Kapitel 14

Integralrechnung

Verständnisfragen

Sachfragen

1. Erläutern Sie den Begriff des bestimmten Integrals!
2. Wie lautet der Mittelwertsatz der Integralrechnung?
3. Erläutern Sie den Begriff der Stammfunktion!
4. Ist die Stammfunktion einer Funktion eindeutig bestimmt?
5. Wie hängt das Integral einer stetigen Funktion von der oberen Integrationsgrenze ab?
6. Wie lautet der Hauptsatz der Differenzial- und Integralrechnung?
7. Was versteht man unter partieller Integration?
8. Wie lautet die Substitutionsregel?
9. Was ist ein unbestimmtes Integral?
10. Was ist eine Quadraturformel?
11. Was ist eine Rechtecksregel?
12. Erläutern Sie die Trapezregel und ihre Herleitung!
13. Erläutern Sie die Kepler'sche Fassregel und ihre Herleitung!
14. Wie lautet die Simpson-Regel!
15. Was versteht man unter einer Newton-Côtes-Formel?
16. Erläutern Sie die Gauß-Quadratur!
17. Was ist eine gewöhnliche Differenzialgleichung?
18. Was versteht man unter einer allgemeinen, was unter einer speziellen Lösung einer gewöhnlichen Differenzialgleichung?
19. Wie kann das Euler-Verfahren motiviert werden?

20. Erläutern Sie den Verlauf des Euler-Verfahrens!
21. Was ist ein Einschrittverfahren?
22. Wie ist der lokale Diskretisierungsverfahren eines Diskretisierungsverfahrens definiert?
23. Was ist die Konsistenzordnung eines Diskretisierungsverfahrens?
24. Wie ist der globale Fehler eines Diskretisierungsverfahrens definiert?
25. Welche Konsistenzordnung besitzt das Euler-Verfahren?
26. Welche Konvergenzordnung besitzt das Euler-Verfahren? Welche Voraussetzungen müssen dafür erfüllt sein?
27. Wie kann die Konvergenzordnung eines Diskretisierungsverfahrens visualisiert werden?
28. Was versteht man unter einem Differenzialgleichungssystem?
29. Was ist ein Vektorfeld?
30. Wie kann ein Differenzialgleichungssystem mit dem Euler-Verfahren gelöst werden?
31. Erläutern Sie den bei einem Einzelschrittverfahren verwendete Ansatz!
32. Wie lautet das Verfahren von Heun?
33. Welche Quadraturformel liegt dem Verfahren von Heun zu Grunde?
34. Wie lautet das Verfahren von Runge-Kutta?
35. Welche Quadraturformel liegt dem Verfahren von Runge-Kutta zu Grunde?

Methodenfragen

1. Bestimmte Integrale durch Grenzwerte von Summen berechnen können.
2. Flächeninhalte durch bestimmte Integrale berechnen können.
3. Die Stammfunktion einer gegebenen Funktion angeben können.
4. Mit Hilfe eines Richtungsfelds eine Stammfunktion skizzieren können.
5. Mit Hilfe des Hauptsatzes der Differenzial- und Integralrechnung bestimmte und unbestimmte Integrale berechnen können.
6. Partielle Integration anwenden können.
7. Die Substitutionsregel anwenden können.
8. Entscheiden können, ob ein unbestimmtes Integral existiert.
9. Ein existierendes unbestimmtes Integral berechnen können.
10. Mit Hilfe von Quadraturformeln ein Integral annähern können.
11. Newton-Côtes-Formeln implementieren und anwenden können.
12. Gauss-Legendre-Formeln anwenden und implementieren können.
13. Für einfache gewöhnliche Differenzialgleichungen die allgemeine und die spezielle Lösung angeben können.

14. Das Euler-Verfahren anwenden und implementieren können.
15. Den globalen Fehler eines Diskretisierungsverfahrens grafisch darstellen können.
16. Das Heun-Verfahren anwenden und implementieren können.
17. Das Verfahren von Runge-Kutta anwenden und implementieren können.

Übungsaufgaben

1. Berechnen Sie über eine Grenzwertbetrachtung für die Unter- und Obersummen das bestimmte Integral über $f(x) = 4 - x^2$ von $a = 0$ nach $b = 2$.

   Lösung:
   Für den Grenzübergang bilden wir Unterteilungen des Intervalls $[0;2]$ mit Schrittweite $h = \frac{1}{2^k}$. Das entspricht $n = 2^k$ und $x_{i+1} - x_i = \frac{1}{2^{k-1}}$.
   Die Funktion $f(x) = 4 - x^2$ ist streng monoton fallend im Integrationsintervall; also ist die Obersumme gegeben durch

   $$O_k = \frac{1}{2^{k-1}} \sum_{i=1}^{n} f(x_{i-1}) = 8 - \frac{1}{2^{3k-1}} \sum_{i=1}^{n-1} i^2.$$

   Verwendet man die Summenformel

   $$\sum_{i=1}^{n} i^2 = \frac{1}{n(n+1)(2n+1)},$$

   dann erhält man für die Obersumme den Grenzwert $\frac{16}{3}$. Analog ergibt sich der gleiche Grenzwert für die Untersumme.

2. Schätzen Sie die bestimmten Integrale $\int_1^2 e^{-x^2} dx$ und $\int_{2\pi}^{3\pi} \frac{\sin(x)}{x} dx$ nach unten und oben ab!

   Lösung:
   Der Integrand ist monoton steigend, also werden Minimum und Maximum an den Rändern angenommen:

   $$e^{-1} \leq \int_1^2 e^{-x^2} dx \leq e^{-2},$$

   $$0 \leq \int_{2\pi}^{3\pi} \frac{\sin x}{x} dx \leq 0,13\pi.$$

3. Skizzieren Sie das Richtungsfeld für die Funktion $f(x) = -\frac{1}{x^2}$ und einige mögliche Stammfunktionen. Vergleichen Sie Ihre Skizzen mit der exakten Stammfunktion!

   Lösung:
   Die exakte Stammfunktion für das Richtungsfeld ist gegeben durch

   $$F(x) = x^{-1} + C.$$

   In Abbildung 14.1 sehen Sie eine Skizze des Richtungsfelds und zwei eingezeichnete Lösungen.

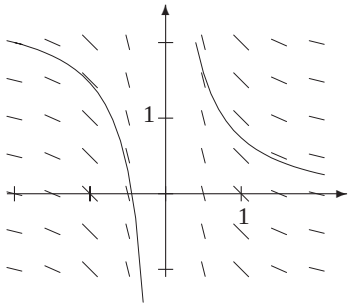

Abbildung 14.1: Das Richtungsfeld für die Funktion $f(x) = -\frac{1}{x^2}$; die linke Stammfunktion ist gegeben durch den Anfangswert $(-2, 2)$, die Stammfunktion im ersten Quadranten ist gegeben durch den Startwert $(0,4, 2)$

4. Geben Sie die folgenden unbestimmten Integrale an: $\int(2e^x - 10^x)dx$, $-\int a^x dx$, $\int(e^t - 4^t)dt$, $\int \frac{1-\cos^2(x)}{\cos^2(x)}dx$ und $\int \frac{1}{2-2x^2}dx$!

   Lösung:

$$\int(2e^x - 10^x)dx = e^x - \frac{10^x}{\ln(10)} + C,$$

$$-\int a^x dx = \frac{-a^x}{\ln(a)} + C,$$

$$\int(e^t - 4^t)dt = e^t - \frac{4^t}{\ln(4)} + C,$$

$$\int \frac{1-\cos^2(x)}{\cos^2(x)}dx = \frac{\sin(x) - x\cos(x)}{\cos(x)} + C,$$

$$\int \frac{1}{2-2x^2}dx = \frac{1}{4}(\ln(x-1) - \ln(x+1)) + C.$$

5. Berechnen Sie die folgenden bestimmten Integrale: $\int_0^3(x-1)dx$, $\int_0^{2\pi}\sin(x)dx$, $\int_1^4(x^2-4x)dx$ und $\int_1^5 \frac{5}{x^2}dx$!

   Lösung:
   $\int_0^3(x-1)dx = \frac{3}{2}$,
   $\int_0^{2\pi}\sin(x)dx = 0$
   $\int_1^4(x^2-4x)dx = -9$,
   $\int_1^5 \frac{5}{x^2}dx = 4$

6. Berechnen Sie mit Hilfe der Substitutionsregel oder partieller Integration: $\int \frac{8x-3}{4x^2-3x+2}dx$, $\int \frac{1}{x\ln x}dx$, $\int \frac{x}{\cos^2 x^2}dx$, $\int xe^x dx$, $\int \frac{\ln x}{x}dx$ und $\int e^x \sin x dx$.

   Lösung:
   $\int \frac{8x-3}{4x^2-3x+2}dx = \ln(2 - 3x + 4x^3) + C$,
   $\int \frac{1}{x\ln(x)}dx = \ln(\ln(x)) + C$,
   $\int \frac{x}{\cos^2 x^2}dx = \frac{1}{2}\tan(x^2) + C$,

$\int xe^x dx = e^x(x-1) + C$,
$\int \frac{\ln(x)}{x} dx = \frac{1}{2}\ln(x)^2 + C$,
$\int e^x \sin(x) dx = -\frac{1}{2}e^x(\cos(x) - \sin(x)) + C$

7. Bestimmen Sie eine Rekursionsformel für die Integrale $I_n = \int_0^1 x^n e^{-x} dx$.

Lösung:
Der Startwert der Rekursion ist das Integral

$$I_0 = \int_0^1 e^{-x} dx = \int_{-1}^0 e^x dx = 1 - \frac{1}{e}.$$

Dabei wurde die Substitution $x = -x$ verwendet.

Partielle Integration führt dann für I_{n+1} zu

$$I_{n+1} = 1 - (n+1)I_n.$$

8. Berechnen Sie die Fläche, die vom Graphen der Funktion $f(x) = x^2 - 5x + 6$ und der x-Achse im Intervall $[1; b]$ eingeschlossen wird!

Lösung:
Die Funktion hat Nullstellen in $x = 2$ und $x = 3$. Links von $x = 2$ und rechts von $x = 3$ ist die Funktion oberhalb der x-Achse. In Abbildung 14.2 sehen Sie den Verlauf der Funktion mit Definitionsbereich $[1; 4]$.

Für ein $b \leq 2$ ist die Fläche oberhalb der x-Achse gegeben durch

$$A(b) = \frac{1}{3}b^3 + 6b - \frac{5}{2}b^2 - \frac{23}{6}.$$

Für $b = 2$ erhält man die Fläche $A(2) = \frac{5}{6}$.

Für ein $b \in (2; 3]$ muss das Vorzeichen für den Bereich von 2 bis b umgekehrt werden:

$$A(b) = \int_1^2 f(x)dx - \int_2^b f(x)dx = \frac{5}{6} - \int_2^b (x^2 - 5x + 6)dx = -\frac{1}{3}b^3 + \frac{5}{2}b^2 - 6b + \frac{11}{2}.$$

Für $b = 3$ erhält man dann die Fläche $A(3) = 1$.

Rechts von $x = 3$ ist dann

$$A(b) = 1 + \int_3^b f(x)dx = \frac{1}{3}b^3 - \frac{5}{2}b^2 + 6b - \frac{7}{2}.$$

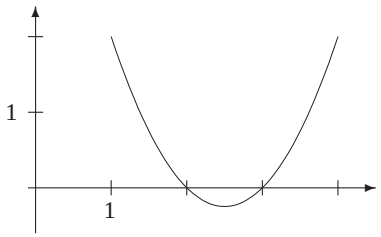

Abbildung 14.2: Der Verlauf des Integranden in Aufgabe 8

9. Berechnen Sie die Fläche zwischen den Graphen von $f(x) = x + 1$ und $g(x) = e^{-x}$ im Intervall $[0; 3]$!

   Lösung:
   Beide Integranden verlaufen oberhalb der x-Achse; und schneiden sich im Punkt $(0, 1)$. Also ist die gesuchte Fläche gegeben durch

$$A = \int_0^3 x + 1 - e^{-x} dx = \left[x + \frac{1}{2}x^2 - e^{-x} \right]_0^3 = \frac{17}{2} - \frac{1}{e^3}.$$

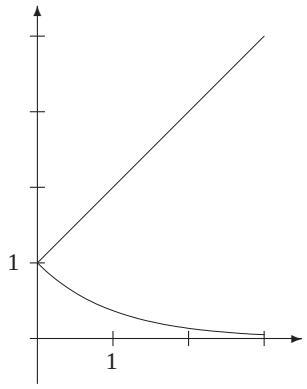

Abbildung 14.3: Der Verlauf des Integranden in Aufgabe 9

10. Untersuchen Sie, ob die folgenden uneigentlichen Integrale existieren: $\int_0^\infty \cos(x)dx$, $\int_{-\infty}^\infty \frac{1}{1+x^2}dx$, $\int_1^4 \frac{x}{\sqrt{|x^2-4|}}dx$, $\int_0^4 \frac{2}{x\sqrt{x}}dx$ und $\int_0^{32} \frac{1}{\sqrt[5]{x^3}}dx$!

    Lösung:
    $\int_0^\infty \cos(x)dx$ existiert nicht;
    $\int_{-\infty}^\infty \frac{1}{1+x^2}dx$ existiert und hat den Wert π;
    $\int_1^4 \frac{x}{\sqrt{|x^2-4|}}dx$ existiert und hat den Wert $3\sqrt{3}$,
    $\int_0^4 \frac{2}{x\sqrt{x}}dx$ existiert nicht,
    $\int_0^{32} \frac{1}{\sqrt[5]{x^3}}dx$ existiert und hat den Wert 10.

11. Leiten Sie die Kepler'sche Fassregel durch Integration des quadratischen Interpolationspolynoms her!

    Lösung:
    Die Stützstellen seien durch die äquidistanten Punkte x_{i-1}, x_i und x_{i+1} gegeben. Wenn die Teilintervalle äquidistant sind, dann ist das Polynom mit $h = x_i - x_{i-1} = x_{i+1} - x_i$ bestimmt als

$$p(x) = f_{i-1} + \frac{f_i - f_{i-1}}{h}(x - x_{i-1}) + \frac{f_{i+1} - 2f_i + f_{i-1}}{2h^2}(x - x_{i-1})(x - x_i).$$

Dann ist $\int_{x_{i-1}}^{x_{i+1}} p(x)dx$ gegeben durch

$$\begin{aligned}\int_{x_{i-1}}^{x_{i+1}} p(x)dx = f_{i-1}\int_{x_{i+1}}^{x_{i+1}} dx \\ + \frac{f_i - f_{i-1}}{h}\int_{x_{i+1}}^{x_{i+1}} (x - x_{i-1})dx \\ + \frac{f_{i+1} - 2f_i + f_{i-1}}{2h^2}\int_{x_{i+1}}^{x_{i+1}} (x - x_{i-1})(x - x_i)dx.\end{aligned}$$

Alle Integrale können mit der Substitution $u = x - x_{i-1}$ gelöst werden es ist dann

$$\int_{x_{i+1}}^{x_{i+1}} (x - x_{i-1})dx = 2(f_i - f_{i-1})h, \int_{x_{i+1}}^{x_{i+1}} (x - x_{i-1})(x - x_i)dx = \frac{h}{3}(f_{i+1} - 2f_i + f_{i-1}).$$

Addiert man diese drei Ergebnisse, ergibt sich die Fassregel.

12. Nähern Sie die Integrale $\int_0^1 \sqrt{3x+4}\,dx$ und $\int_0^1 4\sqrt{1-x^2}\,dx$ mit der Trapez- und der Simpsonregel an und vergleichen Sie die Ergebnisse mit den exakten Werten!

Lösung:
Die Stammfunktion von $\int \sqrt{3x+4}\,dx$ finden Sie als Beispiel für die Anwendung der Substitutionsregel im Buch. Es ist

$$\int \sqrt{3x+4}\,dx = \frac{2}{9}(3x+4)^{\frac{3}{2}}.$$

Für das bestimmte Integral gilt dann $I \approx 2,337\,835$.

Die Trapezregel mit $n = 2$ ergibt die Näherung $2,322\,875\,656$. Für $n = 5$ und der Zerlegung $x_0 = 0, x_i = \frac{i}{4}$ ergibt sich die Näherung $2,336\,883\,252$.

Für $n = 11$ und $h = \frac{1}{10}$ erhält man $2,337\,682\,861$.

Die Fassregel ergibt die Näherung $2,337\,763\,805$.

Für $k = 2$ und $h = 0,25$ ergibt die Simpson-Regel die Näherung $2,337\,830\,413$.

$k = 5$ und $h = 0,1$ bei der Simpson-Regel ergibt $2,337\,835\,241$.

Die exakte Lösung des zweiten Integrals ist π, Sie finden die Integration wiederum als Beispiel der Substitutionsregel; das Ergebnis ist die Fläche eines Kreises mit Radius 1.

Die Trapezregel für $n = 2$ ergibt die Näherung 2, $n = 5$ ergibt $2,995\,709\,068$, $n = 11$ die Näherung $3,104\,518\,326$.

Die Fassregel ergibt die Näherung $2,976\,067\,743$; $k = 2$ die Näherung $3,083\,595\,155$ und $k = 5$ die Näherung $3,127\,008\,159$.

Implementierungen in Java finden Sie in der Lösung der Aufgabe 14 auf Seite 175!

13. Berechnen Sie das Integral $\int_{1,05}^{1,35} f(x)dx$ mit Hilfe der Simpson-Regel, falls für f die Werte in Tabelle 14.1 bekannt sind!

Tabelle 14.1: Wertetabelle für Aufgabe 13

x	1,05	1,10	1,15	1,2	1,25	1,3	1,35
$f(x)$	2,36	2,5	2,74	3,04	3,46	3,98	4,6

Lösung:
Die Näherung durch die Simpson-Regel für $k = 3$ und $h = 0,05$ ist $0,957\,333\,333$.

Die Implementierung in Java finden Sie in der Lösung der Aufgabe 14 auf Seite 175!

14. Implementieren Sie verschiedene Newton-Côtes-Formeln und die Gauß-Quadratur in der Programmiersprache Ihrer Wahl und testen Sie die Implementierung an Hand des Integrals $\int_0^{\frac{\pi}{2}} x^2(x^2-2)\sin(x)\,dx$!

Lösung:
Die Stammfunktion von $\int x^2(x^2-2)\sin(x)\,dx$ ist gegeben durch

$$-(28-14x^2+x^4))\cos(x)+4x(-7+x^2)\sin(x)).$$

Das bestimmte Integral ergibt sich dann als

$$\int_0^{\frac{\pi}{2}} x^2(x^2-2)\sin(x)\,dx = 28-14\pi+\frac{\pi^3}{2} \approx -0,479\,159.$$

Die Trapezregel ergibt die folgenden Näherungen: für $n = 3$ $-0,020\,944\,909$, für $n = 5$ $-0,361\,461\,274\,608$ und für $n = 10$ $-0,455\,783\,021\,609$.

Die Simpson-Regel ergibt die folgenden Näherungen: bei der Fassregel erhält man $-0,329\,850\,875$, für $k = 2$ die Näherung $-0,474\,966\,73$ und für $k = 5$ die Näherung $-0,479\,079\,334$.

Gauss-Legendre mit $n = 2$: $-0,583\,495\,559$,; Gauss-Legendre mit $n = 3$: $-0,473\,832\,940$, mit $n = 10$ (die Gewichte und Stützstellen finden Sie im Buch) erhält man $-0,479\,158\,453$.

Bei der Implementierung in *Java* wird zuerst eine abstrakte Oberklasse implementiert, das für besetzte Gewichte und der fest implementierten Funktion der Aufgabe in der `berechne()`-Funktion die Quadraturformel bildet. Die Gauß-Legendre-Quadratur überschreibt diese Funktion.

```
//-------------------------------------------------------------
// quadraturFormel.java
// Virtuelle Oberklasse für die Algorithmen für
// Quadraturformeln.
//-------------------------------------------------------------
//Autor:           Manfred Brill
//Letzte Änderung: 14. 11. 2004
//-------------------------------------------------------------
public abstract class quadraturFormel {

    protected double a, b; // Integrationsgrenzen
    protected int n;       // Anzahl der Stützstellen
    protected double h;    // Schrittweite
    protected double y[];  // Funktionswerte
    protected double w[];  // Gewichte

    protected quadraturFormel() {
        a = 0.0;
        b = 1.0;
        n = 3;
        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];
    }

    protected quadraturFormel(double aa, double bb, int nn) {
        a = aa; b = bb;
        n = nn;
        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];
```

```
    }

    public void setGrenzen(double aa, double bb) {
        a = aa; b = bb;
    }

    public void setA(double aa) {
        a = aa;
    }

    public void setB(double bb) {
        b = bb;
    }

    public void setN(int j) {
        n = j;
    }

    public void setSchrittweite(double delta) {
        h = delta;
    }

    public int getAnzahlStuetzstellen() {
        return n;
    }

    public double getSchrittweite() {
        return h;
    }

    public void setY(double yy[]) {
        for (int i=0; i<n; i++) y[i] = yy[i];
    }

    public void setGewichte(double ww[]) {
        for (int i=0; i<n; i++) w[i] = ww[i];
    }

    protected double funktion(double x) {
        // Funktion für Aufgabe 14
        return x*x*(x*x-2.0)*Math.sin(x);
    }

    public double berechne() {
        double summe = 0.0;

        for (int i=0; i<n; i++) {
            summe += w[i]*y[i];
        }
        return summe;
    }
}
```

```
// Die Kepler'sche Fassregel
public class fassRegel extends quadraturFormel {

    public fassRegel() {
        double ww;
```

```
        a = 0.0;
        b = 1.0;
        n = 3;
        h = (b-a)/(double)(n-1);

        y[0] = funktion(a);
        y[1] = funktion((b-a)/2.0);
        y[2] = funktion(b);

        ww = (b-a)/6.0;
        w[0] = ww;
        w[1] = 4.0*ww;
        w[2] = ww;
    }

    public fassRegel(double aa, double bb, double yy[]) {
        double ww;
        a = aa; b = bb; n = 3;
        h = (b-a)/(double)(n-1);

        for (int i=0; i<n; i++) y[i] = yy[i];

        ww = (b-a)/6.0;
        w[0] = ww;
        w[1] = 4.0*ww;
        w[2] = ww;
    }

    //! Konstruktor mit eingebauter Funktion der Aufgabe
    public fassRegel(double aa, double bb) {
        double ww;
        a = aa; b = bb; n = 3;
        h = (b-a)/(double)(n-1);

        y[0] = funktion(a);
        y[1] = funktion((b-a)/2.0);
        y[2] = funktion(b);

        ww = (b-a)/6.0;
        w[0] = ww;
        w[1] = 4.0*ww;
        w[2] = ww;
    }
}
```

```
// Die Trapezregel
public class trapezRegel extends quadraturFormel {

    public trapezRegel() {

        a = 0.0;
        b = 1.0;
        n = 3;
        h = (b-a)/(double)(n-1);

        y = new double[3];
        y[0] = funktion(a);
        y[1] = funktion((b-a)/2.0);
        y[2] = funktion(b);
```

```
        w = new double[3];
        w[0] = h/2.0;
        w[1] = h;
        w[2] = h/2.0;
    }

    public trapezRegel(double aa, double bb, int nn, double yy[]) {
        a = aa; b = bb; n = nn;
        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];

        for (int i=0; i<n; i++) y[i] = yy[i];

        w[0] = h/2.0;
        for (int i=1; i<n-1; i++) w[i] = h;
        w[n-1] = h/2.0;
    }

    //! Konstruktor mit eingebauter Funktion der Aufgabe
    public trapezRegel(double aa, double bb, int nn) {
        int i;

        a = aa; b = bb; n = nn;
        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];

        y[0] = funktion(a);
        for (i=1; i<n-1; i++) y[i] = funktion(a + (double)i *h);
        y[n-1] = funktion(b);

        w[0] = h/2.0;
        for (i=1; i<n-1; i++) w[i] = h;
        w[n-1] = h/2.0;
    }
}
```

```
// Die Simpson-Regel
public class simpsonRegel extends quadraturFormel {
    private int k;

    public simpsonRegel() {

        a = 0.0;
        b = 1.0;
        n = 3;
        k = 1;
        h = (b-a)/(double)(n-1);

        y = new double[2];
        y[0] = funktion(a);
        y[1] = funktion((b-a)/2.0);
        y[2] = funktion(b);

        w = new double[2];
        w[0] = h/3.0;
        w[1] = 4.0*h;
```

```
        w[2] = h/3.0;
    }

    public simpsonRegel(double aa, double bb, int nn, double yy[]) {
        int i;

        a = aa; b = bb; n = nn; k = (n-1)/2;
        h = (n-1)/2;

        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];

        for (i=0; i<n; i++) y[i] = yy[i];

        w[0] = h/3.0;
        for (i=1; i<k; i++) w[2*i] = (2.0*h)/3.0;
        for (i=1; i<=k; i++) w[2*i-1] = (4.0*h)/3.0;
        w[n-1] = h/3.0;
    }

    //! Konstruktor mit eingebauter Funktion der Aufgabe
    public simpsonRegel(double aa, double bb, int nn) {
        int i;

        a = aa; b = bb; n = nn; k = (n-1)/2;
        h = (b-a)/(double)(n-1);
        y = new double[n];
        w = new double[n];

        y[0] = funktion(a);
        for (i=1; i<n-1; i++) y[i] = funktion(a + (double)i *h);
        y[n-1] = funktion(b);

        w[0] = h/3.0;
        for (i=1; i<k; i++) w[2*i] = (2.0*h)/3.0;
        for (i=1; i<=k; i++) w[2*i-1] = (4.0*h)/3.0;
        w[n-1] = h/3.0;
    }
}
```

```
// Gauss-Legendre-Quadratur für n=2, 3 und 10
public class gaussLegendre extends quadraturFormel {

    public gaussLegendre() {

        a = -1.0;
        b = 1.0;
        n = 2;
        h = (b-a)/(double)(n-1);

        w = new double[2];
    }

    public gaussLegendre(double aa, double bb, int nn) {
        a = aa; b = bb; n = nn;

        h = (b-a)/(double)(n-1);
        w = new double[n]; y = new double[n];
```

```
    }

    // Berechne-Funktion wird hier überschrieben!
    public double berechne() {
        double ergebnis;

        // Nur die Fälle n=2, 3 und 10 sind implementiert!
        switch (n) {
         case 2:   ergebnis = legendre2();
                   break;
         case 3:   ergebnis = legendre3();
                   break;
         case 10:  ergebnis = legendre10();
                   break;
         default:  n=2;
                   ergebnis = legendre2();
                   break;
        }
        return ergebnis;
    }

    // Koordinatentransformation vom Intervall [a,b] auf [-1,1]
    private double phi(double x) {
        return 0.5*(a+b) + 0.5*(b-a)*x;
    }

    private double legendre2() {
        double x = Math.sqrt(3.0)/3.0;

        return 0.5*(b-a)*(funktion(phi(-x)) + funktion(phi(x)));
    }

    private double legendre3() {
        double y = Math.sqrt(15.0)/5.0;
        double w1 = 5.0/9.0;
        double w2 = 8.0/9.0;

        return 0.5*(b-a)*(w1*funktion(phi(-y)) + w2*funktion(phi(0.0)) +
        w1*funktion(phi(y)));
    }

    private double legendre10() {
        double x[] = new double[10];
        double yy, summe;
        int i;

        x[0] = -0.9739075285;
        x[1] = -0.8650633666;
        x[2] = -0.67940956582;
        x[3] = -0.4333953941;
        x[4] = -0.1488743389;
        x[5] = 0.1488743389;
        x[6] = 0.4333953941;
        x[7] = 0.67940956582;
        x[8] = 0.8650633666;
        x[9] = 0.9739075285;

        w[0] = 0.0666713443;
```

```
        w[1] = 0.1494513491;
        w[2] = 0.2190863625;
        w[3] = 0.2692667193;
        w[4] = 0.2955242247;
        w[5] = 0.2955242247;
        w[6] = 0.2692667193;
        w[7] = 0.2190863625;
        w[8] = 0.1494513491;
        w[9] = 0.0666713443;

        summe = 0.0;
        for (i=0; i<10; i++) {
            summe += w[i]*funktion(phi(x[i]));
        }
        return 0.5*(b-a)*summe;
    }
}
```

Und hier das verwendete Hauptprogramm für die Berechnungen:

```
/*
 * Quadraturen für die Lösung der Aufgabe 14, Kapitel 14
 */

public class int14 {

// ------------------------------------------------
public static void main (String[] args) {
    System.out.println("-------------------------------------");

    kepler();

    System.out.println("-------------------------------------");

    trapez();

    System.out.println("-------------------------------------");

    simpson();

    System.out.println("-------------------------------------");

    gauss();
}

  public static void kepler() {
    double solution=0.0;
    fassRegel quad = new fassRegel(0.0, Math.PI/2.0);

    solution = quad.berechne();
    System.out.println(
    "Die berechnete Lösung der Fass-Regel: "+solution);
  }

  public static void trapez() {
    double solution=0.0;
    int n=10;
    trapezRegel quad = new trapezRegel(0.0, Math.PI/2.0, n);
```

```
    solution = quad.berechne();
    System.out.println(
    "Die berechnete Lösung der Trapez-Regel für n = "+n+": "+solution);
  }

  public static void simpson() {
    double solution=0.0;
    int n=11;
    simpsonRegel quad = new simpsonRegel(0.0, Math.PI/2.0, n);

    solution = quad.berechne();
    System.out.println(
    "Die berechnete Lösung der Simpson-Regel für n = "+n+": "+solution);
  }

  public static void gauss() {
    double solution=0.0;
    int n=10;
    gaussLegendre quad = new gaussLegendre(0.0, Math.PI/2.0, n);

    solution = quad.berechne();
    System.out.println(
    "Die berechnete Lösung der Gauss-Legendre-Quadratur für n = "+n+": "
        +solution);
  }
}
```

15. Verwenden Sie Ihre Implementierungen und Ergebnisse aus Aufgabe 14, um die Fehler der Newton-Côtes-Formeln in einer doppelt-logarithmischen Darstellung aufzutragen! Welche Form muss die Kurve besitzen?

    Lösung:
    Die Ordnung der Quadraturformel ist die Steigung der Geraden, falls eine doppelt-logarithmische Skala verwendet wird. Es ist $e(h) = Ch^p(b-a)f^{(p)}(\xi)$. Durch Logarithmieren auf beiden Seiten erhält man eine Gerade mit Steigung p. Abbildung 14.4 zeigt das grafische Ergebnis, Tabelle 14.2 die numerischen Resultate.

Tabelle 14.2: Die numerischen Ergebnisse für Trapez- und Simpson-Regel für Aufgabe 15

n	h	Trapez-Regel	Simpson-Regel
2	1,0	$1,409 \cdot 10^{-1}$	$4,789 \cdot 10^{-1}$
3	0,5	$3,565 \cdot 10^{-2}$	$5,793 \cdot 10^{-4}$
5	0,25	$8,940 \cdot 10^{-3}$	$3,701 \cdot 10^{-5}$
9	0,125	$2,237 \cdot 10^{-3}$	$2,326 \cdot 10^{-6}$
17	0,062 5	$5,593 \cdot 10^{-4}$	$1,456 \cdot 10^{-7}$
33	0,031 25	$1,398 \cdot 10^{-4}$	$9,103 \cdot 10^{-9}$
65	0,0156 25	$3,496 \cdot 10^{-5}$	$5,69 \cdot 10^{-10}$

16. Bestimmen Sie für das Anfangswertproblem $y' = y + 2$, $y(0) = 1$ eine Näherung für $y(2)$ durch die Verfahren von Euler, Heun und Runge-Kutta mit Schrittweite $h = 1$!

    Lösung:
    Die exakte Lösung des Anfangswertproblems ist gegeben durch $y(x) = -2 + Ce^x$; mit dem Anfangswert ist dann $y(x) = -2 + 3e^x$.

    Die exakte Lösung ist $y(2) \approx 20,167$.

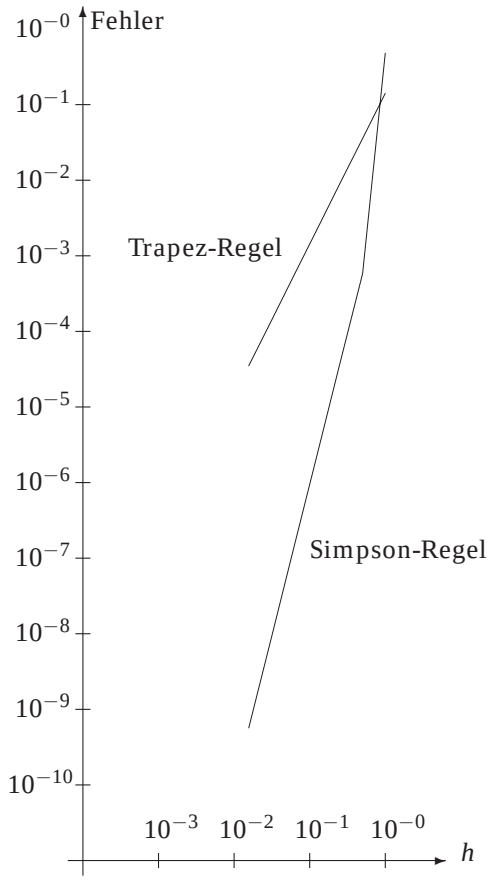

Abbildung 14.4: Die Fehler bei Trapez- und Simpson-Regel in einer doppelt-logarithmischen Skala für Aufgabe 15

Das Euler-Verfahren mit $h = 1$ berechnet die Näherung $y_2 = 10,0$; Heun berechnet $y_2 = 16,75$; Runge-Kutta berechnet $y_2 = 20,005$.

17. Implementieren Sie das Euler-, das Heun- und das Runge-Kutta-Verfahren in der Programmiersprache Ihrer Wahl und testen Sie Ihre Implementierung an Hand des Anfangswertproblems $y' = y + 2, y(0) = -1$ und $y' = e^{2x} - 2y, y(0) = 1$ auf dem Intervall $[0; 2]$. Skizzieren Sie Ihre Ergebnisse, und erstellen Sie eine doppelt-logarithmische Darstellung der globalen Fehler!

Lösung:
Die exakte Lösung des Anfangswertproblems

$$y' = y + 2, y(0) = -1$$

ist $y(x) = e^x - 2$.

Die exakte Lösung des Anfangswertproblems

$$y' = e^{2x} - 2y, y(0) = 1$$

ist gegeben durch

$$y(x) = \frac{1}{4}e^{2x} + \frac{3}{4}e^{-2x}$$

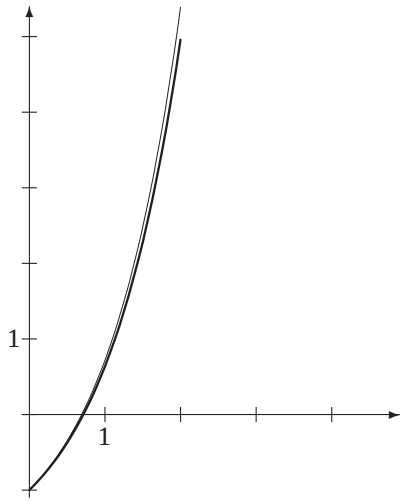

Abbildung 14.5: Ergebnis des Euler-Verfahrens ($n = 17$) für Aufgabe 17, $y' = y + 2$. Die berechnete Näherung ist fett dargestellt.

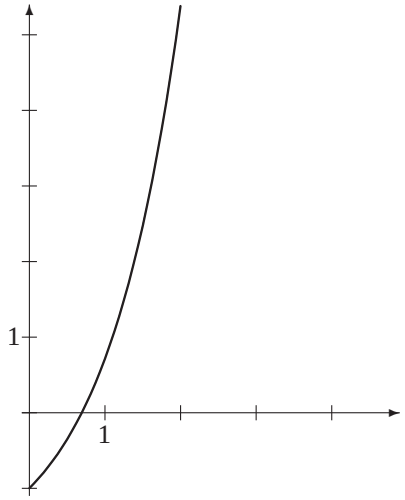

Abbildung 14.6: Ergebnis des Heun-Verfahrens ($n = 17$) für Aufgabe 17, $y' = y + 2$. Die berechnete Näherung ist fett dargestellt.

mit $y(2) \approx 13,663$.

In den Abbildungen 14.5, 14.6 und 14.7 sehen Sie die Ergebnisse für die rechte Seite $f(x, y) = y + 2$; in den Abbildungen 14.8, 14.9 und 14.10 die für die rechte Seite $f(x, y) = e^{2x} - 2y$. Die Lösungen wurden mit dem folgenden Java-Code berechnet:

```
//-------------------------------------------------------
// einzelschrittVerfahren.java
// Virtuelle Oberklasse für die Algorithmen für
// Einzelschrittverfahren.
//-------------------------------------------------------
//Autor:           Manfred Brill
//Letzte Änderung: 14. 11. 2004
```

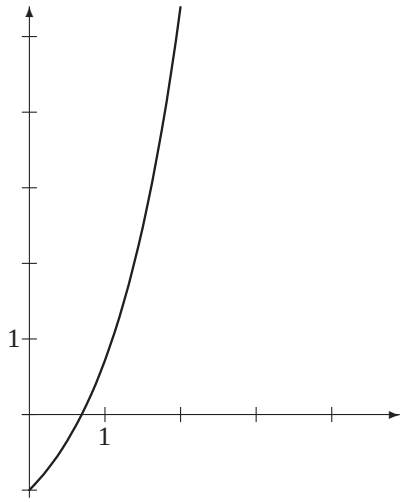

Abbildung 14.7: Ergebnis des Runge-Kutta-Verfahrens ($n = 17$) für Aufgabe 17, $y' = y + 2$. Die berechnete Näherung ist fett dargestellt.

```
//-----------------------------------------------------------
public abstract class einzelschrittVerfahren {

    protected double a, b;     // Integrationsgrenzen
    protected int n;           // Anzahl der Stützstellen
    protected double h;        // Schrittweite
    protected double y0;       // Anfangswert für x=a
    protected double x[], y[]; // Ergebnisvektoren des Verfahrens

    protected einzelschrittVerfahren() {
        a = 0.0;
        b = 1.0;
        n = 3;
        h = (b-a)/(double)(n-1);
        y0 = 0.0;
        x = new double[n];
        y = new double[n];
    }

    protected einzelschrittVerfahren(double aa, double bb,
                                     double yy, int nn) {
        a = aa; b = bb;
        n = nn;
        h = (b-a)/(double)(n-1);
        y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public void getErgebnis(double xw[], double yw[]) {
        for (int i=0; i<n; i++) {
            xw[i] = x[i];
            yw[i] = y[i];
        }
```

```
    }

    public void printErgebnis() {

        for (int i=0; i<n; i++)
            System.out.println("("+x[i]+","+y[i]+")");
    }

    public void setGrenzen(double aa, double bb, double yy) {
        a = aa; b = bb;
        y0 = yy;
    }

    public void setA(double aa) {
        a = aa;
    }

    public void setB(double bb) {
        b = bb;
    }

    public void setAnfangsWert(double yy) {
        y0 = yy;
    }

    public void setN(int j) {
        n = j;
    }

    public void setSchrittweite(double delta) {
        h = delta;
    }

    public double getSchrittweite() {
        return h;
    }

    // Rechte Seite der gewöhnlichen Differenzialgleichung
    protected double funktionBuch(double x, double y) {
        // Buchfunktion, Seite 388
        return x+y;
    }

    // Rechte Seite der gewöhnlichen Differenzialgleichung
    protected double funktion(double x, double y) {
        // Funktion für Aufgabe 17
        return y+2.0;
    }

    protected double funktion2(double x, double y) {
        // Funktion für Aufgabe 17
        return Math.exp(2.0*x) - 2.0*y;
    }

    // Die implementieren Einzelschrittverfahren
    // müssen diese Funktion implementieren!
    protected abstract void berechne();
}
```

```
\end{lstlistling}

\begin{lstlisting}{}
// Das Eulerverfahren
public class eulerVerfahren extends einzelschrittVerfahren {

    public eulerVerfahren (double aa, double bb, double yy, int nn) {
        a = aa; b = bb;
        n = nn;
        h = (b-a)/(double)(n-1);
        y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public void berechne() {
        int i;

        x[0] = a;
        y[0] = y0;

        for (i=1; i<n; i++) {
            x[i] = x[i-1]+h;
            y[i] = y[i-1] + h*funktion(x[i-1], y[i-1]);
        }
    }
}
```

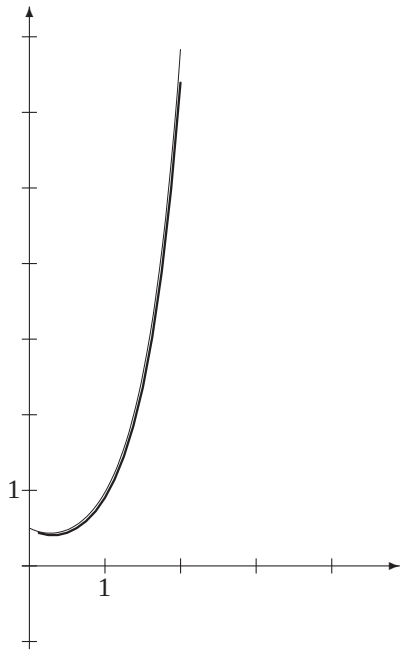

Abbildung 14.8: Ergebnis des Euler-Verfahrens ($n = 17$) für Aufgabe 17, $y' = e^{2x} - 2y$. Die exakte Lösung ist fett dargestellt

In Abbildung 14.11 finden Sie die globalen Fehler für diese Verfahren und der rechten Seite

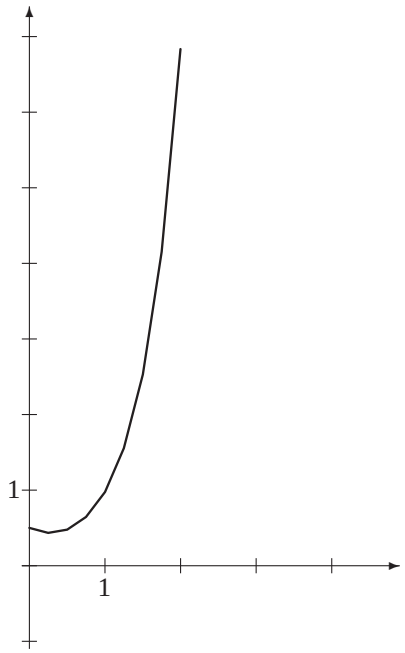

Abbildung 14.9: Ergebnis des Runge-Kutta-Verfahrens ($n = 9$) für Aufgabe 17, $y' = e^{2x} - 2y$.

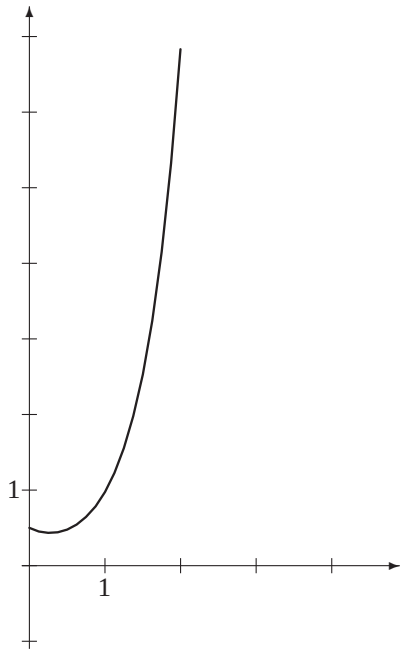

Abbildung 14.10: Ergebnis des Runge-Kutta-Verfahrens ($n = 17$) für Aufgabe 17, $y' = e^{2x} - 2y$.

$f(x, y) = y + 2$ in doppelt-logarithmischer Darstellung.

```
// Das Heun-Verfahren
public class heunVerfahren extends einzelschrittVerfahren {
```

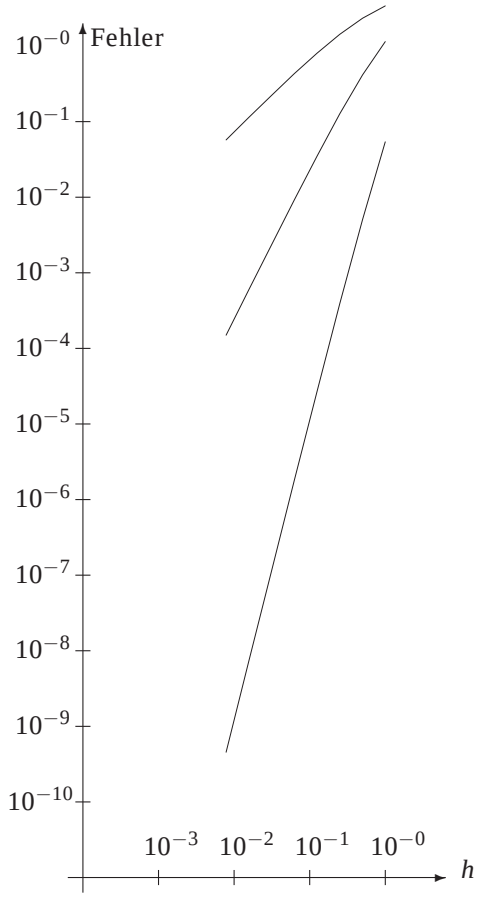

Abbildung 14.11: Die globalen Fehler bei Euler-, Heun- und Runge-Kutta-Verfahren für $f(x, y) = y + 2$ in einer doppelt-logarithmischen Skala für Aufgabe 17

```
public heunVerfahren (double aa, double bb, double yy, int nn) {
    a = aa; b = bb;
    n = nn;
    h = (b-a)/(double)(n-1);
    y0 = yy;

    x = new double[n];
    y = new double[n];
}

public void berechne() {
    int i;
    double k1, k2;

    x[0] = a;
    y[0] = y0;

    for (i=1; i<n; i++) {
        k1 = funktion(x[i-1], y[i-1]);
        k2 = funktion(x[i-1]+h, y[i-1]+h*k1);
```

```
            x[i] = x[i-1]+h;
            y[i] = y[i-1] + (h/2.0)*(k1+k2);
        }
    }
}
```

```
// Das Runge-Kutta-Verfahren
public class rungekuttaVerfahren extends einzelschrittVerfahren {
    public rungekuttaVerfahren (double aa, double bb, double yy, int nn) {
        a = aa; b = bb;
        n = nn;
        h = (b-a)/(double)(n-1);
        y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public void berechne() {
        int i;
        double k1, k2, k3, k4, hh=h/2.0;

        x[0] = a;
        y[0] = y0;

        for (i=1; i<n; i++) {
            k1 = funktion(x[i-1], y[i-1]);
            k2 = funktion(x[i-1]+hh, y[i-1]+hh*k1);
            k3 = funktion(x[i-1]+hh, y[i-1]+hh*k2);
            k4 = funktion(x[i-1]+h, y[i-1]+h*k3);

            x[i] = x[i-1]+h;
            y[i] = y[i-1] + h*(k1/6.0+k2/3.0 + k3/3.0 + k4/6.0);
        }
    }
}
```

18. Skizzieren Sie die Vektorfelder $\mathbf{f}(x,y) = (y,x)$, $\mathbf{f}(x,y) = (-x,2y)$, $\mathbf{f}(x,y) = \frac{1}{\sqrt{x^2+y^2}}(y,-x)$ und $\mathbf{f}(x,y) = (y^2-2xy, 3xy-6x^2)$!

    Lösung:
    Die Skizzen finden Sie in den Abbildungen 14.12 bis 14.15.

19. Erweitern Sie Ihre Implementierung der Einzelschrittverfahren, sodass Sie damit Differenzialgleichungssysteme lösen können. Nähern Sie damit das System mit dem Vektorfeld $\mathbf{f}(x,y) = (-y,x)$ und $(y_1(0) = 1, y_2(0) = 0)$ an. Vergleichen Sie Ihre Ergebnisse mit den Abbildungen 14.16 und 14.20!

    Lösung:
    Sie sollten einen geschlossenen Kreis beim Euler-Verfahren nur bei kleinen Schrittweiten erhalten.

    Hier die Implementierung in Java:

```
//-------------------------------------------------------------
// einzelschrittVerfahren2D.java
// Virtuelle Oberklasse für die Algorithmen für
// Einzelschrittverfahren zur Lösung eines
```

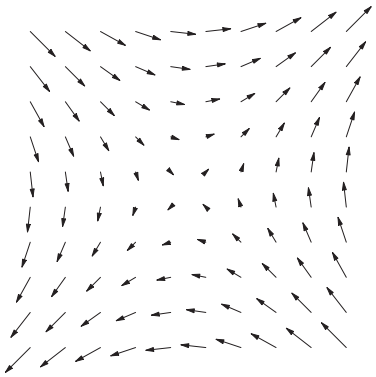

Abbildung 14.12: $f(x,y) = (y,x)$

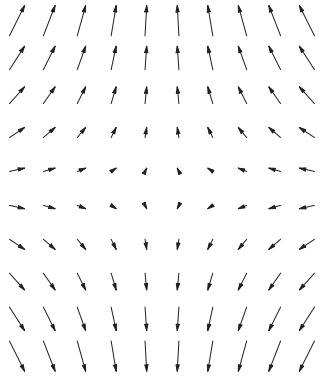

Abbildung 14.13: $f(x,y) = (-x,2y)$

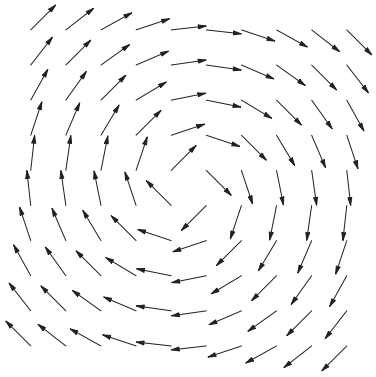

Abbildung 14.14: $f(x,y) = \frac{1}{\sqrt{x^2+y^2}}(y,-x)$

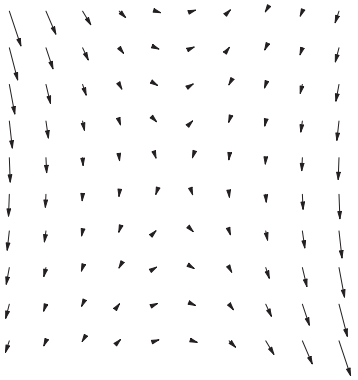

Abbildung 14.15: $f(x,y) = (y^2 - 2xy, 3xy - 6x^2)$

```
// zweidimensionalen Vektorfelds.
//-------------------------------------------------------------
//Autor:           Manfred Brill
//Letzte Änderung: 14. 11. 2004
//-------------------------------------------------------------
public abstract class einzelschrittVerfahren2D {

    protected int n;          // Anzahl der Schritte
    protected double h;       // Schrittweite
    protected double x0, y0;  // Anfangswert
    protected double x[],y[]; // Ergebnisvektoren des Verfahrens

    protected einzelschrittVerfahren2D() {
        n = 3;
        h = 0.1;
        x0 = 0.0; y0 = 0.0;
        x = new double[n];
        y = new double[n];
    }

    protected einzelschrittVerfahren2D(double xx, double yy,
                                       int nn, double delta) {
        n = nn;
        h = delta;
        x0 = xx;
```

```
        y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public void getErgebnis(double xw[], double yw[]) {

        for (int i=0; i<n; i++) {
            xw[i] = x[i];
            yw[i] = y[i];
        }
    }

    public void printErgebnis() {

        for (int i=0; i<n; i++)
            System.out.println("("+x[i]+","+y[i]+")");
    }


    public void setAnfangsWert(double xx, double yy) {
        x0 = xx; y0 = yy;
    }

    public void setN(int j) {
        n = j;
    }

    public void setSchrittweite(double delta) {
        h = delta;
    }

    public double getSchrittweite() {
        return h;
    }

    // Rechte Seite der gewöhnlichen Differenzialgleichung,
    // x-Komponente
    protected double funktionx(double x, double y) {
        // Funktion 1 für Aufgabe 17
        return -y;
    }

    // Rechte Seite der gewöhnlichen Differenzialgleichung,
    // y-Komponente
    protected double funktiony(double x, double y) {
        // Funktion 2 für Aufgabe 17
        return x;
    }

    // Die implementieren Einzelschrittverfahren
    // müssen diese Funktion implementieren!
    protected abstract void berechne();
    protected abstract double xSchritt(double x, double y);
    protected abstract double ySchritt(double x, double y);
}
```

```
// Das Eulerverfahren
```

```
public class eulerVerfahren2D extends einzelschrittVerfahren2D {

    public eulerVerfahren2D (double xx, double yy,
                             int nn, double delta) {
        n = nn;
        h = delta;
        x0 = xx;
        y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public double xSchritt(double x, double y) {
            return x + h*funktionx(x, y);
    }

    public double ySchritt(double x, double y) {
        return y + h*funktiony(x, y);
    }

    public void berechne() {
        int i;
        double savex;

        x[0] = x0;
        y[0] = y0;

        for (i=1; i<n; i++) {
            savex = x[i-1];
            x[i] = xSchritt(x[i-1], y[i-1]);
            y[i] = ySchritt(savex, y[i-1]);
        }
    }
}
```

```
// Das Runge-Kutta-Verfahren
public class rungekuttaVerfahren2D extends einzelschrittVerfahren2D {
    public rungekuttaVerfahren2D (double xx, double yy,
                                  int nn, double delta) {
        n = nn;
        h = delta;
        x0 = xx; y0 = yy;

        x = new double[n];
        y = new double[n];
    }

    public double xSchritt(double x, double y) {
        double k1, k2, k3, k4, hh;

        hh = h/2.0;

        k1 = funktionx(x, y);
        k2 = funktionx(x+hh, y+hh*k1);
        k3 = funktionx(x+hh, y+hh*k2);
        k4 = funktionx(x+h, y+h*k3);

        return x + h*(k1/6.0 + k2/3.0 + k3/3.0 + k4/6.0);
```

```
    }

    public double ySchritt(double x, double y) {
        double k1, k2, k3, k4, hh;

        hh = h/2.0;

        k1 = funktiony(x, y);
        k2 = funktiony(x+hh, y+hh*k1);
        k3 = funktiony(x+hh, y+hh*k2);
        k4 = funktiony(x+h, y+h*k3);

        return y + h*(k1/6.0 + k2/3.0 + k3/3.0 + k4/6.0);
    }
}
```

20. Das *Räuber-Beute-Modell* nach Lotka-Volterra beschreibt die Größe der Beute-Population y_1 und der Räuberpopulation y_2 über die Zeit. Das Modell ist gegeben durch das Differenzialgleichungssystem

$$y_1'(t) = 10y_1(t)(1 - y_2(t)), y_2'(t) = y_2(t)(y_1(t) - 1).$$

Verwenden Sie Ihre Implementierung der Einzelschrittverfahren für die Anfangswerte $y_1(0) = 3$, $y_2(0) = 1$ und vergleichen Sie Ihre Ergebnisse!

Lösung:
In Abbildung 14.16 sehen Sie die mit Runge-Kutta berechneten Paare (y_1, y_2). Die Abbildung 14.17 zeigt die y_1- und y_2-Werte über der Zeitachse aufgetragen. Für die Lösung wurde der Code aus Aufgabe 19 verwendet.

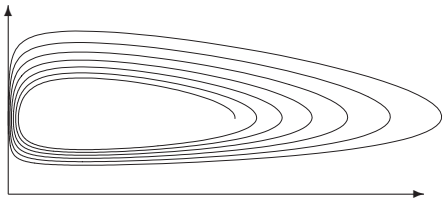

Abbildung 14.16: Das Räuber-Beute-Modell mit Runge-Kutta, $h = 0,01$ und $n = 1\,600$

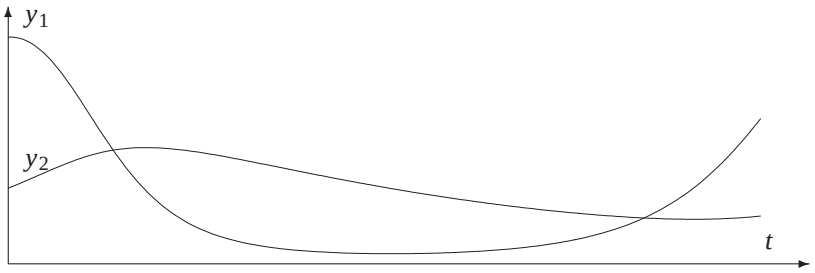

Abbildung 14.17: Die berechneten y_1 und y_2-Werte über der Zeitachse