Matlab实验

**实验1-1** Mupad基本操作



ln(-3)：在未加假设的情况下，MuPAD默认在复数范围内计算，但是在复数范围内没有公式

**实验1-2**

**1**、在同一个图形中绘制正弦和反正弦函数的图像

要求：正弦曲线为蓝色划线，线宽0.5；反正弦曲线为绿色点线，线宽1:  
【f1 := plot::Function2d(sin(x), x = 0..2\*PI,LineStyle=Dotted,LineWidth=0.5,Color=[0,0,1]);  
 f2 := plot::Function2d(arcsin(x), x = 0..2\*PI,LineStyle=Dashed,LineWidth=1,Color = [0,1,0]);  
 plot(f1,f2,GridVisible = TRUE,Scaling = Constrained);  
**2**、在同一个图形中绘制指数函数 y=exp(x) 与对数函数 y=ln(x) 的图形，要求绘制的函数图形的部分关于一三象限对角线 y=x 严格对称，对角线用蓝色线绘制。  
【f1 := plot::Function2d(exp(x), x = -2..2,Color=[0,1,0]);  
 f2 := plot::Function2d(ln(x), x = -2..2,Color = [1,0,0]);  
 f3:=plot::Function2d(x,x = -2..2,Color = [0,0,1]);  
 plot(f1,f2,f3)

**实验1-3**  
**1**、阿基米德螺线：

【curve := plot::Curve2d([r\*sin(r), r\*cos(r)], r = 0..35);

plot(curve)

**2**、三维曲线：

【curve := plot::Curve3d([sin(thet)\*cos(20\*thet),sin(thet)\*sin(20\*thet),cos(thet)],

thet = 0..a, a = 0..PI):

plot(curve)

**3**、绘制了方程(x-y)\*(x+y)=0所确定的隐函数的图像：

【plot(plot::Implicit2d((x-y)\*(x+y), x = -1..1, y = -1..1))  
4、心形线：

plot(plot::Polar([2\*(1-cos(r)),r], r = 0..2\*PI))  
或者plot(plot::Polar([arccos(sin(r)),r], r = 0..2\*PI))  
5、P1 := plot::Polar([r, 4\*r^2], r = 0..PI, Mesh = 400);

plot(P1)圈圈  
6、绘制点：

p1 := plot::Point2d(1, 3, PointSize = 4\*unit::mm,Color= RGB::Blue,PointStyle=Squares);

p2 := plot::Point2d(2, 2, PointSize = 5\*unit::mm,Color= [0.5,0.5,0.5],PointStyle=Circles);

p3:=plot::Point2d(3,1,Color= RGB::Green, PointSize = 6\*unit::mm,PointStyle=FilledDiamonds);

plot(p1, p2, p3)  
7、绘制点列：  
PL1 := plot::PointList2d([[1,1],[2,4],[3,9],[4,16],[5,25]],PointStyle=Stars);

plot(PL1)  
8、画圆：（方程）  
f1:=plot::Function2d(sqrt(1-x^2),x=-1..1):

f2:=plot::Function2d(-sqrt(1-x^2),x=-1..1):

plot(f1,f2,#C)

9、画圆：（参数方程）

curve:=plot::Curve2d([sin(r),cos(r)],r=0..2\*PI); 注:极坐标方程： x=cosα

plot(curve,#C) y=sinα  
10、画圆：（隐函数）

plot(plot::Implicit2d(x^2+y^2=1,x=-1..1,y =-1..1),#C) 注:函数方程：x^2+y^2=1

11、画圆：（参数方程）

plot(plot::Polar([1,r], r = 0..2\*PI),#C) 注:参数方程:p=1

12、玫瑰线：

f1:=plot::Polar([a\*sin(3\*r),r], r = 0..2\*PI,a=0..1);

plot(f1) //三叶玫瑰

f2:=plot::Polar([a\*cos(2\*r),r], r = 0..2\*PI,a=0..1);

plot(f2) //四叶玫瑰

实验1-4 数列极限的几何解释

round向最近的偶数取整

一、

1、floor 返回向负无穷方向的下一个整数（下取整）.

系统说明：floor rounds a number to the next smaller integer.

2、ceil返回向正无穷方向的下一个整数（上取整）

系统说明：ceil rounds a number to the next larger integer.

3、round返回最接近的整数（四舍五入）

系统说明：round rounds a number to the nearest integer.

4、trunc返回向0的方向最近的整数（截断尾数取整）

系统说明：trunc rounds a number to the next integer in the direction of 0.

1. 绘制线段：

L2:=plot::Line2d([-3,-3],[3,3],LineStyle = Dashed,LineWidth = 2.5\*unit::mm,

LineColor = RGB::Green):

plot(L2,#C)

1. 绘制矩形：

P1 := plot::Rectangle(-2..2, -2..2, Filled = TRUE,FillColor = RGB::Red);

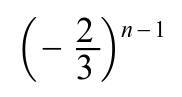
P2 := plot::Rectangle(1..5, 1..5, Filled = FALSE,LineColor = RGB::Black,LineStyle = Dashed);

plot(P1,P2)  
3、自变量趋向无穷大可以用关键字infinity来表示：limit((1 + 1/n)^n, n = infinity)

4、绘制数列图像：

S1:=plot::Sequence((-1)^n/(n+1)^2,n=1..20);

plot(S1);

5、求当n趋于无穷时极限为0的几何解释

delete n;

k:=1; //k取正整数

epsilon:=1/10^k;//为了方便地调整ε的值，令ε等于1/10^k，当整数k的取值增加时，ε的值迅速减小

N:=floor(ln(epsilon)/ln(2/3)+1);//N对epsilon的依赖关系

x:=n->(-2/3)^(n-1);

x(N-1),x(N),x(N+1);//细致地观察第N项前后的变化

S1:=plot::Sequence(x(n),n=N-1..N+20);//函数图像

L1:=plot::Line2d([N,-epsilon],[N+20,-epsilon],LineStyle=Dashed);

L2:=plot::Line2d([N,epsilon],[N+20,epsilon],LineStyle=Dashed);

L3:=plot::Line2d([N,-epsilon],[N,epsilon],LineStyle=Dashed);

plot(S1,L1,L2,L3);//绘制S1,L1,L2,L3的图像

**实验1-5 函数极限的几何解释**

1. 无穷大

设函数f(x)=1/(x-1)，给出当x趋向于1时f(x)以无穷大为极限的几何解释。

首先绘制函数f(x)在1附近的图像：

plot(plot::Function2d(1/(x-1), x=-5..7),

#C, ViewingBox=[-5..7, -4..4]);

根据证明的思路，运行以下MuPAD语句，绘图给出几何解释：

k:=6: // k是非负整数

M:=10^k;

//通过k来控制M的大小

delta:=1/M;

//根据函数的性质给出delat和M之间的关系

F:= plot::Function2d(1/(x-1), x=1-2\*delta..1+2\*delta):

L1:= plot::Line2d([1-delta,-3\*M], [1-delta,3\*M], LineStyle=Dashed):

L2:= plot::Line2d([1+delta,-3\*M], [1+delta,3\*M], LineStyle=Dashed):

L3:= plot::Line2d([1-2\*delta,M], [1+2\*delta,M], LineStyle=Dashed):

L4:= plot::Line2d([1-2\*delta,-M], [1+2\*delta,-M], LineStyle=Dashed):

plot(F,L1,L2,L3,L4,ViewingBox=[1-2\*delta..1+2\*delta, -3\*M..3\*M]);

1. 用MuPAD给出函数y=x\*sin(x)当x趋向于负无穷大时不是无穷大的几何

limit(x\*sin(x),x=-infinity);//使用limit函数来计算该极限

X:=-15;

//X是任意正数

plot(plot::Function2d(x\*sin(x),x=X..X+2\*PI),

plot::Line2d([X,1],[X+2\*PI,1],LineStyle=Dashed),

plot::Line2d([X,-1],[X+2\*PI,-1],LineStyle=Dashed),

ViewingBox=[X..X+2\*PI,-2..2]);

要求：从方便理解你的求解思路和代码作用的角度开辟文本区域或使用注释来说明你的探索过程.

函数y=x\*sin(x)当x趋向于负无穷大时不是无穷大的几何解释： 对任意正数X，f(x)=x\*sin(x)在(X，X+2\*PI]总恰有两个零点，任取其中一个零点为x0，就有|x0 sin(x0)|=0<M0=1。所以函数y=xsin(x)当x趋向于负无穷大时不是无穷大。

1. 用MuPAD给出f(x)=2x-1当x趋向于1时以1为极限的几何解释

limit(2\*x-1,x=1);//使用limit函数来计算该极限

k:=2;

//k取非负整数

epsilon:=1/2^k;

//通过k来控制epsilon的大小

delta:=1/2\*epsilon;

//根据函数的性质给出delat和epsilon之间的关系

F1:=plot::Function2d(2\*x-1,x=1/2..3/2);

//绘制函数的图像

P1:=plot::Point2d([1,1],PointStyle=Circles);

//极限点

R1:=plot::Rectangle(1-delta..1+delta,1-epsilon..1+epsilon,

LineStyle=Dashed);

//函数在给定点处的epsilon带和delat带

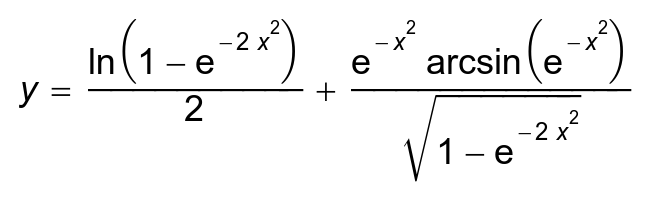
L1:=plot::Line2d([1,1-epsilon],[1,1+epsilon],LineStyle=Dashed);

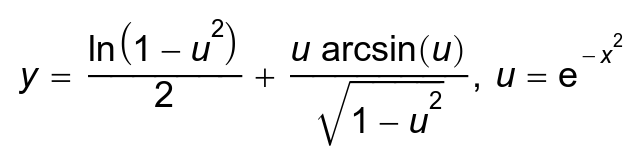
plot(F1,P1,R1,L1,ViewingBox=[1/2..2,0..2]);

f(x)=2x-1当x趋向于1时以1为极限的几何解释：对任意的正数eplison()（通过改变k值实现），存在相应的delta(),只要x在1的delta相邻领域内，函数值f(x)的值就在相邻领域内（函数图像在表示1-epsilon和1+epsilon的两条水平蓝色虚线之内）

**实验1-6 导数的计算**

diff(sin(x),x$n) $ n=1..8;

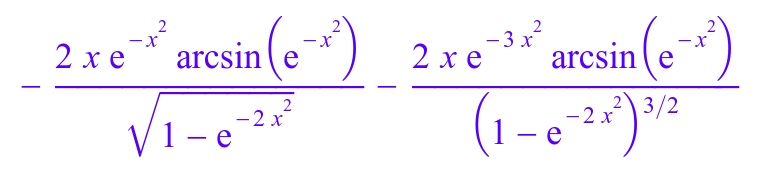
1. 对函数求导，验证复合函数求导法则.

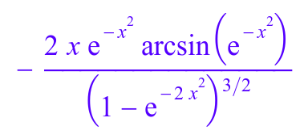
理论分析：该函数可以看作的复合

步骤一 直接由函数表达式对x求导：

delete x;

diff(1/2\*ln(1-E^(-2\*x^2))+E^(-x^2)\*arcsin(E^(-x^2))/sqrt(1-E^(-2\*x^2)), x); Simplify(%)



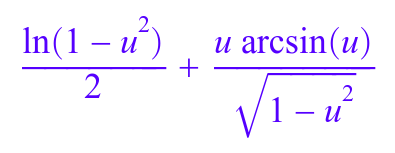


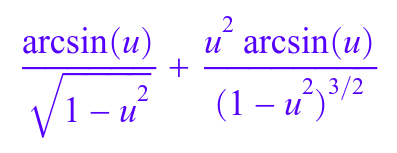
步骤二:由被复合的两个函数表达式分别对u和x求导，然后相乘，验证复合函数求导法则：

delete u,x,y;

y:=1/2\*ln(1-u^2)+u\*arcsin(u)/sqrt(1-u^2);

dy\_du:=diff(y, u);

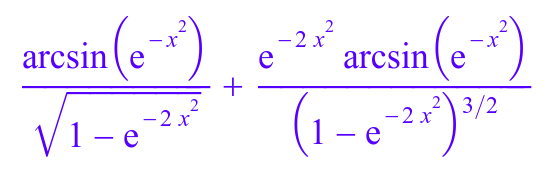




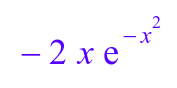
u:=E^(-x^2);

dy\_du;

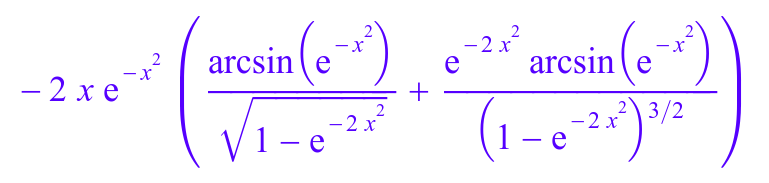




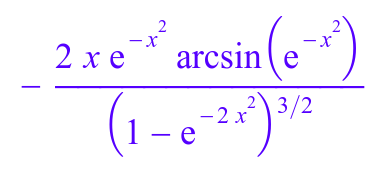
du\_dx:=diff(u, x);

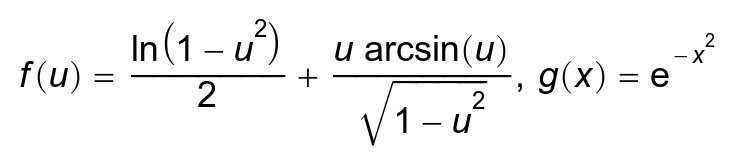


dy\_du\*du\_dx;



Simplify(%)；

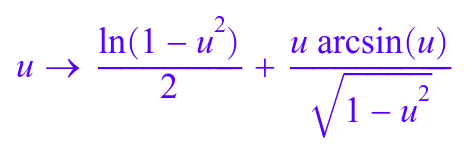


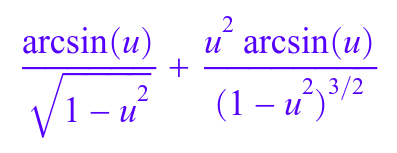
步骤三：视作，两个函数的复合，采用映射形式，分别求导，然后相乘，验证复合函数求导法则：

delete u,x,f,g;

f:=u-->1/2\*ln(1-u^2)+u\*arcsin(u)/sqrt(1-u^2);

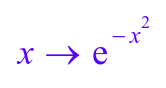
dy\_du:=f'(u);

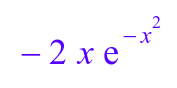




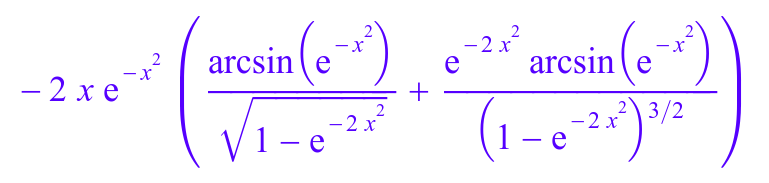
g:=x-->E^(-x^2);

du\_dx:=g'(x);

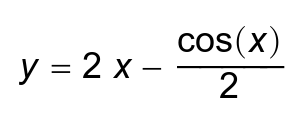


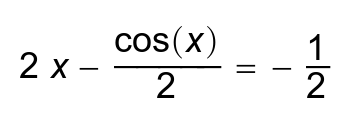


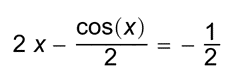
f'(g(x))\*g'(x);



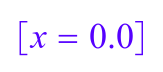
Simplify(%);

1. 反函数求导求函数的反函数在y=-1/2处的导数

步骤1 求解非线性方程的符号解，解不出来  
solve(2\*x-cos(x)/2=-1/2, x);

求解非线性方程的数值解

numeric::fsolve(2\*x-cos(x)/2=-1/2, x);

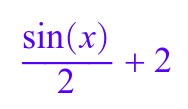


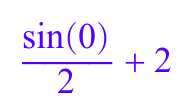
步骤3 用反函数求导公式计算所求：

diff(2\*x-cos(x)/2, x);

subs(%, x=0);

1/Simplify(%);







步骤4 绘函数图像验证上述解答的正确性：

首先，绘制函数的图像：

plot(2\*x-cos(x)/2, x=-5..5);

然后，由于反函数y=f-1(x)的图像与原函数y=f(x)的图像关于直线y=x对称，

可以用plot::Reflect2d绘制反函数的图像：

Gf:=plot::Function2d(2\*x-cos(x)/2, x=-3..3, Color=RGB::Blue):

P:=plot::Point2d([0,-1/2], Color=RGB::Blue):

Gi:=plot::Reflect2d([0, 0], [1, 1],

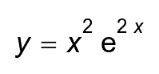
plot::modify(Gf, Color=RGB::Red)):

Q:=plot::Point2d([-1/2,0], Color=RGB::Red):

L:=plot::Function2d(x, x=-4..4, Color=RGB::Black, LineStyle=Dashed):

plot(Gf, P, Gi, Q, L, #C, ViewingBox=[-4..4,-4..4],

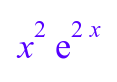
Height=120, Width=120);

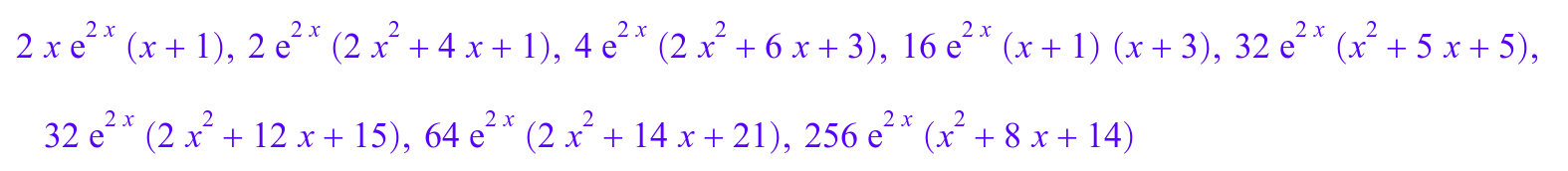
1. 设，求n阶导数并化简

delete x;

y:=x^2\*E^(2\*x);

Simplify(diff(y,x$n))$ n=1..8; //先观察一般规律

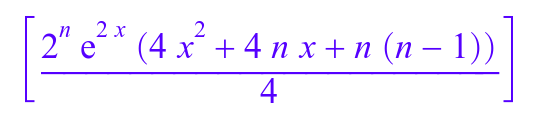




用莱布尼茨公式可以得出:n阶导数为：n\*(n-1)+4\*n\*x+4\*(x^2)]\*2^(n-2)\*e^(2\*x)

delete x,n;

Simplify([n\*(n-1)+4\*n\*x+4\*(x^2)]\*2^(n-2)\*E^(2\*x)); //化简由莱布尼茨公式得到的结果

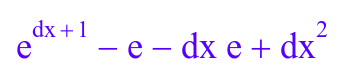


**实验1-7 微分的几何意义**

1、微分的几何意义

步骤1：计算函数在x=1处的函数值，导数值，微分值；

计算在(1+dx)处函数值与函数的微分之间的误差：simplify(f(1+dx)-f(1)-f'(1)\*dx);



步骤2：绘制动画，演示函数的微分以及误差项如何随着自变量的微分的变化而改变.

下面的代码中：

1）GF是函数图像，P是定点，PT是切线；

2）DX是表示自变量的微分dx的水平线段，绿色，动画参数dx从-1变化到1；

3）DY是表示函数的微分=的铅直线段，蓝色，动画参数dx从-1变化到1；

4）Er是表示误差项f(1+dx)-f(1)-f'(1)\*dx=的铅直线段，红色半透明，2毫米宽，动画参数dx从-1变化到1；

Gf:=plot::Function2d(f(x), x=0..2, Color=RGB::Black):

P:=plot::Point2d([1,f(1)], Color=RGB::Black):

PT:=plot::Function2d(f(1)+f'(1)\*(x-1), x=0..2,

LineStyle=Dashed,Color=RGB::Black):

DX:=plot::Line2d([1,f(1)], [1+dx,f(1)], dx=-1..1,

Color=RGB::Green):

DY:=plot::Line2d([1+dx,f(1)], [1+dx,f(1)+f'(1)\*dx], dx=-1..1,

Color=RGB::Blue):

Er:=plot::Line2d([1+dx,f(1)+f'(1)\*dx], [1+dx,f(1+dx)], dx=-1..1,

LineWidth=2, Color=RGB::Red.[0.5]):

plot(Gf,P,PT,DX,DY,Er,#C);

步骤3：绘制图像，表示微分三角形：

1）自变量的微分dx可以取任意的非零实数值，可正可负；

2）当dx>0时，微分三角形在定点P的右侧，图形的横坐标范围宜取;

当dx<0时，微分三角形在定点P的左侧，图形的横坐标范围宜取;

3）微分三角形由DX、DY、DS三线段组成，是直角三角形，无论dx如何取值，都是相似的三角形；

由于选取的横坐标范围总是dx的两倍，并且将微分三角形安放在居中的位置，还通过#C选项，

设置两个坐标轴具有相等的单位长度，所以无论输入dx任何值，看到的微分三角形是全等的；

4）DX是表示dx的线段，DY是表示dy的线段，DX与DY垂直；

5）DS是表示弧微分ds的线段，DS总是落在切线上，是直角的对边；

6）随着dx变小，误差项趋于零（实际上是dx的高阶无穷小项）,图像演示了化曲为直的思想.

dx:=0.003: // dx可取任意的非零实数

xmin:=piecewise([dx>0,1-0.5\*dx],[dx<0,1+1.5\*dx]):

xmax:=piecewise([dx>0,1+1.5\*dx],[dx<0,1-0.5\*dx]):

Gf:=plot::Function2d(f(x), x=xmin..xmax, Color=RGB::Black):

P:=plot::Point2d([1,f(1)], Color=RGB::Black, PointSize=2):

PT:=plot::Function2d(f(1)+f'(1)\*(x-1), x=xmin..xmax,

LineStyle=Dashed, Color=RGB::Black):

DX:=plot::Line2d([1,f(1)], [1+dx,f(1)],

Color=RGB::Green, LineWidth=1):

DY:=plot::Line2d([1+dx,f(1)], [1+dx,f(1)+f'(1)\*dx],

LineWidth=1, Color=RGB::Blue):

DS:=plot::Line2d([1,f(1)], [1+dx,f(1)+f'(1)\*dx],

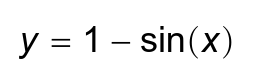
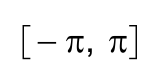
LineWidth=1, Color=RGB::Orange.[0.7]):

Er:=plot::Line2d([1+dx,f(1)+f'(1)\*dx], [1+dx,f(1+dx)],

LineWidth=2, Color=RGB::Red.[0.5]):

plot(Gf,P,PT,DX,DY,DS,Er,#C);

2、切线和法线

演示在上的任意点的切线和法线

步骤1：定义

delete f,x,dx,X;

f:=x->1-sin(x);

步骤2：

1）大写X是函数的自变量；小写x是动画参数.

2）动点P坐标(x, f(x))，动切线PT方程Y=f(x)+f'(x)\*(X-x)，动法线PN方程Y=f(x)-1/f'(x)\*(X-x).

3）在绘得的动画中，切线和法线落在坐标区域的部分全部被显示，长度随着切点运动而变化；

4）plot命令用#C设置了x轴、y轴的单位长度相等，所以切线和法线看起来总是互相垂直，否则观察不到这种关系.

Gf:=plot::Function2d(f(X), X=-PI..PI, Color=RGB::Black):

P:=plot::Point2d([x, f(x)], x=-PI..PI):

PT:=plot::Function2d(f(x)+f'(x)\*(X-x), X=-4.5..4.5, x=-PI..PI):

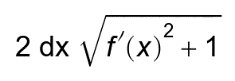
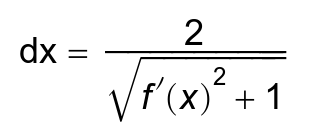
PN:=plot::Function2d(f(x)-1/f'(x)\*(X-x), X=-4.5..4.5, x=-PI..PI):

plot(Gf, P, PT, PN, #C, ViewingBox=[-4.5..4.5, -2..4]);

步骤3：改为利用微分重新绘制动画.

1）根据微分的几何意义，连接点(x-dx, f(x)-f'(x)\*dx)和(x+dx, f(x)+f'(x)\*dx)的线段恰好是切线段，中点(x, f(x))恰好是切点；

2）切线段直角旋转后可得，连接点(x+f'(x)\*dx, f(x)-dx)和(x-f'(x)\*dx, f(x)+dx)的线段恰好是法线段，中点(x, f(x))恰好是法线与曲线的交点.

3）如上的切线段和法线段长度等于，所以如果取，切线段和法线段长度就等于4

dx:=2/sqrt(1+f'(x)^2):

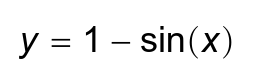
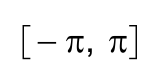
Gf:=plot::Function2d(f(X), X=-PI..PI, Color=RGB::Black):

P:=plot::Point2d([x, f(x)], x=-PI..PI):

PT:=plot::Line2d([x-dx, f(x)-f'(x)\*dx], [x+dx, f(x)+f'(x)\*dx], x=-PI..PI):

PN:=plot::Line2d([x+f'(x)\*dx, f(x)-dx], [x-f'(x)\*dx, f(x)+dx], x=-PI..PI):

plot(Gf, P, PT, PN, #C,ViewingBox=[-4.5..4.5, -2..4]);

1. 演示函数在上任意点的微分三角形

delete f,x,dx,X;

f:=x->1-sin(x);//函数表达式

dx:=0.5: //dx可取任意的非零实数

xmin:=piecewise([dx>0,1-0.5\*dx],[dx<0,1+1.5\*dx])://当dx>0时,微分三角形在定点P的右侧,图形的横坐标范围宜取;

xmax:=piecewise([dx>0,1+1.5\*dx],[dx<0,1-0.5\*dx])://当dx<0时,微分三角形在定点P的左侧,图形的横坐标范围宜取;

Gf:=plot::Function2d(f(X), X=-PI..PI, Color=RGB::Black)://绘制函数图像

P:=plot::Point2d([x,f(x)], x=-PI..PI,Color=RGB::Black, PointSize=2)://绘制点X

PT:=plot::Function2d(f(x)+f'(x)\*(X-x), X=-4.5..4.5, x=-PI..PI,

LineStyle=Dashed, Color=RGB::Black)://绘制切线图像

DX:=plot::Line2d([x,f(x)], [x+dx,f(x)],x=-PI..PI,

Color=RGB::Green, LineWidth=1)://绘制水平直角边即自变量的微分dx

DY:=plot::Line2d([x+dx,f(x)], [x+dx,f(x)+f'(x)\*dx],x=-PI..PI,

LineWidth=1, Color=RGB::Blue)://绘制铅直直角边恰好函数即因变量的微分dy

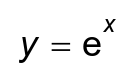
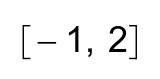
DS:=plot::Line2d([x,f(x)], [x+dx,f(x)+f'(x)\*dx],x=-PI..PI,

LineWidth=1, Color=RGB::Orange.[0.7])://绘制表示弧微分ds的线段

Er:=plot::Line2d([x+dx,f(x)+f'(x)\*dx], [x+dx,f(x+dx)],x=-PI..PI,

LineWidth=2, Color=RGB::Red.[0.5])://绘制误差项的铅直线段

plot(Gf,P,PT,DX,DY,DS,Er,#C,ViewingBox=[-4..4, -2.3..2.3]);//绘制动画

1. 演示经过曲线在上任意点的法线和切线

delete f,x,dx;

f:=x->E^x;//函数表达式

Gf:=plot::Function2d(f(X), X=-4..2, Color=RGB::Black)://函数表达式

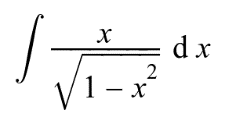
P:=plot::Point2d([x, f(x)], x=-1..2)://点

PT:=plot::Function2d(f(x)+f'(x)\*(X-x), X=-2..2, x=-1..2)://切线

PN:=plot::Function2d(f(x)-1/f'(x)\*(X-x), X=-2..2, x=-1..2)://法线

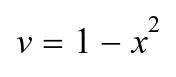
plot(Gf, P, PT, PN, #C, ViewingBox=[-4..4, -8..8]);//绘制图像

**实验1-8 不定积分**

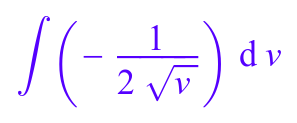
1. 第一类换元法：

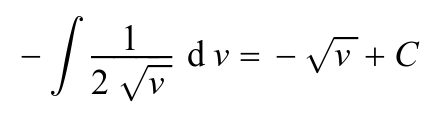
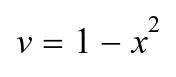
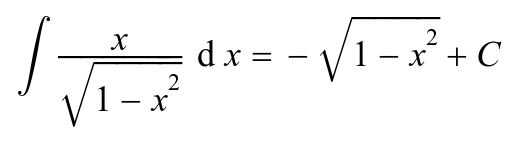
步骤1：用int直接计算：

int(x/sqrt(1-x^2),x);

第2步，用命令intlib::changevar计算换元

intlib::changevar(hold(int(x/sqrt(1-x^2),x)), v=1-x^2, v);

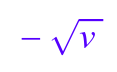


由不定积分的基本公式可得，然后回代，即可得

第3步，用命令int和subs完成以上计算.

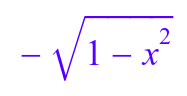
先把第2步换元后得到的新的不定积分复制粘贴到下面这个输入区域，然后按回车键计算：

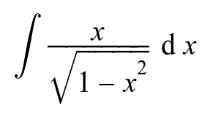
int(-1 / (2 \* v^(1/2)), v);

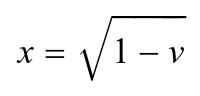


接着用命令subs回代：

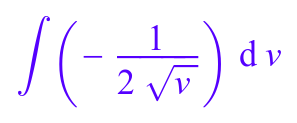
subs(%, v = 1 - x^2);

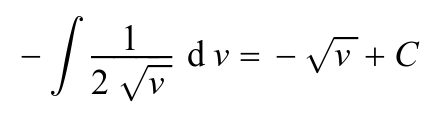
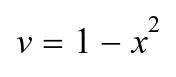
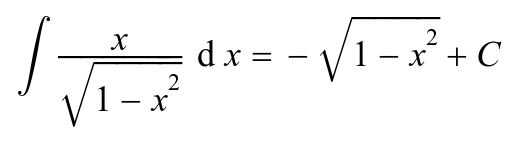


1. 第二类换元法：

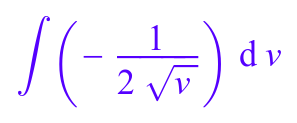
方法一，换元.

intlib::changevar(hold(int(x/sqrt(1-x^2),x)), x=sqrt(1-v), v);



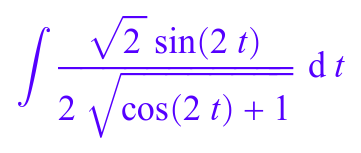
，回代，即得

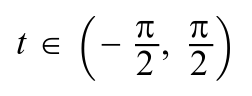
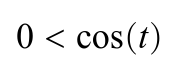
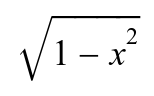
intlib::changevar(hold(int(x/sqrt(1-x^2),x)), x=-sqrt(1-v), v);



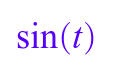
方法二，换元x=sin（t）

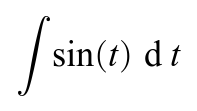
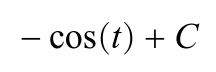
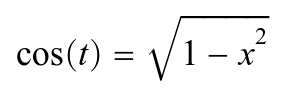
intlib::changevar(hold(int(x/sqrt(1-x^2),x)), x=sin(t), t);



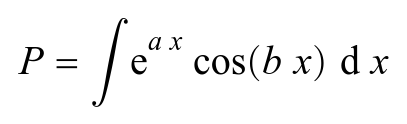
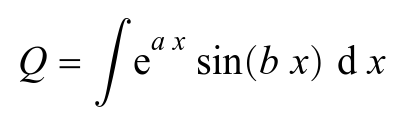
定义域可以取，从而有= 

Simplify((2^(1/2)\*sin(2\*t))/(2\*(cos(2\*t) + 1)^(1/2))) assuming cos(t)>0

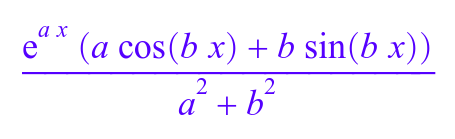
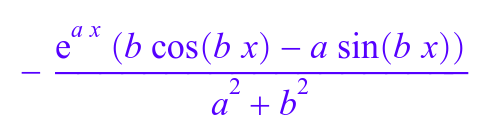


积分= ，回代，解毕

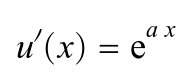
1. 分部积分法：

计算和（ab!=0）

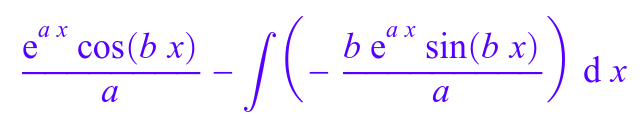
第1步，两个不定积分都可以用命令int直接计算出结果.

int(exp(a\*x)\*cos(b\*x), x); int(exp(a\*x)\*sin(b\*x), x);  
 

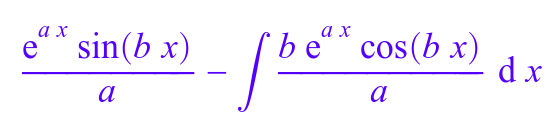
第2步，演示分部积分的中间步骤.

这两个不定积分都是运用分部积分法求解的，都是取

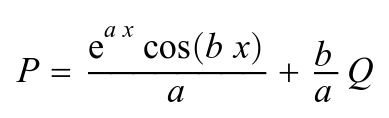
intlib::byparts(hold(int(exp(a\*x)\*cos(b\*x), x)), exp(a\*x));

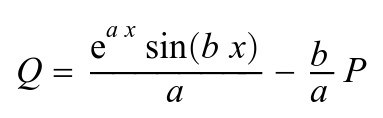


intlib::byparts(hold(int(exp(a\*x)\*sin(b\*x), x)), exp(a\*x));



由上述两个分部积分的结果可见，P和Q有互相循环表示的关系，可以改写成两个方程：

；

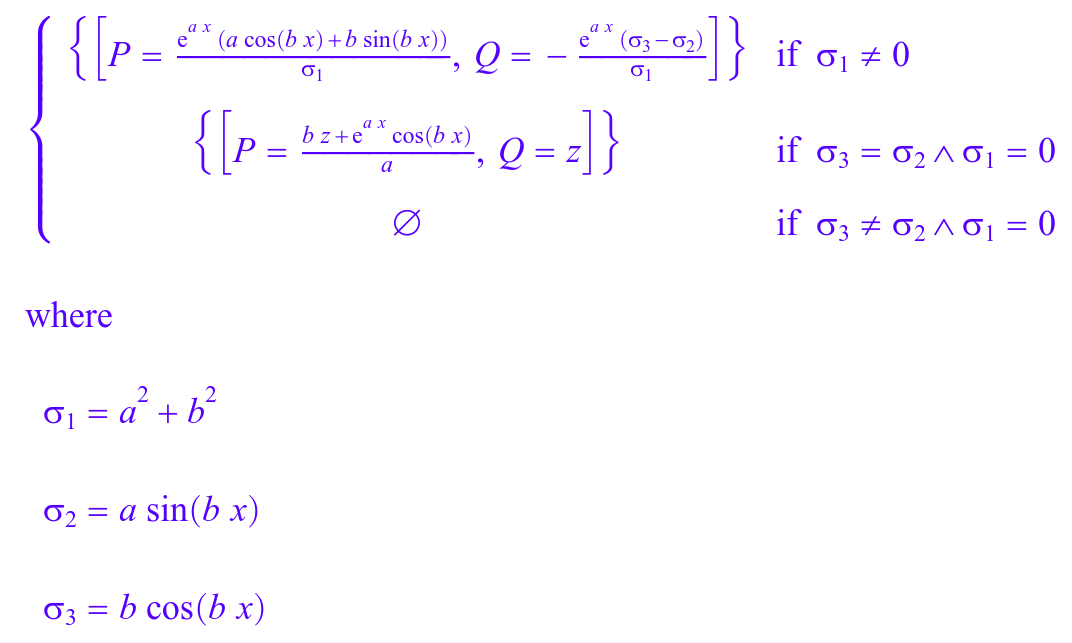
.

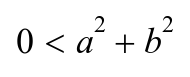
可以通过解方程组而求出P和Q.

第3步，解方程组.

delete a,b,P,Q;

solve({P=exp(a\*x)\*cos(b\*x)/a+b/a\*Q, Q=exp(a\*x)\*sin(b\*x)/a-b/a\*P}, {P,Q});

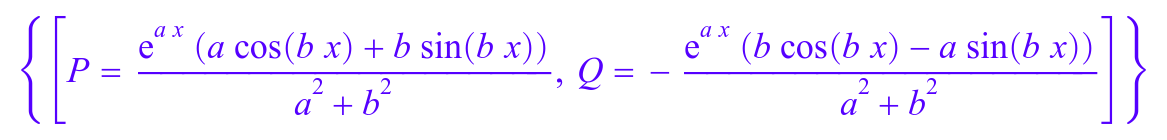


从MuPAD分情况讨论的条件看，增加假设，可以简化输出：

delete a,b,P,Q;

assume(a^2+b^2>0); // a\*b<>0 蕴含 a^2+b^2>0.

solve([P=exp(a\*x)\*cos(b\*x)/a+b/a\*Q, Q=exp(a\*x)\*sin(b\*x)/a-b/a\*P], [P,Q]);

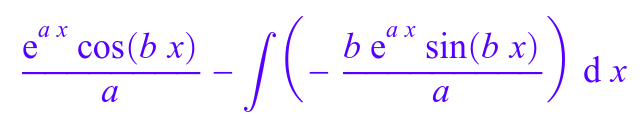


第4步，改用两步分部积分和解方程，分别计算P和Q.

(1) 计算P.

先做第一步分部积分：

intlib::byparts(hold(int(exp(a\*x)\*cos(b\*x), x)), exp(a\*x));

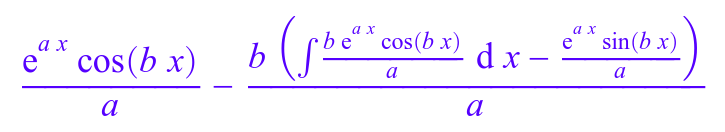


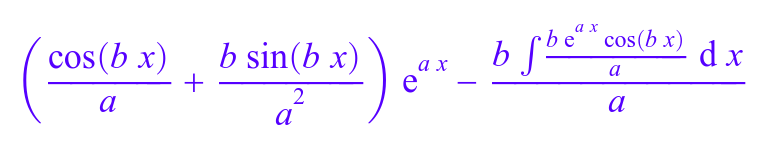
然后，做第二步分部积分：

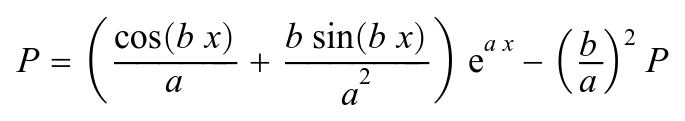
exp(a\*x)\*cos(b\*x)/a + b/a\*

intlib::byparts(hold(int(exp(a\*x)\*sin(b\*x), x)), exp(a\*x));

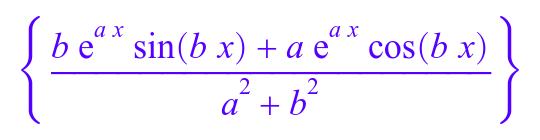
collect(%,exp(a\*x)); //合并含有exp(a\*x)的两个函数表达式





人工整理，

最后解方程：solve(P = (cos(b\*x)/a + (b\*sin(b\*x))/a^2)\*exp(a\*x) - (b/a)^2\*P, P);

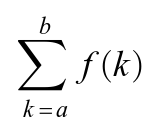


同理可求出Q

**实验1-9 定积分之有限与估计**

计算有限和的浮点数近似值：

命令numeric::sum返回求和的浮点数（小数）近似值结果，格式也有两种：

(1) numeric::sum(f(k), k=a..b) 计算的浮点数值

(2) numeric::sum(f(k), k in S) 计算的浮点数值

命令coerce(object, T)把对象object转换成属于类型T的相应的对象.

命令matrix(m, n)创建m行n列的零矩阵.

for循环语句的语法格式有：

(1)

for i from start to stop do

body

end\_for

循环变量i从初值start至终值stop以1为步长逐步取值，每取一次值，执行一遍语句体body.

注意stop>start，即i递增.

(2)

for i from start to stop step stepwidth do

body

end\_for

循环变量i从初值start至终值stop以stepwidth为步长逐步取值，每取一次值，执行一遍语句体body.

注意stop>start，即i递增.

(3)

for i from start downto stop do

body

end\_for

循环变量i从初值start至终值stop以-1为步长逐步取值，每取一次值，执行一遍语句体body.

注意stop<start，即i递减.

(4)

for i from start to stop step stepwidth do

body

end\_for

循环变量i从初值start至终值stop以stepwidth为步长逐步取值，每取一次值，执行一遍语句体body.

注意stop<start，即i递减.

(5)

for x in object do

body

end\_for

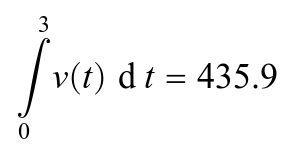
循环变量x依次取遍对象object的所有分量，每取一次值，执行一遍语句体body.

sum(k^2,k in {2,4,6,8,10});

sum(k^2,k in {$2..10 step 2});

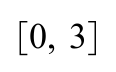
sum(k^2,k in coerce([2,4,6,8,10],DOM\_SET));

1. 假设火箭垂直射向空中的速度函数是v = 160 - 9.8\*t，火箭在最初的三秒总共升高了少米？这些估计值与精确值435.9米相比较有多大误差？

第1步，用定积分验算火箭在头三秒升高的高度的精确值

delete t;

int(160-9.8\*t,t=0..3);

第2步，把闭区间三等分，考虑速度在子区间的左端点取值，用有限和估计高度.

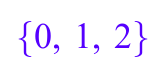
v:=160-9.8\*t:

n:=3:

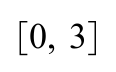
dt:=3/n:

Lt:={k\*dt $k=0..n-1};

numeric::sum(v\*dt, t in Lt);





第3步，把闭区间三等分，考虑速度在子区间的右端点取值，用有限和估计高度.

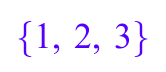
v:=160-9.8\*t:

n:=3:

dt:=3/n:

Rt:={k\*dt $k=1..n};

numeric::sum(v\*dt, t in Rt);





第4步，把闭区间【0,3】三等分，考虑速度在子区间的中间点取值，用有限和估计高度.

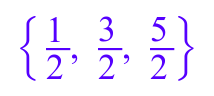
v:=160-9.8\*t:

n:=3:

dt:=3/n:

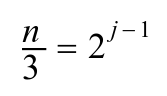
Mt:={(k+1/2)\*dt $k=0..n-1};

numeric::sum(v\*dt, t in Mt);





综合起来，因为v = 160 - 9.8\*t严格单调递减，所以在左端点估值偏大，在右端点估值偏小.由于v = 160 - 9.8\*t是线性函数，所以在中间点估值恰好精确.

第5步，把上述步骤推广到对【0,3】施行更细密的等分区间，例如，就是每一秒都不断地二等分下去.

delete t,j;

m:=10: //m-1是二等分一秒长度的区间的步数

M:=matrix(m,9): //初始化m行9列的零矩阵

v:=160-9.8\*t:

for j from 1 to m do

n:=3\*2^(j-1): //把闭区间[0,3]n等分

dt:=3/n: //小区间的长度

Lt:={k\*dt $k=0..n-1}: //小区间左端点的集合

Mt:={(k+1/2)\*dt $k=0..n-1}: //小区间中间点的集合

Rt:={k\*dt $k=1..n}: //小区间右端点的集合

M[j,1]:=j:

M[j,2]:=n:

M[j,3]:=dt:

M[j,4]:=numeric::sum(v\*dt,t in Lt): //在左端点估计高度

M[j,5]:=(M[j,4]-435.9)://在左端点估计高度的绝对误差

M[j,6]:=numeric::sum(v\*dt,t in Mt): //在中间点估计高度

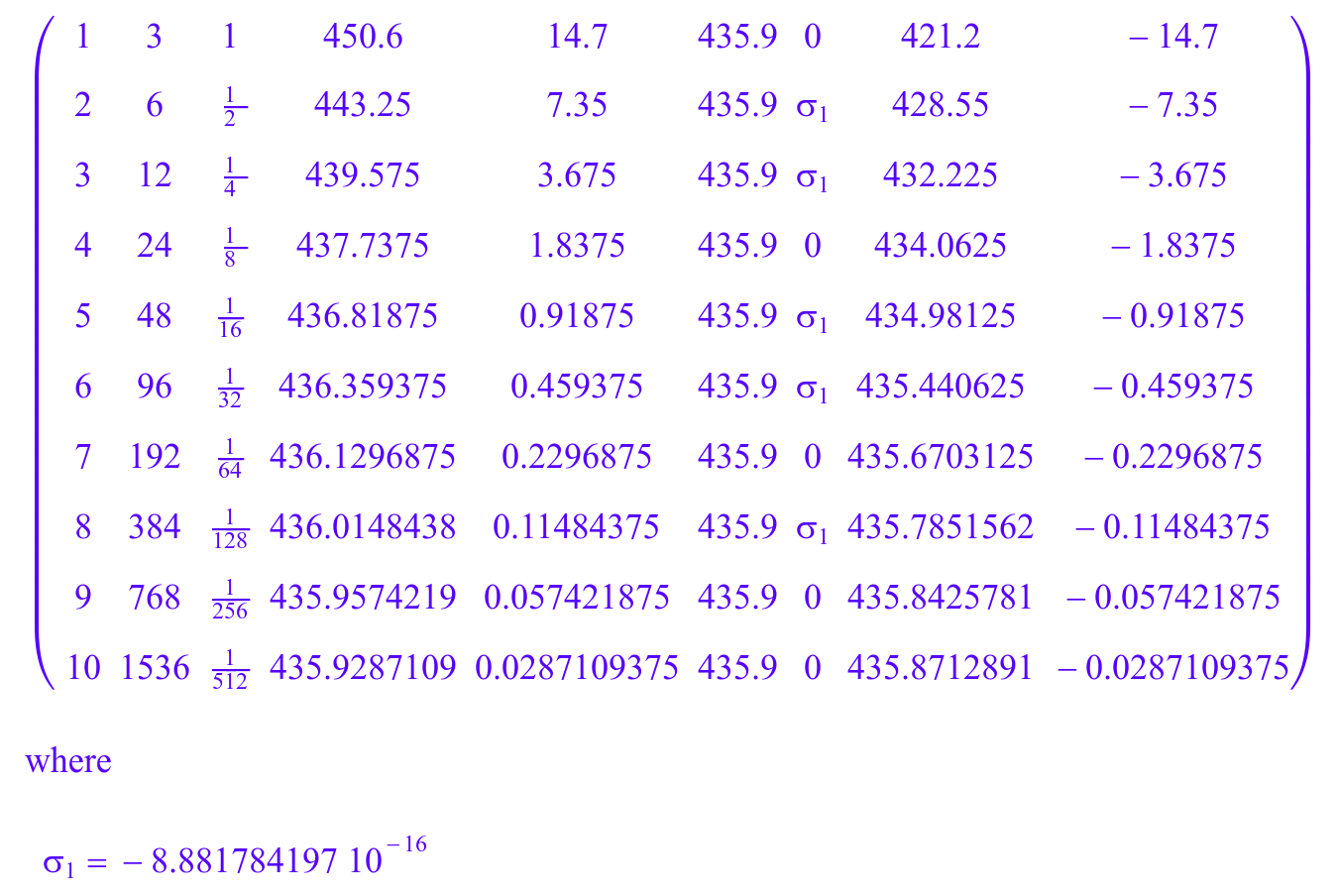
M[j,7]:=(M[j,6]-435.9)://在中间点估计高度的绝对误差

M[j,8]:=numeric::sum(v\*dt,t in Rt): //在右端点估计高度

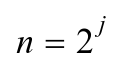
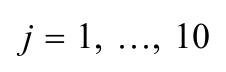
M[j,9]:=(M[j,8]-435.9)://在右端点估计高度的绝对误差

end\_for:

M;



1. 利用有限和估计半径等于4的球体的体积.

把区间等分，用有限和估计球体积. 取，用for循环语句计算，并利用矩阵列表：

delete x;

M:=matrix(10,9): //初始化零矩阵

V:=float(256\*PI/3); //体积精确值的浮点数表示

y:=sqrt(16-x^2):

for j from 1 to 10 do

n:=2^j:

dx:=8/n:

Lx:={-4+k\*dx $k=0..n-1}:

Mx:={-4+(k+1/2)\*dx $k=0..n-1}:

Rx:={-4+k\*dx $k=1..n}:

M[j,1]:=j:

M[j,2]:=n:

M[j,3]:=dx:

M[j,4]:=numeric::sum(PI\*y^2\*dx, x in Lx):

M[j,5]:=M[j,4]-V:

M[j,6]:=numeric::sum(PI\*y^2\*dx, x in Mx):

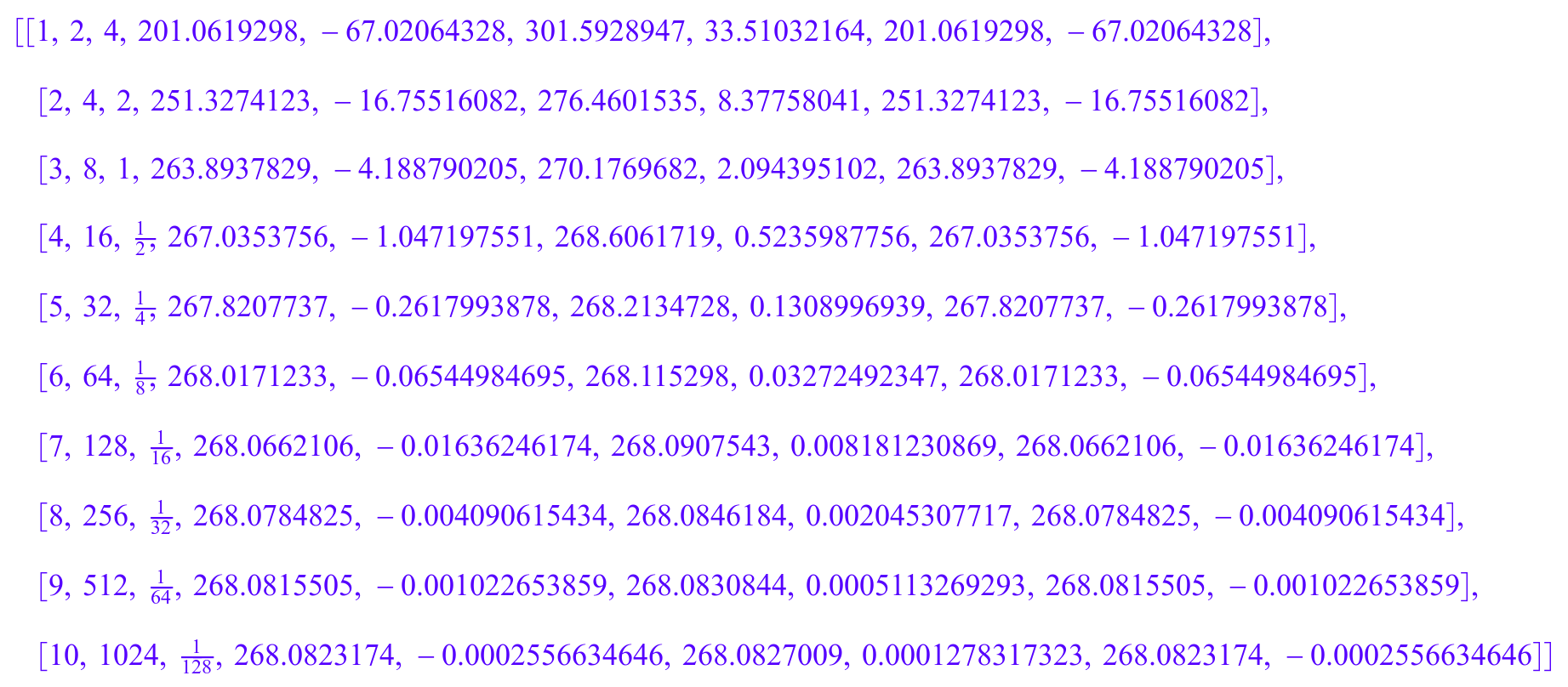
M[j,7]:=M[j,6]-V:

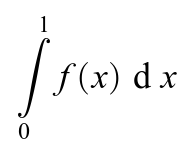
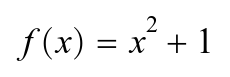
M[j,8]:=numeric::sum(PI\*y^2\*dx, x in Rx):

M[j,9]:=M[j,8]-V:

end\_for:

M;



1. 绘图计算定义计算定积分，其中

第1步，绘制f(x)图像与x轴、直线x=0和x=1围成的曲边梯形，用阴影表示.

直线x=0就是y轴，不需要另外用命令来绘制；

直线x=1用连接点(1,0)和点(1,f(1))的线段表示.

delete x,k,n;

f:=x->x^2+1:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

L:=plot::Line2d([1,0],[1,f(1)],Color=RGB::Black):

plot(plot::Hatch(Gf), Gf, L);

第2步，取分划，n等分区间[0,1]，图示曲边梯形被分割为n个小的曲边梯形.

第k个分点x=k/n，相应的线段连接点(k/n,0)和点(k/n,f(k/n))，

用命令plot::Group2d绘制n条直线段

.n:=10:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

L:=plot::Group2d(plot::Line2d([k/n,0], [k/n,f(k/n)],

Color=RGB::Black)

$ k=1..n):

plot(plot::Hatch(Gf), Gf, L);

第3步，取一个不均匀的分划，图示曲边梯形被分割为n个小的曲边梯形：

h:=[0, 0.3, 0.38, 0.4, 0.5, 0.58, 0.7, 0.78, 0.92, 1]:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

L:=plot::Group2d(plot::Line2d([h[k],0], [h[k],f(h[k])],

Color=RGB::Black)

$ k=1..10):

plot(plot::Hatch(Gf), Gf, L);

第4步，用小矩形代替小曲边梯形，高取在子区间左端点.（为方便起见，下面的过程中对区间进行均匀划分）

用命令plot::Integral就能绘制这些小矩形，还能显示近似值（黎曼和，即小矩形面积之和）与精确值（定积分），

设置IntMethod=RiemannLeft，就是高取在子区间左端点.

n:=10:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RL:=plot::Integral(Gf, n, IntMethod=RiemannLeft):

plot(Gf, RL);

第5步，动画演示当n增大时，黎曼和逼近定积分的过程.

只需要清除变量n，然后在命令plot::Integral中增加n做动画参数.

细致的考虑，设置动画帧数恰好等于n取值的数目.

delete n;

m:=30:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RL:=plot::Integral(Gf, n, n=1..m,

IntMethod=RiemannLeft,

Frames=m):

plot(Gf, RL);

第6步，小矩形的高取在子区间右端点.

设置IntMethod=RiemannRight.

n:=10:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RR:=plot::Integral(Gf, n, IntMethod=RiemannRight):

plot(Gf, RR);

函数图像被遮挡了，只要把RR和Gf的顺序换过来：

n:=10:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RR:=plot::Integral(Gf, n, IntMethod=RiemannRight):

plot(RR,Gf);

第7步，动画演示当n增大时，黎曼和逼近定积分的过程.

delete n;

m:=30:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RR:=plot::Integral(Gf, n, n=1..m,

IntMethod=RiemannRight,

Frames=m):

plot(RR, Gf);

第8步，小矩形的高取在子区间中间点.

设置IntMethod=RiemannMiddle.

n:=10:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RM:=plot::Integral(Gf, n, IntMethod=RiemannMiddle):

plot(RM, Gf);

第9步，动画演示当n增大时，黎曼和逼近定积分的过程.

delete n;

m:=30:

Gf:=plot::Function2d(f(x), x=0..1, Color=RGB::Black):

RM:=plot::Integral(Gf, n, n=1..m,

IntMethod=RiemannMiddle,

Frames=m):

plot(RM, Gf);

