实验报告一

题目： 非线性方程求解

摘要：非线性方程的解析解通常很难给出，因此线性方程的数值解法就尤为重要。本实验采用两种常见的求解方法二分法和牛顿法及改进的牛顿法。

前言：（目的和意义）

掌握二分法与牛顿法的基本原理和应用。

数学原理：

对于二分法，其数学实质就是说对于给定的待求解的方程$f(x)$，其在(a,b)上连续，$f(a)f(b)<0$，且$f(x)$在(a,b)内仅有一个实根x^*，取区间中点c，若，则c恰为其根，否则根据$f(a)f(c)<0$是否成立判断根在区间(a,c)和(c,b)中的哪一个，从而得出新区间，仍称为(a,b)。重复运行计算，直至满足精度为止。这就是二分法的计算思想。

牛顿法通常预先要给出一个猜测初值x_0，然后根据其迭代公式

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

产生逼近解x^*的迭代数列$\{x_k\}$，这就是牛顿法的思想。当x_0接近x^*时收敛很快，但是当x_0选择不好时，可能会发散，因此初值的选取很重要。另外，若将该迭代公式改进为

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

其中r为要求的方程的根的重数，这就是改进的牛顿法，当求解已知重数的方程的根时，在同种条件下其收敛速度要比牛顿法快的多。

拟牛顿法：

以x^0为初始近似值利用递推关系

$$x^{k+1}=x^k-H_kF(x^k)$$
$$H_{k+1}(F(x^{k+1})-F(x^k))=x^{k+1}-x^k$$
$$H_{k+1}=H_k+\Delta H_k \quad ,k=0,1,2,\cdots$$

产生近似于方程组$F(x)=0$的根x^*的迭代序列$\{x^k\}$，这个递推关系式就是拟牛顿法，实际计算时，只要选取较好的初始近似值x^0和初始矩阵H_0，一般可得到较好的近似解。

程序设计：

实验采用 Mathematica 编写源程序，并借助 Mathematica 强大的符号运算能力自动处理和简化部分问题，下面给出主要程序。

二分法：

```
dichotomy[f_,interval_,eps_]:=FixedPoint[
  With[{a=Min[#],b=Max[#],c=Mean[#]},
   Which[
    f[a]f[c]<0,{a,c},
    f[c]f[b]<0,{c,b},
    (*既不在左半区间，也不在右半区间，直接抛出中点*)
    True,Throw[{c}]
   ]
  ]&,interval,
  SameTest→(Abs[#[[1]]-#[[2]]]<eps&)
 ]//Catch//Mean
```

牛顿法及改进的牛顿法：

```
newton[f_,x0_,r_:1,maxIter_:1000]:=With[
  (*符号计算并化简牛顿法迭代函数*)
  {φ=x↦Evaluate@FullSimplify[x-r f[x]/f'[x]]},
  FixedPoint[φ,x0,maxIter,SameTest→Equal]
 ]
```

割线法：

```
secant[f_,{x0_,x1_},maxIter_:1000]:=NestWhile[
  With[{xkk=#[[1]],xk=#[[2]]},
   {xk,
    xk-f[xk]/(f[xk]-f[xkk])(xk-xkk)
   }]&,{x0,x1},
  Apply[Unequal],1,
  maxIter
 ][[2]]
```

拟牛顿法：

（求雅可比矩阵的辅助函数）

```
jacobian[f_, dim_] := With[{vars = Table[Unique["var$$"], dim]},
  Function[Evaluate@vars, Evaluate[D[f@@vars, {vars}]]]
 ]
```

```
quasiNewton[f_,dim_,x0_,maxIter_:1000]:=
 With[{h0=Inverse@*jacobian[f,dim]@@x0},
  FixedPoint[
   Block[{x=#[[1]],h=#[[2]],fx,xnext,r,y,rhy},
    fx=f@@x;
    xnext=x-h.fx;
    r=xnext-x;y=f@@xnext-fx;
    rhy=r.h.y;
    {xnext,
     h+KroneckerProduct[(r-h.y),r.h/rhy]}
    ]&,
   {x0,h0},maxIter,
   SameTest→(#1[[1]]==#2[[1]]&)
  ][[1]]
 ]
```

结果分析和讨论：

1. 用二分法计算方程 $\sin x-\frac{x^2}{2}=0$ 在 $(1,2)$ 内的根（$\varepsilon=0.5\times10^{-5}$）

```
In[•]:= dichotomy[x ↦ Sin[x] - x^2/2, {1., 2.}, 0.5*^-5]
Out[•]= 1.40441417694
```

2. 用牛顿法求解下列方程

a) $xe^x-1=0$，$x_0=0.5$；

```
In[•]:= newton[x ↦ x e^x - 1, 0.5]
Out[•]= 0.56714329041
```

b) $x^3-x-1=0$，$x_0=1$；

```
In[•]:= newton[x ↦ x^3 - x - 1, 1.]
Out[•]= 1.32471795724
```

c) $(x-1)^2(2x-1)=0$，$x_0=0.45$ 和 $x_0=0.65$；

```
In[•]:= newton[x ↦ (x - 1)^2 (2 x - 1), 0.45]
Out[•]= 0.5
In[•]:= newton[x ↦ (x - 1)^2 (2 x - 1), 0.65]
Out[•]= 0.5
```

3. 用割线法计算方程 $xe^x-1=0$ 在 $[0,1]$ 内的近似根，取初始点为 $x_0=0.4$ 以及 $x_1=0.6$

```
In[•]:= secant[x ↦ x e^x - 1, {0.4, 0.6}]
Out[•]= 0.56714329041
```

4. 用改进的牛顿法求解，有 2 重根，取 $r=2$

$(x-1)^2(2x-1)=0$，并与 3.中的 c)比较结果（$x_0=0.55$）。

当 x_0=0.55 时，在 10^6 次的最大迭代次数下

```
In[•]:= newton[x ↦ (x - 1)^2 (2 x - 1), 0.55, 2, 1*^6]
Out[•]= 0.500176649656
```

显然这个结果不是很好，而且也不是收敛至方程的 2 重根上。

实际上，牛顿法向方程的哪个根收敛与初值的选取密切相关，如图 1。可以看到大致以初值$x_0 \approx 0.67$为界，改进的牛顿法向方程的两个不同的根收敛。

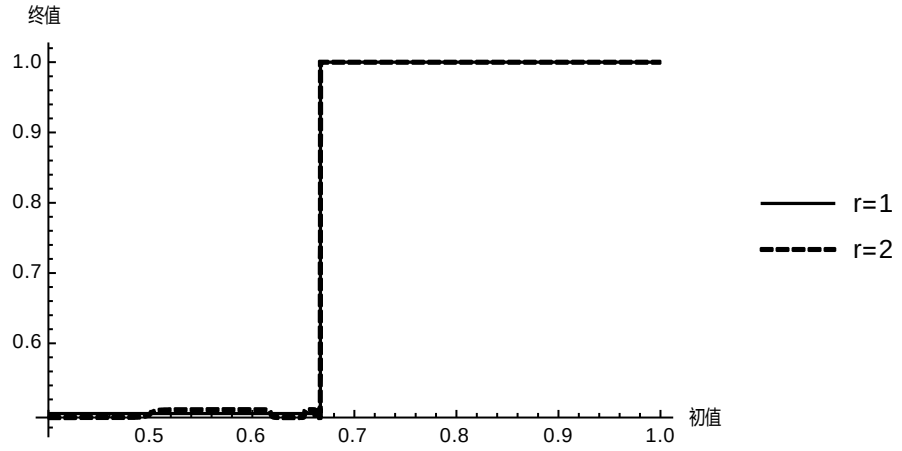

图 1 终值与初值间的关系

另一方面，作为比较，考虑不同初值时改进牛顿法的收敛速度，图 2 给出牛顿法（$r=1$）和改进的牛顿法（$r=2$）取不同初值时的迭代次数（最大迭代次数限制为 100）。对比图 1，可以发现改进的牛顿法在向二重根（$x^*=1$）收敛时确实显著地改善了收敛速度，但在向单根（$x^*=0.5$）收敛时，收敛速度则极为缓慢，甚至远远无法在 100 次迭代内收敛。

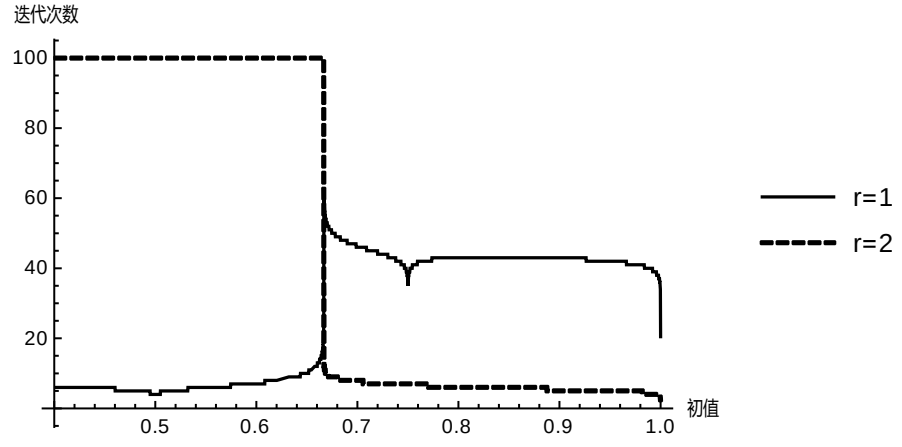

图 2 迭代次数与初值间的关系

这说明初值的选取很重要，直接关系到方法的收敛性。而只有选取合适初值使其向对应的重根收敛时，改进的牛顿法才能有效改善收敛速度。

5. 用拟牛顿法求解下列方程组

$$F(x,y,z)=\begin{bmatrix} xy-z^2-1 \\ xyz+y^2-x^2-2 \\ e^x+z-e^y-3 \end{bmatrix}=0$$

的近似解，取初始值$(x^0,y^0,z^0)=(1.0,1.0,1.0)$

```
In[•]:= quasiNewton[{x, y, z} ↦ {
          x y - z^2 - 1,
          x y z + y^2 - x^2 - 2,
          e^x + z - e^y - 3},
        3, {1., 1., 1.}]

Out[•]= {1.77767191801, 1.42396059789, 1.23747111773}
```

结论：

对于二分法，只要能够保证在给定的区间内有根，使能够收敛的，当时收敛的速度和给定的区间有关，二且总体上来说速度比较慢。牛顿法，收敛速度要比二分法快，但是最终其收敛的结果与初值的选取有关，初值不同，收敛的结果也可能不一样，也就是结果可能不时预期需要得结果。改进的牛顿法求解重根问题时，如果初值不当，可能会不收敛，这一点非常重要，当然初值合适，相同情况下其速度要比牛顿法快得多。

实验报告二

题目： Gauss 列主元消去法

摘要：求解线性方程组的方法很多，主要分为直接法和间接法。本实验运用直接法的 Guass 消去法,并采用选主元的方法对方程组进行求解。

前言：（目的和意义）

1. 学习 Gauss 消去法的原理。
2. 了解列主元的意义。
3. 确定什么时候系数阵要选主元

数学原理：

由于一般线性方程在使用 Gauss 消去法求解时，从求解的过程中可以看到，若 $a_{kk}^{(k-1)}=0$，则必须进行行交换，才能使消去过程进行下去。有的时候即使 $a_{kk}^{(k-1)} \neq 0$，但是其绝对值非常小，由于机器舍入误差的影响，消去过程也会出现不稳定得现象，导致结果不正确。因此有必要进行列主元技术，以最大可能的消除这种现象。这一技术要寻找行 r，使得

$$| a_{rk}^{(k-1)} |= \max_{i>k} \left| a_{ik}^{(k-1)} \right|$$

并将第 r 行和第 k 行的元素进行交换，以使得当前的 $a_{kk}^{(k-1)}$ 的数值比 0 要大的多。这种列主元的消去法的主要步骤如下：

1. 消元过程

对 k=1,2,…,n-1,进行如下步骤。

1) 选主元，记

$$| a_{rk} |= \max_{i>k} \left| a_{ik} \right|$$

若 $| a_{rk} |$ 很小，方程可能为病态。

2) 交换增广阵 A 的 r，k 两行的元素。

$$a_{rj} \leftrightarrow a_{kj} \qquad (j=k,\cdots,n+1)$$

3) 计算消元

$$a_{ij} = a_{ij} - a_{ik}a_{kj} / a_{kk} \qquad (i=k+1,\cdots,n;\ j=k+1,\cdots\cdots,n+1)$$

2. 回代过程

对 $k= n, n-1,\cdots,1$,进行如下计算

$$x_k = (a_{k,n+1} - \sum_{j=k-1}^{n} a_{kj}x_j / a_{kk})$$

至此，完成了整个方程组的求解。

程序设计：

实验采用 Mathematica 编写源程序，借助向量化运算简化表达。

Gauss 消去法：

```
eliminate[mat_?MatrixQ] := Module[{a = mat},
  Do[
   (*直接消元*)
   a[[k+1;;,k;;]]-=a[[k+1;;,{k}]].a[[{k},k;;]]/a[[k,k]],
   {k,Min@Dimensions@a - 1}
   ];
  a
 ]
```

Gauss 列主元消去法：

```
eliminatePE[mat_?MatrixQ] := Module[{a = mat},
  Do[
   With[(*找主元*)
    {r=k-1+First@Ordering[Abs@a[[k;;,k]], 1, GreaterEqual]},
    a[[{k,r},All]]=a[[{r,k},All]](*交换*)
   ];
   a[[k+1;;, k;;]]-=a[[k+1;;,{k}]].a[[{k},k;;]]/a[[k, k]],
   {k, Min@Dimensions@a - 1}
   ];
  a
  ]
```

回带求解过程：

```
eliminationSolve[a_?MatrixQ,b_?VectorQ,eliminator_]:=
 With[{n=Length[a^T],
  aa=eliminator[MapThread[Append,{a,b}]]},
  Module[{x=ConstantArray[0,n]},
   Do[(*回带*)
    x[[k]]=(aa[[k,n+1]]-Sum[x[[j]] aa[[k,j]],{j,1+k,n}])/aa[[k,k]],
    {k,n,1,-1}
   ];
   x
  ]
 ]
```

结果分析和讨论：

分别用高斯消去法和高斯列主元消去法求解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，其中

1. $\boldsymbol{A}=\begin{bmatrix}10^{-8} & 2 & 3\\ -1 & 3.712 & 4.623\\ -2 & 1.072 & 5.643\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}1\\2\\3\end{bmatrix}$

1) 高斯消去法

```
In[•]:= eliminationSolve[{{10.^-8, 2, 3}, {-1, 3.712, 4.623}, {-2, 1.072, 5.643}}, {1, 2, 3}, eliminate]

Out[•]= {-0.491058216312, -0.0508860701412, 0.367257381731}
```

2) 高斯列主元消去法

```
In[•]:= eliminationSolve[{{10.^-8, 2, 3}, {-1, 3.712, 4.623}, {-2, 1.072, 5.643}}, {1, 2, 3}, eliminatePE]

Out[•]= {-0.491058221222, -0.0508860774424, 0.367257386598}
```

可以看到尽管主对角线上有10^{-8}这样看似接近 0 的元素，但是否选取主元在这个情况下只影响到小数点后第 8 位以后的部分。事实上，即

使将其替换为10^{-15}，高斯消元法产生的误差才到 0.05 左右。

```
In[•]:= eliminationSolve[{{10.^-15, 2, 3}, {-1, 3.712, 4.623}, {-2, 1.072, 5.643}}, {1, 2, 3}, eliminate]

Out[•]= {-0.44408920985, 0.0178571428571, 0.321428571429}
```

2. $\boldsymbol{A}=\begin{bmatrix}4 & -2 & 4\\ -2 & 17 & 10\\ -4 & 10 & 9\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}10\\ 3\\ 7\end{bmatrix}$

1) 高斯消去法

```
In[•]:= eliminationSolve[{{4, -2, 4}, {-2, 17, 10}, {-4, 10, 9}}, {10, 3, 7}, eliminate]

Out[•]= {11/56, -25/28, 13/7}
```

2) 高斯列主元消去法

```
In[•]:= eliminationSolve[{{4, -2, 4}, {-2, 17, 10}, {-4, 10, 9}}, {10, 3, 7}, eliminatePE]

Out[•]= {11/56, -25/28, 13/7}
```

结论：

采用 Gauss 消去法时，如果在消元时对角线上的元素始终较大，那么本方法不需要进行列主元计算，计算结果一般就可以达到要求，否则必须进行列主元这一步，以减少机器误差带来的影响，使方法得出的结果正确。

实验报告三

题目： 龙格现象产生和克服

摘要：由于高次多项式插值不收敛，会产生龙格现象，本实验在给出具体的实例后，采用分段线性插值和三次样条插值的方法有效的克服了这一现象，而且还取的很好的插值效果。

前言：（目的和意义）

1. 深刻认识多项式插值的缺点。
2. 明确插值的不收敛性怎样克服。
3. 明确精度与节点和插值方法的关系。

数学原理：

在给定 $n+1$ 个节点和相应的函数值以后构造 n 次的拉格朗日插值多项式，实验结果表明（见后面的图）这种多项式并不是随着次数的升高对函数的逼近越来越好，这种现象就是龙格现象。

解决龙格现象的方法通常有分段线性插值、三次样条插值等方法。

分段线性插值：

设在区间$[a, b]$上，给定 n+1 个插值节点

$$a=x_0<x_1<\cdots<x_n=b$$

和相应的函数值 y_0，y_1，…，y_n，，求作一个插值函数 $\phi(x)$，具有如下性质：

3) $\phi(x_j)=y_j$，j=0，1，…，n。

4) $\phi(x)$ 在每个区间$[x_i, x_j]$上是线性连续函数。则插值函数 $\phi(x)$ 称为区间$[a, b]$上对应 n 个数据点的分段线性插值函数。

三次样条插值：

给定区间$[a, b]$一个分划

$\triangle$：$a=x_0<x_1<\cdots<x_N=b$

若函数 $S(x)$满足下列条件：

1) $S(x)$在每个区间$[x_i, x_j]$上是不高于 3 次的多项式。

2) $S(x)$及其 2 阶导数在$[a, b]$上连续。则称 $S(x)$使关于分划$\triangle$的三次样条函数。

程序设计：

本实验使用 Mathematica 编写程序，借助 Mathematica 繁多的语法糖简化表达。

拉格朗日插值：

```
lagrangianInterpolation[points:{{_,_}..}]:=
 With[{px=points[[All,1]],pfx=points[[All,2]],n=Length@points},
  x↦∑_(i=1)^n pfx[[i]]∏_(j=1)^n {(x-px[[j]])/(px[[i]]-px[[j]])  i≠j
                                 1                              True
 ]
```

分段线性插值：

```
piecewiseLinear[points:{{_,_}..}/;¬OrderedQ[points[[All,1]]]]:=
 (*保证输入点按横坐标顺序排列*)
 piecewiseLinear[points//SortBy[First]]
```

```
piecewiseLinear[points:{{_,_}..}][x_]:=
 With[{px=points[[All,1]],pfx=points[[All,2]],n=Length@points},
  Do[
   If[px[[i]]≤x≤px[[i+1]],(*在相邻两插值点间则计算并抛出结果*)
    Throw[(pfx[[i+1]]-pfx[[i]])/(px[[i+1]]-px[[i]])(x-px[[i]])+pfx[[i]]]
   ],
   {i,n-1}];
  Undefined(*插值区间外未定义*)
 ]//Catch
```

三次样条插值源程序：（三弯矩）

```
(*计算插值参数*)
spline3GetParams[pts_,type_,boundary_]:=
 Module[{px=pts[[All,1]],pfx=pts[[All,2]],
  h,nh,λ,μ,m,a,b},
  h=Differences[px];nh=Length[h];
  λ=Table[h[[i+1]]/(h[[i]]+h[[i+1]]),{i,nh-1}];
  μ=1-λ;
  m=spline3GetM[type,px,pfx,h,nh,μ,λ,boundary];
  b=pfx[[;;-2]]-h^2/6 m[[;;-2]];
  a=Table[(pfx[[i+1]]-pfx[[i]])/h[[i]]-h[[i]]/6 (m[[i+1]]-m[[i]]),{i,nh}];
  {px,pfx,h,m,a,b}
 ]
```

```
(*差商*)
dq[_,ys_List,0]:=ys
dq[xs_List,ys_List,n_]/;Length[xs]==Length[ys]:=
 Differences[dq[xs,ys,n-1]]/Differences[xs,1,n]
```

```
(*第一边界条件*)
spline3GetM[1,px_,pfx_,h_,nh_,μ_,λ_,{dfa_,dfb_}]:=
 Module[{cof=SparseArray[{Band[{1,1}]→2,
    Band[{2,1}]→μ,{nh+1,nh}→1,
    Band[{2,3}]→λ,{1,2}→1
   },nh+1],
  d=6dq[px,pfx,2]},
  PrependTo[d,6/h[[1]] (pfx[[2]]-pfx[[1]])/h[[1]]-dfa)];
  AppendTo[d,6/h[[-1]] (dfb-(pfx[[-1]]-pfx[[-2]])/h[[-1]])];
  LinearSolve[cof,d]
 ]
```

```
(*第二边界条件*)
spline3GetM[2,px_,pfx_,h_,nh_,μ_,λ_,{d2fa_,d2fb_}]:=
 Module[{cof=SparseArray[{Band[{1,1}]→2,
    Band[{2,1}]→μ[[2;;]],Band[{1,2}]→λ[[;;-2]]
   },nh-1],
  d=6dq[px,pfx,2],m},
  d[[{1,-1}]]-={μ[[1]]d2fa,λ[[-1]]d2fb};
  m=LinearSolve[cof,d];
  m~Prepend~d2fa~Append~d2fb
 ]
```

```
spline3[points:{{_,_}..},type:1|2,boundary:{_,_}:{0,0}]:=
 With[{n=Length[points],
 params=spline3GetParams[SortBy[points,First],type,boundary]},
 With[{px=params[[1]],pfx=params[[2]],h=params[[3]],
 m=params[[4]],a=params[[5]],b=params[[6]]},
  x↦Catch[
   Do[
    If[px[[i]]≤x≤px[[i+1]],(*在相邻两插值点间则计算并抛出结果*)
     Throw[(px[[i+1]]-x)^3/(6h[[i]])m[[i]]+(x-px[[i]])^3/(6h[[i]])m[[i+1]]+a[[i]](x-px[[i]])+b[[i]]]
    ]
    ,{i,n-1}];
   Undefined(*插值区间外未定义*)
  ]
 ]]
```

结果分析和讨论：

本实验采用函数 $f(x)=\dfrac{1}{1+25x^2}$ 进行数值插值，插值区间为[-1，1]，给定节点为 $x_j=-1+jh, h=0.1$，$j=0,\cdots$，n。下面分别给出拉格朗日插值，三次样条插值，线性插值的函数曲线。

1. 拉格朗日插值：

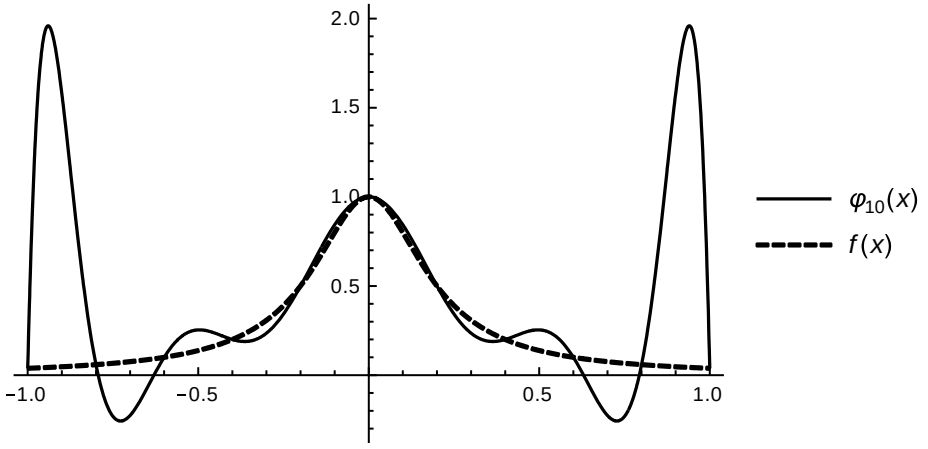

图 1 拉格朗日插值函数与原函数

2. 分段线性插值结果：

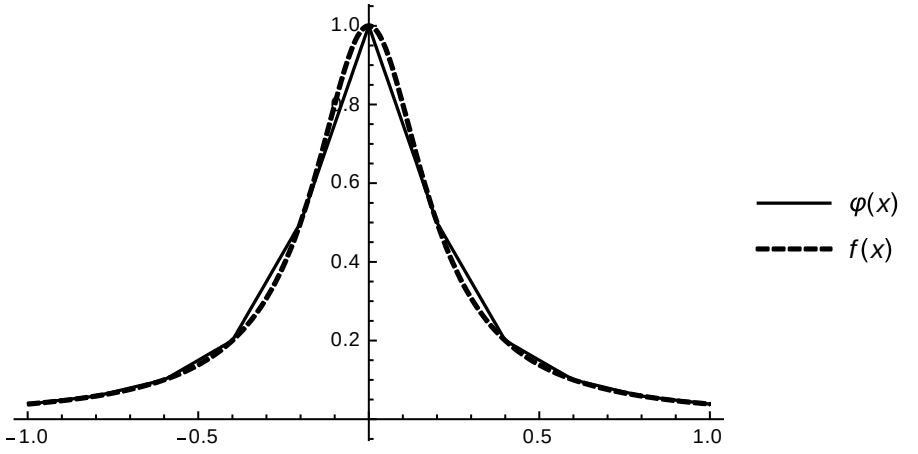

图 2 分段线性插值函数与原函数

3. 三次样条插值结果：

由于三次样条插值曲线与原函数曲线图像上基本重合，肉眼已经无法分辨，这里就不给出这两者的图像了。作为对比，下面给出样条插值和原函数的差值的图像，以及两种不同的边界条件下样条差值的结果之差。

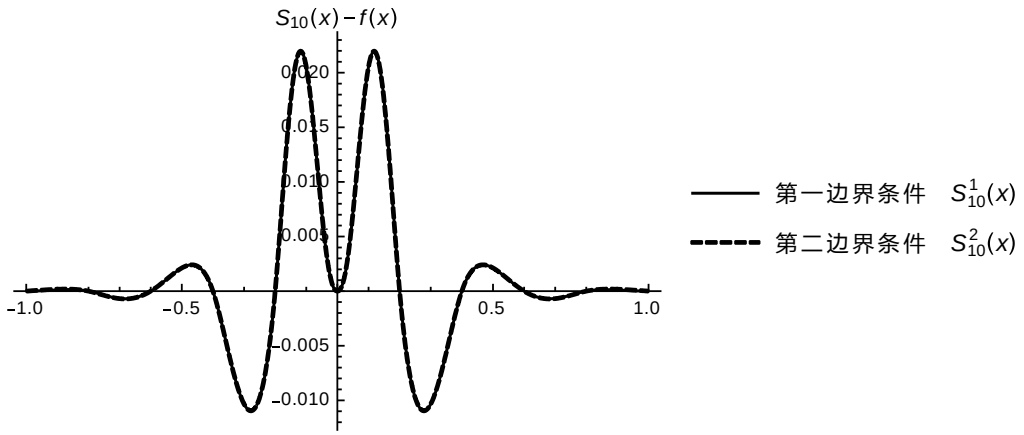

图 3 三次样条插值函数与原函数间的差

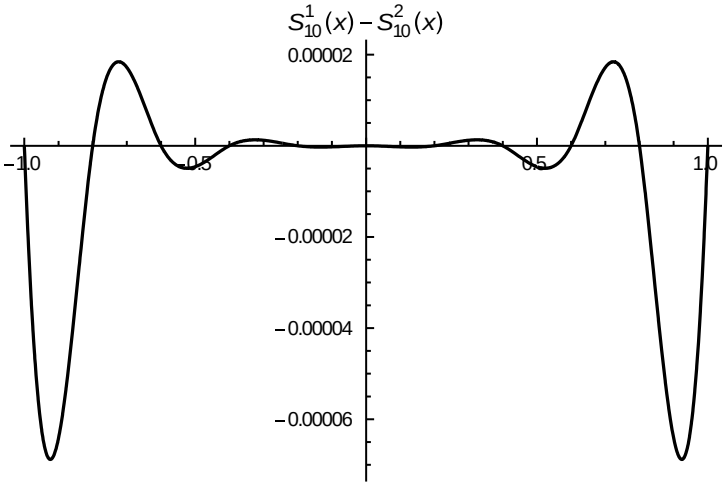

图 4 两种不同边界条件下三次样条插值结果之差

从以上结果可以看到，用三次样条插值和线性分段插值，不会出现多项式插值是出现的龙格现象，插值效果明显提高。进一步说，为了提高插值精度，用三次样条插值和线性分段插值是可以增加插值节点的办法来满足要求，而用多项式插值函数时，节点数的增加必然会使多项式的次数增加，这样会引起数值不稳定，所以说这两种插值要比多项式插值好的多。而且在给定节点数的条件下，三次样条插值的精度要优于线性分段插值，曲线的光滑性也要好一些。

实验报告四

题目： 龙贝格积分法

摘要：对于实际的工程积分问题，很难应用-莱布尼兹公式去求解。因此应用数值方法进行求解积分问题已经有着很广泛的应用，本文基于龙贝格积分法来解决一类积分问题。

前言：（目的和意义）

1. 理解和掌握龙贝格积分法的原理；
2. 学会使用龙贝格积分法；
3. 明确龙贝格积分法的收敛速度及应用时容易出现的问题。

数学原理：

考虑积分 $I(f)=\int_a^b f(x)dx$，欲求其近似值，通常有复化的梯形公式、辛普森公式和科特斯公式。但是给定一个精度，这些公式达到要求的速度很缓慢。如何提高收敛速度，自然是人们极为关心的课题。为此，记 $T_{1,k}$ 为将区间[*a*,*b*]进行 2^k 等分的复化的梯形公式计算结果，记 $T_{2,k}$ 为将区间[*a*,*b*]进行 2^k 等分的复化的辛普森公式计算结果，记 $T_{3,k}$ 为将区间[*a*,*b*]进行 2^k 等分的复化的科特斯公式计算结果。根据 *Richardson* 外推加速方法，可以得到收敛速度较快的龙贝格积分法。其具体的计算公式为：

1. 准备初值，计算

$$T_{1,1}=\frac{a-b}{2}[f(a)+f(b)]$$

2. 按梯形公式的递推关系，计算

$$T_{1,k+1}=\frac{1}{2}T_{1,k}+\frac{b-a}{2^k}\sum_{i=0}^{2^{k-1}-1}f(a+\frac{b-a}{2^{k-1}}(i+0.5))$$

3. 按龙贝格积分公式计算加速值

$$T_{m,k-m}=\frac{4^{m-1}T_{m-1,k+1-m}-T_{m-1,k-m}}{4^{m-1}-1} \qquad m=2,\cdots,\ k$$

4. 精度控制。对给定的精度 R，若

$$\left|T_{m,1}-T_{m-1,1}\right|<R$$

则终止计算，并取 $T_{m,1}$ 为所求结果；否则返回 2 重复计算，直至满足要求的精度为止。

程序设计：

本实验采用 Mathematica 编写。

龙贝格积分法：

```
romberg[f_,{a_,b_},monitor_:None]:=
 Module[{k=1,h=b-a,n=1,t={(b-a)/2. (f[a]+f[b])}},
  While[
   monitor[t];
   (*梯形法递推*)
   AppendTo[t,t[[-1]]/2+h/2 Sum[f[a+h(i+1/2)],{i,0,n-1}]];
   (*外推加速*)
   Do[t[[k-m+1]]=(4^m t[[k-m+2]]-t[[k-m+1]])/(4^m-1),{m,1,k}];
   h/=2;n*=2;++k;
   t[[1]]≠t[[2]]
  ];
  monitor[t];
  t[[1]]
 ]
```

```
incompleteTranspose[m_]:=(*辅助函数-不完全数组转置*)
 PadRight[m,{Length[m],Max[Length/@m]},"p"]^T/.{"p"→Nothing}
(*从原始数据构造T-数表*)
makeTTable=incompleteTranspose@*incompleteTranspose;
```

结果分析和讨论：

1. 求积分 $\int_6^{100} x^3 dx$。精确解 I= 24999676。

```
In[•]:= {result, {t}} = romberg[x ↦ x^3, {6, 100}, Sow] // Reap;
        makeTTable[t] // TableForm // Style[#, Medium] &
        result

Out[•]= 4.7010152×10^7    3.0502295×10^7    2.637533075×10^7
        2.4999676×10^7    2.4999676×10^7
        2.4999676×10^7

Out[•]= 2.4999676×10^7
```

龙贝格积分给出的结果为2.4999676×10^7，与精确解相同，没有误差。

2. 求积分 $\int_0^1 \frac{\sin x}{x} dx$。

直接按前面方法进行积分，会发现系统报错：“碰到不定表达式”。出现这种情况的原因就是当$x=0$时，被积函数分子分母都出现了 0，但我们知道该点其实是一个可去间断点，通过分段函数补充这点的定义就可以避免这个问题。

```
In[•]:= {result, {t}} = romberg[x ↦ {Sin[x]/x   x ≠ 0
                                    1           True}, {0, 1}, Sow] // Reap;
        makeTTable[t] // TableForm // Style[#, Small] &
        result

Out[•]= 0.920735492404    0.939793284806    0.944513521665    0.945690863583    0.945985029934    0.946058560963
        0.946145882274    0.946086933952    0.946083310888    0.946083085385    0.946083071306
        0.946083004064    0.946083069351    0.946083070351    0.946083070367
        0.946083070387    0.946083070367    0.946083070367
        0.946083070367    0.946083070367
        0.946083070367

Out[•]= 0.946083070367
```

故该函数的积分约为 0.946083060367。

3. 求积分 $\int_0^1 \sin x^2 dx$

本题的解析解很难给出，但运用龙贝格积分可以很容易给出近似解：

```
In[ ]:= {result, {t}} = romberg[x ↦ Sin[x^2], {0, 1}, Sow] // Reap;
        makeTTable[t] // TableForm // Style[#, Tiny] &
        result
```

```
Out[ ]=
0.420735492404    0.334069725829    0.315975360759    0.311680239481    0.310620366809    0.310356260655    0.310290287874
0.305181136971    0.309943905736    0.310248532388    0.310267075919    0.31026822527     0.310268296948
0.310261423654    0.310268840832    0.310268312154    0.310268301893    0.310268301726
0.310268958565    0.310268303763    0.31026830173     0.310268301723
0.310268301195    0.310268301722    0.310268301723
0.310268301723    0.310268301723
0.310268301723
```

```
Out[ ]= 0.310268301723
```

故该函数的积分约为 0.310268301723。

结论：

龙贝格积分通常要求被积函数在积分区间上没有奇点。如有奇点，且奇点为第一间断点，那么采用例 2 的方法，还是能够求出来的，否则，可能需要采用其它的积分方法。当然，龙贝格积分的收敛速度还是比较快的。