数值计算实验报告

16340217

王晶

一、已知 sin(0.32)=0.314567，sin(0.34)=0.333487，sin(0.36)=0.352274，sin(0.38)=0.370920。请采用线性插值、二次插值、三次插值分别计算 sin(0.35)的值。

可以直接通过拉格朗日插值多项式来得到，其中线性插值，二次插值，三次插值分别对应 n=1,n=2 和 n=3 的情况。

$$l_k(x)=\frac{(\mathrm{x}-\mathrm{x}_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(\mathrm{x_k}-\mathrm{x}_0)\ldots(\mathrm{x_k}-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)}$$

然后得到

$$L_n(x_j)=\sum_{k=0}^{n}\mathrm{y_k}l_k(x_j)$$

最终得到结果为，其中 an1，2，3 分别对应线性，二次，三次插值：

```
>> test1

ans1 =

   0.342880500000000


ans2 =

   0.342897125000000


ans3 =

   0.342897625000000
```

准确值应为：

```
>> sin(0.35)

ans =

   0.342897807455451
```

可以发现，随着取点个数地增加，误差也会越来越小

在代码中，可直接运行 test1 来获取三种方法的结果

二、请采用下述方法计算 115 的平方根，精确到小数点后六位。

(1)二分法。选取求根区间为[10, 11]。

(2)牛顿法。

(3)简化牛顿法。

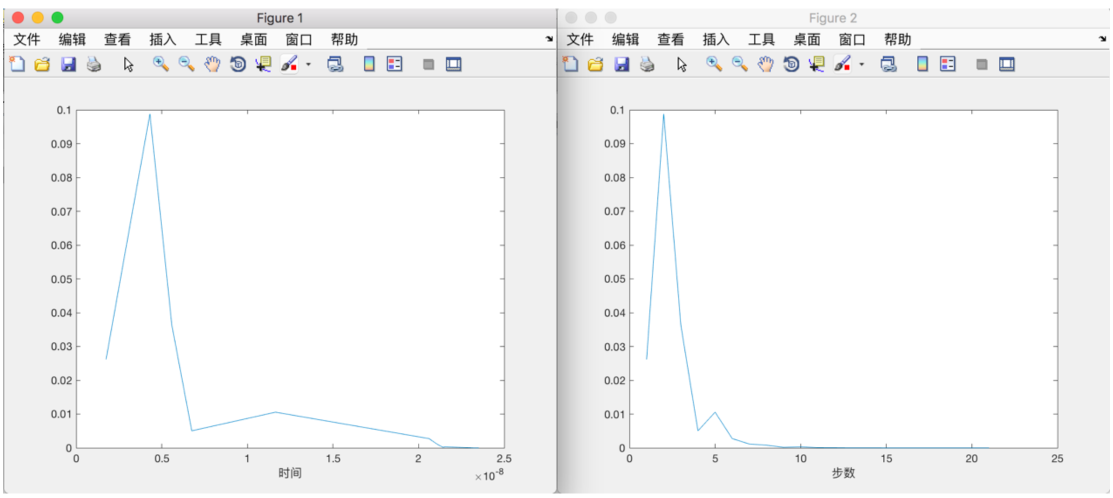
(4)弦截法。

绘出横坐标分别为计算时间、迭代步数时的收敛精度曲线。

精度皆设置为 10^-6

以下各图，左边的横坐标为时间，右边的横坐标为迭代步数

首先是二分法：

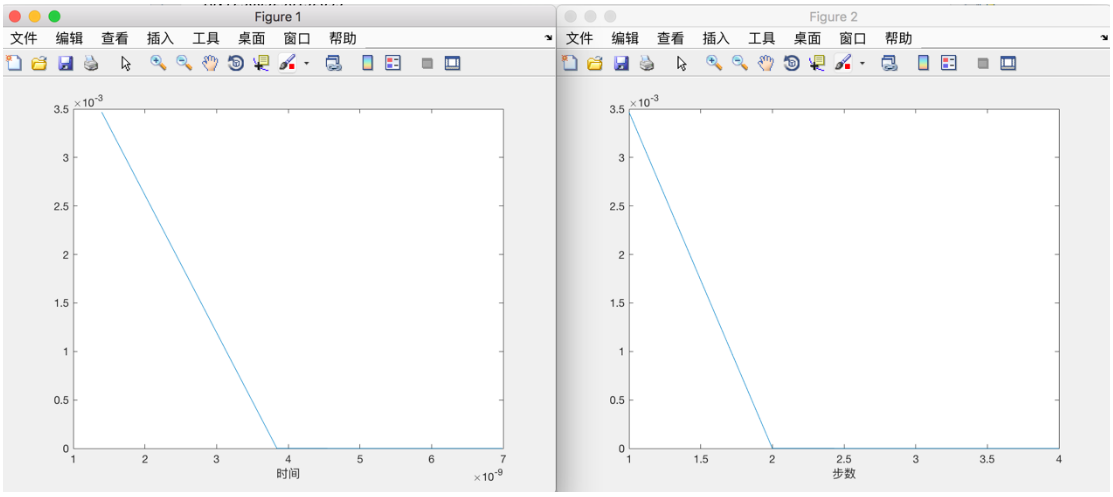


函数位于 binary.m 文件中

主要迭代步骤：

```matlab
if(((b+a)/2)^2 >=115)
    b = (b+a)/2;
else
    a = (b+a)/2;
end
```

然后是牛顿法：

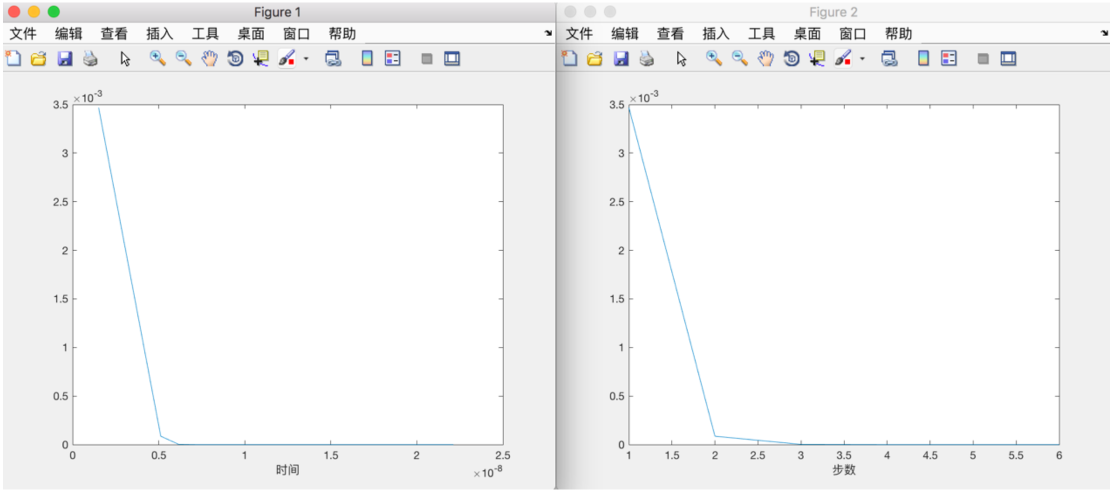


函数位于 newton.m 文件中

主要迭代步骤

```matlab
xk2 = xk1 - (xk1^2-115)/(2*xk1);
```

然后是简化牛顿法：



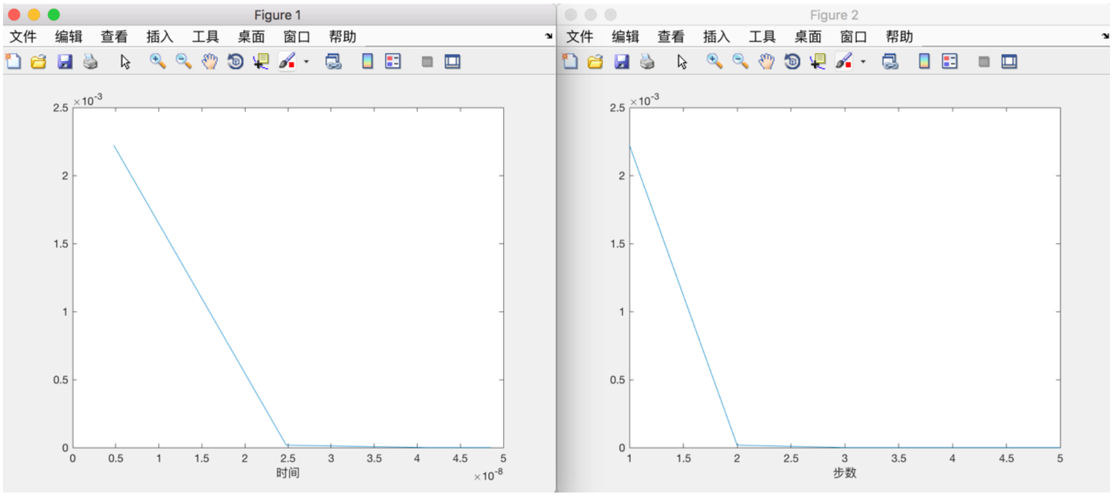
函数位于 simNewton.m 文件中

主要迭代步骤，将牛顿法中导数改为定值

```
C = 1/22;

xk2 = xk1 - C*(xk1^2-115);
```

然后是弦截法：

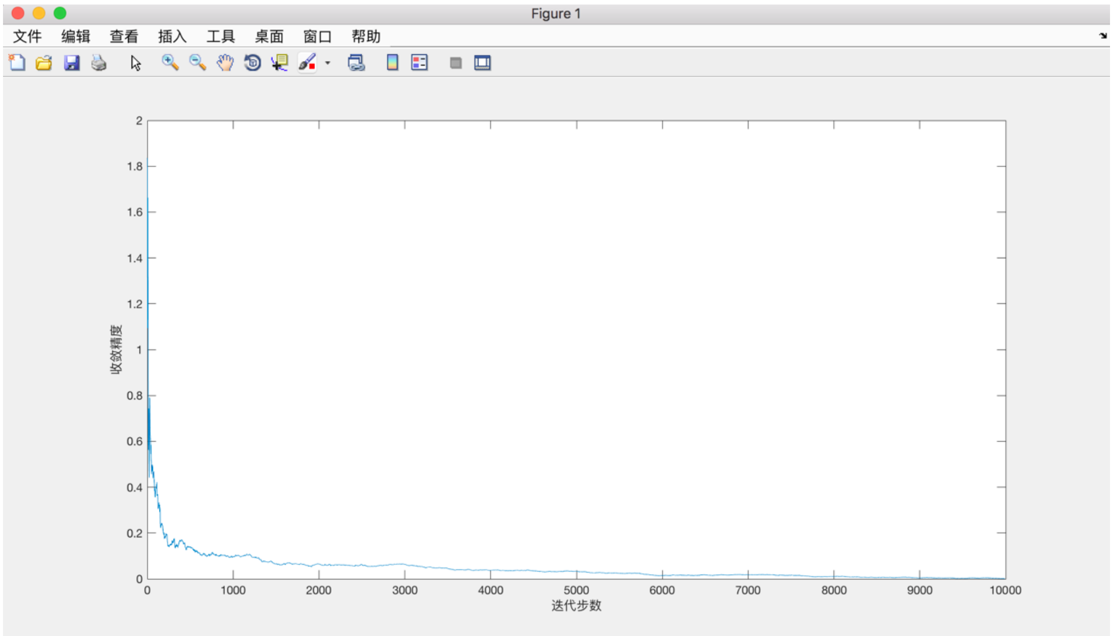


函数位于 xianjie.m 文件中

主要迭代步骤，离散化：

```
xk3 = xk2 - ((xk2^2-115)/((xk2^2-115)-(xk1^2-115)))*(xk2-xk1);
```

三、请采用递推最小二乘法求解超定线性方程组 Ax=b，其中 A 为 mxn 维的已知矩阵，b 为 m 维的已知向量，x 为 n 维的未知向量，其中 n=10，m=10000。A 与 b 中的元素服从独立同分 布的正态分布。绘出横坐标为迭代步数时的收敛精度曲线。



其中对比精度通过与使用最小二乘法所得的结果，求得二范数。

```
P = P-(P*f*f'*P)./(1+f'*P*f);
a = am + P*f*(b(k)-f'*am);
pre(k-n)=norm((a-result),2);
```

以上是关键的递推公式

可直接运行 RLS.m 获取结果

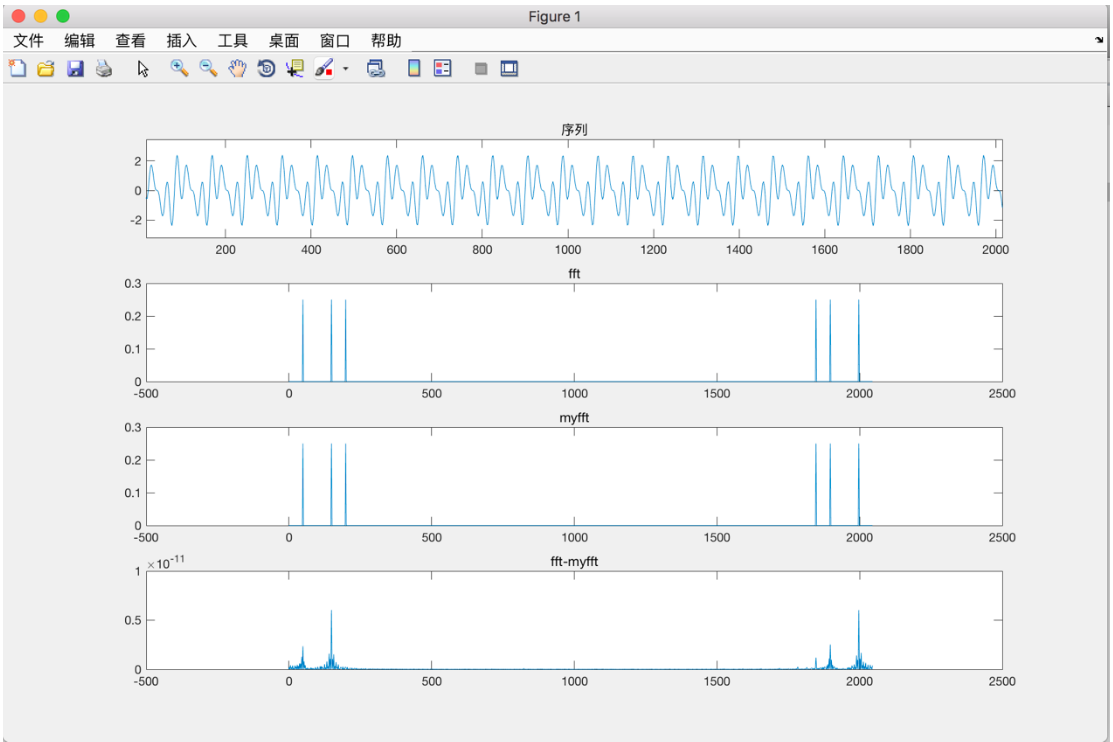
四、请编写 1024 点快速傅里叶变换的算法。自行生成一段混杂若干不同频率正弦的信号， 测试所编写的快速傅里叶变换算法。

首先要取 1024 个采样点，那么要先构造一个符合条件的信号。

```
function [x,t] = createSin()
%UNTITLED35 此处显示有关此函数的摘要
%   此处显示详细说明

f1=50;%信号频率Hz
f2=150;%信号频率Hz
f3=200;%信号频率Hz
fs=2048;%采样频率Hz
N=1024;%采样点数
t=(0:N-1)/fs;%采样时间s
x1=sin(2*pi*f1*t);%信号采样值
x2=sin(2*pi*f2*t);%信号采样值
x3=sin(2*pi*f3*t);%信号采样值
x=x1+x2+x3;
plot(t,x,'.')
end
```

然后进行快速傅立叶变换，并和 matlab 自带的所得结果进行对比



可直接运行 useFFT.m 获取结果

主要代码如下：

```
function X=myfft(x)
    N=length(x);
    h=log2(N);
    for i=1:h
        s=[];
        for j=1:2^(i-1)
            M=2^(h-i+1);
            xj=x([1:M]+(j-1)*M);
            [y,z]=disbutterfly(xj)
            s=[s,y,z];
        end
        x=s;
    end

X=rader(x,N);
end
```

纠正输出序列的输出顺序

```
function y=rader(x,N)
    n=[0:N-1];
    bn=dec2bin(n);
    rbn=fliplr(bn);
    rn=bin2dec(rbn);
    y = x(rn+1);
end
```

将序列分解为偶采样点和奇采样点

```
function [y,z] = disbutterfly(x
    N=length(x);
    n=0:N/2-1;
    w=exp(-2*1i*pi/N).^n;
    x1=x(n+1);
    x2=x(n+1+N/2);
    y=x1+x2;
    z=(x1-x2).*w;
end
```

五、请采用复合梯形公式与复合辛普森公式，计算 sin(x)/x 在[0, 1]范围内的积分。采样 点数目为 5、9、17、33。

首先定义函数：

```
function t = f(x)
    if(x==0)
        t = 1;
    else
        t = sin(x)/x;
    end
end
```

复合梯形公式：

主要步骤：

```
h=(b-a)/n(j);
sum=0;
for k=1:n(j)-1
    sum=sum+f(a+k*h);
end
y(j) = (f(a)+2*sum+f(b))*h/2;
```

其中 h 为间隔，f 为之前定义的函数，

得到结果：

```
>> txing()

ans =

   0.945078780953402   0.945773188549752   0.945996225242376   0.946060023888043
```

函数位于 txing.m 文件中

复合辛普森公式:

主要步骤:

```
for i = 0:n(k)-1
    sum1 = sum1 + f(a+(i+1/2).*h);
end
for j = 1:n(k)-1
    sum2 = sum2 + f(a+j.*h);
end
y(k) = h/6*(f(a)+4*sum1+2*sum2+f(b));
end
```

得出结果:

```
ans =

   0.946083168838073   0.946083079742053   0.946083071103489   0.946083070419036
```

函数位于 simpson.m 文件中

从左到右分别是采样点数目为 5, 9, 17, 33 时的所得的结果。

六、请采用下述方法，求解常微分方程初值问题 y'=y-2x/y，y(0)=1，计算区间为[0, 1]， 步长为 0.1。

(1)前向欧拉法。

(2)后向欧拉法。

(3)梯形方法。

(4)改进欧拉方法。

前向欧拉方法:

```
>> frontEuler

ans =

   1.784770832497982
```

函数位于 frontEuler.m 中

后向欧拉方法：

```
>> backEuler

ans =

   1.661808338519370
```

函数位于 backEuler.m 中

梯形方法：

```
>> trapzd

ans =

   1.734150601354133
```

函数位于 trapzd.m 文件中

改进欧拉方法：

```
>> improEuler

ans =

   1.737867401035414
```

函数位于 improEuler.m 中

真实的 3 的平方根：

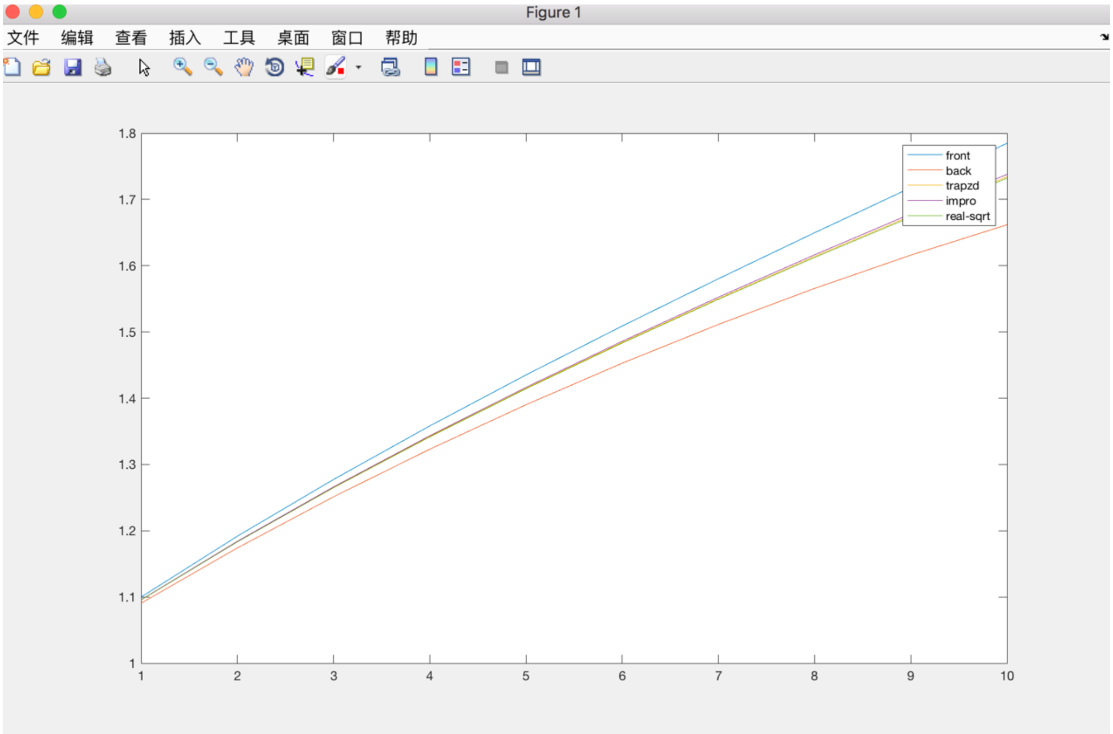
```
>> sqrt(3)

ans =

   1.732050807568877
```

然后进行对比：

对比函数可以直接运行 test2.m 文件函数



发现前向和后向欧拉方法的误差较大，梯形方法和改进欧拉方法的误差极小。

实现原理：

前向欧拉：

```
for n=1:10
    yn2 = yn1 + h*f(x,yn1);
    yn1 = yn2;
    y(n)=yn2;
    x = x + h;
```

其中 yn2 为 yn+1，yn1 为 yn

后向欧拉：

```
yt = yn1 + h*f(x,yn1);
%迭代求y(n+1)
x = x+h;
done = 0;
while  ~done
    yn2 = yn1 + h * f(x,yt);
    done = (abs(yn2-yt)<1e-6);
    yt = yn2;
end
yn1 = yn2;
y(n)=yn2;
```

其中 yt 为 yn+1(0)，最终迭代得到 yn+1(k)

梯形方法：

```
yn2 = yn1 + h * f(x,yn1);
done = 0;
while ~done
    yt = yn1 + 0.5*h*(f(x,yn1)+f(x+h,yn2));
    done = (abs(yt-yn2)<1e-6);
    yn2 = yt;
end
x = x + h;
yn1 = yn2;
y(n)=yn2;
```

其中 yn2 为 yn+1(0)，最终迭代得到 yn+1(k)，yt 为临时变量。

改进欧拉：

```
for n=1:10
    yt1 = yn1 + h*f(x,yn1);
    yt2 = yn1 + h*f(x+h,yt1);
    yn2 = 0.5*(yt1+yt2);
    yn1 = yn2;
    y(n)=yn2;
    x = x+h;
```

其中 yt1 和 yt2 为 yp，yc，yn2 为 yn+1，yn1 为 yn