数值分析 HW2

16340041　　　陈亚楠

实验一

1.问题描述：

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。

2.算法设计：

拉格朗日插值法：

对某个多项式函数，已知有给定的 k + 1 个取值点：

$$(x_0, y_0), \ldots, (x_k, y_k)$$

其中 x_j 对应着自变量的位置，而 y_j 对应着函数在这个位置的取值。

假设任意两个不同的 x_j 都互不相同，那么应用拉格朗日插值公式所得到的拉格朗日插值多项式为：

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

其中每个 $l_j(x)$为拉格朗日基本多项式（或称插值基函数），其表达式为：

$$\ell_j(x) := \prod_{i=0,\, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}$$

拉格朗日基本多项式 $l_j(x)$的特点是在 x_j 上取值为 1，在其它的点 x_i，$i\neq j$ 上取值为 0。

3.数值实验：

（1）线性插值：

①设 y（k）和 y（k+1）分别是 sin(0.34) 和 sin(0.36)；

②实验源码：

```
function ans = Linear()
format long;
x = [.34, .36];
y = sin(x);
x0 = .35;
ans = (x(2)-x0)/(x(2)-x(1))*y(1) + (x0-x(1))/(x(2)-x(1))*y(2)
ans = [ans; sin(.35)];
end
```

③实验结果：

```
命令行窗口
>> Linear

ans =

   0.342880662707952


ans =

   0.342880662707952
   0.342897807455451
```

（2）二次插值：

①设 y（k-1），y（k）和 y（k+1）分别是 sin(0.32)，sin(0.34)，sin(0.36)；

②实验源码：

```
function ans = Quadratic()
format long;
x = [.32, .34, .36];
y = sin(x);
x0 = .35;
ans = (x0-x(2)) * (x0-x(3)) / (x(1)-x(2)) / (x(1)-x(3)) * y(1);
ans = ans + (x0-x(1)) * (x0-x(3)) / (x(2)-x(1)) / (x(2)-x(3)) * y(2);
ans = ans + (x0-x(1)) * (x0-x(2)) / (x(3)-x(1)) / (x(3)-x(2)) * y(3);
ans = [ans; sin(.35)];
end
```

③实验结果：

```
命令行窗口
>> Quadratic

ans =

   0.342897336506755
   0.342897807455451

fx >>
```

（3）三次插值：

①设定 y（k-1），y（k），y（k+1）和 y（k+2）分别是 sin(0.32)，sin(0.34) 和 sin(0.36) 和 sin(0.38)；

②实验源码：

```
function ans = Cubic()
format long;
x = [.32, .34, .36, 38];
y = sin(x);
x0 = .35;
ans = (x0-x(2)) * (x0-x(3)) * (x0 - x(4)) / (x(1)-x(2)) / (x(1)-x(3)) /
(x(1)-x(4)) * y(1);
ans = ans + (x0-x(1)) * (x0-x(3)) * (x0 - x(4)) / (x(2)-x(1)) / (x(2)-x(3))
/ (x(2)-x(4)) * y(2);
ans = ans + (x0-x(1)) * (x0-x(2)) * (x0 - x(4)) / (x(3)-x(1)) / (x(3)-x(2))
/ (x(3)-x(4)) * y(3);
ans = ans + (x0-x(1)) * (x0-x(2)) * (x0 - x(3)) / (x(4)-x(1)) / (x(4)-x(2))
/ (x(4)-x(3)) * y(4);
ans = [ans; sin(.35)];
end
```

③实验结果：

```
命令行窗口
>> Cubic

ans =

   0.342897325220493
   0.342897807455451

fx >>
```

4.结果分析：

在使用线性插值法进行计算时，误差值约为 1.7*10e-5，使用二次插值法计算时，误差值约为 4.7*10e-6，使用三次插值法时，误差值约为 4.8*10e-6，由此可见，不一定次数越多，插值法的结果就越准确。

实验二

1.问题描述：

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。

2.算法设计：

（1）二分法：

若要求已知函数 f(x) = 0 的根 (x 的解)，则：

①先找出一个区间 [a, b]，使得 f(a)与 f(b)异号。根据介值定理，这个区间内一定包含着方程式的根；

②求该区间的中点 m = (a + b) / 2，并找出 f(m) 的值；

③若 f(m) 与 f(a) 正负号相同则取 [m, b] 为新的区间, 否则取 [a, m]；

④重复第 2 和第 3 步至理想精确度为止。

（2）牛顿法：

首先，选择一个接近函数 f(x)零点的 x_0，计算相应的 $f(x_0)$和切线斜率 $f'(x_0)$。然后我们计算穿过点 $(x_0,f(x_0))$并且斜率为 $f'(x_0)$的直线和 x 轴的交点的 x 坐标，也就是求如下方程的解：

$$0=(x-x_0)\cdot f'(x_0)+f(x_0)$$

我们将新求得的点的 x 坐标命名为 x_1，通常 x_1 会比 x_0 更接近方程 f(x)=0 的解。因此我们现在可以利用 x_1 开始下一轮迭代。迭代公式可化简为如下所示：

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

（3）简化牛顿法：

简化牛顿法，也称平行弦法，其迭代公式为

$$x_{k+1}=x_k-Cf(x_k)\qquad C\neq 0,\ k=0,1,\cdots$$

取

$$C=\frac{1}{f'(x_0)},$$。

（4）弦截法：

设 x_k，x_{k-1} 是 f(x) = 0 的近似根，我们利用 $f(x_k)$，$f(x_{k-1})$ 构造一次插值多项式 p1(x)，并用 p1(x) = 0 的根作为 f (x) = 0 的新的近似根 x_{k+1}.由于

$$p_1(x)=f(x_k)+\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}(x-x_k)$$

因此有

$$x_{k+1}=x_k-\frac{f(x_k)}{f(x_k)-f(x_{k-1})}(x_k-x_{k-1})$$

这样导出的迭代公式可以看做牛顿公式

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

中的导数 $f'(x_k)$用差商

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

取代的结果。

3.数值实验：

（1）二分法：

①这里我们使用迭代法实现二分法求根，迭代初始条件是给定的求根区间，之后根据二二分区间中值的函数值，确定新的求根区间，求根下限或者求根上限随之改变。迭代的终止条件是如果某一点的函数值小于一个小量 eps，那么迭代停止。

②实验源码：

```
function [ans, iter] = Dichotomy(a, left, right, eps)
MAX = 100;
format long;
if nargin == 0
    a = 115;
    left = 10.0;
    right = 11.0;
    eps = 10e-6;
elseif nargin == 1
    left = floor(sqrt(a));
    right = ceil(sqrt(a));
    eps = 10e-6;
elseif nargin == 3
    eps = 10e-6;
end
iter = 1;
ans = [];
while iter < MAX
    mid = (left + right)/2;
    iter = iter+1;
    ans = [ans, mid];
    val = mid*mid - a;
    if abs(val) < eps
        ans = [ans, mid];
        return;
```

```
    end
    if val > 0
        right = mid;
    elseif val < 0
        left = mid;
    end
end
end
```

③实验结果：

```
命令行窗口
ans =

  1 至 5 列

  10.500000000000000   10.750000000000000   10.625000000000000   10.687500000000000   10.718750000000000

  6 至 10 列

  10.734375000000000   10.726562500000000   10.722656250000000   10.724609375000000   10.723632812500000

  11 至 15 列

  10.724121093750000   10.723876953125000   10.723754882812500   10.723815917968750   10.723785400390625

  16 至 20 列

  10.723800659179688   10.723808288574219   10.723804473876953   10.723806381225586   10.723805427551270

  21 列

  10.723805427551270

fx >>
```

（2）牛顿法：

①通过迭代来实现牛顿法求根，设定迭代初始条件为 x = 10，之后不断根据 x 点出切线与 x 轴的交点迭代，确定新的结果。

②实验源码：

```
function [ans, iter] = Newton(a, x, eps)
MAX = 100;
format long;
if nargin == 0
    a = 115;
    x = 10.0;
    eps = 10e-6;
elseif nargin == 1
    x = floor(sqrt(a));
    eps = 10e-6;
```

```
elseif nargin == 2
    eps = 10e-6;
end
ans  =[];
iter = 1;
while iter < MAX
    iter = iter + 1;
    ans = [ans, x];
    x = (x + a/x)/2;
    if abs(x*x - a) < eps
        ans = [ans, x];
        return;
    end
end
end
```

③实验结果：

```
命令行窗口
>> Newton

ans =

  10.000000000000000   10.750000000000000   10.723837209302324   10.723805294811097

fx >>
```

（3）简化牛顿法：

①此方法前面同牛顿法，之后不断根据 x 点处切线的估值（C = 20）与 x 轴的交点迭代，确定新的结果。

②实验源码：

```
function [ans, iter] = NewtonPro(a, x, eps)
MAX = 100;
format long;
if nargin == 0
    a = 115;
    x = 10.0;
    eps = 10e-6;
elseif nargin == 1
    x = floor(sqrt(a));
    eps = 10e-6;
elseif nargin == 2
```

```
    eps = 10e-6;
end
ans  =[];
iter = 1;
c = 2*x;
while iter < MAX
    iter = iter + 1;
    ans = [ans, x];
    x = x - (x*x-a)/c;
    if abs(x*x - a) < eps
        ans = [ans, x];
        return;
    end
end
end
```

③实验结果：

```
命令行窗口
>> NewtonPro

ans =

  1 至 5 列

  10.000000000000000   10.750000000000000   10.721875000000001   10.723944824218750   10.723795194574345

  6 至 7 列

  10.723806025815554   10.723805241849655

fx >>
```

（4）弦截法：

①设定迭代初始条件为 x0 = 10，x = 11。同牛顿法，之后不不断根据 x 点处弦与 x 轴的交点迭代，确定新的结果。

②实验源码：

```
function [ans, iter] = Secant(a, x0, x, eps)
MAX = 100;
format long;
if nargin == 0
    a = 115;
    x0 = 10.0;
    x = 11.0;
    eps = 10e-6;
elseif nargin == 1
```

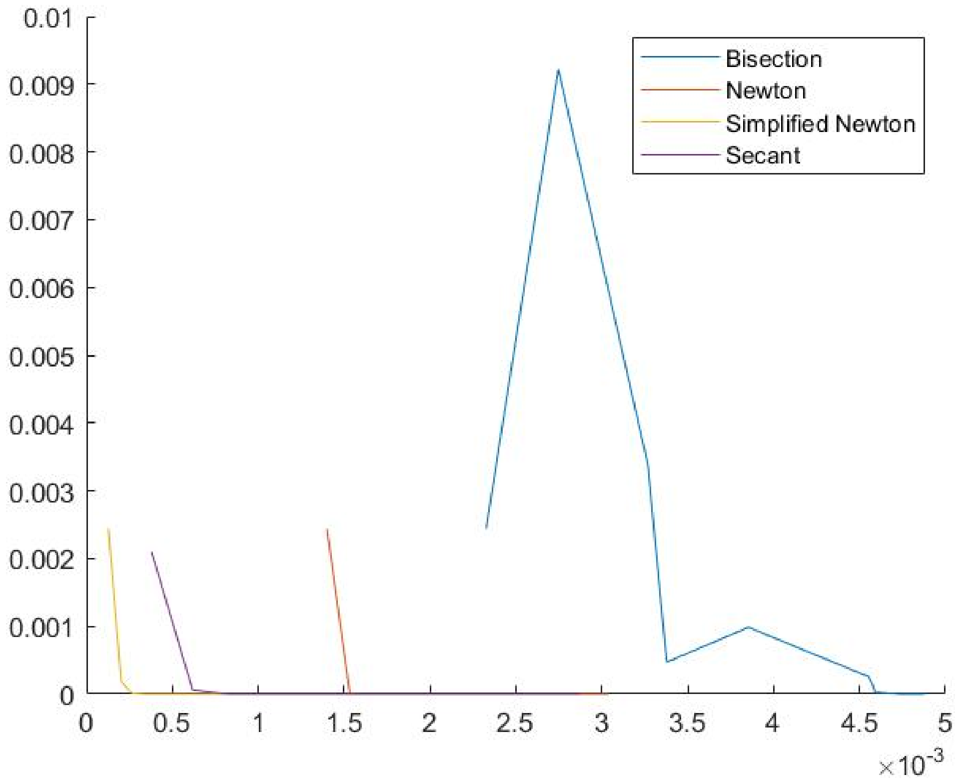
```matlab
    x0 = floor(sqrt(a));
    x = ceil(sqrt(a));
    eps = 10e-6;
elseif nargin == 2
    eps = 10e-6;
end
ans  =[];
iter = 1;
while iter < MAX
    iter = iter + 1;
    ans = [ans, x];
    tmp = x;
    x = x - (x*x-a)/(x+x0);
    x0 = tmp;
    if abs(x*x - a) < eps
        ans = [ans, x];
        return;
    end
end
end
```

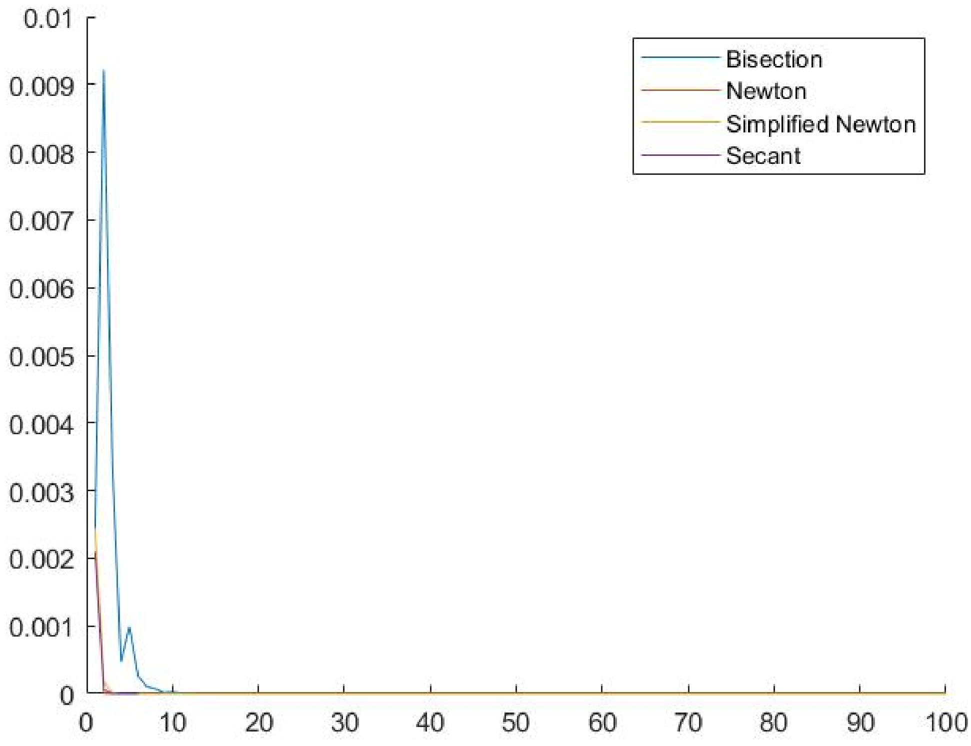
③实验结果：

命令行窗口

```
>> Secant

ans =

  11.000000000000000   10.714285714285714   10.723684210526315   10.723805348531346

fx >>
```

4.结果分析：





二分法使用了 21 次迭代得出结果，牛顿法使用了 4 次迭代得出结果，牛顿

法简化法使用了 7 次迭代读出结果，弦截法使用了 4 次迭代得出结果；虽然牛顿法相比迭代次数很少，但实际上每一步的迭代还多了一些其他的工作量，简化牛顿法在时间上相较于牛顿法慢；弦截法利用弦进行点更新，其更新速度更快，效果更明显，在性能比牛顿法有更多优越性。

实验三

1.问题描述：

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 m x n 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。

2.算法设计：

设线性方程组 Ax=b，其中 A 为 m x n 维的已知矩阵，b 为 m 维的已知向量，x 为 n 维的未知向量。如果矩阵的行数大于矩阵的列数，也就是 m > n，那么称这个方程组为超定方程组。最小二乘法常常用于超定方程组的求解，求得的解是最小二乘解。把每一行看作是一次迭代，那么共可以迭代 m 次，找到方程的最小二乘解。超定方程 Ax=b 的最小二乘解 x* 可以看作是 f(x) = Ax - b 的零点逼近值。如果矩阵的每一列分别计作 a1、a2、a3…… 最小二乘解 x* 的每一行计作 x1、x2、x3…… 那么，矩阵的最小二乘解 x* 可以看成是 a1*x1+a2*x2+……an*xn - b，对每一行 i，进行 fminunc 函数极小值运算 abs(a1i*x1+a2i*x2+……ani*xn - bi)，得到相应的 x* 的每一行。用第 i 行的计算结果作为第 i+1 行的初始值，不断迭代，最终可以得到该超定方程的总体

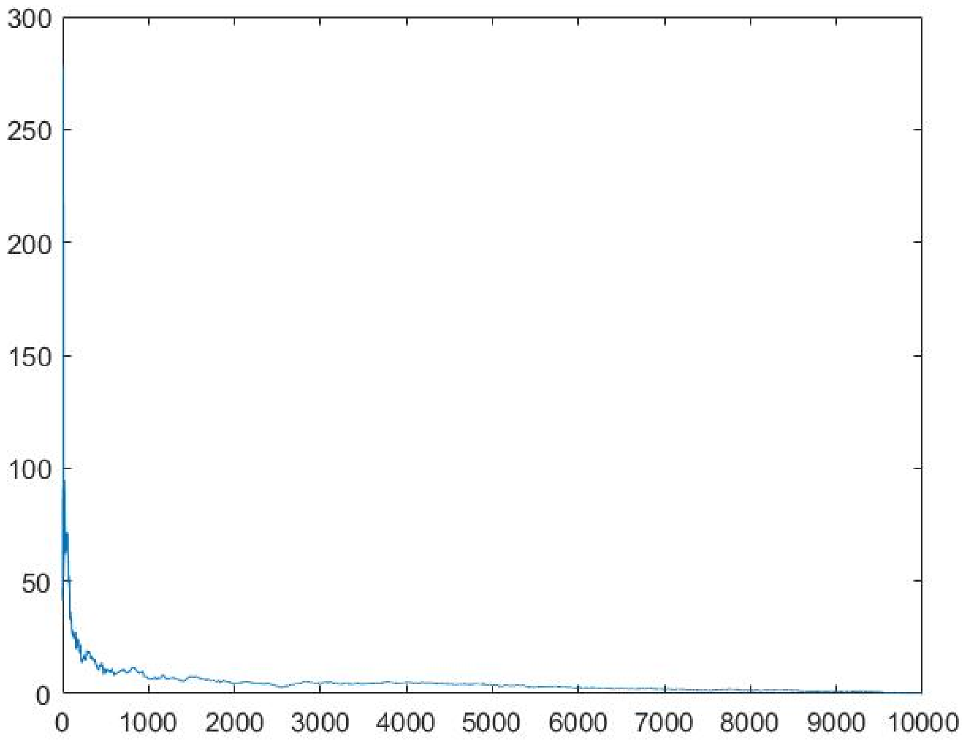
最小二乘。

3.数值实验：

①实验源码：

```
function [rsteps, results] = RLS(phi, y, m, n)
    rsteps = ones(1, m);
    results = ones(n, m);
    p = eye(n) * 100000;
    result = zeros(n, 1);
    for index = 1 : m
        k = p * phi(index, :)' / (1 + phi(index, :) * p * phi(index, :)');
        result = result + k * (y(index, 1) - phi(index, :) * result);
        rsteps(1, index) = index;
        results(:, index) = result;
        p = (eye(n) - k * phi(index, :)) * p;
    end
end
```

②实验结果：



实验四

1.问题描述：

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。

2.算法设计：

根据采样定理，fft 能分辨的最高频率为采样频率的一半（即 Nyquist 频率），函数 fft 返回值是以 Nyqusit 频率为轴对称的，Y 的前一半与后一半是复数共轭关系。作 FFT 分析时，幅值大小与输入点数有关，要得到真实的幅值大小，只要将变换后的结果乘以 2 除以 N 即可（但此时零频—直流分量—的幅值为实际值的 2 倍）。对此的解释是：Y 除以 N 得到双边谱，再乘以 2 得到单边谱（零频在双边谱中本没有被一分为二，而转化为单边谱过程中所有幅值均乘以 2，所以零频被放大了）。

若分析数据时长为 T，则分析结果的基频就是 f0=1/T，分析结果的频率序列为 [0:N-1]*f0。

使用 N 点 FFT 时，不应使 N 大于 y 向量的长度，否则将导致频谱失真。

最后用 plot 图画出原始信号的波形，用 stem 画出输出的复数序列。幅值显著增高的点对应的值除以频率就是快速傅立叶变换的余弦参数。

3.数值实验：

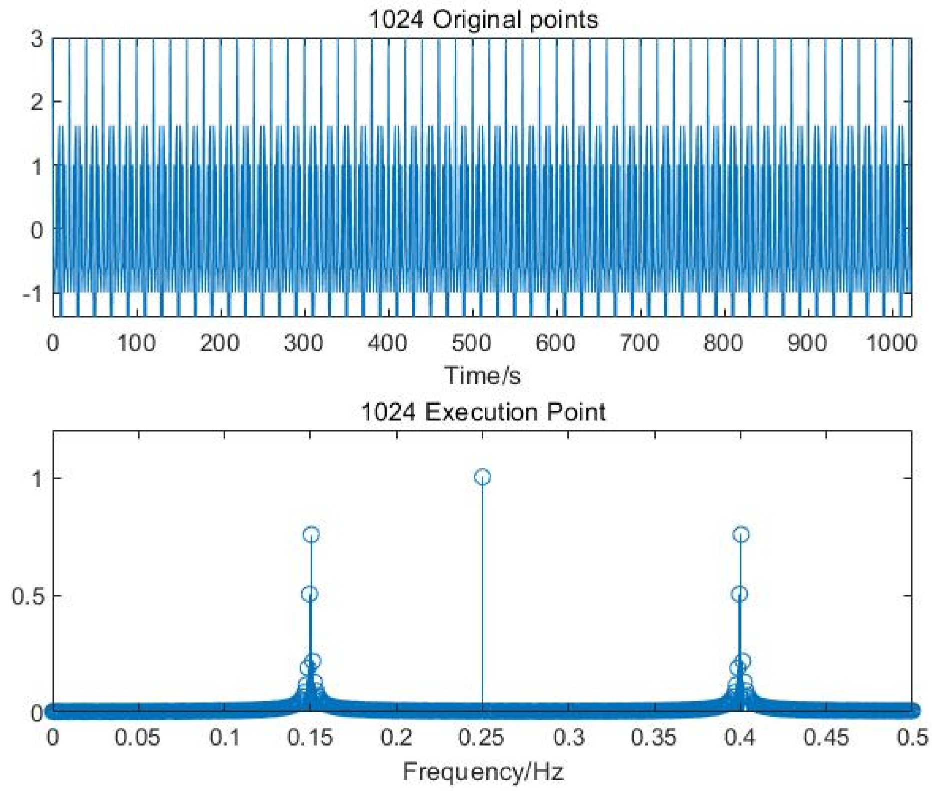
①实验源码：

```
function ans = FFT(N,Fs,c)
dt=1/Fs;
t=[0:N-1]*dt;
xn = 0;
for i=1:size(c)
    xn=xn+cos(2*pi*c(i)*[0:N-1]);
```

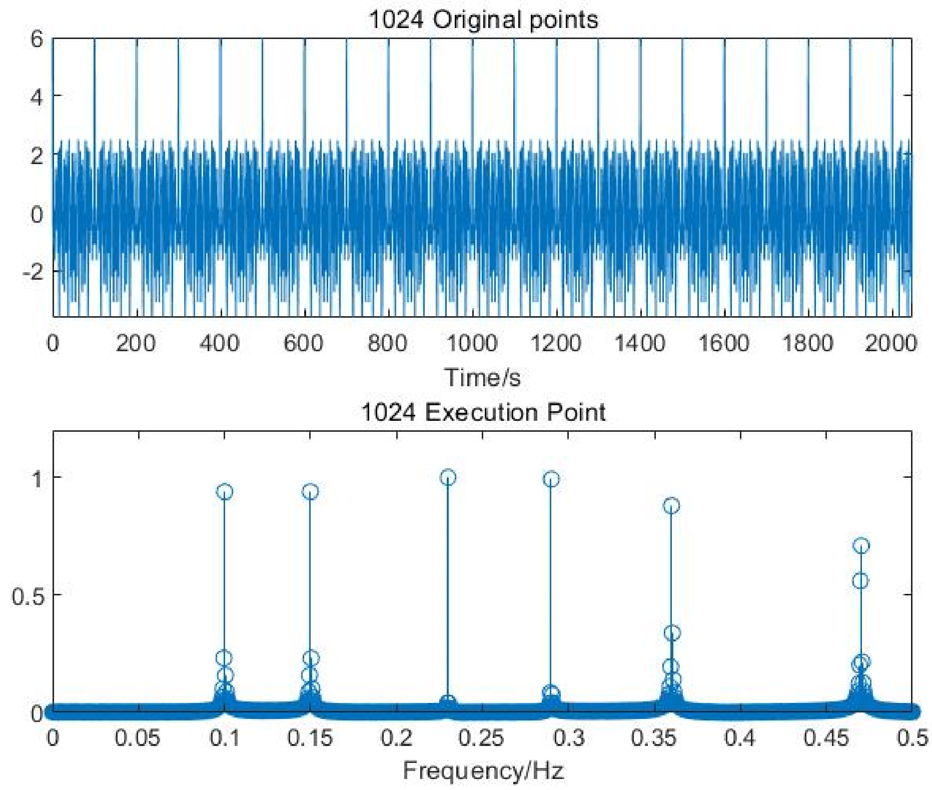
```
end
subplot(2,1,1);
plot(t,xn);
axis([0 N min(xn') max(xn')]);
xlabel('Time/s'),title('1024 Original points');
f0=1/(dt*N);
f=[0:ceil((N-1)/2)]*f0;
XN=fft(xn,N)/N;
XN=abs(XN);
subplot(2,1,2);
stem( f,2*XN(1:ceil((N-1)/2)+1) );
xlabel('Frequency/Hz');
axis([0 Fs/2 0 max(2*XN(1:ceil((N-1)/2)+1))+0.2]);
title('1024 Execution Point');
end
```

②实验结果：

N = 1024:



N = 2048:



实验五

1.问题描述：

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。

2.算法设计：

（1）复合梯形公式：

将区间[a, b]划分为 n 等分，分点 $x_k = kh$, $h = (b - a)/n$, $k = 0, 1, \cdots, n$, 在每个子区间$[x_k, x_{k+1}]$ $(k = 0, 1, \cdots, n-1)$上采用梯形公式，则得

$$I = \int_a^b f(x)\,\mathrm{d}x = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)\,\mathrm{d}x$$

$$= \frac{h}{2}\sum_{k=0}^{n-1}[f(x_k) + f(x_{k+1})] + R_n(f)$$

记

$$
\begin{aligned}
T_n &= \frac{h}{2}\sum_{k=0}^{n-1}[f(x_k)+f(x_{k+1})] \\
&= \frac{h}{2}\quad f(a)+2\sum_{k=1}^{n-1}f(x_k)+f(b)
\end{aligned}
$$

称为复合梯形公式。

（2）复合辛普森公式：

将区间[a, b] 分为 n 等分，在每个子区间[x_k , x_{k+1}] 上采用辛普森公式，

若记 $x_{k+1/2} = x_k + 1/2*h$，则得

$$
\begin{aligned}
I &= \int_a^b f(x)\,\mathrm{d}x = \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)\,\mathrm{d}x \\
&= \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+1/2})+f(x_{k+1})]+R_n(f)
\end{aligned}
$$

记

$$
\begin{aligned}
S_n &= \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+1/2})+f(x_{k+1})] \\
&= \frac{h}{6}\quad f(a)+4\sum_{k=0}^{n-1}f(x_{k+1/2})+2\sum_{k=1}^{n-1}f(x_k)+f(b)
\end{aligned}
$$

称为复合辛普森公式。

3.数值实验：

（1）复合梯形公式：

①实验源码：

```
function ans = CTF(left, right, steps)
syms x;
f(x) = sin(x)/x;
lv = sin(left)/left;
if isnan(lv)
    lv = limit(f(x),x,left,'right');
```

```
end
rv = sin(right)/right;
if isnan(rv)
    rv = limit(f(x),x,right,'left');
end
h = (right-left)/steps;
sum = (lv+rv)/2;
for i=1 : steps-1
    sum = sum + f(left+i*h);
end
ans = sum * h;
vpa(ans, 6);
end
```

②实验结果：

命令行窗口

```
>> CTF(0,1,5)

ans =

0.945079

>> CTF(0,1,9)

ans =

0.945773

>> CTF(0,1,17)

ans =

0.945996

>> CTF(0,1,33)

ans =

0.94606
```

（2）复合辛普森公式：

①实验源码：

```
function ans = CSPS(left, right, steps)
syms x;
f(x) = sin(x)/x;
lv = sin(left)/left;
if isnan(lv)
    lv = limit(f(x),x,left,'right');
```

```
end
rv = sin(right)/right;
if isnan(rv)
    rv = limit(f(x),x,right,'left');
end
h = (right-left)/steps;
sum = lv+rv+4*f(left+h/2);
for i=1 : steps-1
    sum = sum + 4*f(left+i*h+h/2) + 2*f(left+i*h);
end
ans = sum * h / 6;
vpa(ans,6);
end
```

②实验结果：

```
命令行窗口
>> CSPS(0,1,5)

ans =

0.946083

>> CSPS(0,1,9)

ans =

0.946083

>> CSPS(0,1,17)

ans =

0.946083

>> CSPS(0,1,33)

ans =

0.946083
```

4.结果分析

通过以上两种方法，我们可以发现随着取点数的增加，结果逐渐靠近真实值，可见区间划分越细，结果越准确。

实验六

1.问题描述：

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。

2.算法设计：

（1）前向欧拉法：

一般地, 设已做出该折线的顶点 P_n，过 P_n（x_n，y_n）依方向场的方向再推进到 P_{n+1} (x_{n+1} , y_{n+1}) ,显然两个顶点 P_n , P_{n+1} 的坐标有关系

$$\frac{y_{n+1} - y_n}{x_{n+1} - x_n} = f(x_n, y_n),$$

$$y_{n+1} = y_n + hf(x_n, y_n)$$

这就是著名的欧拉公式。若初值 y_0 已知, 则依公式可逐步算出

$$y_1 = y_0 + hf(x_0, y_0),$$

$$y_2 = y_1 + hf(x_1, y_1),$$

$$\cdots$$

（2）后向欧拉法：

现有方程

$$y' = f(x, y)$$

对方程从 x_n 到 x_{n+1} 积分, 得

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t))\,\mathrm{d}t.$$

如果在上式中右端积分用右矩形公式 $hf(x_{n+1}, y(x_{n+1}))$ 近似，则得另一个公式

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}),$$

称为后向欧拉法。

（3）梯形方法：

现有方程

$$y' = f(x, y)$$

对方程从 x_n 到 x_{n+1} 积分，得

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t))\,\mathrm{d}t.$$

如果在上式中右端积分用梯形求积公式近似，并用 y_n 代替 $y(x_n)$，y_{n+1} 代替 $y(x_{n+1})$，则得

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})],$$

称为梯形方法。

（4）改进欧拉方法：

先用欧拉法求得一个初步的近似值，称为预报值，然后用它替代梯形法右端的 y_{i+1} 再直接计算 f_{i+1}，得到校正值 y_{i+1}，这样建立的预报-校正系统称为改进的欧拉格式：

$$\text{预测}\quad \bar{y}_{n+1} = y_n + hf(x_n, y_n),$$

$$\text{校正}\quad y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, \bar{y}_{n+1})]$$

它有下列平均化形式：

$$y_p = y_n + hf(x_n, y_n),$$
$$y_c = y_n + hf(x_{n+1}, y_p),$$
$$y_{n+1} = \frac{1}{2}(y_p + y_c).$$

3.数值实验：

（1）前向欧拉法：

①实验源码：

```
function [x, y] = ForwardEular(init, left, right, h)
x = left:h:right;
y = zeros(size(x));
y(1) = init;
for i=1:length(x)-1
    y(i+1) = y(i) + h*(y(i) - 2*x(i)/y(i));
end
end
```

②实验结果：

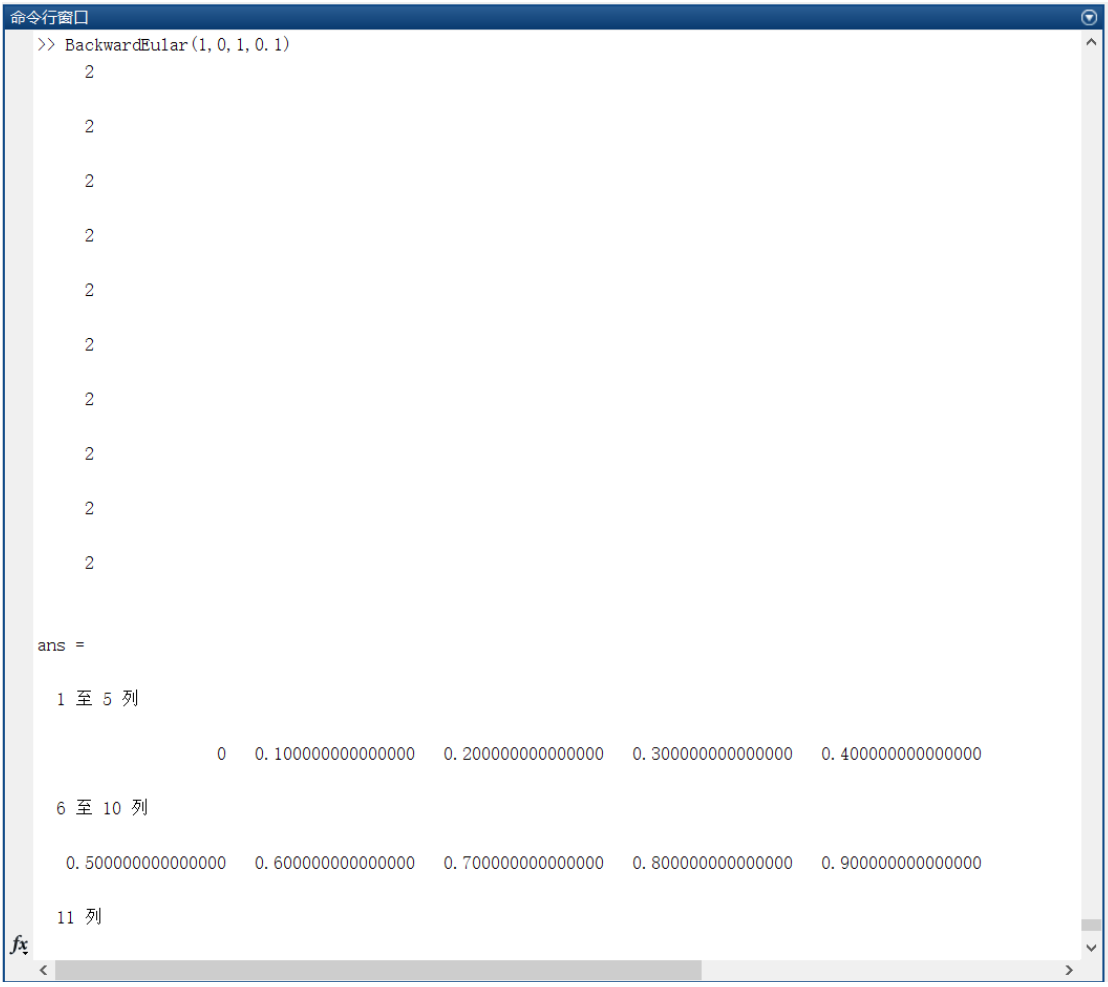
```
命令行窗口
>> ForwardEular(1, 0, 1, 0.1)

ans =

  1 至 2 列

                   0   0.100000000000000

  3 至 4 列

   0.200000000000000   0.300000000000000

  5 至 6 列

   0.400000000000000   0.500000000000000

  7 至 8 列

   0.600000000000000   0.700000000000000

  9 至 10 列

   0.800000000000000   0.900000000000000

  11 列

   1.000000000000000
```

（2）后向欧拉法：

①实验源码：

```
function [x, y] = BackwardEular(init, left, right, h)
MAX = 100;
eps = 10e-8;
x = left:h:right;
y = zeros(size(x));
y(1) = init;
for i=1:length(x)-1
    % implicit calculate formula
    tmp = y(i) + h*(y(i) - 2*x(i)/y(i));
    prev = tmp;
    for j=1:MAX
        y(i+1) = y(i) + h*(tmp - 2*x(i+1)/tmp);
        if abs(y(i+1) - prev) < eps
            disp(j);     %display iteration number
            break;
        end
        prev = y(i+1);
    end
end
end
```

②实验结果：



（3）梯形方法：

①实验源码：

```
function [x, y] = LadderShape(init, left, right, h)
MAX = 100;
eps = 10e-18;
x = left:h:right;
y = zeros(size(x));
y(1) = init;
for i=1:length(x)-1
    tmp = y(i) + h*(y(i) - 2*x(i)/y(i));
    prev = tmp;
    for j=1:MAX
        y(i+1) = y(i) + h/2*( (y(i)-2*x(i)/y(i)) + (tmp-2*x(i+1)/tmp) );
        if abs(y(i+1) - prev) < eps
            disp(j);            break;
        end
        prev = y(i+1);
    end
```

```
end
end
```

②实验结果：

```
命令行窗口
>> LadderShape(1, 0, 1, 0.1)
     2

     2

     2

     2

     2

     2

     2

     2

     2

     2


ans =

  1 至 5 列

                   0   0.100000000000000   0.200000000000000   0.300000000000000   0.400000000000000

  6 至 10 列

   0.500000000000000   0.600000000000000   0.700000000000000   0.800000000000000   0.900000000000000

  11 列
```

（4）改进欧拉方法：

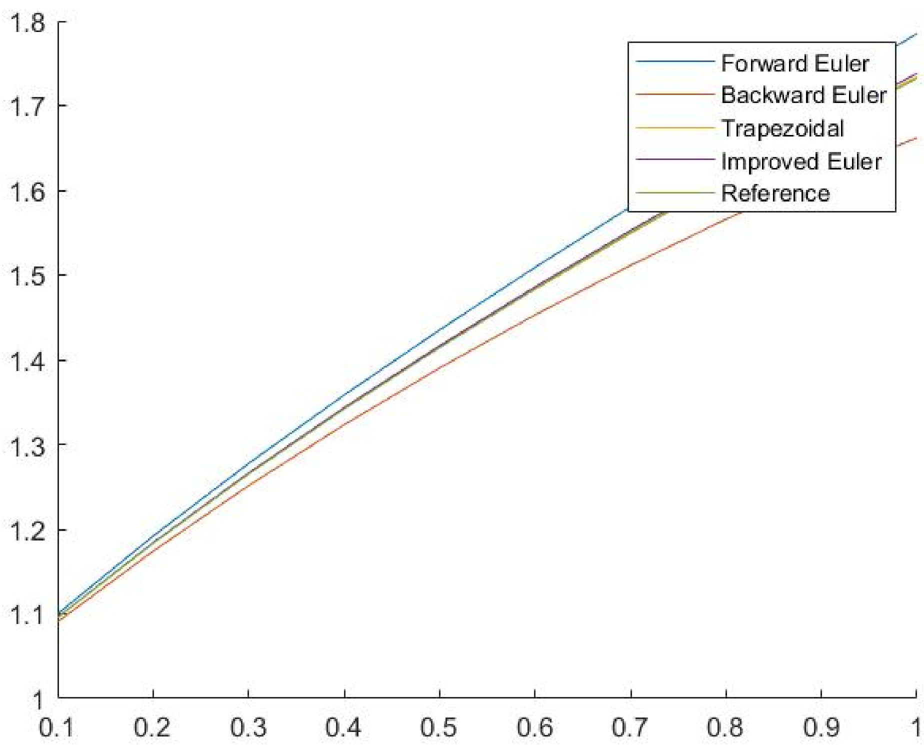
①实验源码：

```
function [x, y] = EularPro(init, left, right, h)
x = left:h:right;
y = zeros(size(x));
y(1) = init;
for i=1:length(x)-1
    y(i+1) = y(i) + h*(y(i) - 2*x(i)/y(i));
    yp = y(i+1);
    y(i+1) = y(i)+h/2*( (y(i)-2*x(i)/y(i)) + (yp-2*x(i+1)/yp) );
    yc = y(i+1);
    y(i+1) = (yp+yc)/2;
end
end
```

②实验结果：

```
命令行窗口
>> EularPro(1,0,1,0.1)

ans =

  1 至 5 列

                   0   0.100000000000000   0.200000000000000   0.300000000000000   0.400000000000000

  6 至 10 列

   0.500000000000000   0.600000000000000   0.700000000000000   0.800000000000000   0.900000000000000

  11 列

   1.000000000000000
```

4.结果分析：



由图，四条曲线都是逐步上升并靠近，最终都达到了了 1.7 附近，且四种方法得到的结果差距都不是很大。