1 Solution (Homework 2)

1. We have

$$\begin{aligned}\Phi(y,x_i) &= e^{-(y-x_i)^2} = e^{-[y-x_*-(x_i-x_*)]^2} = e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}e^{2(x_i-x_*)(y-x_*)} \\ &= e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}\sum_{m=0}^{\infty}\frac{2^m(x_i-x_*)^m(y-x_*)^m}{m!} \\ &= e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}\left[\sum_{m=0}^{p-1}\frac{2^m(x_i-x_*)^m(y-x_*)^m}{m!}+error_p\right]\end{aligned}$$

$$|error_p| \leqslant \frac{|y-x_*|^p}{p!}\sup_{0\leqslant y\leqslant 1}\left|\frac{\partial^p e^{2(x_i-x_*)(y-x_*)}}{\partial y^p}\right| = \frac{2^p|y-x_*|^p|x_i-x_*|^p}{p!}\sup_{0\leqslant y\leqslant 1}e^{2(x_i-x_*)(y-x_*)}.$$

Let us select $x_* = 0.5$, then

$$|y-x_*|^p \leqslant \frac{1}{2^p}, \quad |x_i-x_*|^p \leqslant \frac{1}{2^p}, \quad \sup_{0\leqslant y\leqslant 1}e^{2(x_i-x_*)(y-x_*)} \leqslant e^{2\cdot 0.5\cdot 0.5} = e^{0.5}.$$

$$|error_p| \leqslant \frac{2^p\sqrt{e}}{2^p 2^p p!} = \frac{1}{2^p p!}.$$

Since

$$e^{-(y-x_*)^2} \leqslant 1, \quad e^{-(x_i-x_*)^2} \leqslant 1,$$

The absolute error in computation of $\Phi(y,x_i)$ does not exceed $|error_p|$.

For fast summation we have

$$\begin{aligned}v_j &= \sum_{i=1}^{N}\Phi(y_j,x_i)u_i = \sum_{i=1}^{N}e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}\left[\sum_{m=0}^{p-1}\frac{2^m(x_i-x_*)^m(y-x_*)^m}{m!}+error_p\right]u_i \\ &= \sum_{m=0}^{p-1}\frac{2^m(y_j-x_*)^m}{m!}e^{-(y_j-x_*)^2}\sum_{i=1}^{N}e^{-(x_i-x_*)^2}(x_i-x_*)^m u_i + \sum_{i=1}^{N}e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}u_i error_p.\end{aligned}$$

The absolute error in fast computation of v_j is

$$\begin{aligned}\epsilon &= \left|v_j - \sum_{m=0}^{p-1}\frac{2^m(y_j-x_*)^m}{m!}e^{-(y_j-x_*)^2}\sum_{i=1}^{N}e^{-(x_i-x_*)^2}(x_i-x_*)^m u_i\right| \\ &= \left|\sum_{i=1}^{N}e^{-(y-x_*)^2}e^{-(x_i-x_*)^2}u_i error_p\right| \leqslant \sum_{i=1}^{N}\left|e^{-(y-x_*)^2}\right|\left|e^{-(x_i-x_*)^2}\right||u_i||error_p| \\ &\leqslant \sum_{i=1}^{N}|error_p| \leqslant \frac{N\sqrt{e}}{2^p p!}.\end{aligned}$$

The truncation number can be found as the smallest integer that satisfies inequality

$$2^p p! > \frac{N\sqrt{e}}{\epsilon}.$$

We have the following values of $2^p p! = 2\cdot 4\cdot \ldots \cdot (2p)$

p	1	2	3	4	5	6	7	8	9	10	11
$2^p p!$	2	8	48	384	3,840	46,080	645,120	$>10^7$	$>1.8{\cdot}10^8$	$>3.7{\cdot}10^9$	$>8.1{\cdot}10^{10}$

Since

$$\sqrt{e} < 1.7,$$

a truncation number of $p = 11$ is sufficient for computations with $\epsilon = 10^{-6}$ and $N \leqslant 10^4$. Indeed, for $p = 11$, we have

$$\epsilon \leqslant \frac{N\sqrt{e}}{2^p p!} < \frac{1.7 \cdot 10^4}{8.1 \cdot 10^{10}} < 2.1 \cdot 10^{-7} < 10^{-6}.$$

For computations with $N = 10^3$ the truncation number can be $p = 10$,since

$$\epsilon \leqslant \frac{N\sqrt{e}}{2^p p!} < \frac{1.7 \cdot 10^3}{3.7 \cdot 10^9} < 5 \cdot 10^{-7} < 10^{-6},$$

and for computations with $N = 10^2$ the truncation number can be $p = 9$, since

$$\epsilon \leqslant \frac{N\sqrt{e}}{2^p p!} < \frac{1.7 \cdot 10^2}{1.8 \cdot 10^8} < 10^{-6}.$$

The formula for fast computations will then be

$$\begin{aligned} v_j &= e^{-(y_j - x_*)^2} \sum_{m=0}^{p-1} c_m \left(y_j - x_*\right)^m, \quad j = 1, ..., M. \\ c_m &= \frac{2^m}{m!} \sum_{i=1}^{N} e^{-(x_i - x_*)^2} \left(x_i - x_*\right)^m u_i, \quad x_* = 0.5. \end{aligned}$$

2. Matlab code is attached.

3. See Figure 1.

4. See Figure 2. This graph shows the performance of two programs. Squares and circles shows the results of computations and lines show the linear (for "Fast") and quadratic (for "Standard") dependences in the loglog coordinates. See continuous lines on the previous graph. They are close to the computational results. So the standard method behaves as $O(N^2)$ algorithm, while the "Fast" method behaves as

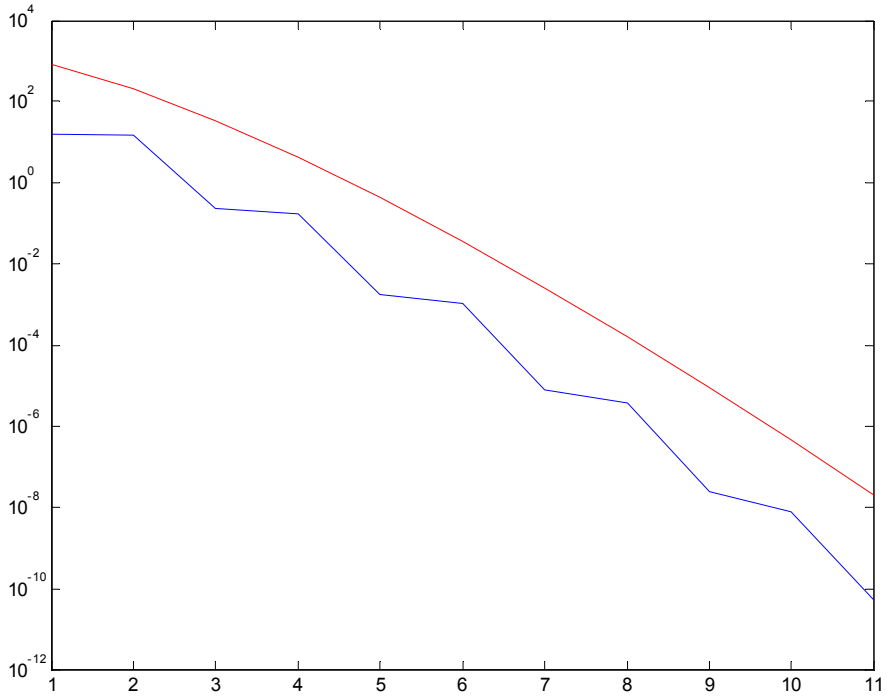
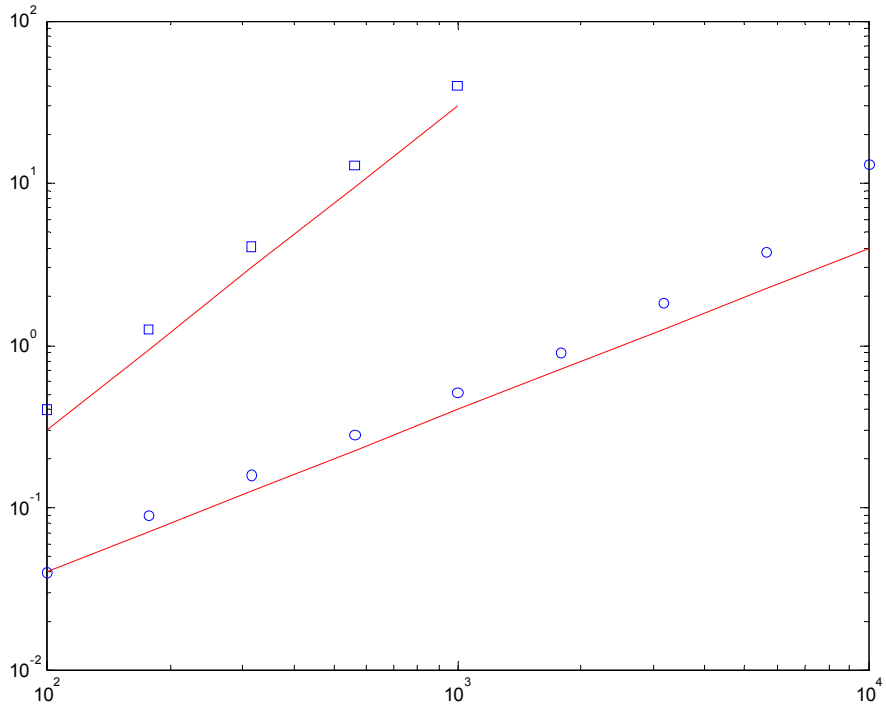


Figure 1:



$O(N)$.

5. See Figure 3.

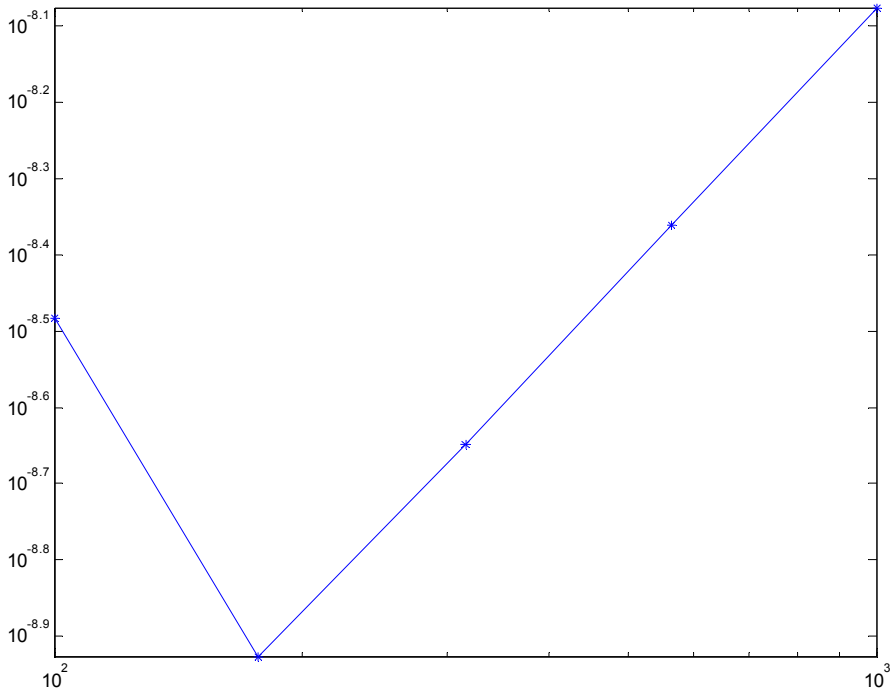


Figure 2:

```
%homework 2
clear all;
x_star=0.5;
% generate a graph for varying truncation number p
N=1000;
M=2*N;
X=rand(1,N);
Y=rand(1,2*N);
U=rand(1,N);
[time_standard,Vs]=h2_standard(X,Y,U);

for p=1:11
    [time_fast,Vm]=h2_fast(X,Y,U,p,x_star);
    err_actual(p)=max(abs(Vm-Vs));
    err_theory(p)=Get_error(N,p);
end;

figure;
semilogy([1:11],err_actual,'b');
hold on;
semilogy([1:11],err_theory,'r');

% generate a graph for varying number of sources;

npoint=9;
NN=floor(logspace(2,4,npoint));
epsilon=1e-6;

for i=1:npoint
    N=NN(i);
    M=2*N;
    X=rand(1,N);
    Y=rand(1,M);
    U=rand(1,N);

    p=Get_p(N,epsilon);
    [tm(i),Vm]=h2_fast(X,Y,U,p,x_star);

    if N<=1000
        [ts(i),Vs]=h2_standard(X,Y,U);
        Nst(i)=N;
        Nst2(i)=N*N;
        err(i)=max(abs(Vm-Vs));
    end;
end;

figure;
loglog(NN,tm,'o');
hold on;
loglog(NN,0.0004*NN,'r');
loglog(Nst,ts,'s');
loglog(Nst,0.00003*Nst2,'r');
figure;
loglog(Nst,err,'-*');
%=====================================================================
function epsilon=Get_error(N,p)
% computes theoretical error bound for given N and truncation number
```

```
% (sources are distributed on [0,1], x_star=0.5)

epsilon = N*exp(0.5)/(2^p*factorial(p));
%=========================================================================
function p=Get_p(N,epsilon)

% computes theoretical truncation number for given N and epsilon
% (sources are distributed on [0,1], x_star=0.5)

Bound = N*exp(0.5)/epsilon;
num = 1;
p=0;
while num < Bound
    p=p+1;
    num=num*2*p;
end;
%=========================================================================
function [time,V]=h2_standard(X,Y,U)

%standard matrix vector multiplication
%homework 1

t=cputime;

N=length(X);
M=length(Y);

for j=1:M
    V(j)=0;
    for i=1:N,
        V(j)=V(j)+U(i)*exp(-(Y(j)-X(i))^2);
    end;
end;

time=cputime-t;
%=========================================================================
function [time,V]=h2_fast(X,Y,U,p,x_star)

%middleman matrix vector multiplication
%homework 1

t=cputime;

N=length(X);
M=length(Y);

% find coefficients c(m)

factor=1;

for m=0:p-1
    sum=0;
    for i=1:N
        arg=X(i)-x_star;
        sum=sum+U(i)*exp(-arg*arg)*arg^m;
    end;
    c(m+1)=factor*sum;
```

```
    factor=2*factor/(m+1);
end;

% final summation

for j=1:M
    sum=0;
    arg=Y(j)-x_star;
    for m=0:p-1
       sum=sum+c(m+1)*arg^m;
    end;
    V(j)=exp(-arg*arg)*sum;
end;

time=cputime-t;
```