《符号计算系统Maple教程》

图书基本信息

书名：《符号计算系统Maple教程》

13位ISBN编号：9787312020674

10位ISBN编号：7312020674

出版时间：2007-10

出版社：中国科学技术大学出版社

作者：张韵华等

页数：221

版权说明：本站所提供下载的PDF图书仅提供预览和简介以及在线试读，请支持正版图书。

更多资源请访问：www.tushu000.com

《符号计算系统Maple教程》

内容概要

符号计算系统Maple教程，ISBN：9787312020674，作者：张韵华、王新茂

《符号计算系统Maple教程》

精彩短评

1、对maple 的运用讲得比较清楚哈。很喜欢这本书。
2、maple
3、谁有这本书啊，纸质版的。急需

章节试读

1、《符号计算系统Maple教程》的笔记-第24页

注，调用反函数的符号是 map

2、《符号计算系统Maple教程》的笔记-第1000页

&lt;原文开始&gt;&lt;/原文结束&gt;seq - create a sequence
Calling Sequence
seq(f, i = m..n)
seq(f, i = m..n, step)
seq(f, i = 1..m, step)
seq(f, i = x)
seq(f, i in x)
seq(m .. n, step)
Parameters
f
-
any expression
i
-
name
m, n
-
numerical values
x
-
expression
step
-
(optional) numerical value
Description
The seq command is used to construct a sequence of values. The most typical calling sequence is seq(f(i), i = 1..n) which generates the sequence
fApplyFunction(1)commafApplyFunction(2)comma...commaf

ApplyFunction(n)
. More generally, seq(f(i), i = m..n) generates the sequence
fApplyFunction(m)commafApplyFunction(m + 1)comma...commaf

ApplyFunction(n)
.
The seq(f(i), i = x) calling sequence generates a sequence by applying f to each operand or entry of x. Here, x would most commonly be a set or list, but it could be any other data structure to which op can be applied, such as a sum or product. For tables and rtables, the entries of x are scanned rather than the operands. This form of the seq command can also iterate over the characters within a string.
The seq(m..n) and seq(m..n, step) calling sequences generate numerical sequences starting with m, followed by m

+ step. The last value does not exceed n. When step is omitted, the default increment is 1.
 The seq command is related to the for-loop construct. The precise semantics of the first two versions of the seq call can best be understood by defining them in terms of the for-loop as follows. In this description, the expression f is typically a function of i.

```
seq(f, i=m..n) == S := NULL;
            old := i;
            for i from m to n do S := S,f end do;
            i := old;
            S; # the result
seq(f, i=x)    == S := NULL;
            old := i;
            for i in x do S := S,f end do;
            i := old;
            S; # the result
```

  Note: As written above, the seq version is more efficient than the for-loop version because the for-loop version constructs many intermediate sequences in the process of building the final result. Specifically, here the cost of the seq version is linear in the length of the sequence, but the for-loop version is quadratic.
  Note: The limits m and n must evaluate to literal constants, that is integers, fractions, floating-point numbers, or characters. As a special case, m may evaluate to infinity, or n may evaluate to -infinity. If m is greater than n, seq returns the empty sequence (NULL).
  Note: The index variable i is NOT private to the seq invocation. It is recommended that you always explicitly declare the index variable to be local inside procedures.
 When x is a sparse Matrix, Vector or rtable, only the non-zero entries are scanned. Otherwise, regardless of the indexing function, or storage, the entire index-space of the object is scanned.
 See also the add and mul commands which work similarly to seq except that instead of constructing a sequence, they construct a sum and product respectively.
 The seq(f(i), i in x) calling sequence is equivalent to seq(f(i), i = x).
 There is also a procedure parameter modifier, seq, which declares that multiple arguments of a specific type within a procedure invocation will be assigned to a single parameter (as a sequence).
Thread Safety
 The seq command is thread safe as of Maple 15 provided that evaluating f is itself thread safe.   Further the index variable i must not be shared between threads. Using a procedure local is advised.
 For more information on thread safety, see index/threadsafe.
Examples

```
&gt;                        / 2      \
&gt;             seqApplyFunction\i ,i=1..5/
                1, 4, 9, 16, 25
&gt;                /            /? i\      \
&gt;        seqApplyFunction|sinApplyFunction|---|,i=0..6|
&gt;                \            \ 6 /      /
          1  1  (1/2)     1  (1/2)  1
       0, -, - 3     , 1, - 3     , -, 0
          2  2            2         2
&gt;             seqApplyFunction(x[i],i=1..5)
          x[1], x[2], x[3], x[4], x[5]
&gt;            X:=[seqApplyFunction(i,i=0..6)]
           X := [0, 1, 2, 3, 4, 5, 6]
&gt;            /            / 2        \\
&gt;           { seqApplyFunction\i  mod 7,i=X/ }
```

```
&gt;             \                    /
                 {0, 1, 2, 4}
&gt;                 [             / 2    \]
&gt;               Y:=[seqApplyFunction\i ,i=X/]
            Y := [0, 1, 4, 9, 16, 25, 36]
&gt;    [seqApplyFunction([X[i],Y[i]],i=1..nopsApplyFunction(X))]
 [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36]]
&gt;               seqApplyFunction(i,i="Hello")
             "H", "e", "l", "l", "o"
&gt;               seqApplyFunction(i,i="a".."f")
            "a", "b", "c", "d", "e", "f"
&gt;             [seqApplyFunction(i,i=0..uminus0?)]
                  []
&gt;              L:=[seqApplyFunction(i,i=1..10,2)]
              L := [1, 3, 5, 7, 9]
&gt;                           /i     \
&gt;                seqApplyFunction|-,i ? L|
&gt;                           \2     /
                1  3  5  7  9
                -, -, -, -, -
                2  2  2  2  2
&gt;    A:=MatrixApplyFunction([[seqApplyFunction(i,i=1..3)],[seq
&gt;
&gt;     ApplyFunction(i,i=4..6)]])
A:=[Typesetting:-mtable(Typesetting:-mtr(Typesetting:-mtd(1,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), Typesetting:-mtd(2,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), Typesetting:-mtd(3,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), rowalign = "",

 columnalign = "", groupalign = ""), Typesetting:-mtr(

 Typesetting:-mtd(4, rowalign = "", columnalign = "",

 groupalign = "", rowspan = "1", columnspan = "1"),

 Typesetting:-mtd(5, rowalign = "", columnalign = "",

 groupalign = "", rowspan = "1", columnspan = "1"),

 Typesetting:-mtd(6, rowalign = "", columnalign = "",
 groupalign = "", rowspan = "1", columnspan = "1"),

 rowalign = "", columnalign = "", groupalign = ""),

 align = "axis", rowalign = "baseline", columnalign = "right",

 groupalign = "{left}", alignmentscope = "true",

 columnwidth = "auto", width = "auto", rowspacing = "1.0ex",

 columnspacing = "0.8em", rowlines = "none",

 columnlines = "none", frame = "none",

 framespacing = "0.4em 0.5ex", equalrows = "false",

 equalcolumns = "false", displaystyle = "false", side = "right",

 minlabelspacing = "0.8em")]
&gt;              s:=seqApplyFunction(i,i ? A)
               s := 1, 4, 2, 5, 3, 6
&gt;              seqApplyFunction(0..100,10)
       0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
C := proc(f,x)
local i; # declare index variable to be local
 [seq( coeff(f,x,i), i=0..degree(f,x) )];
end proc:
&gt;                     /       3 \
&gt;              CApplyFunction\2 x - 4+x ,x/
                [-4, 2, 0, 1]
&gt;        T:=tableApplyFunction({color="red",size="XL"})
        T := TABLE([size = "XL", color = "red"])
&gt;        seqApplyFunction(e,e=evalApplyFunction(T))
                 "XL", "red"
```

See Also
add, dollar, for, map, mul, op, sequence, The seq Modifier, Threads[Seq]

3、《符号计算系统Maple教程》的笔记-第1000页

seq - create a sequence

Calling Sequence

```
seq(f, i = m..n)
seq(f, i = m..n, step)
seq(f, i = 1..m, step)
seq(f, i = x)
seq(f, i in x)
seq(m .. n, step)
```

Parameters
f
-
any expression
i
-
name
m, n
-
numerical values
x
-
expression
step
-
(optional) numerical value
Description
The seq command is used to construct a sequence of values. The most typical calling sequence is seq(f(i), i = 1..n) which generates the sequence
fApplyFunction(1)commafApplyFunction(2)comma...commaf

ApplyFunction(n)
. More generally, seq(f(i), i = m..n) generates the sequence
fApplyFunction(m)commafApplyFunction(m + 1)comma...commaf

ApplyFunction(n)
.
The seq(f(i), i = x) calling sequence generates a sequence by applying f to each operand or entry of x. Here, x would most commonly be a set or list, but it could be any other data structure to which op can be applied, such as a sum or product. For tables and rtables, the entries of x are scanned rather than the operands. This form of the seq command can also iterate over the characters within a string.
The seq(m..n) and seq(m..n, step) calling sequences generate numerical sequences starting with m, followed by m + step. The last value does not exceed n. When step is omitted, the default increment is 1.
The seq command is related to the for-loop construct. The precise semantics of the first two versions of the seq call can best be understood by defining them in terms of the for-loop as follows. In this description, the expression f is typically a function of i.

```
seq(f, i=m..n) == S := NULL;
          old := i;
          for i from m to n do S := S,f end do;
          i := old;
          S; # the result
seq(f, i=x)    == S := NULL;
          old := i;
          for i in x do S := S,f end do;
          i := old;
          S; # the result
```

Note: As written above, the seq version is more efficient than the for-loop version because the for-loop version constructs many intermediate sequences in the process of building the final result. Specifically, here the cost of the seq version is linear in the length of the sequence, but the for-loop version is quadratic.

Note: The limits m and n must evaluate to literal constants, that is integers, fractions, floating-point numbers, or characters. As a special case, m may evaluate to infinity, or n may evaluate to -infinity. If m is greater than n, seq returns the empty sequence (NULL).
Note: The index variable i is NOT private to the seq invocation. It is recommended that you always explicitly declare the index variable to be local inside procedures.
When x is a sparse Matrix, Vector or rtable, only the non-zero entries are scanned. Otherwise, regardless of the indexing function, or storage, the entire index-space of the object is scanned.
See also the add and mul commands which work similarly to seq except that instead of constructing a sequence, they construct a sum and product respectively.
The seq(f(i), i in x) calling sequence is equivalent to seq(f(i), i = x).
There is also a procedure parameter modifier, seq, which declares that multiple arguments of a specific type within a procedure invocation will be assigned to a single parameter (as a sequence).
Thread Safety
The seq command is thread safe as of Maple 15 provided that evaluating f is itself thread safe.   Further the index variable i must not be shared between threads. Using a procedure local is advised.
For more information on thread safety, see index/threadsafe.
Examples

```
&gt;                          / 2      \
&gt;              seqApplyFunction\i ,i=1..5/
             1, 4, 9, 16, 25
&gt;                   /             /? i\     \
&gt;        seqApplyFunction|sinApplyFunction|---|,i=0..6|
&gt;                   \             \ 6 /     /
          1  1  (1/2)    1  (1/2)  1
        0, -, - 3    , 1, - 3    , -, 0
          2  2          2        2
&gt;             seqApplyFunction(x[i],i=1..5)
          x[1], x[2], x[3], x[4], x[5]
&gt;            X:=[seqApplyFunction(i,i=0..6)]
           X := [0, 1, 2, 3, 4, 5, 6]
&gt;            /            / 2       \\
&gt;           { seqApplyFunction\i  mod 7,i=X/ }
&gt;            \                        /
             {0, 1, 2, 4}
&gt;               [            / 2   \]
&gt;             Y:=[seqApplyFunction\i ,i=X/]
          Y := [0, 1, 4, 9, 16, 25, 36]
&gt;   [seqApplyFunction([X[i],Y[i]],i=1..nopsApplyFunction(X))]
 [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36]]
&gt;             seqApplyFunction(i,i="Hello")
            "H", "e", "l", "l", "o"
&gt;             seqApplyFunction(i,i="a".."f")
          "a", "b", "c", "d", "e", "f"
&gt;           [seqApplyFunction(i,i=0..uminus0?)]
                []
&gt;           L:=[seqApplyFunction(i,i=1..10,2)]
             L := [1, 3, 5, 7, 9]
&gt;                         /i     \
&gt;              seqApplyFunction|-,i ? L|
&gt;                       \2     /
               1  3  5  7  9
               -, -, -, -, -
               2  2  2  2  2
&gt;   A:=MatrixApplyFunction([[seqApplyFunction(i,i=1..3)],[seq
&gt;
&gt;    ApplyFunction(i,i=4..6)]])
A:=[Typesetting:-mtable(Typesetting:-mtr(Typesetting:-mtd(1,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), Typesetting:-mtd(2,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), Typesetting:-mtd(3,

 rowalign = "", columnalign = "", groupalign = "",

 rowspan = "1", columnspan = "1"), rowalign = "",

 columnalign = "", groupalign = ""), Typesetting:-mtr(

 Typesetting:-mtd(4, rowalign = "", columnalign = "",

 groupalign = "", rowspan = "1", columnspan = "1"),

 Typesetting:-mtd(5, rowalign = "", columnalign = "",

 groupalign = "", rowspan = "1", columnspan = "1"),

 Typesetting:-mtd(6, rowalign = "", columnalign = "",

 groupalign = "", rowspan = "1", columnspan = "1"),

 rowalign = "", columnalign = "", groupalign = ""),

 align = "axis", rowalign = "baseline", columnalign = "right",

 groupalign = "{left}", alignmentscope = "true",

 columnwidth = "auto", width = "auto", rowspacing = "1.0ex",

 columnspacing = "0.8em", rowlines = "none",

 columnlines = "none", frame = "none",

 framespacing = "0.4em 0.5ex", equalrows = "false",
 equalcolumns = "false", displaystyle = "false", side = "right",

 minlabelspacing = "0.8em")]
&gt;            s:=seqApplyFunction(i,i ? A)
           s := 1, 4, 2, 5, 3, 6
&gt;            seqApplyFunction(0..100,10)
      0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
C := proc(f,x)
local i; # declare index variable to be local
  [seq( coeff(f,x,i), i=0..degree(f,x) )];
end proc:
&gt;                    /      3 \
&gt;            CApplyFunction\2 x - 4+x ,x/
              [-4, 2, 0, 1]
&gt;        T:=tableApplyFunction({color="red",size="XL"})
      T := TABLE([size = "XL", color = "red"])
&gt;        seqApplyFunction(e,e=evalApplyFunction(T))
              "XL", "red"
```

See Also
add, dollar, for, map, mul, op, sequence, The seq Modifier, Threads[Seq]

4、《符号计算系统Maple教程》的笔记-第23页

在调用函数的时候通常有两种调用方法，第一种：
函数名称：= 变量-&gt; 表达式
第二种：
函数：= 函数表达式

ff := unapply(函数，函数变量)
举例：
定义函数f(x)=x^2 +1,计算f(2);
f:= x-&gt; x^2 +1;
f(2);

另一个例子：
P ：= x -1
 p：=x -1# 显示项目
ff ：= unapply(p,x);
 ff := x-&gt; x-1 #显示项目

5、《符号计算系统Maple教程》的笔记-第20页

我不熟悉的数学函数
argument()　 # 复数的辐角
conjuagte()　 #共轭复数
Re(), Im()　 # 复数的实部，复数的虚部

factorial()　　# 阶乘
GAMMA()　　# GAMMA函数,
Ln (x) 或者 log(x) # 自然对数
log10(x)　　# 常用对数
log[b](x)　　# 以b 为底的对数
sqrt()　　#平方根
ceil()　　#不小于某个数的对数
floor ()　　# 不大于某个数的对数
round ()　　#四舍五入
trunc()　　#取某个数的整数部分
frac()　　#取某个数的小数部分
gcd(a,b)　　# 取整式或者多项式的最大公约数
lcm(a1,a2,a3,a4....an)　# 取整式或者多项式的最小公倍数
max(a1,a2,a3.. an)　# 取最大值
min(a1,a2,a3..an)　#取最小值
rand()　　#随机取整
randomize()　　# 初始化随机数形成
sinh(),cosh(),tanh(),csch(),sech(),coth()　　#双曲函数
arcsinh(), arccsch(),arccosh(),arcsech(),arctanh(),arccoth()　# 反双曲函数

6、《符号计算系统Maple教程》的笔记-第22页

　　运用maple 求三个数的最大公约数；

求公约数的函数符号是gcd;求最小公倍数的符号是lcm;
求三个数的公约数的方法要这样：先求两个数的最大公约数，再求这两个数字的最大公约数与第三个数字的最大公约数；而求最小公倍数却可以一次性直接求。
例如：求135，179和468的最大公约数和最小公倍数；
输入如下：
gcd(gcd(135,279),468),lcm(135,279,468);

7、《符号计算系统Maple教程》的笔记-第20页

　　变量替换的一般形式为：
subs(变量1=变量2，表达式)
subs(等式1，等式2，等式3.......等式k,表达式)；
例子；
subs(x=1,x+1);
2 #输出结果
subs(a=x+1,b= 2-y,(a+b+2)^2);
(x+2-y+c)^2 # 输出结果
subs（a+b=1,sin(a+b+c)）;# subs的功力不够，需要用到代数替换algsubs
 sin(a+b+c)
修改好的代码如下：
algsubs(a+b=1,sin(a+b+c));
sin(1+c)　#输出结果

8、《符号计算系统Maple教程》的笔记-第24页

表达式序列是maple的一个数据类型，序列的元素用逗号操作符连接，可以直接讲序列赋值到一个变量中，逗号操作是所有操作符中优先级别最低的操作符号，我们会在多种数据类型遇到它，序列的一般形式是；
表达1，表达式2，........................表达式n
可以用函数seq 构造数值或者函数序列，例如；
seq(f(i),i =1....n); # 生成序列 f(1),f(2),f(3)...........f(n);
seq(f(i),i= m....n) # 生成序列f(m),f(m+1),f(m+2)......f(n);
命令seq(f(i),i=x)中的x 可以为集合(set) 或者列表（list）

《符号计算系统Maple教程》

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:www.tushu000.com