1D Finite Element Method Matlab Vectorization Implementation Details [*†]

Wenqiang Feng [‡]

Abstract

This is the project report of MATH 574. In this project, I implement the Finite Element Method (FEM) for two-point boundary value Poisson problem by using sparse assembling and Matlab 's vectorization techniques. After using the sparse assembling and vectorization techniques, I find it's 10 times faster than the classical programming and 2 times faster than only using the sparse assembling techniques. This work is partially supported by Dr.Wise's summer research assistant.

1 Model Problem

In this corse project, I focus on the Finite Element Method (FEM) implementation for two-point boundary value Poisson problem

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

2 Mesh

Since Matlab can not use 0 as subscript, then I will use uniform mesh which is described in Figure.1. (But the data structures support the nonuniform mesh.) The element number and the nodal number are denoted by NT and N, respectively.

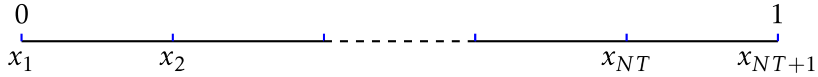


Figure 1: Uniform partition in 1D for finite element method

In the following subsections, I will give the demo mesh and demo data structure for the degrees of polynomial=1,2,3 with 4 elements.

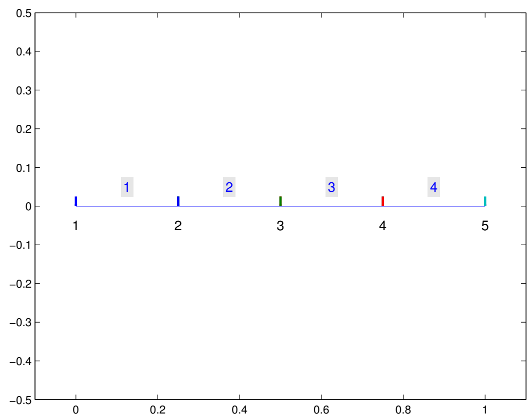
[*]This work is partially supported by Dr.Wise's summer research assistant.

[†]Key words: 1 Dimension, finite elements, Matlab, vectorization.

[‡]Department of Mathematics,University of Tennessee, Knoxville, TN, 37909, wfeng@math.utk..edu

2.1 Demo mesh and data structure for q=1

The demo mesh for q=1 is in Figure.2 and the corresponding demo data structure is in (2).

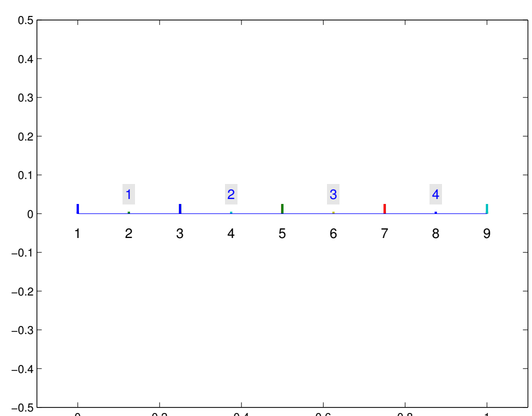


$$elem = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix}, node = \begin{bmatrix} 0 \\ 0.25 \\ 0.75 \\ 1 \end{bmatrix}. \quad (2)$$

Figure 2: Demo mesh for q=1

2.2 Demo mesh and data structure for q=2

The demo mesh for q=2 is in Figure.3 and the corresponding demo data structure is in (3).

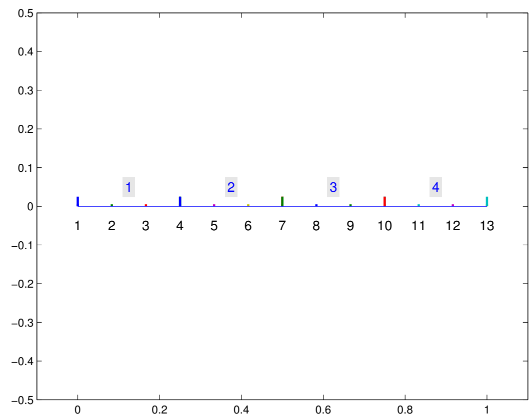


$$elem = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 5 & 6 & 7 \\ 7 & 8 & 9 \end{bmatrix}, node = \begin{bmatrix} 0.000 \\ 0.125 \\ 0.250 \\ 0.375 \\ 0.500 \\ 0.625 \\ 0.750 \\ 0.875 \\ 1.000 \end{bmatrix}. \quad (3)$$

Figure 3: Demo mesh for q=2

2.3 Demo mesh and data structure for q=3

The demo mesh for q=3 is in Figure.3 and the corresponding demo data structure is in (4).



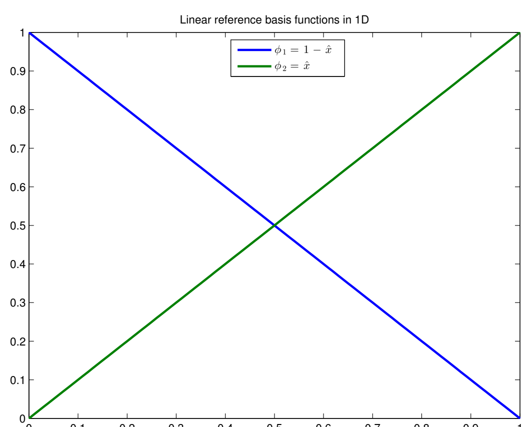
$$elem = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 7 & 8 & 9 & 10 \\ 10 & 11 & 12 & 13 \end{bmatrix}, node = \begin{bmatrix} 0.0000 \\ 0.0833 \\ 0.1667 \\ 0.2500 \\ 0.3333 \\ 0.4167 \\ 0.5000 \\ 0.5833 \\ 0.6667 \\ 0.7500 \\ 0.8333 \\ 0.9167 \\ 1.0000 \end{bmatrix}. \quad (4)$$

Figure 4: Demo mesh for q=3

3 Reference Basis function

3.1 Reference basis function

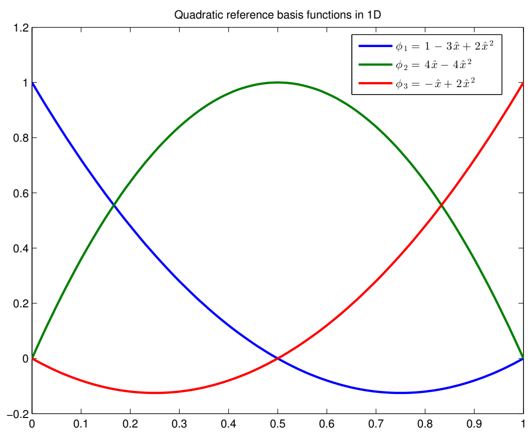
1. Linear reference basis function



$$\begin{cases} \phi_1 & = 1 - \hat{x}, \\ \phi_2 & = \hat{x}. \end{cases}$$

Figure 5: Linear reference basis function

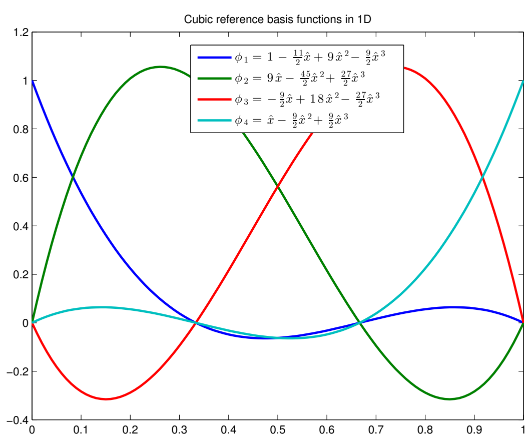
2. Quadratic reference basis function



$$\begin{cases} \phi_1 & = 1 - 3\hat{x} + 2\hat{x}^2, \\ \phi_2 & = 4\hat{x} - 4\hat{x}^2, \\ \phi_3 & = -\hat{x} + 2\hat{x}^2. \end{cases}$$

Figure 6: Quadratic reference basis function

3. Cubic reference basis function



$$\begin{cases} \phi_1 & = 1 - \frac{11}{2}\hat{x} + 9\hat{x}^2 - \frac{9}{2}\hat{x}^3, \\ \phi_2 & = 9\hat{x} - \frac{45}{2}\hat{x}^2 + \frac{27}{2}\hat{x}^3, \\ \phi_3 & = -\frac{9}{2}\hat{x} + 18\hat{x}^2 - \frac{27}{2}\hat{x}^3, \\ \phi_4 & = \hat{x} - \frac{9}{2}\hat{x}^2 + \frac{9}{2}\hat{x}^3. \end{cases}$$

Figure 7: Cubic reference basis function

3.2 Quadrature Points

3.3 Reference Quadrature Points on [-1, 1]

Since gauss-legendre formula with n_q points is exact for polynomial of degree $q \leq 2n_q - 1$, I choose $n_1 = 1$, $n_2 = 2$, $n_3 = 3$.

1. Gauss quadrature points and the corresponding weight on [-1, 1] for q=1

$$points = \begin{bmatrix} 0.0000000000000000 \end{bmatrix}, weights = \begin{bmatrix} 2.0000000000000000 \end{bmatrix}. \tag{5}$$

2. Gauss quadrature points and the corresponding weight on [-1, 1] for q=2

$$points = \begin{bmatrix} -0.5773502691896257 \\ 0.5773502691896257 \end{bmatrix}, weights = \begin{bmatrix} 1.0000000000000000 \\ 1.0000000000000000 \end{bmatrix}. \tag{6}$$

3. Gauss quadrature points and the corresponding weight on [-1, 1] for q=3

$$points = \begin{bmatrix} 0.0000000000000000 \\ -0.7745966692414834 \\ 0.7745966692414834 \end{bmatrix}, weights = \begin{bmatrix} 0.8888888888888888 \\ 0.5555555555555556 \\ 0.5555555555555556 \end{bmatrix}. \tag{7}$$

3.4 Reference Quadrature Points on $[0, 1]$

Let $\hat{K}$ spanned by $\hat{A}_1 = 0$, $\hat{A}_2 = 1$ be the reference triangle (Figure. 8).

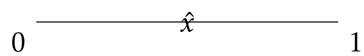


Figure 8: The interval reference element.

For a given physical element $K \in \mathcal{T}_h$, we treat it as the image of $\hat{K}$ under the affine map (Figure.9):

$$F : \hat{K} \to K.$$

If K has vertices x_i, x_{i+1}, then the map F can be defined by

$$\mathbf{x} = F(\hat{\mathbf{x}}) = h_i \hat{\mathbf{x}} + \mathbf{x}_i,$$

where

$$h_i = x_{i+1} - x_i.$$

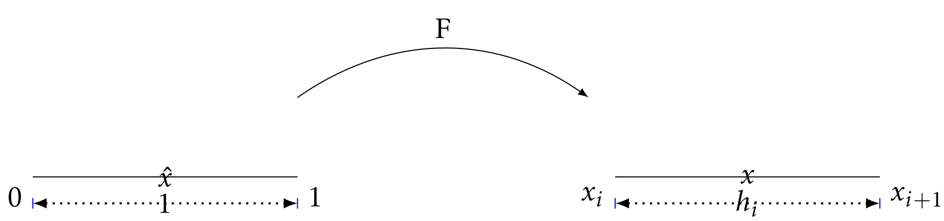


Figure 9: The affine mapping of interval element.

4 Templates

4.1 Quadrature points templates

1. $q = 1$

$$points = \begin{bmatrix} 0.0000000000000000 \end{bmatrix}, weights = \begin{bmatrix} 2.0000000000000000 \end{bmatrix}.$$

2. $q = 2$

$$points = \begin{bmatrix} -0.5773502691896257 \\ 0.5773502691896257 \end{bmatrix}, weights = \begin{bmatrix} 1.0000000000000000 \\ 1.0000000000000000 \end{bmatrix}.$$

3. $q = 3$

$$points = \begin{bmatrix} 0.0000000000000000 \\ -0.7745966692414834 \\ 0.7745966692414834 \end{bmatrix}, weights = \begin{bmatrix} 0.8888888888888888 \\ 0.5555555555555556 \\ 0.5555555555555556 \end{bmatrix}.$$

Vectorization of reference quadrature points and corresponding weights:

```
function    [points, weights]=refQuadPoint1D(number)
%REFQUADPOINTS:  generate the 1D Gauss quadrature points and the
    corresponding
% weights on the reference element
%function    [points, weights]=refQuadPoints(number)
%--------------------------------------------------------------------------%
% USAGE
%     function  [points, weights]=refQuadPoint1d(number)
%
%INPUT
%     number:  the specific number of the Gauss pionts
%
%OUTPUT
%     points:  the coordinate of the Gauss points
%     weights: the corresponding weights
%REFERENCE
%     http://pomax.github.io/bezierinfo/legendre-gauss.html
% This work is supported by Dr.Wise's summer Research Assistant.
% Created by Wenqiang Feng on 5/8/14.
% Copyright (c) 2014 WENQIANG FENG. All rights reserved.
%--------------------------------------------------------------------------%
switch number
    case 1
        points=0.000000;
        weights=2.000;
    case 2
        points=[-0.5773502691896257
                0.5773502691896257]';

        weights =[1.0000000000000000
                  1.0000000000000000]';
    case 3
        points=[0.0000000000000000
                -0.7745966692414834
                0.7745966692414834]';
        weights=[0.8888888888888888
                 0.5555555555555556
                 0.5555555555555556]';
    case 4
        points=[-0.3399810435848563
                0.3399810435848563
                -0.8611363115940526
                0.8611363115940526]';
        weights=[0.6521451548625461
                 0.6521451548625461
                 0.3478548451374538
                 0.3478548451374538]';
    case 5
        points=[0.0000000000000000
```

```
                -0.5384693101056831
                0.5384693101056831
                -0.9061798459386640
                0.9061798459386640
                ]';
        weights=[0.5688888888888889
                 0.4786286704993665
                 0.4786286704993665
                 0.2369268850561891
                 0.2369268850561891
                 ]';
    case 6
        points=[-0.6612093864662645
                0.6612093864662645
                -0.2386191860831969
                0.2386191860831969
                -0.9324695142031521
                0.9324695142031521]';
        weights=[0.3607615730481386
                 0.3607615730481386
                 0.4679139345726910
                 0.4679139345726910
                 0.1713244923791704
                 0.1713244923791704]';

    otherwise
        error('Quadrature rule not chosen!');
end
```

4.2 Reference basis template

1. $q = 1$

$$\phi = \begin{bmatrix} 1-\hat{x} \\ \hat{x} \end{bmatrix}, D\phi = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

2. $q = 2$

$$\phi = \begin{bmatrix} 1-3\hat{x}+2\hat{x}^2 \\ 4\hat{x}-4\hat{x}^2 \\ -\hat{x}+2\hat{x}^2 \end{bmatrix}, D\phi = \begin{bmatrix} -3+4\hat{x} \\ 4-8\hat{x} \\ -1+4\hat{x} \end{bmatrix}.$$

3. $q = 3$

$$\phi = \begin{bmatrix} 1-\frac{11}{2}\hat{x}+9\hat{x}^2-\frac{9}{2}\hat{x}^3 \\ 9\hat{x}-\frac{45}{2}\hat{x}^2+\frac{27}{2}\hat{x}^3 \\ -\frac{9}{2}\hat{x}+18\hat{x}^2-\frac{27}{2}\hat{x}^3 \\ \hat{x}-\frac{9}{2}\hat{x}^2+\frac{9}{2}\hat{x}^3 \end{bmatrix}, D\phi = \begin{bmatrix} -\frac{11}{2}+18\hat{x}-\frac{27}{2}\hat{x}^2 \\ 9-45\hat{x}+\frac{81}{2}\hat{x}^2 \\ -\frac{9}{2}+36\hat{x}-\frac{81}{2}\hat{x}^2 \\ 1-9\hat{x}+\frac{27}{2}\hat{x} \end{bmatrix}.$$

Vectorization of reference basis functions:

```

function [ phi,Dphi,weight] = refBasis1d(T)
% REFBASIS1D: The reference basis function for 1d FEM
```

```
% function [ phi,Dphi,weight, fvec] = refBasis1d(T)
%-----------------------------------------------------------------------%
% USAGE
%     [ phi,Dphi,weight, fvec] = refBasis1d(T)
% INPUT
%     T:      auxiliary data structure
%
% OUTPUT
%     phi:  the value of the basis function  at the quadrature points
%     Dphi: the derivative value of the basis at the quadrature points
%     weight: the quadrature weight of the corresponding quadrature points
%     fvec:   the value of the RHS function at the global quadrature points
%
% This work is supported by Dr.Wise's Research Assistant.
% Created by Wenqiang Feng on 9/7/14.
% Copyright (c) 2014 WENQIANG FENG. All rights reserved.
%-----------------------------------------------------------------------%
  [xi, weight]=QuadPoint1D(T.nQuad,0,1);

switch T.polyDeg
%% Linear basis function
  case 1
        % Linear basis functions
        phi=[1-xi; xi];
        % The derivative of the linear basis
        Dphi=[-1+0.*xi;1+0.*xi];

%% Quadratic basis function
    case 2
        % Quadratic basis functions
        phi=[1-3.*xi+2*xi.^2; 4*xi-4*xi.^2;-xi+2*xi.^2];

        % The derivative of the Quadratic basis
        Dphi=[-3+4.*xi; 4-8.*xi; -1+4.*xi];
%% Cubic basis function
    case 3
        % cubic basis functions
        phi=[1-11/2.*xi+9*xi.^2-9/2*xi.^3;
             9.*xi-45/2*xi.^2+27/2*xi.^3;
             -9/2*xi+18*xi.^2-27/2*xi.^3;
             xi-9/2*xi.^2+9/2*xi.^3];
        % The derivative of the Cubic basis
        Dphi=[-11/2+18*xi-27/2*xi.^2;
              9-45.*xi+81/2*xi.^2;
              -9/2+36.*xi-81/2.*xi.^2;
              1-9*xi+27/2*xi.^2];
end
```

5 Stiffness Matrix and Load Vector (RHS)

Sparse assembling and MATLAB vectorization techniques are described in the fowling code:

```
function [ A, F ] = assemble1dvector(elem,T)
% ASSEMBLE1D: Assembling [rocess of the 1d FEM
```

```
%function [AV,AE,AI]=elemAssemble(T,bdFlag2Elem,pde)
%-----------------------------------------------------------------------%
% USAGE
%      [ a, f ] = assemble1d(elem,node,T)
% INPUT
%      elem:    elem indexes
%      node:    node points xy value
%      T    :   the auxiliary data structure
%
% OUTPUT
%      A:       Stiffness Matrix
%      f:       Load vector
%
% This work is supported by Dr.Wise's summer Research Assistant.
% Created by Wenqiang Feng on 9/7/14.
% Copyright (c) 2014 WENQIANG FENG. All rights reserved.
%-----------------------------------------------------------------------%

A=sparse(T.N,T.N);
F=zeros(T.N,1);
ia=zeros(T.Nloc^2*T.NT,1);
iF=zeros(T.Nloc*T.NT,1);
Fi=zeros(T.N,1);
ja=zeros(T.Nloc^2*T.NT,1);
aij=zeros(T.Nloc^2*T.NT,1);

% Computing the reference stiffness matrix and referenece load vector
refA=zeros(T.Nloc,T.Nloc);
refF=zeros(1,T.Nloc);

[ phi,Dphi,weight] = refBasis1d(T);

% reference stiffess matrix
for i=1:T.Nloc
    for j=1:T.Nloc
        refA(i,j)=sum(weight.*Dphi(j,:).*Dphi(i,:));
    end
end

% the value of RHS function at global quadrature points
rhsvec=rhsfun(T.Quadx);

findx=1;aindx=1;
 for ie=1:T.NT
     %local stiffness matrix and local load vector
     %Ae=locA{ie};Fe=locF{ie};
     Ae=1/T.area(ie)*refA;
     for ib=1:T.Nloc
         refF(1,ib)=sum(weight.*rhsvec(ie,:).*phi(ib,:));
         Fe=T.area(ie)*refF;
         iF(findx)=elem(ie,ib);
         Fi(findx)=Fe(ib);
         findx=findx+1;
         for jb=1:T.Nloc
             ia(aindx)=elem(ie,ib);ja(aindx)=elem(ie,jb);
             aij(aindx)=Ae(ib,jb);
```

```
                aindx=aindx+1;
            end
        end
    end

    A=sparse(ia,ja,aij,T.N,T.N);
    F=accumarray(iF, Fi,[T.N 1]);
```

6 Numerical Experiments

6.1 test 1

In the first test, we choose the data such that the exact solution of (1) on the unit domain $\Omega = [0,1]$ is given by

$$u(x) = \sin(\pi x).$$

The errors for the FEM approximation using $r = 1,2,3$ and varying h can be found in Table (1). By using linear regression for the errors, we can see that the errors in Table 1 obey the error rules

Table 1: Errors of the computed solution in Test 1.

	NT	$\|u-u_h\|_{L^\infty}$	$\|u-u_h\|_{L^2}$	$\|u-u_h\|_{H^1}$
$q=1$	25	2.63×10^{-3}	1.68×10^{-3}	6.57×10^{-2}
	50	6.58×10^{-4}	4.19×10^{-4}	3.29×10^{-2}
	100	1.64×10^{-4}	1.05×10^{-4}	1.65×10^{-2}
	200	4.11×10^{-5}	2.62×10^{-5}	8.22×10^{-3}
$q=2$	25	1.59×10^{-5}	1.01×10^{-5}	1.59×10^{-3}
	50	1.99×10^{-6}	1.27×10^{-6}	3.98×10^{-4}
	100	2.49×10^{-7}	1.58×10^{-7}	9.95×10^{-5}
	200	3.11×10^{-8}	1.98×10^{-8}	2.49×10^{-5}
$q=3$	25	1.30×10^{-7}	7.35×10^{-8}	1.69×10^{-5}
	50	8.11×10^{-9}	4.59×10^{-9}	2.11×10^{-6}
	100	5.07×10^{-10}	2.87×10^{-10}	2.64×10^{-7}
	200	3.15×10^{-11}	1.81×10^{-11}	3.25×10^{-8}

Using linear regression, we can also see that the errors in Table.1 obey

1. For the polynomial degree $r = 1$.

$$\|u-u_h\|_{L^\infty} \approx 1.6429h^{1.9998},$$
$$\|u-u_h\|_{L^2} \approx 1.0478h^{2.0001},$$
$$\|u-u_h\|_{H^1} \approx 1.6416h^{0.9996}.$$

2. For the polynomial degree $r = 2$.

$$\begin{aligned} ||u - u_h||_{L^\infty} &\approx 0.2474h^{2.9990}, \\ ||u - u_h||_{L^2} &\approx 0.1578h^{2.9993}, \\ ||u - u_h||_{H^1} &\approx 0.9957h^{2.0002}. \end{aligned}$$

3. For the polynomial degree $r = 3$.

$$\begin{aligned} ||u - u_h||_{L^\infty} &\approx 0.0512h^{4.0028}, \\ ||u - u_h||_{L^2} &\approx 0.0285h^{3.9975}, \\ ||u - u_h||_{H^1} &\approx 0.2694h^{3.0058}. \end{aligned}$$

These linear regressions indicate that the finite element method for this problem can converge in the optimal rates, which are $q+1$ order in L^2 norm and q order in H^1 norm.

6.2 test 2

In the second test, we choose the data such that the exact solution of (1) on the domain $\Omega = [0,1]$ is given by

$$u(x) = 4x(1-x).$$

The errors for the FEM approximation using $r = 1,2,3$ and varying h can be found in Table (2). By using linear regression for the errors, we can see that the errors in Table 2 obey the error rules

1. For the polynomial degree $r = 1$.

$$\begin{aligned} ||u - u_h||_{L^\infty} &\approx 1.0000h^{2.0000}, \\ ||u - u_h||_{L^2} &\approx 1.0000h^{2.0001}. \end{aligned}$$

2. For the polynomial degree $r = 2,3$. when $degree(u) \leq r$, our numerical approximation is exact.

Table 2: Errors of the computed solution in Test 1.

	NT	$\|\|u - u_h\|\|_{L^\infty}$	$\|\|u - u_h\|\|_{L^2}$
$q = 1$	25	1.60×10^{-3}	1.68×10^{-3}
	50	4.00×10^{-4}	4.19×10^{-4}
	100	1.00×10^{-4}	1.05×10^{-4}
	200	2.50×10^{-5}	2.62×10^{-5}
$q = 2$	25	4.00×10^{-14}	2.92×10^{-14}
$q = 3$	25	5.53×10^{-14}	3.31×10^{-14}