

```

> restart:
  with(linalg):
Warning, the protected names norm and trace have been redefined and unprotected

```

## – Logarithmusfunktion (10 Punkte)

### – Naehierungswert+Fehlerschranke:

```

> lsumme := proc(x::numeric,n::integer)
  local k,sum,xhochk;
  sum := 0;
  xhochk := x;
  for k from 1 to n do
    sum := sum + xhochk/k;
    xhochk := -x*xhochk;
  end do;
  return sum,abs(xhochk/k);
end proc;

lsumme := proc(x::numeric, n::integer)
local k, sum, xhochk;
  sum := 0;
  xhochk := x;
  for k to n do sum := sum + xhochk / k; xhochk := -x*xhochk end do;
  return sum, abs(xhochk / k)
end proc
> lsumme(1/2, 2);

```

$$\frac{3}{8}, \frac{1}{24}$$

```

[ >

```

### – Naehierungswert mit vorgegebener Genauigkeit:

```

> lapprox := proc(x::numeric, e::numeric)
  local k,sum,xhochk;
  sum := 0;
  xhochk := x;
  k := 1;
  while abs(xhochk/k) > e do
    sum := sum + xhochk/k;
    xhochk := -x*xhochk;
    k := k + 1;
  end do;
  return sum;
end proc;

lapprox := proc(x::numeric, e::numeric)
local k, sum, xhochk;

```

```

    sum := 0;
    xhochk := x;
    k := 1;
    while e < abs(xhochk / k) do sum := sum + xhochk / k; xhochk := -x*xhochk; k := k + 1
    end do;
    return sum
end proc
[ > lapprox(1/2, 1/25);
                                     5
                                     12
[ >

```

## – Partielle Integration

```

[ > funkE := proc(x::numeric, k::integer)
    if k = 0
    then return exp(x) - 1
    else return x^k * exp(x) - k * funkE(x, k - 1)
    end if;
end proc;
funkE := proc(x::numeric, k::integer)
    if k = 0 then return exp(x) - 1 else return x^k * exp(x) - k * funkE(x, k - 1) end if
end proc
[ > funkE(1/2, 2);
                                     5
                                     4 e(1/2) - 2
[ >

```

## – Listen

Die Prozedur `anzahl` zählt die Häufigkeit eines Elements `x` in der Liste `l`

```

[ > anzahl := proc(l::list, x)
    local elem, zaehler;
    zaehler := 0;
    for elem in l do
        if elem = x then zaehler := zaehler + 1 end if;
    end do;
    return zaehler;
end proc;
anzahl := proc(l::list, x)
    local elem, zaehler;
    zaehler := 0;
    for elem in l do if elem = x then zaehler := zaehler + 1 end if end do;

```

```
return zaehler
```

```
end proc
```

Zwei Listen sind Anagramme, wenn sie gleich viele Elemente haben und die Häufigkeit aller Elemente identisch ist:

```
> anagramm := proc(l1::list, l2::list)
  local elem;
  if nops(l1) <> nops(l2) then return false end if;
  for elem in l1 do
    if anzahl(l1,elem) <> anzahl(l2,elem)
      then return false end if;
    end do;
  return true;
end proc;
```

```
anagramm := proc(l1::list, l2::list)
```

```
local elem;
```

```
if nops(l1) ≠ nops(l2) then return false end if;
```

```
for elem in l1 do if anzahl(l1, elem) ≠ anzahl(l2, elem) then return false end if end do;
```

```
return true
```

```
end proc
```

```
> anagramm([1,2,3], [0,2,3,1]);
```

```
false
```

```
> anagramm([1,2,3], [2,3,1]);
```

```
true
```

```
>
```

## – Baeume

Beispielbaeume:

```
> b1 := [10,[]]; b2 := [7, []];
```

```
b1 := [10, [ ]]
```

```
b2 := [7, [ ]]
```

```
> b1[2] := [eval(b1),b2]: b1;
```

```
b3 := [3, [b1,b2,b2]];
```

```
[10, [[10, [ ]], [7, [ ]]]]
```

```
b3 := [3, [[10, [[10, [ ]], [7, [ ]]], [7, [ ]], [7, [ ]]]]
```

## – Loesung mit Zaehlen der Knoten (nicht schoen aber nicht verboten)

```
> knotenzahl := proc(baum)
```

```
  local zahl, sohn;
```

```
  zahl := 1;
```

```
  for sohn in baum[2] do
```

```
    zahl := zahl + knotenzahl(sohn);
```

```
  end do;
```

```
  return zahl;
```

```
end proc;
```

```

knotenzahl := proc(baum)
local zahl, sohn;
    zahl := 1;
    for sohn in baum[2] do zahl := zahl + knotenzahl(sohn) end do;
    return zahl
end proc
> gerade_knotenzahl := proc(baum)
    if knotenzahl(baum) mod 2 = 0
    then return true
    else return false
    end if;
end proc;
gerade_knotenzahl := proc(baum)
    if knotenzahl(baum) mod 2 = 0 then return true else return false end if
end proc
> knotenzahl(b1); gerade_knotenzahl(b1);
                                     3
                                     false

```

## – Effiziente Implementierung ueber Boolesche Algebra

Ein Baum ohne Söhne (Blatt) hat nur einen Knoten, also eine ungerade Zahl von Knoten. Für jeden Sohn, der eine ungerade Zahl von Knoten muss einmal zwischen gerade und ungerade "hin und her"

geschaltet werden:

```

> gerade_knoten := proc(baum)
    local gerade, sohn;
    gerade := false;
    for sohn in baum[2] do
        if not gerade_knoten(sohn)
        then gerade := not gerade
        end if;
    end do;
    return gerade;
end proc;
gerade_knoten := proc(baum)
local gerade, sohn;
    gerade := false;
    for sohn in baum[2] do if not gerade_knoten(sohn) then gerade := not gerade end if
    end do;
    return gerade
end proc
> gerade_knoten(b3);
                                     true

```

## – Matrizen

```
> toeplitz := proc(a::matrix,n::integer)
  local i,j;
  for i from 2 to n do
    for j from 2 to n do
      if not a[i,j] = a[i-1,j-1]
      then return false
      end if:
    end do:
  end do:
  return true;
end proc;
```

```
toeplitz := proc(a::matrix, n::integer)
```

```
local i,j;
```

```
for i from 2 to n do
```

```
  for j from 2 to n do if not (a[i,j]=a[i-1,j-1]) then return false end if end do
```

```
end do;
```

```
return true
```

```
end proc
```

```
> a := matrix(3,3,[3,4,5,2,3,4,1,2,3]);
```

$$a := \begin{bmatrix} 3 & 4 & 5 \\ 2 & 3 & 4 \\ 1 & 2 & 3 \end{bmatrix}$$

```
> toeplitz(a,3);
```

```
true
```

```
>
```