Maple-Test zur LVA 'Computermathematik', SS 2011

Gruppe weiß

***** ANMERKUNGEN UND LÖSUNGSHINWEISE *****

AUFGABE	PUNKTE
1	/ 2
2	/ 3
3	/ 3
4	/ 5
5	/ 6
6	/ 4
7	/ 3
8	/ 4
SUMME:	/ 30

*** AUFGABE 1 (2 Punkte):**

(a) Erklären Sie jeden einzelnen Befehl.
(b) Welchen Typ hat das Objekt r ?
(c) Geben Sie an, wie die Ausgabe X aussieht.

```
> m,n := 2,3;
```

$$m, n := 2, 3$$

```
> r := [1..m,1..n];
```

$$r := [1 .. 2, 1 .. 3]$$

```
> X := ([seq([seq(i*j,i in r[1])],j in r[2])]);
```

$$X := [[1, 2], [2, 4], [3, 6]]$$

*** Lösung:**

(a) Die exprseq m,n wir mit Wert 2,3 belegt.
r besteht aus dann aus den Ranges 1..2,2.3.
X wird als Liste von Listen konstruiert, implizite Schleife basierend auf den in r gespeicherten Ranges.
(b) r ist eine Liste der Länge 2.
(c) siehe Code oben.

*** AUFGABE 2 (3 Punkte):**

(a) Erklären Sie, was sie hier sehen.

```
> unassign('z'):
  add(x,y in z);
```

$$x$$

(b) Erklären Sie, was sie hier sehen.

```
> g(u) := u^3;
  g(u),g(1);
  print(g);
```

$$g(u) := u^3$$

$$u^3, g(1)$$

proc() **option** *remember*; 'procname(args)' **end proc**

*** Lösung:**

(a) z hat keinen definierten Wert, d.h. z='z'.
Der add-Befehl verwendet eine implizite Schleife der Länge 1,
wobei y den Wert z annimmt. Damit ist das Ergebnis dasselbe wie
Summe(x,i=1..1) = x.

(b) g(u) := u^3 legt offenbar den Funktionswert g(u)=g('u') fest. Damit wird
eine Funktion g generiert, für die nur die Auswertung an u definiert ist, und zwar mittels

einer remember table. Alle anderen Auswertungen von g sind nicht definiert.

*** AUFGABE 3 (3 Punkte):**

Was gibt diese Prozedur aus, wenn Sie wie unten spezifiziert, aufgerufen wird?

```
> p := proc(n::nonnegint)
  if (n>0) then
     print(n);
     p(n-1)
  end if;
  print(n)
  end proc;
```

p := **proc**(*n*::*nonnegint*) **if** 0 < *n* **then** print(*n*); p(*n* – 1) **end if**; print(*n*) **end proc**

*** Lösung:**

```
> p(5);
```

5
4
3
2
1
0
1
2
3
4

5

*** AUFGABE 4 (5 Punkte):**

Formulieren Sie eine Prozedur switch(L), die je 2 benachbarte Elemente der Liste L vertauscht und die neue Liste als Wert zurückgibt.
Falls die Länge von L ungerade ist, bleibt das letzte Element unverändert.

Verwenden Sie keine expliziten Schleifen. Überprüfung auf sinnvolle Eingabeparameter ist nicht erforderlich.
ACHTUNG: Eine Zuweisung der folgenden Form ist für Listen nicht zulässig:

```
> L[i..(i+1)] := [a,b];
```

```
Error, invalid subscript on left hand side of assignment
```

*** Lösung: (Code der Prozedur switch):**

```
> switch := proc(L::list)
  local l,n;
  n := nops(L);
  l := [seq(seq(L[j],j in [2*i,2*i-1]),i=1..n/2)];
  if is(n,odd) then
    l := [op(l),L[n]]
  end if;
  return l;
  end proc:

> switch([]);
```

[ ]

```
> switch([1]);
```

[1]

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

[2, 1, 3]

```
> switch([1,2,3,4]);
```

[2, 1, 4, 3]

*** AUFGABE 5 (6 Punkte):**

Formulieren Sie eine Prozedur linesearch(f::procedure,P,s::Vector), die zu einer gegebenen Funktion f: R^2 -> R den minimalen Funktionswert entlang der Geraden P + lambda*s (lambda in R) zurückgibt. (P und s sind Vector-en der Länge 2. f wird als Funktion f(x,y) angenommen (x,y in R). Wir nehmen an, dass f differenzierbar ist und dass die Minimalstelle eindeutig existiert.)

Hinweis: Extremwertaufgabe. Für g(lambda) := f(P+lambda*s) ist
g'(lambda) = (nabla f)(P+lambda*s) . s (inneres Produkt),
wobei (nabla f) der Gradientenvektor ist: (nabla f) = (df/dx,df/dy)
(partielle Ableitungen nach x und y).

Definieren Sie intern die Funktion g(lambda) und verwenden Sie solve, um den Wert für lambda zu finden, der der Minimalstelle entspricht.

*** Lösung (Code der Prozedur linesearch):**

```
> linesearch := proc(f::procedure,P,s::Vector)
  local g,gstrich,lambda,minimalstelle,nablaf;
  g := lambda -> P+lambda*s; # Vector-wertige Funktion
  nablaf := Vector([diff(f(x,y),x),
                    diff(f(x,y),y)]); # Gradientenvektor
  nablaf := subs(x=g(lambda)[1],
                 y=g(lambda)[2],nablaf); # g(lambda) eingesetzt
  gstrich := nablaf[1]*s[1]+nablaf[2]*s[2];
  minimalstelle := g(solve(gstrich=0,lambda)); # Minimalstelle als Vector
  return f(minimalstelle[1],minimalstelle[2])  # f an Minimalstelle ausgewertet
  end proc:

> linesearch((x,y)->-x^2-y^2,Vector([1,2]),Vector([1,2]));
                                                    0

> linesearch((x,y)->-x^2-y^2,Vector([1,2]),Vector([1,3]));
                                                   -1
                                                   --
                                                   10
```

*** AUFGABE 6 (4 Punkte):**

Formulieren Sie eine Prozedur inc(f::procedure,C::list),
die eine gegebene Funktion f(x) n mal integriert und nach jedem einzelnen
Integrationsschritt die Integrationskonstante C[i] addiert (i=1,2,...,n),
wobei n die Anzahl der Elemente in der Liste C ist.
Zurückzugeben ist der entsprechende Formelausdruck
(verwenden Sie irgendeinen Variablennnamen, z.B. x.)

*** Lösung (Code der Prozedur inc):**

```
> inc := proc(f::procedure,C::list)
  local i,integral,x,
        n:=nops(C);
  integral := f(x);
  for i from 1 to n do
      integral := int(integral,x) + C[i]
  end do
  end proc:

> inc(t->t^2,[a,b]);
```

$$\frac{1}{12}x^4 + a\,x + b$$

*** AUFGABE 7 (3 Punkte):**

Analysis (Grenzwerte, Differenzieren, Integrieren,..) in Maple:
Schreiben Sie alles relevante auf, was Ihnen zu diesem Thema einfällt..

*** Lösung:**

```
> limit, diff, Diff, int, Int, taylor; # etc. etc.
```

limit, diff, Diff, int, Int, taylor

*** AUFGABE 8 (4 Punkte):**

Ein bisschen Lineare Algebra:
Nehmen Sie an, p steht für eine Polynomfunktion (wird von der Prozedur nicht überprüft).
-- Was ist der Zweck der Prozedur XXX?
-- Was bedeuten die return-Meldungen?
-- Die Prozedur enthält auch einen Syntaxfehler. Finden Sie ihn.
-- Anmerkung: LinearAlgebra[Equal] vergleicht Vektoren bzw. Matrizen
(Vergleich mit = funktioniert hier nicht).

```
> XXX := proc(p::procedure,A::Matrix)
  uses LinearAlgebra;
  local dim:=Dimension(A),  # Initialisierung von dim: Dimensionen der Matrix
        deg:=degree(p(x)), # Initialisierung von deg: Polynomgrad
        pA,pnormiert,x;
  if dim[1]<>dim[2] then
     error "sinnlos."
  elif # elseif
     evalb(deg<=0) or evalb(deg>dim[1]) then # (p(x)=0 hat Grad -infinity)
     return "Grad? Das kann wohl nur ein Versehen sein."
  else
    pnormiert := p(x)/coeff(p(x),x,deg);
    if simplify(pnormiert-CharacteristicPolynomial(A,x))=0 then
       return "C.P. (oder ein Vielfaches davon)."
    else
       pA := simplify(p(A)); # Was ist das wohl?
       if Equal(pA,ZeroMatrix(dim)) then
          return "Treffer! Grad =",deg
       else
          return "Fehlanzeige."
       end if
    end if
  end if
  end proc:
```

*** Lösung:**

Die Prozedur überprüft, ob das gegebene Polynom p ein Annullatirpolynom von A ist, d.h. ob gilt p(A)=0. Es unterscheidet auch zwischen dem charakteristischen Polynom (C.P.) und einem ggf. existierenen Annullatorpolynom niedrigeren Grades (z.B. das Minimalpolynom). In letzterem Fall wird auch der Grad von p mit zurückgegeben.

Falls der Grad von p 0 oder größer als die Matrixdimension ist, wird nichts berechnet.

Syntaxfehler: In der Angabe stand elseif statt elif.

```
> Q := Matrix(3,3,symbol='q'):
  Lambda := LinearAlgebra[DiagonalMatrix]([a,a,b]):
  A := Q.Lambda.Q^(-1):

> XXX(x->(x-a)^2*(x-b),A);
                                    "C.P. (oder ein Vielfaches davon)."
> XXX(x->(x-a)*(x-b),A);
                                    "Treffer! Grad =", 2
> XXX(x->(x-b),A);
                                    "Fehlanzeige."
```