

## Numerical Finance Reading Course

### Sheet 10 - Sample Solution

#### Exercise 1: Characterization Theorem (Remark 6.3.5 (ii))

Prove the Characterization Theorem:

$$\begin{aligned} J(v) &:= \frac{1}{2}a(v, v) - (f, v)_0 \text{ has its minimum in } u. \\ \iff a(u, v) &= (f, v) \quad \text{for all } v \in H_0^1(\Omega). \end{aligned}$$

**Solution:**

“ $\Rightarrow$ ” Let  $u$  be a minimum point of  $J(v)$ . Then it holds for all  $\epsilon \neq 0$ ,  $\epsilon \in \mathbb{R}$  and all  $v \in H_0^1(\Omega)$ :

$$\begin{aligned} J(u + \epsilon v) &= \frac{1}{2}a(u + \epsilon v, u + \epsilon v) - (f, u + \epsilon v)_0 \geq J(u) = \frac{1}{2}a(u, u) - (f, u)_0 \\ \iff J(u + \epsilon v) - J(u) &= \frac{1}{2}(a(u + \epsilon v, u + \epsilon v) - a(u, u)) - (f, u + \epsilon v)_0 + (f, u)_0 \geq 0 \\ \iff \frac{1}{2}(a(u, u) + 2\epsilon a(u, v) + \epsilon^2 a(v, v) - a(u, u)) - (f, u)_0 - \epsilon f(u, v)_0 + (f, u)_0 &\geq 0 \\ \iff \epsilon(a(u, v) - f(u, v)_0) + \frac{\epsilon^2}{2}a(v, v) &\geq 0 \end{aligned}$$

For  $\epsilon > 0$ , we thus have

$$(a(u, v) - f(u, v)_0) + \frac{\epsilon^2}{2}a(v, v) \geq 0,$$

so that for  $\epsilon \rightarrow 0+$

$$a(u, v) - f(u, v)_0 \geq 0.$$

Similarly, for  $\epsilon \rightarrow 0-$

$$a(u, v) - f(u, v)_0 \leq 0,$$

so that  $a(u, v) - f(u, v)_0 = 0$  for all  $v$ .

“ $\Leftarrow$ ” Let

$$a(u, v) - f(u, v)_0 = 0 \quad \text{for all } v \in H_0^1(\Omega).$$

Then for all  $\epsilon \neq 0$ ,  $\epsilon \in \mathbb{R}$  and any  $v \in H_0^1(\Omega)$ :

$$\begin{aligned} J(u + \epsilon v) - J(u) &= \epsilon(a(u, v) - f(u, v)_0) + \frac{\epsilon^2}{2}a(v, v) \\ &= \frac{\epsilon^2}{2}a(v, v) > 0, \end{aligned}$$

so that  $u$  is indeed a minimum point of  $J(v)$ .

## Exercise 2: Solving the Poisson Equation with FEM

Implement the Finite Element method for the problem given in Exercise 2 on Sheet 9. Plot the approximation and the exact solution for one  $N$ . Obtain the convergence rate by plotting the approximation error for different  $N$ . You can use the supremum error

$$\|u - u_h\|_\infty = \sup_{x \in [0,1]} |u(x) - u_h(x)|,$$

which can be approximated using a fine grid with  $M \gg N$  intervals.

### Solution:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <flens/flens.h>

using namespace std;
using namespace flens;

typedef GeMatrix<FullStorage<double, ColMajor> > GEMatrix;
typedef DenseVector<Array<double> > DEVector;

double sol(double x){
    if(x < 0.5){
        return -0.5*x*x + 0.25*x;
    }
    else{
        return -(-0.5*x*x + 0.75*x - 0.25);
    }
}

int main(){

    int N;
    double h;

    for(int i = 2; i <= 7; i++){
        N = (1 << i)+1;
        h = 1./(N+1);

        // Assemble Stiffness Matrix
        GEMatrix A(N,N);
        for(int j=2; j <= N-1; j++){
            A(j,j) = 2./h;
            A(j,j+1) = -1./h;
            A(j,j-1) = -1./h;
        }
        A(1,1) = 2./h;
        A(N,N) = 2./h;
        A(1,2) = -1./h;
        A(N,N-1) = -1./h;

        // Assemble Right-Hand Side
        DEVector b(N);
        for(int j=1; j < (N+1)/2; j++){
            b(j) = h;
        }
        b((N+1)/2) = 0;
        for(int j=(N+1)/2 + 1; j <= N; j++){
            b(j) = -h;
        }

        // Calculate Solution with cg-method
```

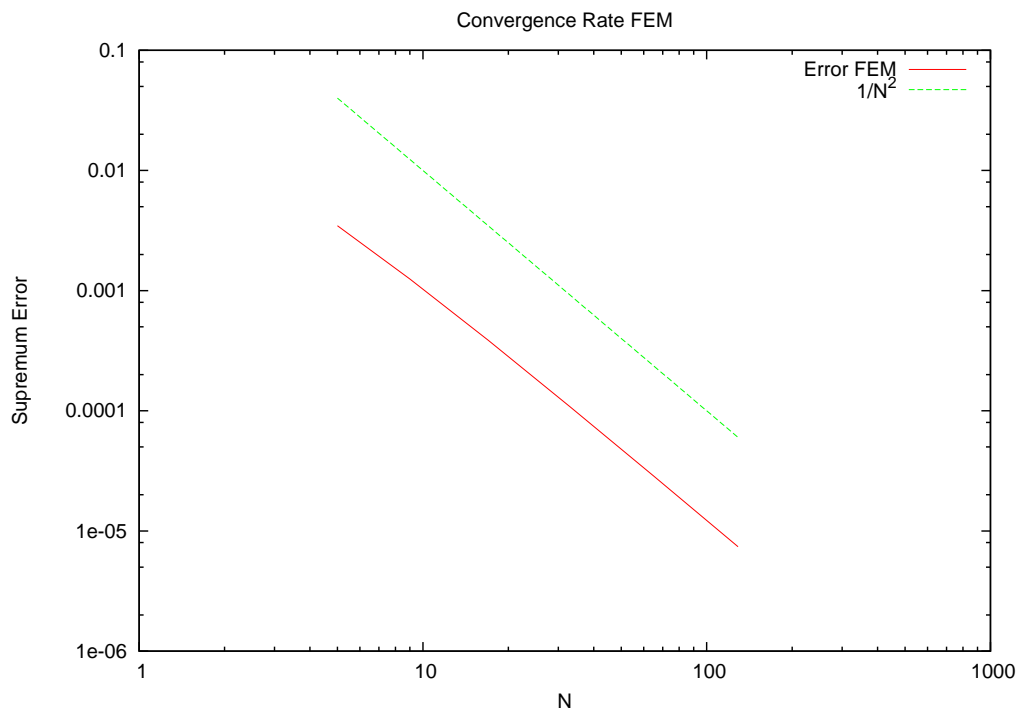
```

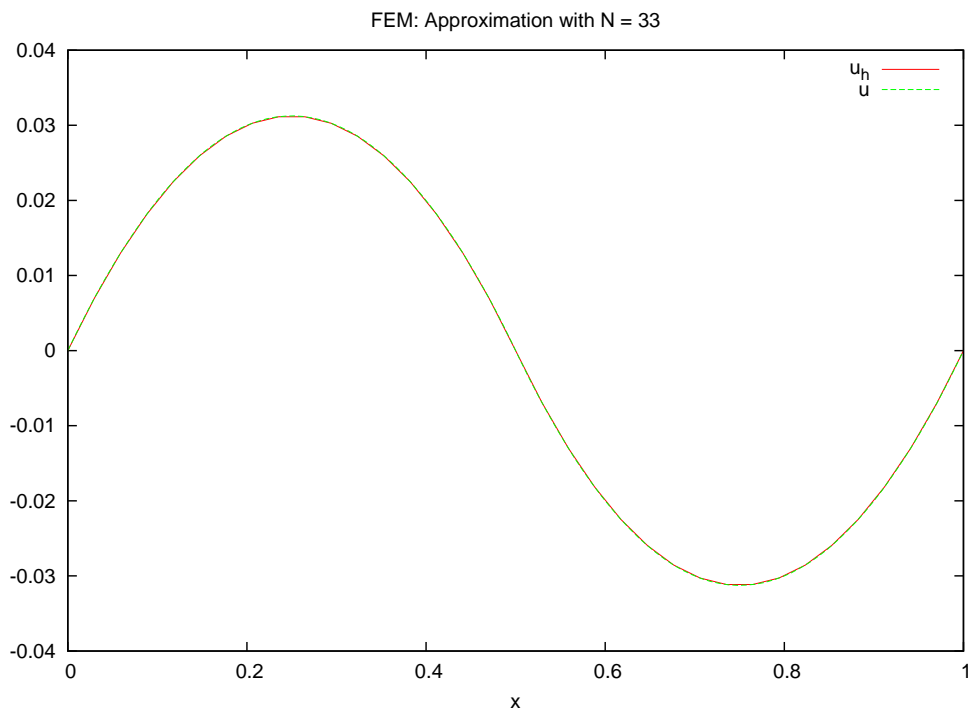
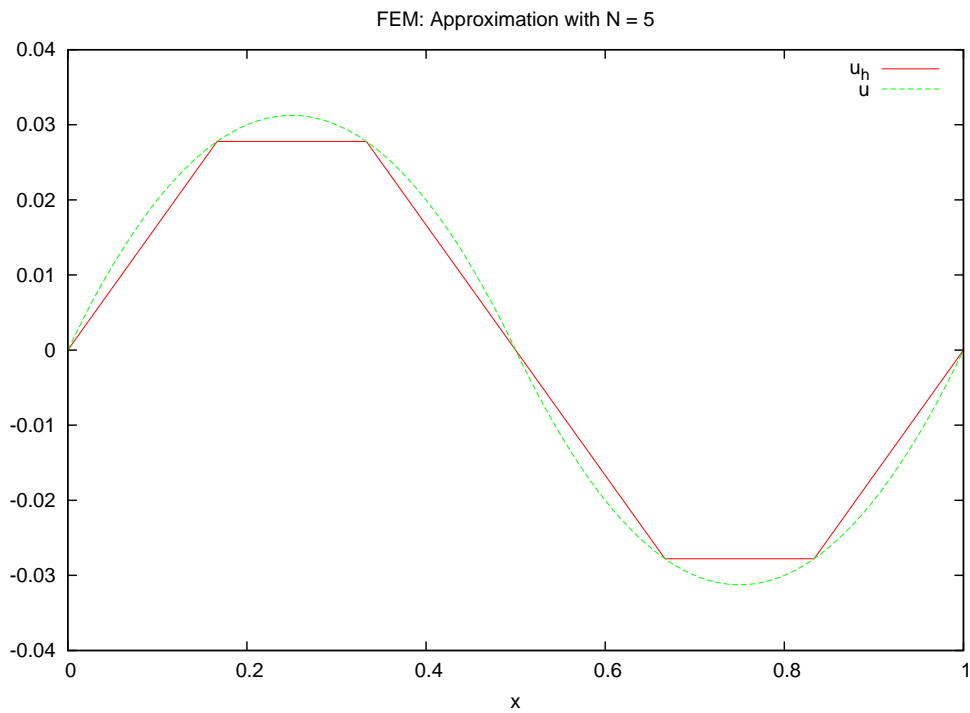
DEVector u(N);
int iterations = cg(A, u, b);
cerr << "Iterations: " << iterations << endl;

// Calculate error
double err = 0;
double diff = 0;
double u_interpol;
int M = 100;
double h_M = h/M;
for(int l = 0; l <= M; l++){
    u_interpol = l*h_M*u(1)/h;
    diff = abs(u_interpol - sol(l*h_M));
    if(diff > err){ err = diff;}
}
for(int k = 1; k < N; k++){
    for(int l = 0; l <= M; l++){
        u_interpol = u(k) + l*h_M*(u(k+1)-u(k))/h;
        diff = abs(u_interpol - sol(k*h + l*h_M));
        if(diff > err){ err = diff;}
    }
}
for(int l = 0; l <= M; l++){
    u_interpol = u(N) + l*h_M*(-u(N))/h;
    diff = abs(u_interpol - sol(N*h + l*h_M));
    if(diff > err){ err = diff;}
}
cout << N << " " << err << endl;
}

return 0;
}

```





### Exercise 3: Simulation of a Cox-Ingersoll-Ross process

A Cox-Ingersoll-Ross (CIR) process has the form

$$dr(t) = \alpha(b - r(t))dt + \sigma\sqrt{r(t)}dW_t$$

for some  $\alpha, b > 0$  and is often used to model short rates.

Compare both methods by calculating the empirical density of  $r(T)$  for  $T = 1$ , i.e. repeatedly simulating  $r(T)$  and plotting a histogram, and compare it to the exact density (you can use e.g. R or MATLAB for the evaluation of your data). Use for example the parameters  $\alpha = 0.2$ ,  $\sigma = 0.1$ ,  $b = 0.05$  and  $r(0) = 0.04$ .

## Solution:

### a) Euler-Maruyama:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <boost/random.hpp>
#include <time.h>

using namespace std;
using namespace boost;

int main(int argc, char* argv[]){

    if(argc != 3){
        cerr << "Usage: " << argv[0] << " N(2^Ntime_steps) M(2^Mpaths)" << endl;
        return 0;
    }

    double T = 1;
    double r_0 = 0.04;
    double sigma = 0.1;
    double alpha = 0.2;
    double b = 0.05;

    int Nexp = atoi(argv[1]);
    int N = 1 << Nexp;

    double deltaT = T/(double)N;
    double sqrtdeltaT = sqrt(T/(double)N);

    vector<double> r(N+1);

    int Mexp = atoi(argv[2]);

    mt19937 generator;
    generator.seed(time(0));
    normal_distribution<> norm_dist;
    variate_generator<mt19937&,normal_distribution<> > norm_rnd(generator, norm_dist);

    // .. generate M paths
    for(int n = 1; n <= 1<<Mexp; n++){
        r[0] = r_0;
        // Euler-Maruyama scheme
        for(int i = 1; i <= N; i++){
            r[i] = r[i-1] + alpha*(b - r[i-1])*deltaT + sigma*sqrt((r[i-1]>0 ? r[i-1] : 0 ))
                *sqrtdeltaT*norm_rnd();
        }

        cout << r[N] << endl;
    }

    return 0;
}
```

### b) Exact Simulation:

```
#include <iostream>
```

```

#include <cmath>
#include <vector>
#include <boost/random.hpp>
#include <time.h>

using namespace std;
using namespace boost;

double chisq(double a, double b, double m, int d ,
             variate_generator<mt19937&,uniform_real<double> > &uni){
    double u1,u2,v;
    while(true){
        u1 = uni();
        u2 = uni();
        v = b*u2/u1;
        if(m*u1 - d + v + 1./v <= 0){
            return 2*a*v;
        }
        if(m*log(u1) - log(v) + v - 1 <= 0){
            return 2*a*v;
        }
    }
}

int main(int argc, char* argv[]){

    if(argc != 2){
        cerr << "Usage: _" << argv[0] << "_M_(2^M_paths)" << endl;
        return 0;
    }

    double T = 1;
    double r_0 = 0.04;
    double sigma = 0.1;
    double alpha = 0.2;
    double b = 0.05;

    int Mexp = atoi(argv[1]);

    mt19937 generator1;
    mt19937 generator2;
    generator1.seed(time(0));
    generator2.seed(time(0));
    normal_distribution<> norm_dist;
    variate_generator<mt19937&,normal_distribution<> > norm_rnd(generator1, norm_dist);
    uniform_real<double> uni_dist(0,1);
    variate_generator<mt19937&,uniform_real<double> > uni_rnd(generator2, uni_dist);

    // Variables for Non-Central Chi-Square Distribution
    int d = 4*b*alpha/(sigma*sigma);
    double c = sigma*sigma*(1.-exp(-alpha))/(4.*alpha);
    double lambda = exp(-alpha)/c;

    // Variables for Random Generator Chi-Square => chisq(a, b_, m, d_, uni_rnd())
    double a_ = (d-1.)/2.;
    double a = a_ - 1.;
    double b_ = (a_ - 1./(6.*a_))/a;
    double m = 2./a;
    double d_ = m+2.;

    // .. generate M paths
    for(int n = 1; n <= 1<<Mexp; n++){
        double Z = norm_rnd();
        cout << c*((Z + sqrt(r_0*lambda))*(Z + sqrt(r_0*lambda))
                + chisq(a, b_, m, d_, uni_rnd())) << endl;
    }

    return 0;
}

```

