目录

问题一

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。

算法设计

线性插值

即求一次多项式$\mathrm{p}(\mathrm{x})$，满足$\mathrm{p}(\mathrm{x}_0) = y_0, p(x_1) = y_1$

推出 $\frac{y-y_0}{x-x_0} = \frac{(y_1-y_0)}{(x_1-x_0)}$

从而得到 $p(x) = y_0 + (x - x_0)\frac{(y_1-y_0)}{x_1-x_0}$

重新整理得 $p(x) = y_0\frac{(x-x_1)}{x_0-x_1} + y_1\frac{(x-x_0)}{x_1-x_0}$

令 $l_0(x) = \frac{x-x_1}{x_0-x_1}, l_1(x) = \frac{x-x_0}{x_1-x_0}$

从而得到公式 $p(x) = y_0 l_0(x) + y_1 l_1(x)$

二次插值：

即求二次多项式$p(x)$，满足

$$p(x_{k-1})=y_{k-1},p(x_k)=y_k,p(x_{k+1})=y_{k+1},$$

通过计算得到

$$p(x)=y_{k-1}\frac{(x-x_k)(x-x_{k+1})}{(x_{k-1}-x_k)(x_{k-1}-x_{k+1})}+y_k\frac{(x-x_{k-1})(x-x_{k+1})}{(x_k-x_{k-1})(x_k-x_{k+1})}$$
$$+y_{k+1}\frac{(x-x_{k-1})(x-x_k)}{(x_{k+1}-x_{k-1})(x_{k+1}-x_k)}$$

三次插值

采用四个点，三次多项式

$$p(x)=\sum_{k=0}^{3}y_k\prod_{i=0,i\neq k}^{3}\frac{x-x_i}{x_k-x_i}$$

数值实验

直接根据公式，将已知点带入，在线性插值中，为了减小截断误差，选取插值点的邻接节点。

首先编写拉格朗日插值函数

```
%拉格朗日插值
%x,y 插值点坐标
%xi 求值点
function res= lagrange(x,y,xi)
  format long
  res = 0
  n = length(x);
  Lb = ones(1,n);
  for i = 1:n
    for j = 1:n
      if(j~=i)
        Lb(i)=Lb(i)*(xi-x(j))/(x(i)-x(j));
      end
    end
    res = res+Lb(i)*y(i);
  end
end
```

对于线性插值，将

```
x=[0.34,0.36]
```

```
y=[0.333487,0.352274]
```

二次插值

```
x=[0.32,0.34,0.36]
y=[0.314567,0.333487,0.352274]
```

三次插值

```
x=[0.32,0.34,0.36,0.38]
y=[0.314567,0.333487,0.352274,0.370920]
```

调用函数，得到结果。

```
lagrange(x,y,0.35)
```

结果分析

方法	结果
线性插值	0.34288050
二次插值	0.342897125
三次插值	0.342897625
精确解	0.342897807

根据结果可知，估计 sin(0.35)时，三次插值结果精确度最高，二次插值次之，线性插值最低。

问题二

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。

$$x^2 - 115 = 0$$

算法设计

二分法

二分法的思想很简单，就是每次将有根区间一分为二，得到长度逐次减半的区间序列$\{(a_k, b_k)\}$,则区间中点 $x_k = (a_k + b_k)/2$ 就是第 k 步迭代的近似解。

1. 计算$f(a), f(b)$，若$f(a)f(b) > 0$,则算法失效，停止计算
2. 令$x = \frac{a+b}{2}$, *计算* $f(x)$
3. 若$|f(x)| < \varepsilon$ 或 $|b - a| < \varepsilon$，停止计算，输出近似解 x
4. 若$f(a) * f(x) < 0$,则令$b = x$,否则令 $a = x$，返回第二步

牛顿法

a. 任取迭代初始值x_0
b. 计算$x_1 = x_0 - f(x_0)/f'(x_0)$
c. 判断收敛性，如果$|x_1 - x_0| < \varepsilon$ 或者 $|f(x_1)| < \varepsilon$，则算法收敛，停止计算，输出近似解x_1
d.令 $x_0 \leftarrow x_1$ 返回第二步

简化牛顿法

用 $f'(x_0)$ 替代牛顿法中的所有的 $f'(x_k)$

弦截法

设置初值 x_0, x_1
迭代公式

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

步骤与牛顿法相似，因此省略。

数值实验

二分法

```
%二分法
%n 为迭代步数
function [timestep,errors,x] = binary(f,a,b,n)
```

```
  solu = 115^(1/2);%准确值
  errors = zeros(1,n); %每步迭代的误差
  timestep = zeros(1,n); %每步迭代的时间
  tic
  for i=1:n
    if(feval(f,a)*feval(f,b)>0) %在该区间内无解
      break;
    end
    x=(a+b)/2;
    errors(i)= abs(solu-x);%计算绝对误差
    timestep(i)=toc %记录时间
    if(feval(f,a)*feval(f,x)<0)
      b = x;
    else
      a = x;
  end
end
```

牛顿法

牛顿法选择初始值为 11，设置精度为 0.000001

```
%牛顿法
function [timestep,errors,res] = Newton(f,a,b,n) %n 为迭代步数
  solu = 115^(1/2);
  errors = zeros(1,n);
  timestep = zeros(1,n);
  syms x;
  res = b; %设置初值
  tic
  for i=1:n
    res = res-feval(f,res)/subs(diff(f(x)),x,res);
    errors(i)= abs(solu-res);
    timestep(i)=toc %记录时间
    if(errors(i)<0.000001)
      break;
    end
  end
end
```

简化牛顿法

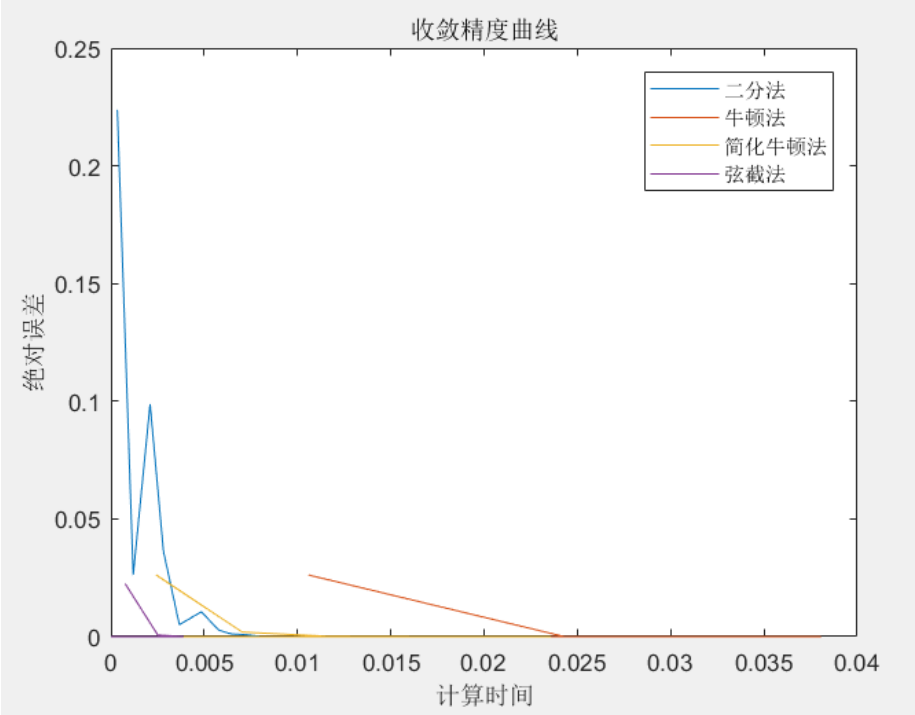
只贴出关键部分，其他部分和牛顿法相似

```
diffx0 = subs(diff(f(x)),x,a);
res = res-feval(f,res)/diffx0;
```

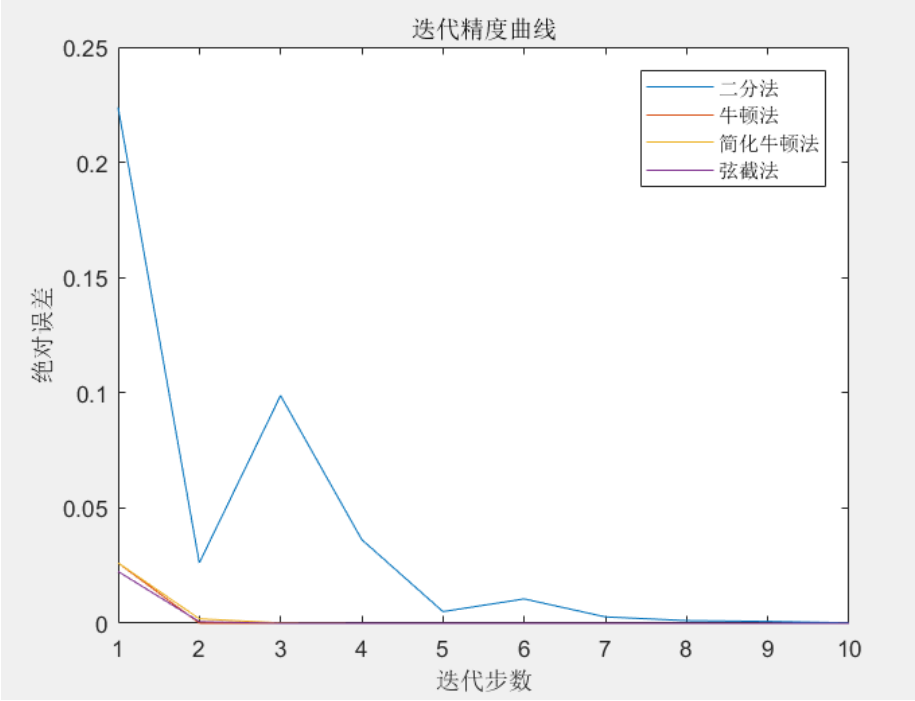
弦截法

```
%弦截法
function [timestep,errors,res] = Secant(f,a,b,n) %n 为迭代步数
  solu = 115^(1/2); %准确值
  errors = zeros(1,n); %每步的误差
  timestep = zeros(1,n); %每步的时间
  x0 = a; %设置初值 x_(k-1)
  x1 = a+0.1; %x_(k)
  tic
  for i=1:n
    res = x1-feval(f,x1)/(feval(f,x1)-feval(f,x0))*(x1-x0);
    x0 = x1;
    x1 = res;
    errors(i)= abs(solu-res);
    timestep(i)=toc %记录时间
    if(errors(i)<0.000001)
      break;
    end
  end
end
```

结果分析



从计算时间看，弦截法收敛最快，牛顿法收敛最慢，因为牛顿法每一步都要求导。简化牛顿法不需要，因此计算时间比牛顿法少。



从迭代步数，弦截法的收敛速度最快，二分法简单易用，总是收敛，但是收慢。弦截法收敛速度相对快，在此次实验中，牛顿法和简化法和简化牛顿法收敛速度相差不大。但我们知道牛顿法在根 x*附近是平方收敛，简化牛顿法是线性收敛的。

问题三

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。

算法设计

最小二乘法

考虑最小二乘问题，

$$\min_{x\in R^{m*n}} \|Ax-b\|_2{}^2$$

其中$A \in R^{n*m}$，$b \in R^{m}$，问题的解称为最小二乘解。

递推最小二乘法迭代步骤

$$A_k = a_k^T$$
$$P^{-1} = A^T A$$

迭代更新

$$\epsilon(n) = b(n) - x^H(n-1)A(n)$$
$$k(n) = \frac{P(n-1)a(n)}{1 + a^H(n)P(n-1)a(n)}$$
$$P(n) = P(n-1) - k(n)a^H(n)P(n-1)$$
$$x(n) = x(n-1) + k(n)\epsilon^*(n)$$

其中，k(n) 增益向量，ϵ(n)为估计误差。

数值实验

编写代码时，使用了 G 而不是 P，$G = A^T A$，

$$G_{k+1} = G_k + A_{k+1}^T A_{k+1}$$
$$x^{k+1} = x^k + G_{k+1}^{-1} A_{k+1}^T (b^{k+1} - A_{k+1}x^k)$$

算法代码：

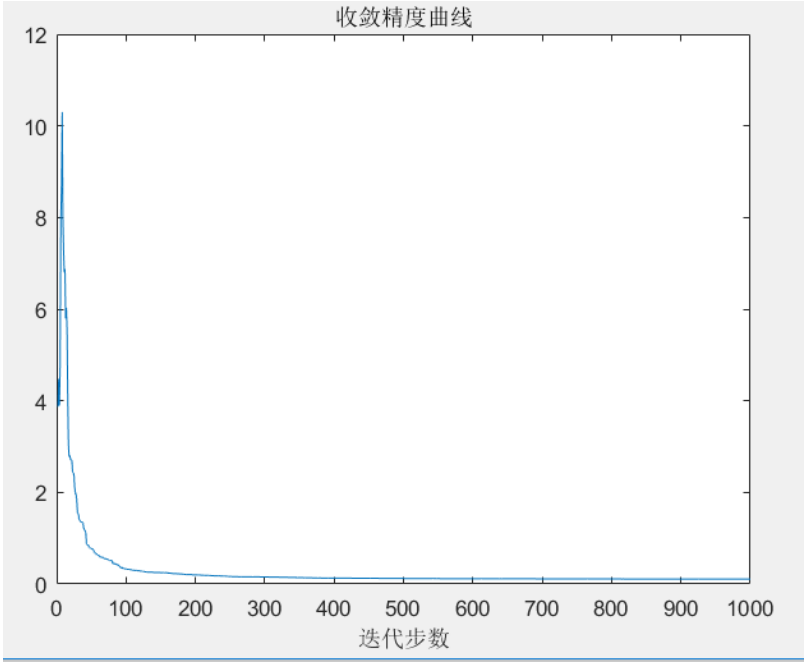
```
%递推最小二乘法
function [e,x] = rls(A,b)
  e = zeros(1,1000);
  m = size(A,1);
  G = A(1,:)'*A(1,:);
  x = inv(G)*A(1,:)'*b(1);
  e(1)=norm(A*x-b,2)/norm(b,2)
  for n=2:1000
```

```
    e(n)=norm(A*x-b,2)/norm(b,2);
    G = G+A(n,:)'*A(n,:);
    x = x+inv(G)*A(n,:)'*(b(n)-A(n,:)*x);
  end
```

使用 random 生成 A，b

```
A = random('Normal',100,10,1000,10);
b = random('Normal',100,10,1000,1);
```

结果分析



可以看到，递推最小二乘法一开始收敛速度较快。

问题四

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。

算法设计

快速傅里叶变换

ω_N^{jk} 具有周期性和对称性。

当 N = 2^p时，ω_N^{jk}只有 N/2 个不同的值，利用这些性质可将

$c_j = \sum_{k=0}^{N-1} x_k \omega_N^{jk}$ 对半折成两个和式，再将对应项相加。

有

$c_j = \sum_{k=0}^{\frac{N}{2}-1} x_k \omega_N^{jk} + \sum_{k=0}^{\frac{N}{2}-1} x_{N/2+k} \omega_N^{j(\frac{N}{2}+k)} = \sum_{k=0}^{\frac{N}{2}-1} [x_k + (-1)^j x_{N/2+k}] \omega_N^{jk}$

依下标奇偶分别考察，则

$$c_{2j} = \sum_{k=0}^{\frac{N}{2}-1} [x_k + x_{N/2+k}] \omega_{N/2}^{jk}$$

$$c_{2j+1} = \sum_{k=0}^{\frac{N}{2}-1} [x_k - x_{N/2+k}] \omega_N^k \omega_{N/2}^{jk}$$

若令

$y_k = x_k + x_{N/2+k}, y_{N/2+k} = (x_k - x_{\frac{N}{2}+k}) \omega_N^k$

则可将 N 点 DFT 归结为两个 N/2 点 DFT

$$\begin{cases} c_{2j} = \sum_{k=0}^{\frac{N}{2}-1} y_k \omega_{N/2}^{jk} \\ c_{2j+1} = \sum_{k=0}^{\frac{N}{2}-1} y_{N/2+k} \omega_{N/2}^{jk} \end{cases}$$

如此反复施行二分手续即刻得到 FFT 算法。

数值实验

根据上面的公式编写 FFT：

```
%快速傅里叶变换
function F = myfft(x)
%% 判断是否是向量
sz=size(x);
N=max(sz(1:2));
F=zeros(1,N);
for k=1:N/2
```

```
    [F(k),F(k+N/2)]=myfftEle(x,N,k);
end
```

通过二分法递归计算

```
function [Fk,Fkn]=myfftEle(x,N,k) %输入信号 x,信号总长度 N，频域坐标 k
  %递归的终止条件
  if N==1
    Fk=x;
    Fkn=x;
    return;
  else
    x1=x(1:2:N-1);%奇数
    x2=x(2:2:N);%偶数
    F1=myfftEle(x1,N/2,k);
    F2=myfftEle(x2,N/2,k);
    Wkn=exp(-i*2*pi*(k-1)/N);
    Fk=F1+Wkn*F2;
    Fkn=F1-Wkn*F2;
  end
```

使用 FFT 求噪声中隐藏的信号的频率分量，首先生成信号

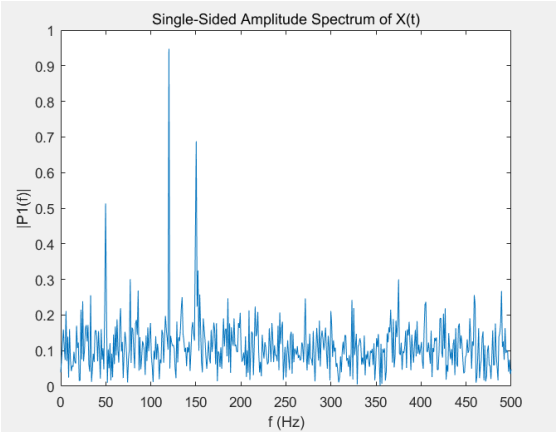
其中包含幅值为 0.7 的 50 Hz 正弦量和幅值为 1 的 120 Hz 和 150Hz 正弦量。

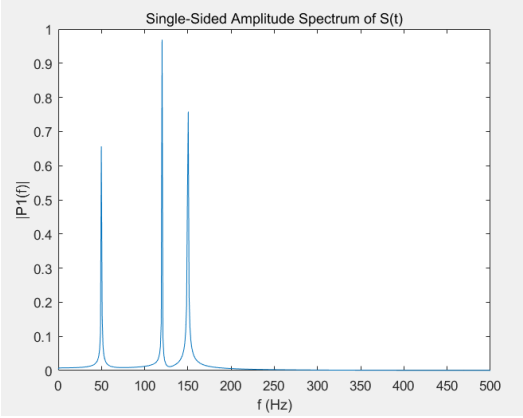
```
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t)+sin(2*pi*150*t);
%用均值为零、方差为 4 的白噪声扰乱该信号。
X = S + 2*randn(size(t));
```

调用 myfft 进行快速傅里叶变换。

详细测试代码在 run.m 中

结果分析





左边为加入干扰信号的，右边不加入干扰信号。可以看到在频率为 50Hz,100Hz，150Hz 时有峰值，因此通过算法可以得到原始信号的频率，实验结果正确。

问题五

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。

算法设计

复合梯形公式

将[a,b] 分成 n 个小区间$[x_i, x_{i+1}]$，其中

$$a = x_0 < x_1 < x \ldots < x_{n-1} < x_n = b$$

得到

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \approx \sum_{i=0}^{n-1} \frac{h_i}{2}[f(x_i) + f(x_{i+1})]$$

取等距节点，得到$\mathrm{T_n} = \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)]$，其中$h = \frac{b-a}{n}$

复合辛普森公式

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

取等距节点，$\mathrm{S_n} = \sum_{i=0}^{n-1} \frac{h}{6}[f(x_i) + 4f\left(x_{i+\frac{1}{2}}\right) + f(x_{i+1}] = \frac{h}{6}[f(a) + 4\sum_{i=0}^{n-1} f\left(x_{i+\frac{1}{2}}\right) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)$

数值实验

编写代码

复合梯形公式

```
%复合梯形
% f 为 函数 [a,b] 为区间,n 代表取点的个数
```

```
function res = compTrapez(f,a,b,n)
  h = (b-a)/n;
  d = 1;
  for i = a+h:h:b-h
    d = d+(2*feval(f,i));
  end
  res = (h/2)*(d+feval(f,b));
```

复合辛普森公式

```
f = inline('(sin(x))/x','x')
```

f(0) = 1
调用函数得到近似积分

结果分析

积分准确值：0.9460831

n	5	9	17	33
复合梯形公式结果	0.9450788	0.9457732	0.9459962	0.9460600
复合辛普森公式结果	0.9460832	0.9460831	0.9460831	0.9460831

由结果可得 n 取值越大，估计值越接近准确值。复合辛普森公式的精度比符合梯形公式的精度高。n=5 时的辛普森公式的结果比 n=33 时梯形公式的结果更准确。实验中的主要误差来自于计算机浮点运算中的截余。

问题六

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。

算法设计

前向欧拉方法

$$\begin{cases} y' = f(x,y) \\ y(x_0) = y_0 \end{cases}$$

任取一点，设为x_i,则有

$$\mathrm{y}'(\mathrm{x_i}) = f(x_i, y(x_i))$$

用差商公式 $\mathrm{y}'(\mathrm{x_i}) = y(x_{i+1}) - y(x_i)/(x_{i+1} - x_i)$ 代替导数项，设步长为 h，用 $\mathrm{y_i}$ 表示 $\mathrm{y}(\mathrm{x_i})$ 的近似值，得到公式：

$$\mathrm{y_{i+1}} = y_i + hf(x_{i+1}, y_{i+1})$$

后向欧拉方法

采用向后差

$y'(x_{i+1}) = y(x_{i+1}) - y(x_i)/(x_{i+1} - x_i)$ 代替导数项，用 $\mathrm{y_i}$ 表示 $y(x_i)$ 的近似值，得到公式：

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1})$$

采用迭代法求解，使用欧拉方法提供的初值。

$$\begin{cases} y_{n+1}^0 = y_n + hf(x_n, y_n) \\ y_{n+1}^{k+1} = y_n + hf\left(x_{n+1}, y_{n+1}^k\right) \end{cases}$$

梯形方法

梯形方法是欧拉法和向后欧拉法两者取平均。使用它求解时得到的相邻两个近似解连线的斜率正好是两个端点处解函数导数的算术平均.。

$$y(x_{n+1}) = y(x_n) + \frac{h}{2}[f\left(x_n, y(x_n)\right) + f\left(x_{n+1}, y(x_{n+1})\right)]$$

迭代公式，初值使用前向欧拉方法提供的初值

$$\begin{cases} y_{n+1}^0 = y_n + hf(x_n, y_n) \\ y_{n+1}^{k+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f\left(x_{n+1}, y_{n+1}^k\right)] \end{cases}$$

改进欧拉方法

先用前向欧拉公式求一个初步的近似值 $\bar{y}_{n+1}$,成为预测值，预测值$\bar{y}_{n+1}$ 的精度可能很差，再用梯形公式将它校正一次，这个结果成为校正值。

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

数值实验

编写算法代码

前向欧拉法

```
% 前向欧拉法
% f 为定义的函数 y0 为函数初值
%[a,b] 是求解区间
% h 为步长
function [x,y] = eulerForward(f,y0,a,b,h)
n = (b-a)/h;
y = ones(1,n);
y(1) = y0;
x = a:h:b;
for i=1:n
  y(i+1)= y(i)+h*feval(f,x(i),y(i));
end;
```

后向欧拉法

只列出关键的部分，其余部分和前向欧拉法相同。

```
%后向欧拉法
y(i+1)= y(i)+h*feval(f,x(i),y(i));
y(i+1)= y(i)+h*feval(f,x(i+1),y(i+1));
```

梯形法：

```
  y(i+1)= y(i)+h*feval(f,x(i),y(i));
  y(i+1)= y(i)+(h/2)*(feval(f,x(i),y(i))+feval(f,x(i+1),y(i+1)));
```

改进欧拉法

```
yp = y(i)+h*feval(f,x(i),y(i));
yr = y(i)+h*feval(f,x(i+1),yp);
y(i+1)= (1/2)*(yp+yr);
```

定义内敛函数

```
fun = inline('y-2*x/y','x','y')
```

分别调用四种算法得到结果，如

```
[x,y]=eulerForward(fun,1,0,1,0.1)
```

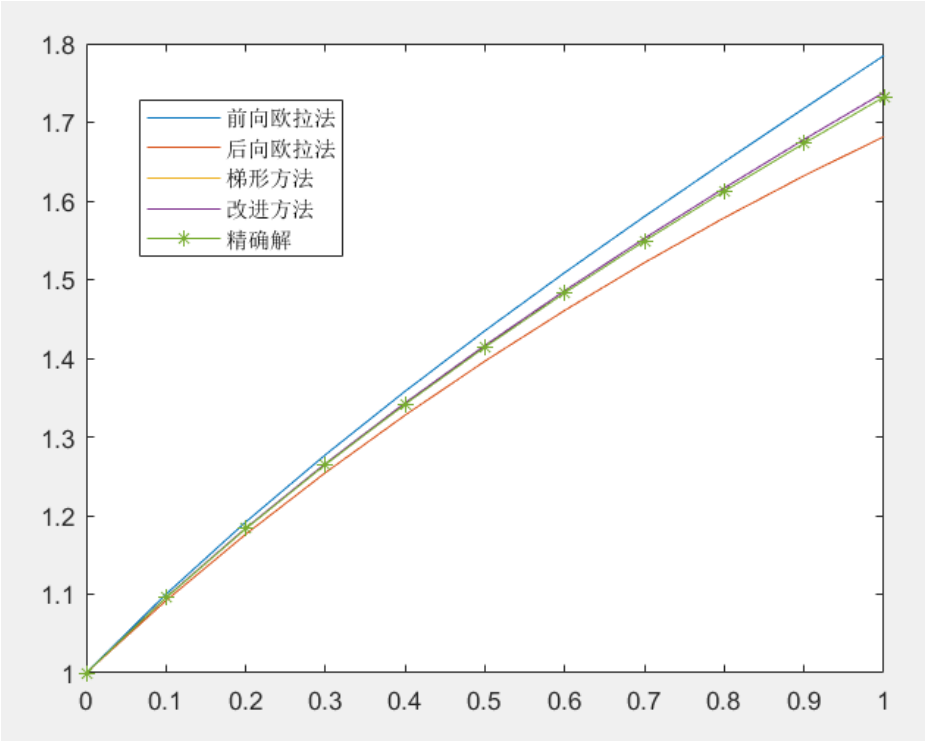
使用 dsolve 求解

```
f = dsolve('Dy==y-2*x/y','y(0)==1','x')
```

得 f = (2*x + 1)^(1/2)

结果分析

各迭代只进行了一次，因此梯形公式和改进欧拉法的结果相同。



由图中可看出梯形公式和改进欧拉法的精度比较高

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
前向欧拉法	1.1000	1.1918	1.2774	1.35821	1.43513	1.5090	1.58033	1.6498	1.7178	1.7848
后向欧拉法	1.09182	1.1763	1.2547	1.3278	1.3964	1.46094	1.52162	1.57864	1.63208	1.68188
梯形公式	1.09591	1.184097	1.266201	1.34336	1.41640	1.48596	1.55251	1.61647	1.67817	1.73787
改进欧拉法	1.09591	1.184097	1.266201	1.34336	1.41640	1.48596	1.55251	1.61647	1.67817	1.73787