Algorithmen des wissenschaftlichen Rechnens II
Übungsblatt 6
Smolyak-Quadratur

```
> restart;
> with(plots):
> with(plottools):
Warning, the name changecoords has been redefined

Warning, the assigned name arrow now has a global binding
```

Ein paar generelle Optionen zum Plotten:

```
> poptions := thickness=2,
              symbol=DIAMOND, symbolsize=20:
```

Eine Quadraturformel wird wie bei der Kombinationstechnik (Blatt 5, Aufgabe 3) dargestellt als ein assoziatives Array (maple: table) mit den Koordinaten der Auswertestellen als Schlüssel und den Gewichten als Werten (jetzt haben wir auch 1D-Regeln, dort sind die Schlüssel einfach Zahlen, im 2D-Fall wie bisher zweikomonentige Sequenzen).
Beim Tabellenerzeugen geben wir wieder 'sparse' als Indexfunktion an - das bewirkt, dass wir das Gewicht 0 für alle Punkte bekommen, die nicht explizit einen anderen Wert bekommen haben (wir müssen ja wieder Formeln mit unterschiedlichen Gittern kombinieren).

```
>
```

Bild malen

Diese Prozeduren malen die Gitter der Quadraturregel qr.
Zunächst eimal für 1D-Regeln (Punkte und Gewichte):

```
> bild1 := proc(qr)
    local i;
    display([
        line([0,0],[1,0],poptions),
        pointplot([seq([op(i),0], i=[indices(qr)])],poptions),
        seq(textplot([op(i),0.05,sprintf("  %a",qr[op(i)])],
                    align={ABOVE},poptions),
            i=[indices(qr)])
      ], font=[HELVETICA,12], axes=NONE, scaling=CONSTRAINED)
  end proc:
```

Dann die bekannte Darstellung für 2D-Regeln (Punkte und Gewichte):

```
> bild2 := proc(qr)
    local i;
    display([
        rectangle([0,0],[1,1],poptions),
        pointplot([seq(i, i=[indices(qr)])],poptions),

  seq(textplot([op(i)[1],op(i)[2]+0.01,sprintf("%a",qr[op(i)])]
  ,
                    align={ABOVE},poptions),
            i=[indices(qr)])
```

```
      ], font=[HELVETICA,12], axes=NONE, scaling=CONSTRAINED)
  end proc:
```

Und dann nochmal 2D nur mit den Gitterpunkten ohne Beschriftung:

```
bild2ohne := proc(qr)
  local i;
  display([
      rectangle([0,0],[1,1],poptions),
      pointplot([seq(i, i=[indices(qr)])],poptions)
    ], font=[HELVETICA,12], axes=NONE, scaling=CONSTRAINED)
end proc:
```

```
>
```

Quadraturregel anwenden

Vielleicht wollen wir unsere Quadraturregel auch mal auf eine Funktion anwenden - das ist einfach (und unabhängig von der Dimension, weil in beiden Fällen die Schlüssel genau die Sequenz der Aktualparameter zur Funktionsauswertung sind):

```
> anwenden := (qr, f) -> add(qr[op(i)]*f(op(i)),
  i=[indices(qr)]):
```

```
>
```

Quadraturregeln kombinieren

Wir bei Blatt 5: alpha1*qr1 + alpha2*qr2 mit Quadraturregeln qr1, qr2 und Koeffizienten alpha1, alpha2:

```
> kombinieren := proc(qr1, alpha1, qr2, alpha2)
     local qr, i;
     qr := table(sparse);
     for i in {indices(qr1)} union {indices(qr2)} do
        qr[op(i)] := alpha1*qr1[op(i)] + alpha2*qr2[op(i)]
     end do;
     return qr;
  end proc:
```

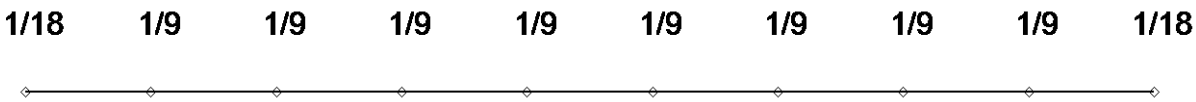
```
>
```

1D Trapezregel

Die 1D Trapezregel mit p^l Intervallen

```
> tr1 := proc(p,l)
     local N,tr,i,s;
     N := p^l;
     tr := table(sparse);
     if l>0 then
       s := 1/N;
       tr[0] := s/2;
       for i from 1 to N-1 do
          tr[i/N] := s;
       end do;
```

```
        tr[1] := s/2;
      end if;
      return tr;
   end proc:
> bild1(tr1(3,2));
```

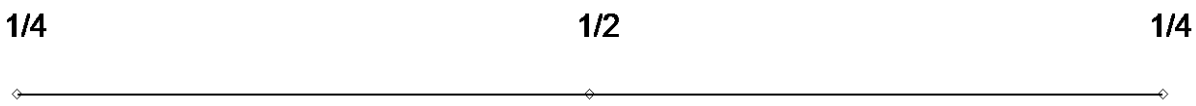


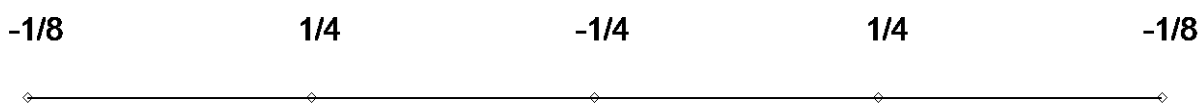

```
>
>
```

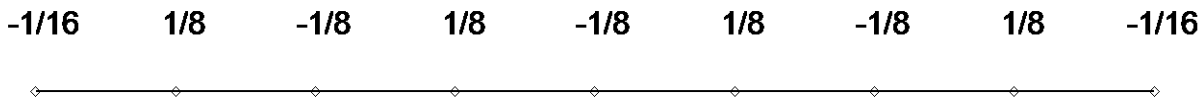
1D Differenzenformel

```
> Delta1 := proc(p,l,qrproc)
    local qr1,qr2,qr;
    qr1 := qrproc(p,l);
    qr2 := qrproc(p,l-1);
    qr := kombinieren(qr1,1,qr2,-1);
    return qr
  end proc:
```

```
> for i to 3 do
     bild1(Delta1(2,i,tr1))
  end do;
```

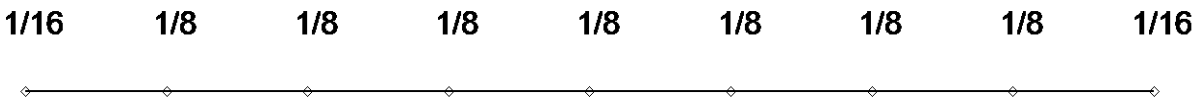



-1/8
1/4
-1/4
1/4
-1/8



Wenn wir die Differenzen aufsammeln, bekommen wir natürlich wieder die Ausgangsregel:

```
> t := table(sparse):
  for i to 3 do
     t := kombinieren(t, 1, Delta1(2,i,tr1), 1)
  end do:
  bild1(t);
```




```
>
```

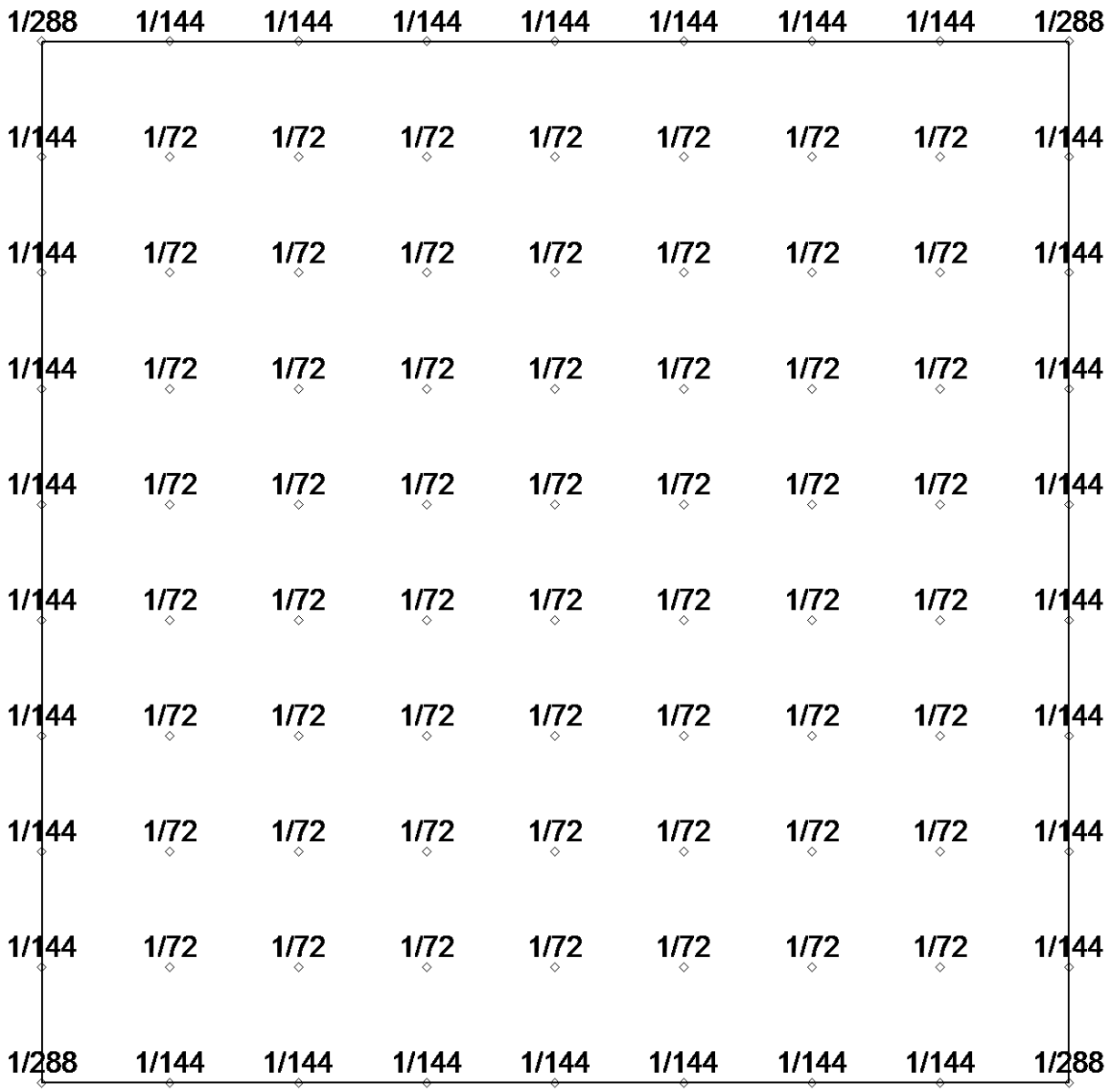
Tensorprodukt

Das Tensorprodukt bekommt zwei 1D-Formeln und konstruiert eine 2D-Formel mit Gitterpunkten (x,y), wobei x ein Gitterpunkt von qr1 und y ein Gitterpunkt von qr2 ist; das Gewicht ist das Produkt der (1D-)Gewichte von x und y.

```
> tensorprodukt := proc(qr1, qr2)
    local qr,i,j;
    qr := table(sparse);
    for i in [indices(qr1)] do
      for j in [indices(qr2)] do
         qr[op(i),op(j)] := qr1[op(i)]*qr2[op(j)]
      end do
    end do;
    return qr
  end proc:
```

Damit bekommen wir z.B. die 2D-Trapezregel wieder:

```
> tr2 := (q1,l1,q2,l2) -> tensorprodukt(tr1(q1,l1),tr1(q2,l2)):
> bild2(tr2(2,3,3,2));
```




```
>
```

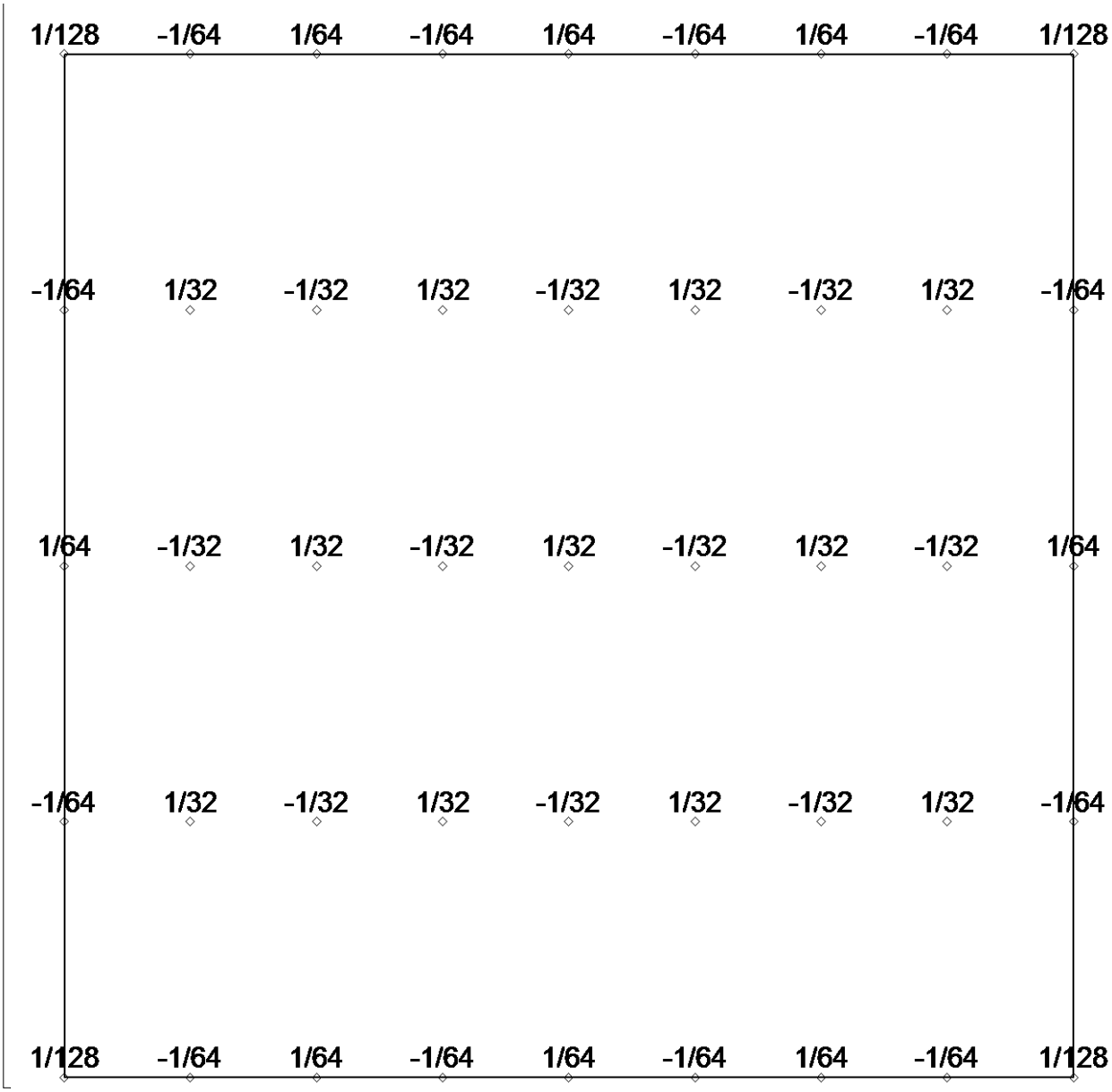
Smolyak-Quadratur

Das ist ein Summand der Smolyak-Regel (in der Darstellung von Folie 94):

```
> summand := (p1,l1,p2,l2,qrproc) ->
  tensorprodukt(Delta1(p1,l1,qrproc),Delta1(p2,l2,qrproc));
```

$$summand := (p1, l1, p2, l2, qrproc) \rightarrow \text{tensorprodukt}(\Delta 1(p1, l1, qrproc), \Delta 1(p2, l2, qrproc))$$

```
> bild2(summand(2,3,2,2,tr1));
```

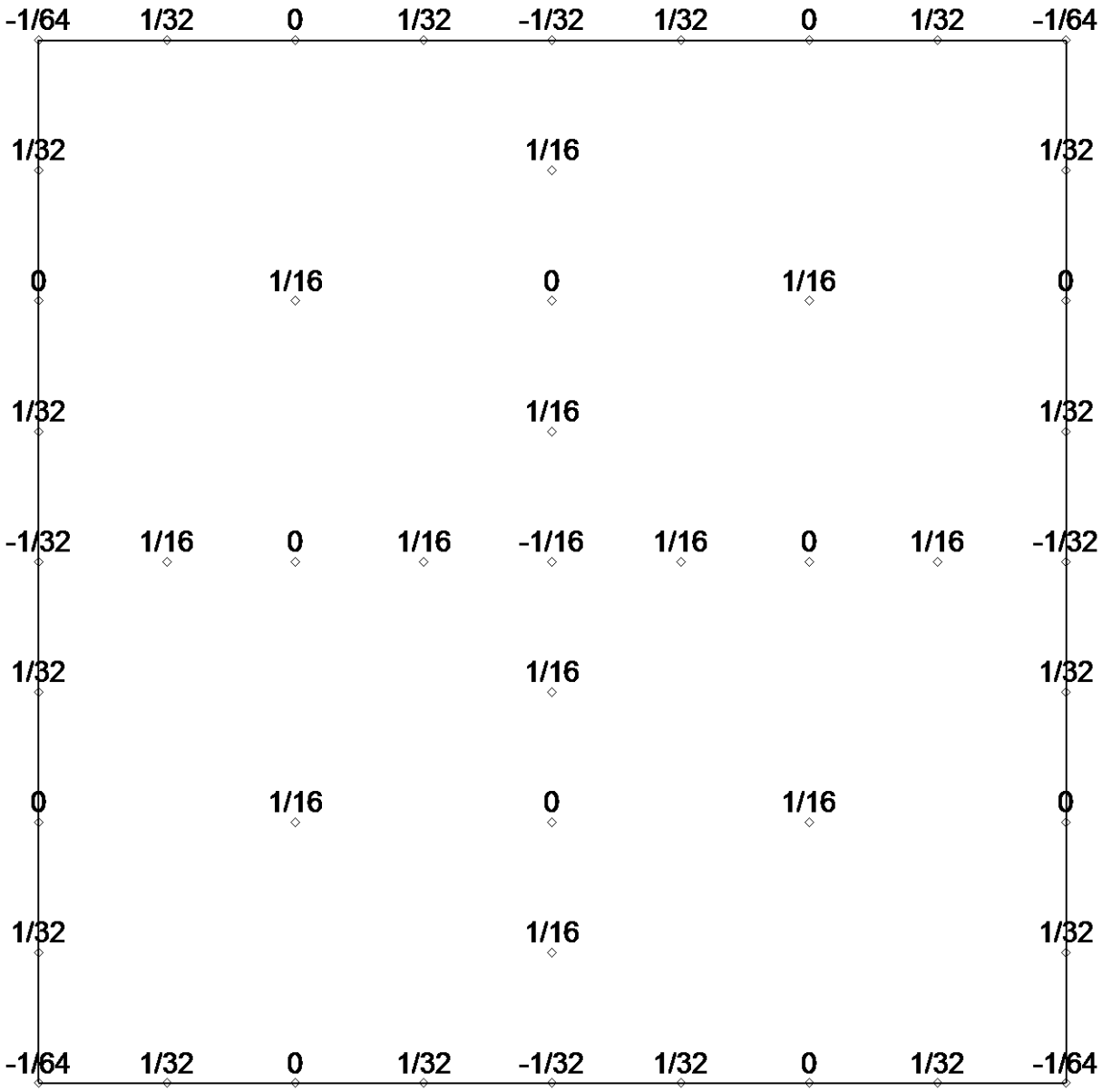


Und das ist die ganze Summe.
Wenn man genau hinschauen würde, würde man sehen, dass die Level hier in Wirklichkeit bei 0 anfangen zu zählen, so dass auch Q_{-1}^{(1)} auftritt. Da das aber genau so 0 ist wie Q_0^{(1)}, stört es nicht. Für die Beispiele von Folie 95 (s.u. werden wir es brauchen, weil wir dort ein anderes Q_0^{(1)} verwenden.

```
> smolyak := proc(p,n,qrproc)
    local qr,i,j;
    qr := table(sparse);
    for i from 0 to n+1 do
       for j from 0 to n+1-i do
          qr := kombinieren(qr, 1, summand(p, i, p, j,
  qrproc), 1)
       end do
    end do;
    return qr;
  end proc:
```

Für p=2 kommt wieder die von der Kombinationstechnik bekannte Regel raus:

```
> bild2(smolyak(2,3,tr1));
```




```
>
```

Auswertung

Wir nehmen wieder unsere beliebte Parabel:

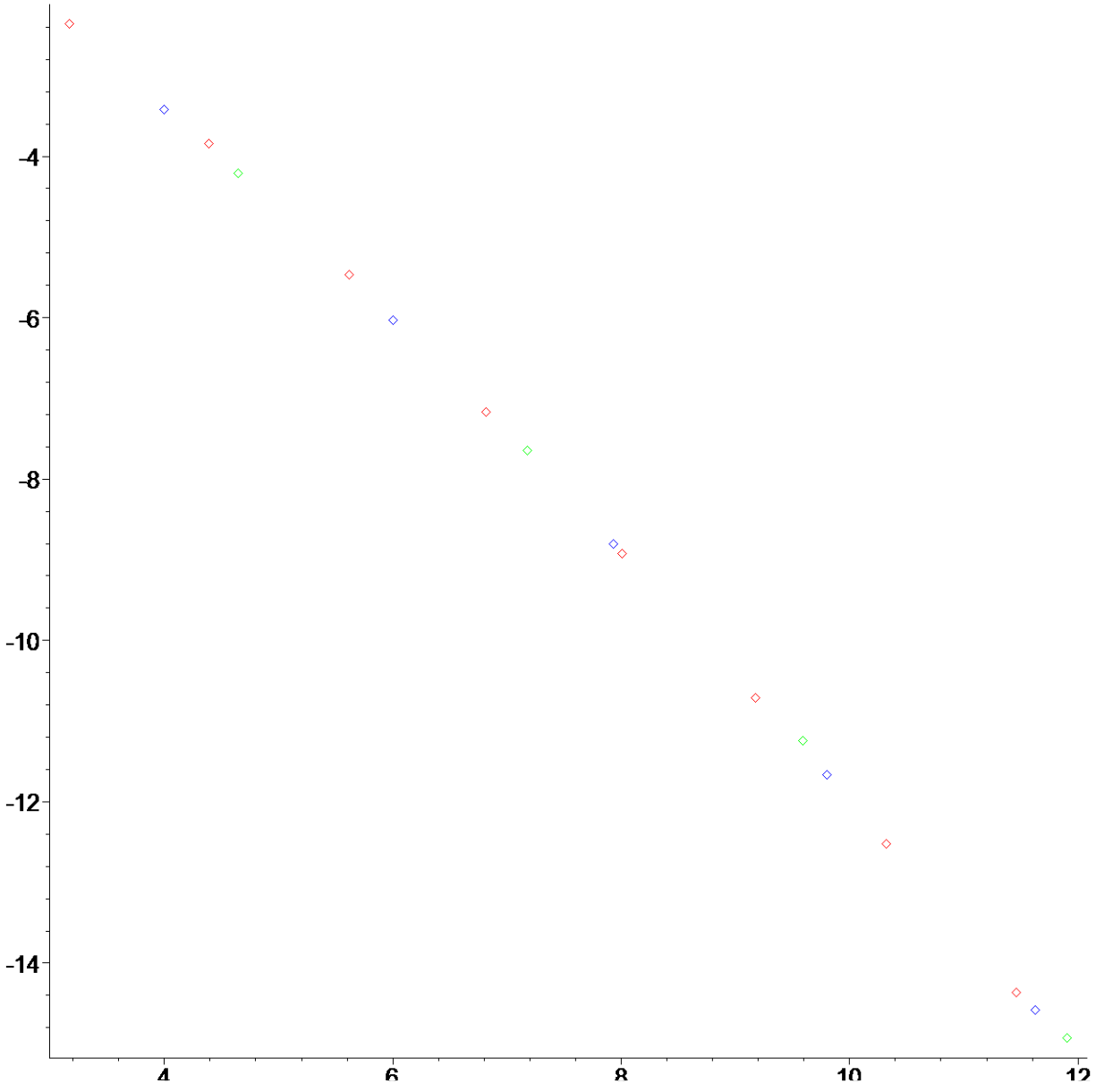
```
> u := (x,y) -> 16*x*(1-x)*y*(1-y):
> ref := int(int(u(x,y),x=0..1),y=0..1);
```

$$ref := \frac{4}{9}$$

Dann rechnen wir für p=2,3,4 immer bis zu n=maxtiefe[p] und merken uns Anzahl der Gitterpunkte und Fehler:

```
> maxtiefe := table():
  maxtiefe[2] := 8:
  maxtiefe[3] := 5:
  maxtiefe[4] := 4:
  erg := table():
```

```
  for p in [2,3,4] do
     for n to maxtiefe[p] do
       qr := smolyak(p, n, tr1):
       erg[p,n] := [nops([indices(qr)]),
                    abs(anwenden(qr, u)-ref)]
     end do
  end do;
> p2 := pointplot(
           [seq(map(log[2],erg[2,n]),n=1..maxtiefe[2])],
           poptions,color=RED):
> p3 := pointplot(
           [seq(map(log[2],erg[3,n]),n=1..maxtiefe[3])],
           poptions,color=BLUE):
> p4 := pointplot(
           [seq(map(log[2],erg[4,n]),n=1..maxtiefe[4])],
           poptions,color=GREEN):
> display([p2,p3,p4]);
```



```
> 
```

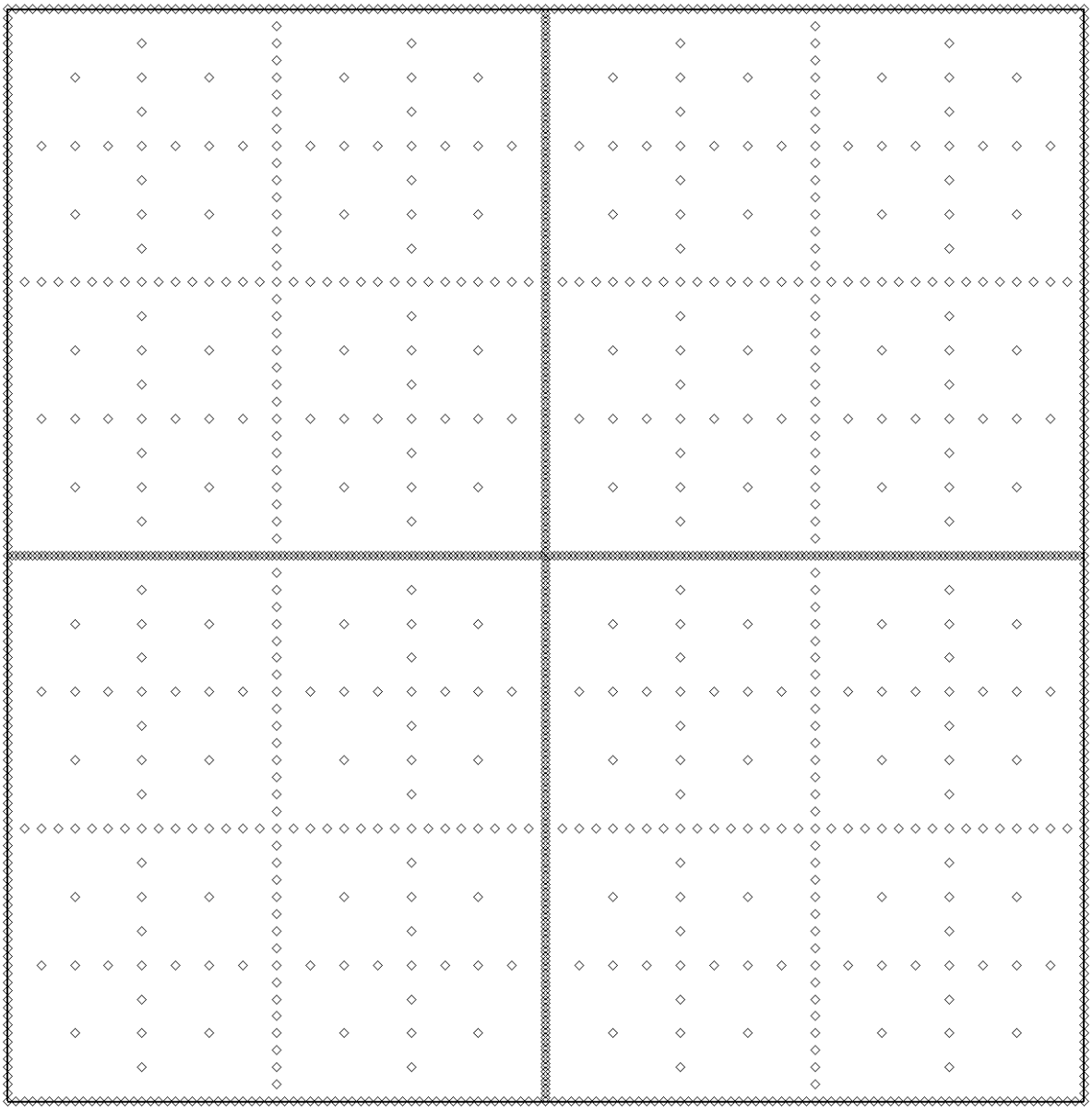
Bilder von Folie 95

Für die Beispiele von Folie 95 wurde die Hierarchie der 1D-Regeln etwas modifiziert: Q_0^{(1)} ist nun die Mittelpunktsregel (Q_{-1}^{(1)} bleibt 0):

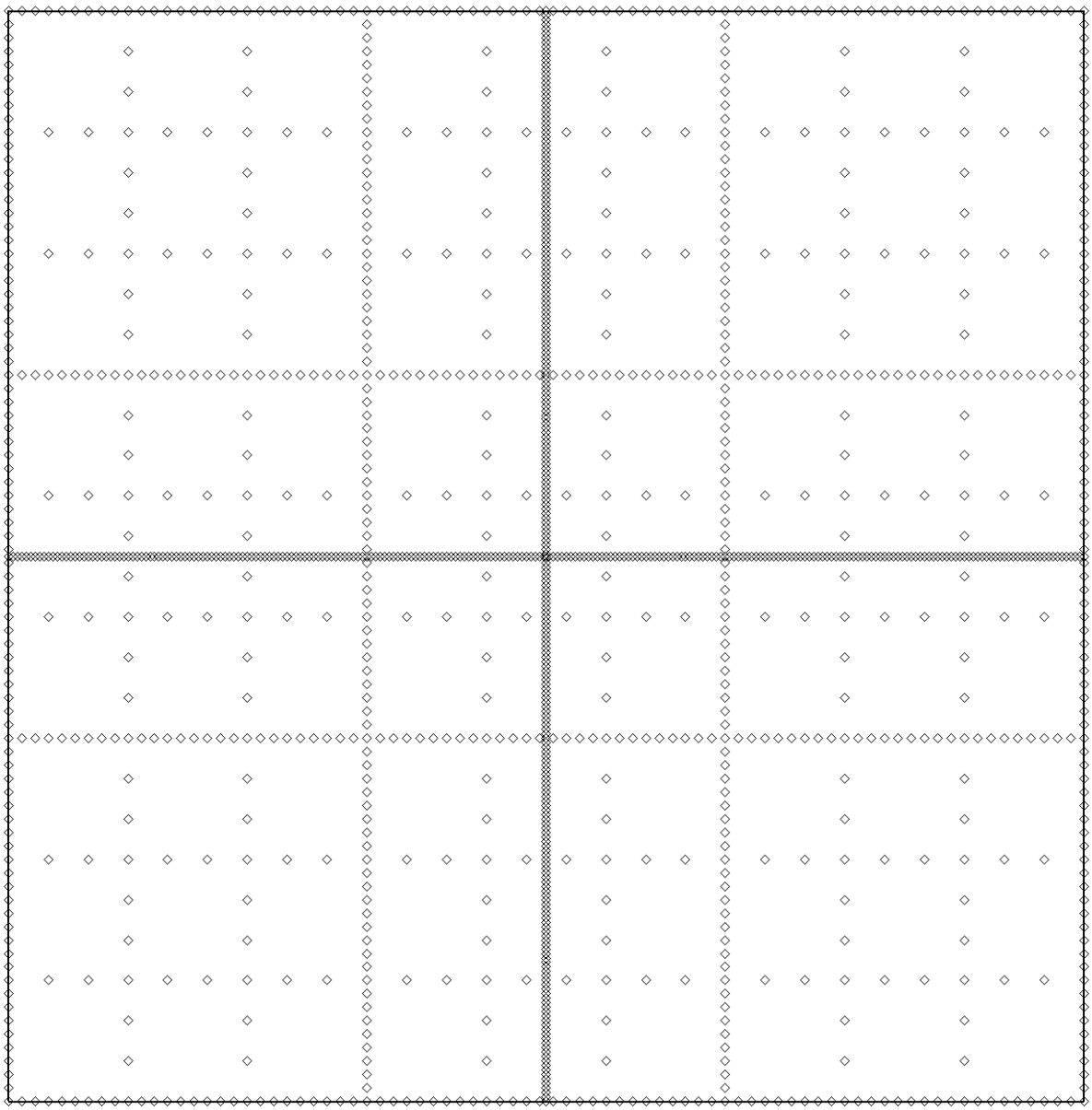
```
> tr1mod := proc(p,l)
     local N,tr,i,s;
     N := p^l;
     tr := table(sparse);
     if l>0 then
       s := 1/N;
       tr[0] := s/2;
       for i from 1 to N-1 do
          tr[i/N] := s;
       end do;
       tr[1] := s/2;
     elif l=0 then    # der Fall ist neu!
       tr[1/2] := 1;
     end if;
     return tr;
  end proc:
```

Bei p=2 zeigt das Bild in Wirklichkeit n=7:

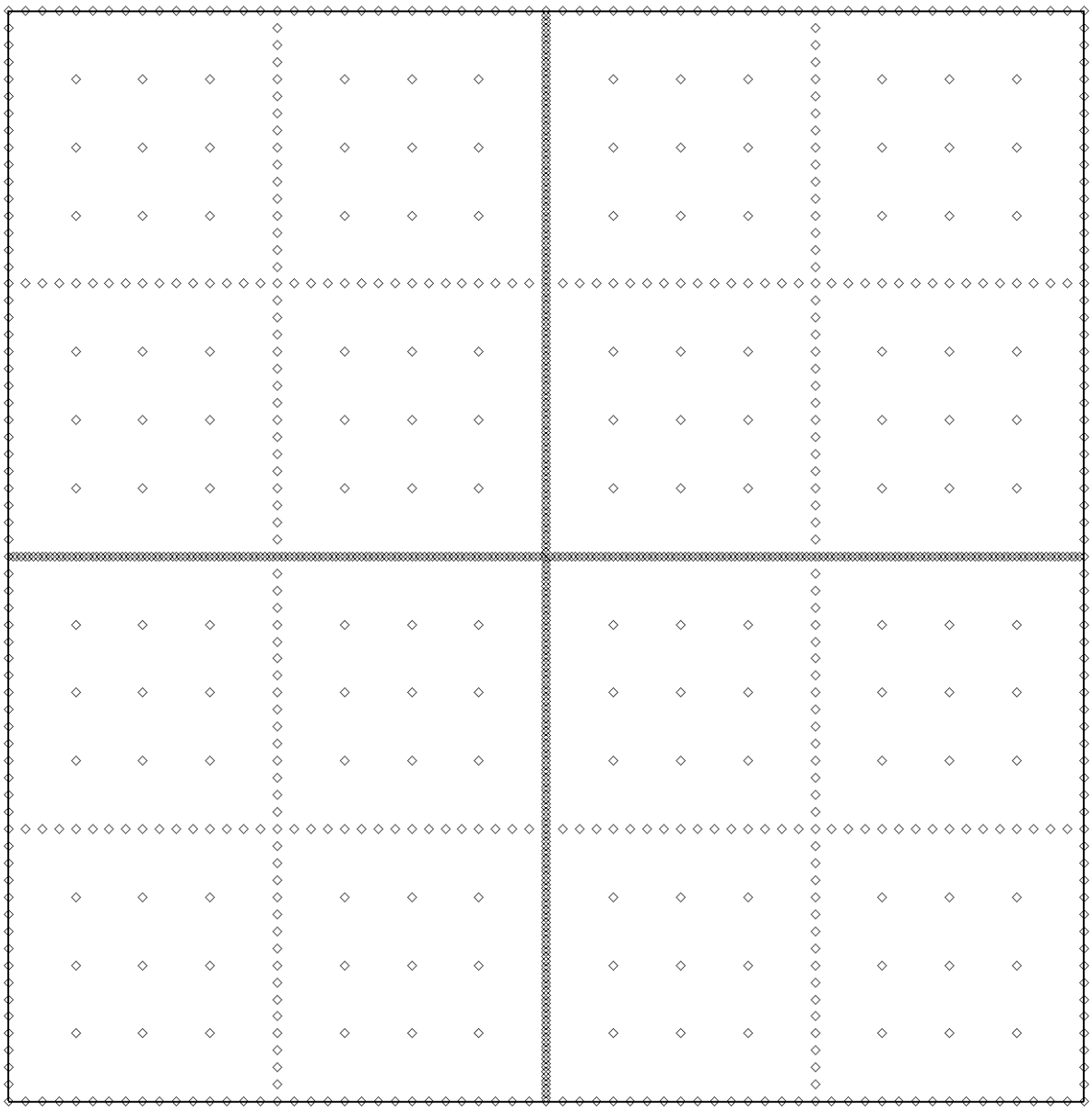
```
> bild2ohne(smolyak(2,7,tr1mod));
```



```
> bild2ohne(smolyak(3,4,tr1mod));
```



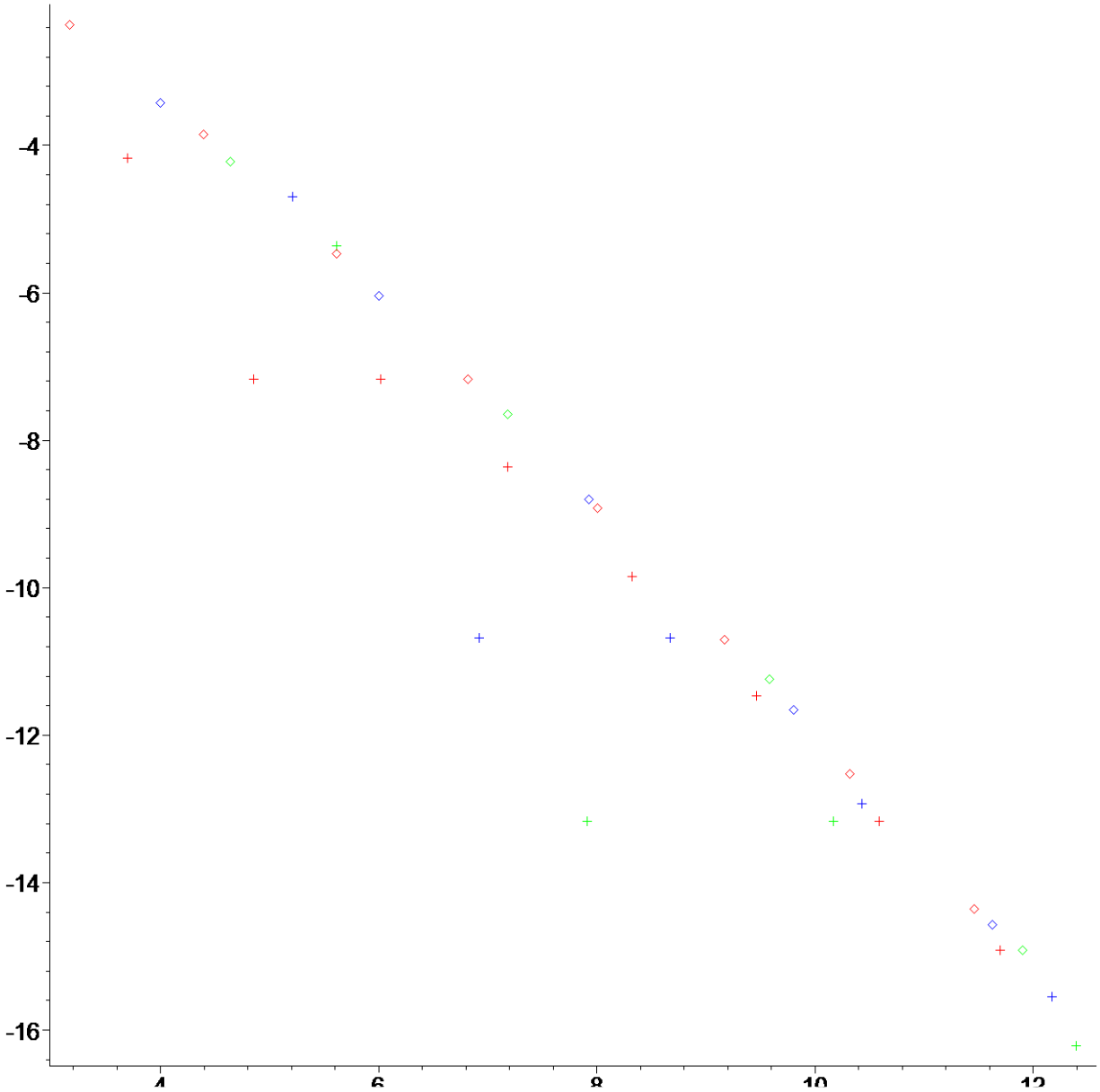
```
> bild2ohne(smolyak(4,3,tr1mod));
```



Dann rechnen wir wieder für p=2,3,4 immer bis zu n=maxtiefe[p] (im Bild sind das die Kreuze, die Färbung ist wie bei den Quadraten der nichtmodifizierten Regel):

```
> ergmod := table():
  for p in [2,3,4] do
     for n to maxtiefe[p] do
       qr := smolyak(p, n, tr1mod):
       ergmod[p,n] := [nops([indices(qr)]),
                       abs(anwenden(qr, u)-ref)]
    end do
  end do;
> p2mod := pointplot(
             [seq(map(log[2],ergmod[2,n]),n=1..maxtiefe[2])],
             poptions,color=RED,symbol=CROSS):
> p3mod := pointplot(
             [seq(map(log[2],ergmod[3,n]),n=1..maxtiefe[3])],
             poptions,color=BLUE,symbol=CROSS):
> p4mod := pointplot(
```

```
                [seq(map(log[2],ergmod[4,n]),n=1..maxtiefe[4])],
                poptions,color=GREEN,symbol=CROSS):
> display([p2,p3,p4,p2mod,p3mod,p4mod]);
```




```
> 
> 
```