    return -1.;
}

double solB2(double x){
    return -(-0.5*x*x + 0.75*x - 0.25);
}

int main(){

    for(int N = 1<<4; N <= 1<<20; N *=2){
        // Variables
        double h = 1./N;
        vector<double> u(N+1); // u_0, ..., u_N
        vector<double> f(N+1);
        vector<double> sol(N+1);

        double r = -1/(h*h);
        double d = 2/(h*h);
        double t = -1/(h*h);

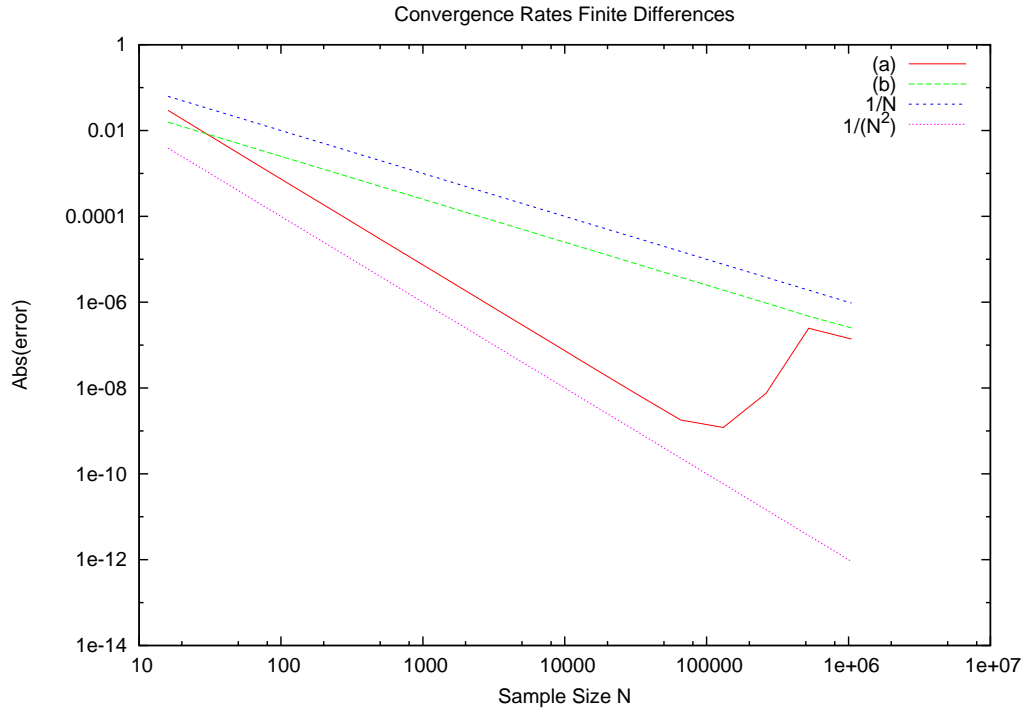
        vector<double> alpha(N-1);
        vector<double> gamma(N-2);
        vector<double> v(N-1); // v_1, ..., v_{N-1}

        // Initialization f
        for(int i = 0; i < N+1; i++){
            if(i*h <= 0.5){
                f[i] = rhsB1(i*h);
                sol[i] = solB1(i*h);
            }
            else{
                f[i] = rhsB2(i*h);
                sol[i] = solB2(i*h);
            }
        }
        // Initialization alpha, gamma
        alpha[0] = d;
        for(int i = 0; i < N-2; i++){
            gamma[i] = t / alpha[i];
            alpha[i+1] = d - gamma[i]*r;
        }

        // Forward Recursion
        v[0] = f[1] / alpha[0];
        for(int i = 1; i < N-1; i++){
            v[i] = 1./alpha[i] * (f[i+1] - r*v[i-1]);
        }
        // Backward Recursion
        u[N] = 0;
        u[0] = 0;
        u[N-1] = v[N-2];
        for(int i = N-3; i >= 0; i--){
            u[i+1] = v[i] - gamma[i]*u[i+2];
        }
        // Calculate Error
        double superr = 0;
        for(int i = 0; i < N+1; i++){
            if(abs(u[i]-sol[i]) > superr){
                superr = abs(u[i]-sol[i]);
            }
        }
        cout << N << " " << superr << endl;
    }

    return 0;
}

```



### Exercise 3: Explicit Euler Method

To solve the Black-Scholes Equation with Finite Differences, one can transform it to a heat equation

$$\begin{cases} \frac{\partial}{\partial \tau} u(x, \tau) - \frac{\partial^2}{\partial x^2} u(x, \tau) = 0 \text{ in } (x_{min}, x_{max}) \times (0, T) \\ u(x, 0) = u_0(x), \quad x \in (x_{min}, x_{max}) \\ u(x_{min}, t) = u_1(t), u(x_{max}, t) = u_2(t), t \in (0, T), \end{cases}$$

using

$$S = Ke^x, \quad t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, \quad V(S, t) = V(Ke^x, T - \frac{\tau}{\frac{1}{2}\sigma^2}) =: v(x, \tau)$$

$$\text{and } v(x, \tau) := K \exp \left\{ -\frac{1}{2}(q-1)x - \left( \frac{1}{4}(q-1)^2 + q \right) \tau \right\} u(x, \tau), \quad q := \frac{2r}{\sigma^2}.$$

Note that the change of variable for  $t$  transforms the original backward PDE into a forward one. The initial condition for a European Put is then

$$u_0(x) = \max \left\{ e^{\frac{x}{2}(q-1)} - e^{\frac{x}{2}(q+1)}, 0 \right\}.$$

The boundary conditions of a European Put are transformed to

$$\begin{aligned} u(x, \tau) \rightarrow u_1(x, \tau) &:= \exp \left\{ \frac{1}{2}(q-1)x + \frac{1}{4}(q-1)^2 \tau \right\} \quad \text{for } x \rightarrow -\infty, \\ u(x, \tau) \rightarrow u_2(x, \tau) &:= 0 \quad \text{for } x \rightarrow \infty. \end{aligned}$$

However, instead of the infinite interval  $x \in (-\infty, \infty)$ , the truncated domain  $x \in (x_{min}, x_{max})$  is used, where the choice of  $x_{min}$ ,  $x_{max}$  depends on assumptions on the range of  $S$ .

Implement the Explicit Euler method to solve the Black Scholes PDE using the above transformation for a European Put with  $K = 12$ ,  $r = 0.04$ ,  $\sigma = 0.4$  and  $S \in (0.00001, 20)$ . Plot the solution  $V(S_i, t_j)$ ,  $i = 1, \dots, N$ ,  $j = 1, \dots, M$ .

### Solution:

```

#include <iostream>
#include <cmath>

using namespace std;

int main(){

    // Problem Definition:
    double K = 12;
    double sigma = 0.4;
    double r = 0.04;
    double T = 1;

    // Grid Sizes:
    // Time:
    int M = 50;
    double deltaTau = 0.5*sigma*sigma*T / (double)M;
    // Space:
    int N = 200;
    // Calculate V(S, t) for S in (0, 20]
    double xmin = log(0.00001/K);
    double xmax = log(20/K);
    double h = (xmax-xmin) / (double)N;

    // Initialization:
    double u_k[N+1];
    double q = (2*r)/(sigma*sigma);
    double x;
    for(int i = 0; i <= N; i++){
        x = xmin + i*h;
        u_k[i] = max(exp(0.5*x*(q-1)) - exp(0.5*x*(q+1)) , 0.0);
        //Print: T S_T V(S_T)
        cout << T << " " << K*exp(x) << " " << K*exp(-0.5*(q-1)*x)*u_k[i] << endl;
    }
    cout << endl;

    // Explicit Euler:
    double gamma = deltaTau/(h*h);
    double tmp[N+1];
    double tau;
    for(int j = 1; j <= M; j++){
        tau = j*deltaTau;
        // Boundary Conditions:
        tmp[0] = exp(0.5*(q-1)*xmin + 0.25*(q-1)*(q-1)*tau);
        tmp[N] = 0;
        // Euler:
        for(int i = 1; i < N; i++){
            tmp[i] = (1 - 2*gamma)*u_k[i] + gamma*(u_k[i-1] + u_k[i+1]);
        }
        // Update y_k:
        double realtime = T - tau/(0.5*sigma*sigma);
        for(int i = 0; i <= N; i++){
            u_k[i] = tmp[i];
            //Print: t S_t V(S_t)
            x = xmin + i*h;

```

```

    cout << realtime << " " << K*exp(x) << " "
        << K*exp(-0.5*(q-1)*x - (0.25*(q-1)*(q-1) + q)*tau)*u_k[i] << endl;
    }
    cout << endl;
}
return 0;
}

```

European Put:  $K = 12$ ,  $r = 0.04$ ,  $\sigma = 0.4$

