Willkommen bei math4u2 V2.1

Neuigkeiten in der Version math4u2 V2.1

Einen kurzen Überblick über die Neuigkeiten im Vergleich zu V2.0. finden Sie als Vorspann vor den eigentlichen Kapiteln.

Schnell starten

Im Kapitel 1 erfahren Sie,

- wie Sie eine Lektion von einen math4u2-Server holen und
- wie Sie selbst ein kleines Modell erstellen.

Bedienung im Detail

Im Kapitel 2 werden Sie detailliert mit der Bedienung der math4u2-Oberfläche vertraut gemacht.

Die Mathematik

Das Kapitel 3 beschreibt, wie Sie aus Daten, Funktionen und Objekten Ihr Modell aufbauen, graphisch darstellen und manipulieren.

Lektions-Skripte

Das Kapitel 4 zeigt, wie Sie mit Hilfe von XML-Skripten math4u2-Lektionen erstellen können.

math4u2 ist ein Projekt der FH Augsburg
Kontakt: www.math4u2.de.

Inhalt:

Neuigkeiten in der Version math4u2 V2.1

Benutzeroberfläche

Die Benutzeroberfläche wurde in einigen Details leicht modifiziert und ergänzt.

Insbesondere kann jetzt beim Drucken naben den Graphiken auch der aktuelle Stand des begleitenden Textes gedruckt werden.

Die Druck-Funktion wird deshalb nicht mehr über das Kontext-Manü eines Zeichenfensters aktiviert, sondern über das Menü der Steuerleiste.

Vektoren und Matrizen durch ASCII-Dateien initialisieren

Vektoren, Dualvektoren und Matrizen können jetzt auch über ASCII-Dateien mit Werten initialisiert werden. Umgekehrt können Ergebnisse für Vektoren, Dualvektoren und Matrizen in Dateien abgelegt werden. Genaueres dazu in den entsprechenden Abschnitten des Kapitels 3.

Diagramme und Textelemente

Neu sind folgende Diagramm-Typen:

- **Diskretes Punkt-Diagramm**
  zur Darstellung diskreter Relationen.
- **Karte** zur farblichen Darstellung einer zweistelligen skalaren Funktion `f(x,y)`.
  Damit lassen sich insbesondere zweidimensionale Potentiale veranschaulichen.
- **Vektorfeld** zur Darstellung zweistelliger vektorwertiger Funktionen.
  Damit lasen sich insbesondere zweidimensionale Kraftfelder veranschaulichen.

Beim Diagramm-Typ **Text-Element** gibt es weitere Layout-Möglichkeiten.

Lektionen, XML-Schema

Das XML-Schema zur Beschreibung von Lektionen wurde überarbeitet und ergänzt, insbesondere im Bereich des Formelsatzes.

Die Syntax ist jetzt im Kapitel 4 dokumentiert, interessierte Nutzer können somit eigene Lektionen verfassen.

1 Erste Schritte

In diesem Kapitel werden Sie durch zwei kurze Beispiele mit den wichtigsten Bedienkonzepten von math4u2 vertraut gemacht.

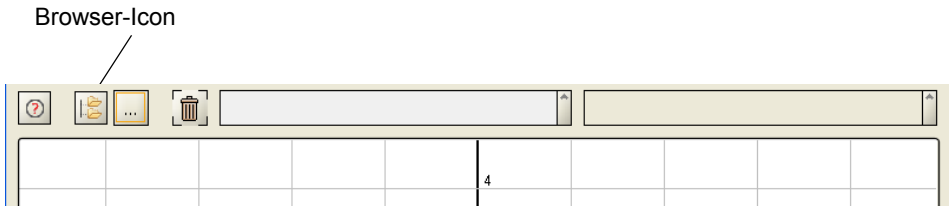
Mehr Details zur Bedienung finden Sie im Kapitel 2, der mathematische Kern von math4u2 ist ausführlich im Kapitel 3 beschrieben.

1.1 Online-Lektionen holen

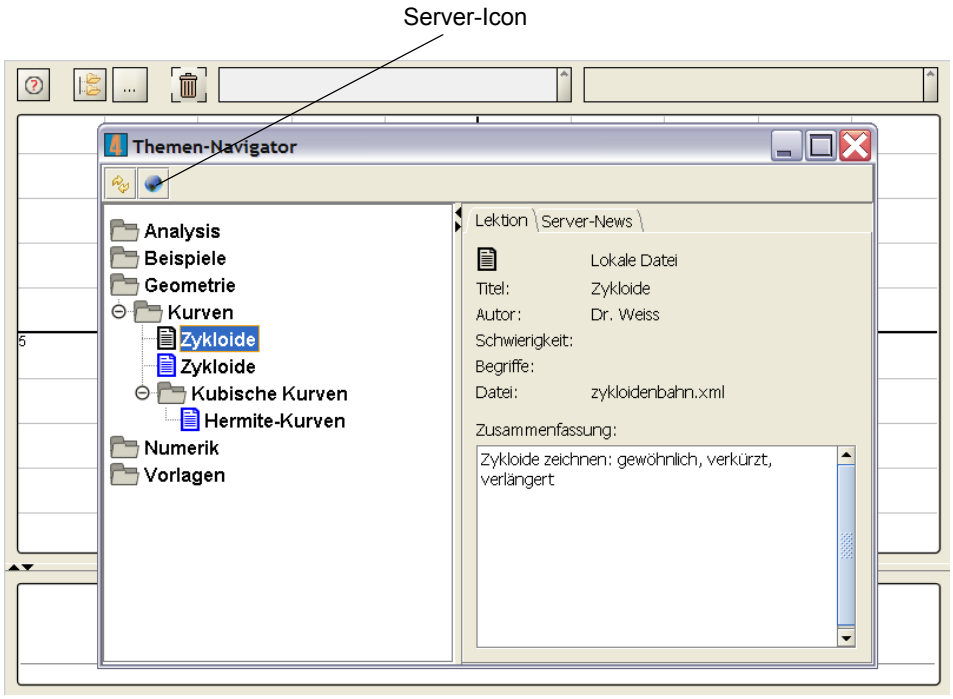
Gehen Sie online, holen Sie neue Lektionen direkt vom Server.

Ohne Umwege über einen zusätzlichen Browser oder das Dateisystem können Sie direkt im Lektionenbestand eines math4u2-Servers stöbern und Lektionen sofort starten.

Schritt 1

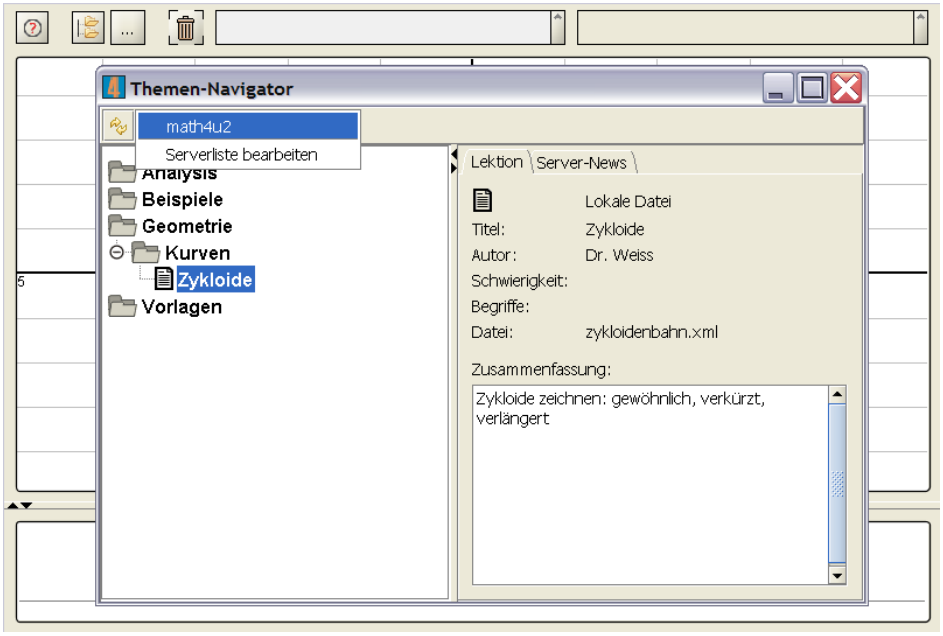


Klicken Sie auf das Browser-Icon, der Themen-Navigator wird geöffnet.



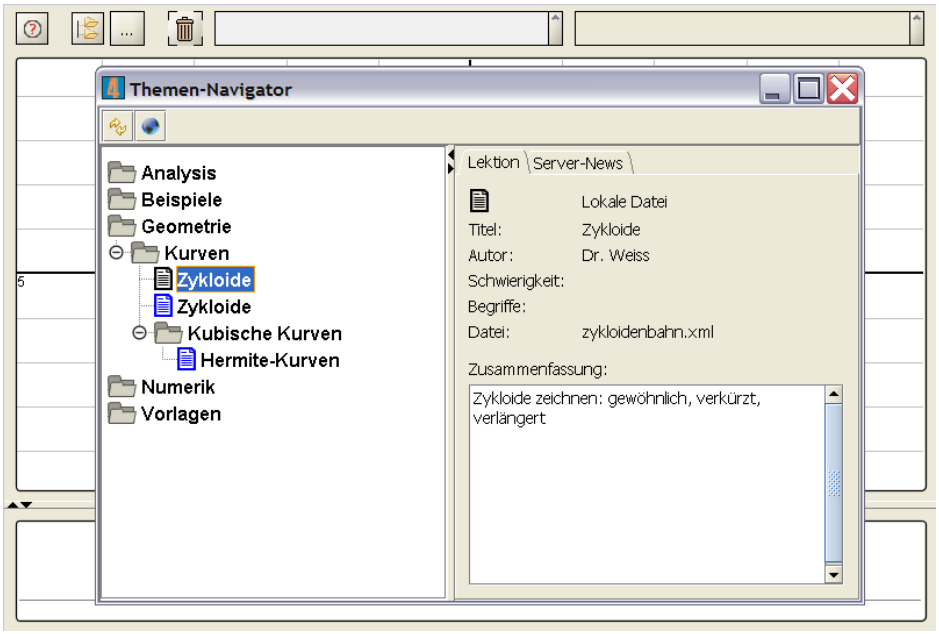
Schritt 2

Klicken Sie auf das Server-Icon und wählen Sie einen Server aus. Sie sind jetzt mit dem Server verbunden, die Liste der aktuellen Server-Lektionen wird geholt.



Schritt 3

Sie sehen jetzt neben den Lektionen, die Sie bereits auf Ihrem Rechner gespeichert haben (schwarz), die Lektionen, die sich auf dem Server befinden (blau).



Schritt 4

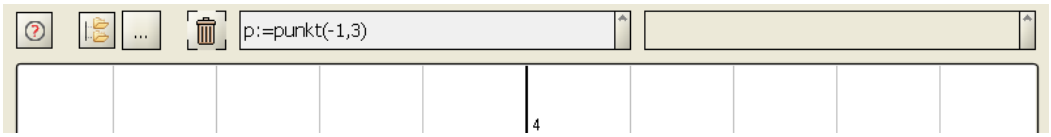
Doppelklicken Sie auf eine Server-Lektion mit der Maus.
Diese wird jetzt heruntergeladen und dargestellt.

1.2 Ein kleines Modell erstellen

Erstellen Sie ein kleines Modell und lernen Sie die Grundlagen des math4u2-Bedienkonzepts kennen.

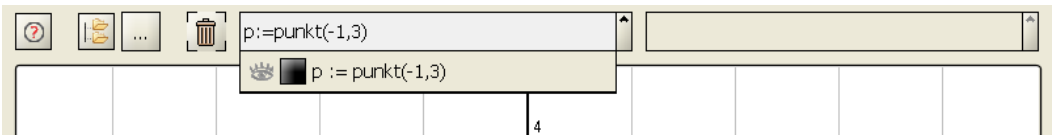
Schritt 1

Sämtliche Definitionen erfolgen über das Definitionsfenster.
Hier definieren wir einen Punkt p, er hat zunächst die Koordinaten (-1, 3).



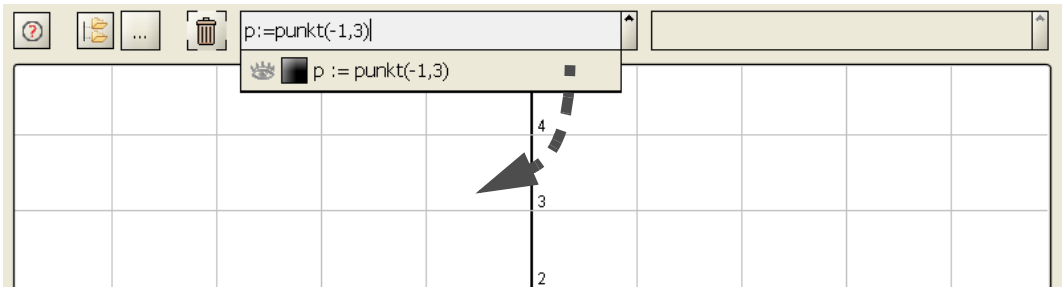
Schritt 2

Sobald wir ENTER drücken, wird die Definition des Punktes in die Definitionsliste aufgenommen.



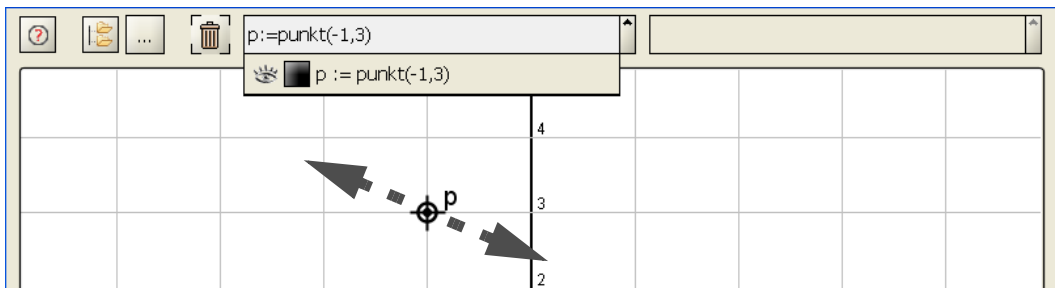
Schritt 3

Wir ziehen die Definition per Drag & Drop auf die Zeichenfläche (Element der Definitionsliste mit gedrückter Maustaste auf die Zeichenfläche ziehen, dort loslassen).



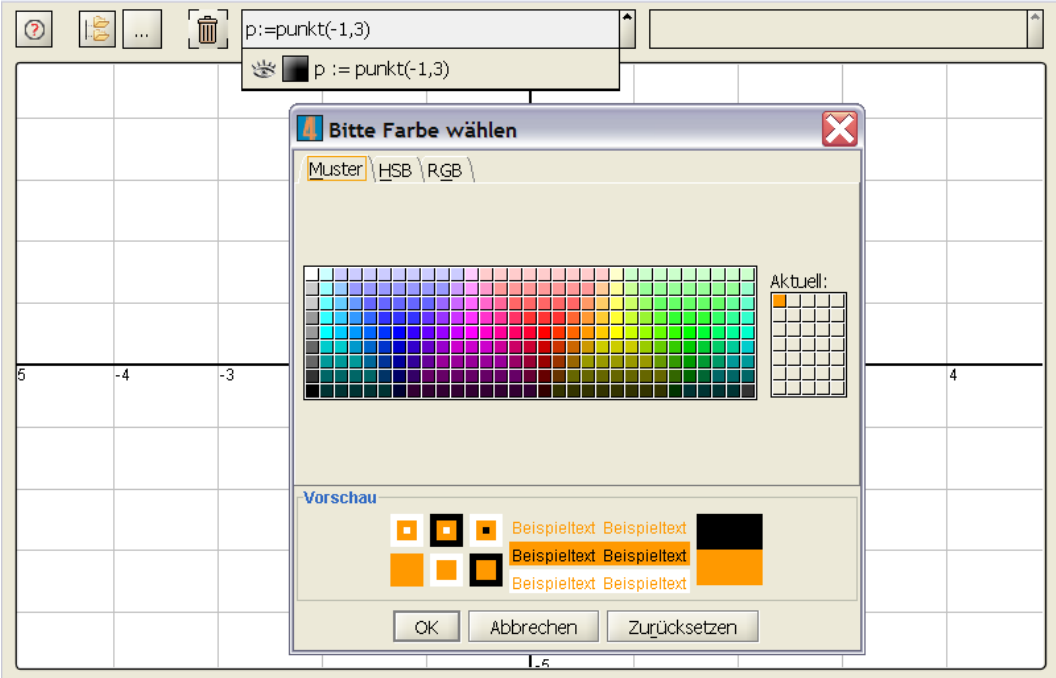
Schritt 4

Jetzt wird der Punkt gezeichnet.



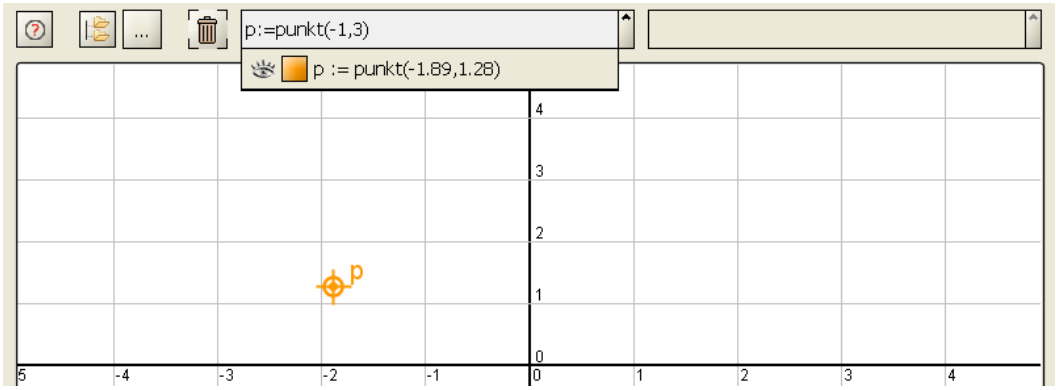
Schritt 5

Durch klicken auf die Farbanzeige verändern wir die Farbe.



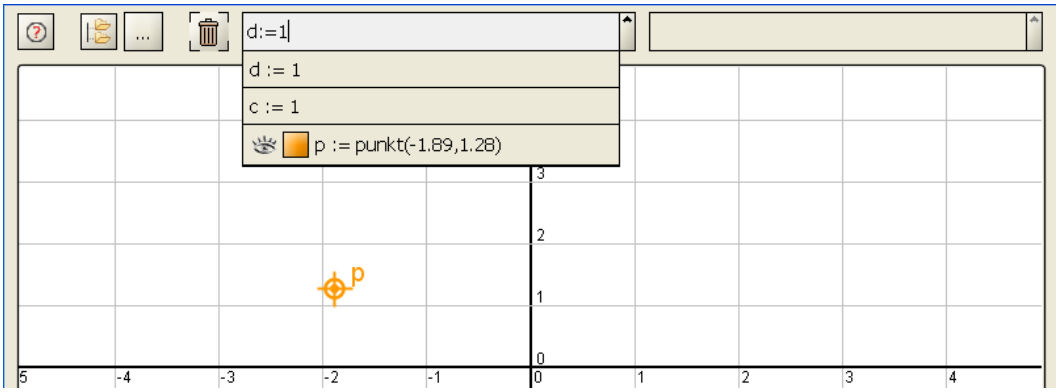
Schritt 6

Wenn wir den Punkt mit der Maus verschieben, sehen wir in seiner Definition die aktuellen Koordinaten.



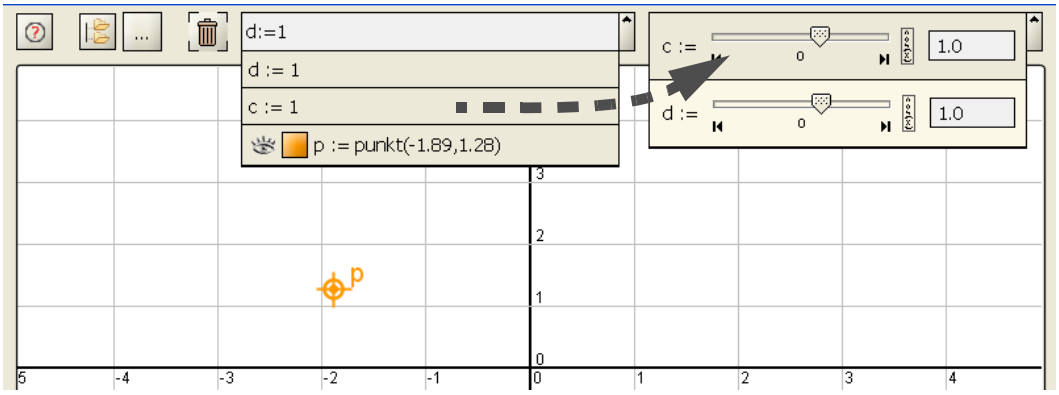
Schritt 7

Als nächstes definieren wir zwei Parameter c und d, beide haben anfangs den Wert 1.



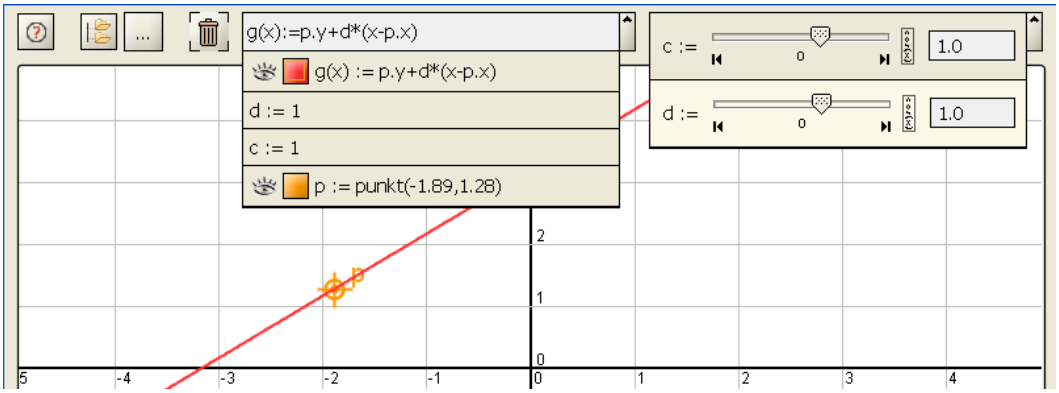
Schritt 8

Per Drag & Drop erzeugen wir zu beiden Definitionen eine Detailsicht (Element der Definitionsliste mit gedrückter Maustaste nach rechts auf das Rechteck ziehen, dort loslassen).



Schritt 9

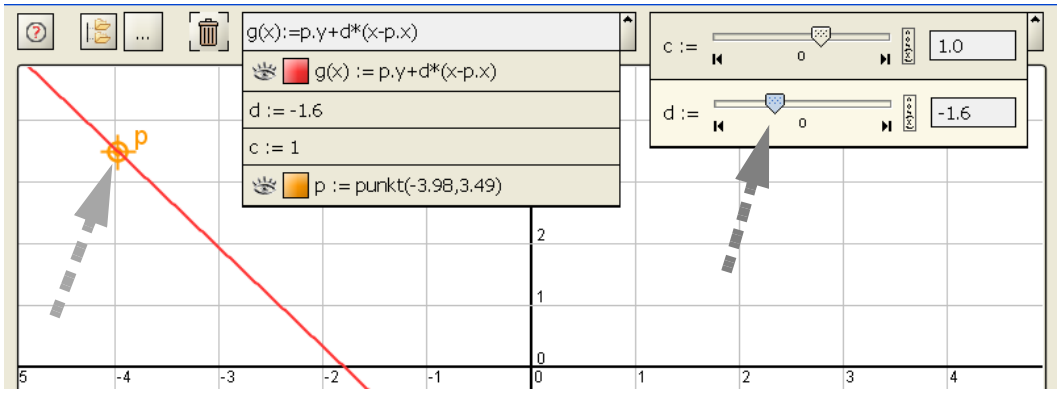
Ferner legen wir eine Gerade mit veränderbarer Steigung (Parameter d) durch unseren Punkt p. Zeichnen und Farbveränderung funktionieren wie oben beschrieben.



Schritt 10

Wir verändern die Position des Punktes (per Maus) und die Steigung durch den Parameter d (per Slider).

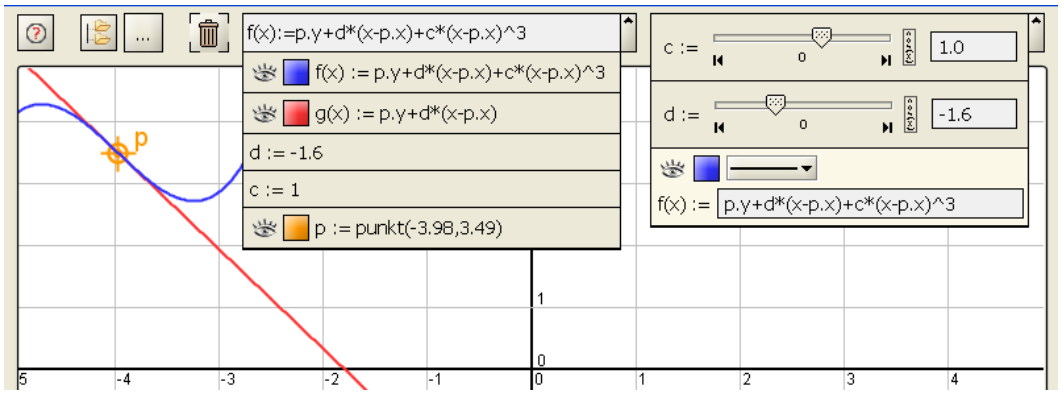
Die Gerade folgt den eingestellten Werten.



Schritt 11

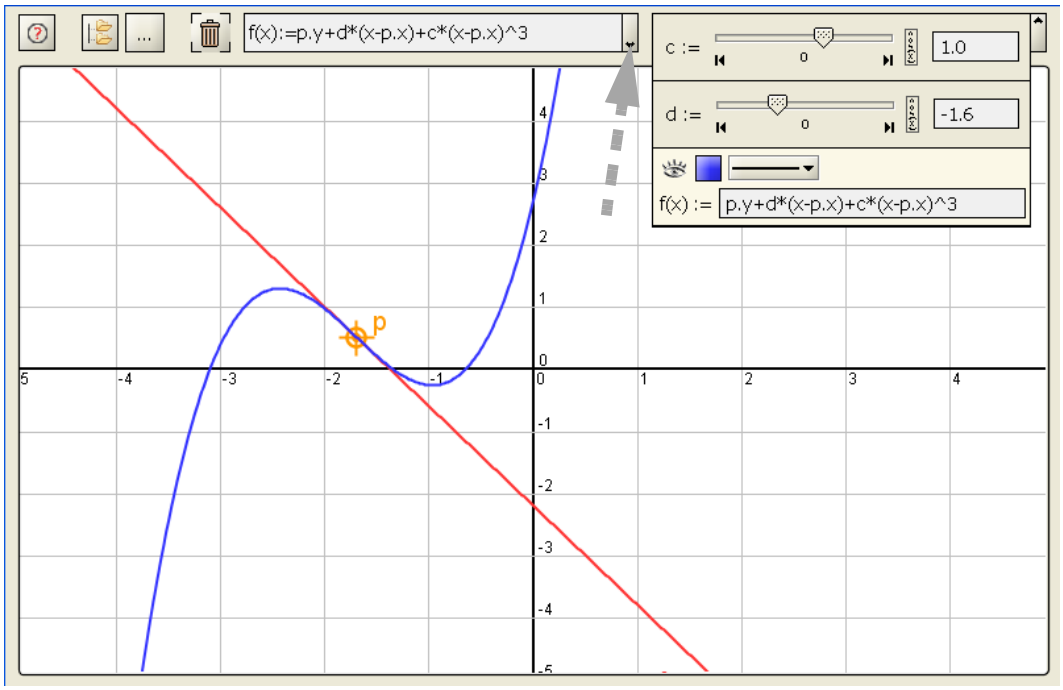
Entsprechend wie bei der Geraden definieren und zeichnen wir eine weitere Funktion f.

Die Detailsicht von f bietet spezielle Einstell-Möglichkeiten für Funktionen.



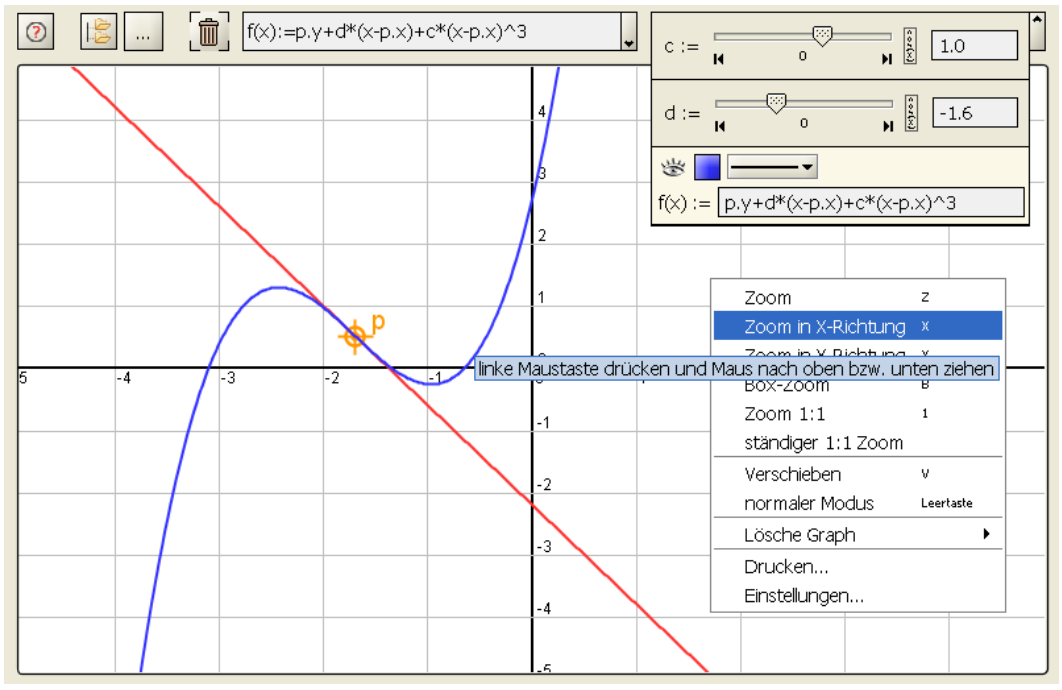
Schritt 12

Wir klappen die Definitionsliste zu (zweimal klicken). Nur noch die zentralen Elemente sind sichtbar.



Schritt 13

Über das Kontextmenü kann Ausschnitt und Maßstab der Zeichenfläche angepasst werden.

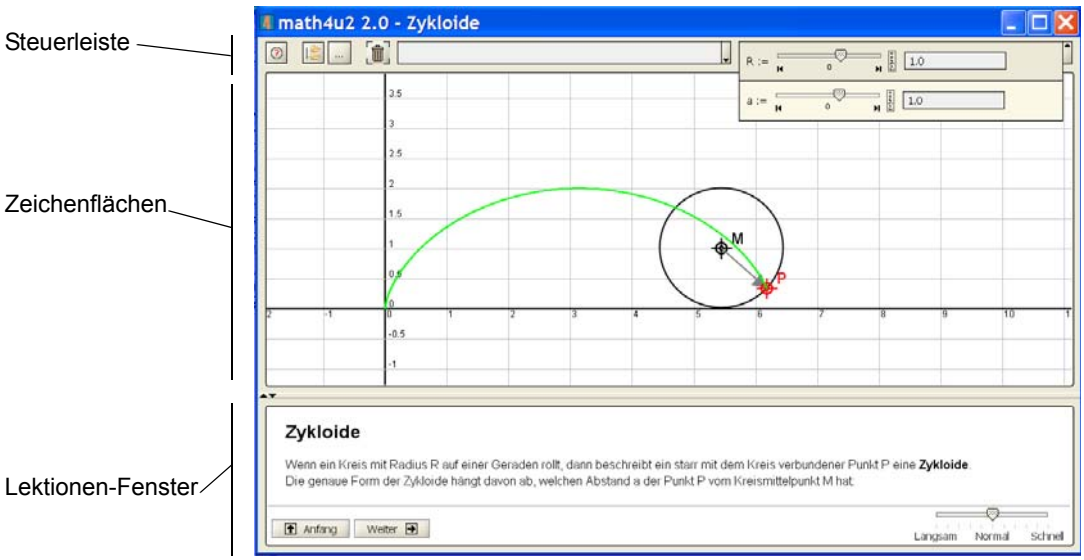


2 Bedienung

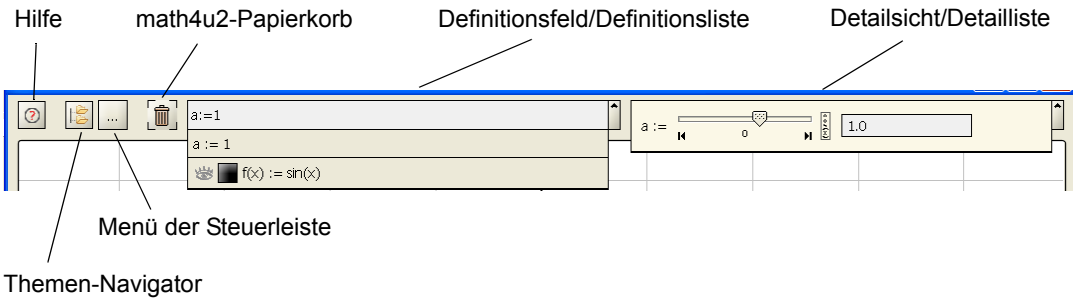
Hier finden Sie Details zur Bedienung von math4u2.
Einen kurzen Einblick in die wichtigsten Bedienkonzepte gibt der Abschnitt 1.2 des Kapitels 1.

2.1 Überblick: Die math4u2-Oberfläche

math4u2 passt die Größe der wichtigsten Steuerelemente und der Graphiken automatisch an den verfügbaren Platz an. Minimal sollten 800 x 600 Pixel zur Verfügung stehen.



2.1.1 Elemente der Steuerleiste



Hilfe: Mit diesem Knopf aktivieren Sie die math4u2-Online-Hilfe.

Themen-Navigator: Mit diesem Knopf aktivieren Sie den math4u2-Themen-Navigator. Näheres finden Sie im Abschnitt 2.5 dieses Kapitels.

Menü der Steuerleiste: Enthält die Elemente

- **Alles löschen**
  Löscht sämtliche Definitionen.
- **Speichern unter ...**
  Führt zum Speicher-Dialog. Näheres unten.
- **Drucken ...**
  Führt zur Druckvorschau.
- **Schrifttyp**
  Zeigt ein Untermenü zum Einstellen der Schriftgröße für das Definitionsfeld.
- **Mauskoordinaten anzeigen**
  Fügt eine Anzeige der Mauskoordinaten als Detailsicht in die Detailliste ein.

math4u2-Papierkorb: Zum Löschen werden Objekte in den math4u2-Papierkorb verschoben. Näheres im Abschnitt 2.2 und 2.3 dieses Kapitels.

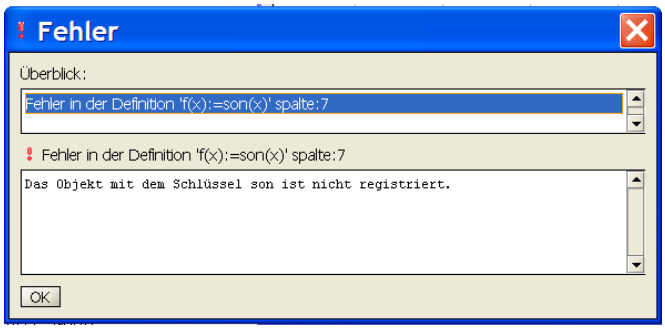
Definitionsfeld und Definitionsliste: Hier werden Objekte definiert und verwaltet. Näheres im Abschnitt 2.2 dieses Kapitels.

Detailsicht und Detailliste: Zum Inspizieren und Manipulieren von Objekten können Sie hier spezielle Sichten erzeugen. Näheres im Abschnitt 2.3 dieses Kapitels.

2.1.2 Fehler-Fenster

Fehler werden in einem speziellen Fehler-Fenster protokolliert.

Im oberen Bereich sind alle aktuellen Fehler aufgelistet. Wenn Sie auf einen dieser Einträge klicken, wird im unteren Bereich die Detail-Information zu diesem Fehler angezeigt.



Sie schließen dieses Fenster durch Klicken auf OK oder durch Drücken der RETURN-Taste.

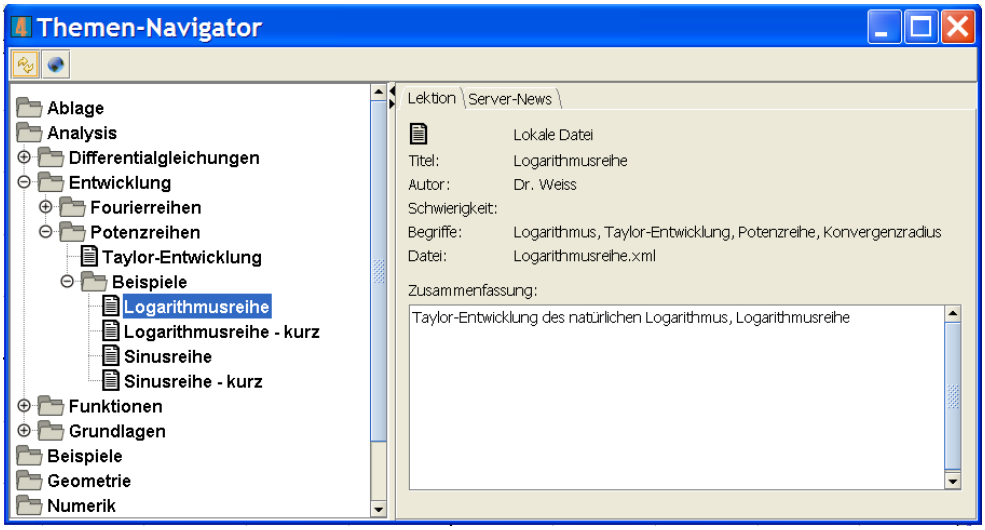
TIPP: Mit RETURN-Taste das Fehlerfenster schnell schließen.

2.1.3 Zeichenflächen

Das Arbeiten mit den Zeichenflächen ist im Abschnitt 2.4 dieses Kapitels beschrieben.

2.1.4 Themen-Navigator

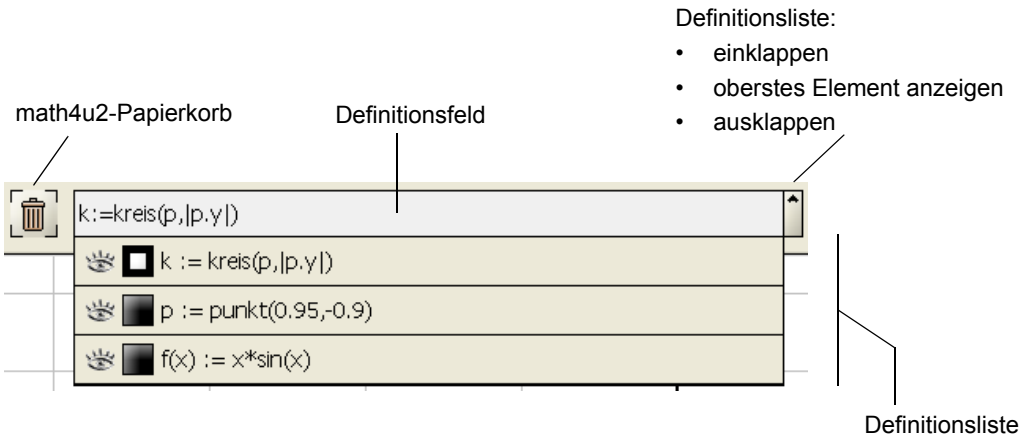
Lektionen suchen, starten und speichern: Abschnitt 2.5 dieses Kapitels.



2.1.5 Lektionen-Fenster

Mehr dazu im Abschnitt 2.6 dieses Kapitels.

2.2 Objekte definieren und löschen: Definitionsfeld und Definitionsliste



Objekt definieren

Objekte werden durch Eingabe der entsprechenden Definition im Definitionsfeld (z.B. `f(x):=x*sin(x)`) erzeugt. Eine Definition wird durch Eingabe von RETURN wirksam.

Jedes Objekt wird durch einen Eintrag in der Definitionsliste repräsentiert.

Neue Einträge werden oben in der Liste angefügt.

Definition ändern

Sie ändern die Definition eines Objekts mit dem Namen `obj`, indem Sie für den gleichen Namen im Definitionsfeld eine neue Definition eingeben.

Wenn Sie die Definition von `obj` nur leicht modifizieren wollen, dann klicken Sie auf den entsprechenden Eintrag der Definitionsliste. Die aktuelle Definition wird dadurch in das Definitionsfeld gestellt. Sie können die Definition jetzt dort editieren und zuletzt mit RETURN bestätigen.

Für ein bereits existierendes Objekt kann eine neue Definition nur dann eingegeben, wenn dadurch ein Objekt des gleichen Typs definiert wird. So kann z.B. eine einstellige Funktion `f(x):=sin(x)` nur durch eine einstellige Funktion wie `f(x):=2*cos(x)` ersetzt werden (und z.B. nicht durch eine zweistellige Funktion, einen Punkt oder eine Strecke).

Wenn die Definition der Funktion `f(x):=sin(x)` nicht mehr benötigt wird und nun ein Punkt mit dem Namen `f` definiert werden soll, dann muss zunächst die Definition der Funktion gelöscht werden (verschieben in den math4u2-Papierkorb), anschließend kann mit `f:=punkt(1,1)` der Punkt definiert werden.

Objekt löschen

Zum Löschen eines Objekts haben Sie zwei Möglichkeiten:

- Markieren Sie das entsprechende Element der Definitionsliste mit der Maus und drücken Sie die Taste ENTF.
- Schieben Sie das entsprechende Element der Definitionsliste mit gedrückter Maustaste in den math4u2-Papierkorb.

Mit einem Objekt werden auch sämtliche zugehörigen Graphen und gegebenenfalls die zugehörige Detailsicht gelöscht.

Achtung: Ein Objekt kann nicht gelöscht werden, wenn es von anderen Objekten noch verwendet wird.

2.3 Objekte inspizieren und verändern: Detailsicht und Detailliste

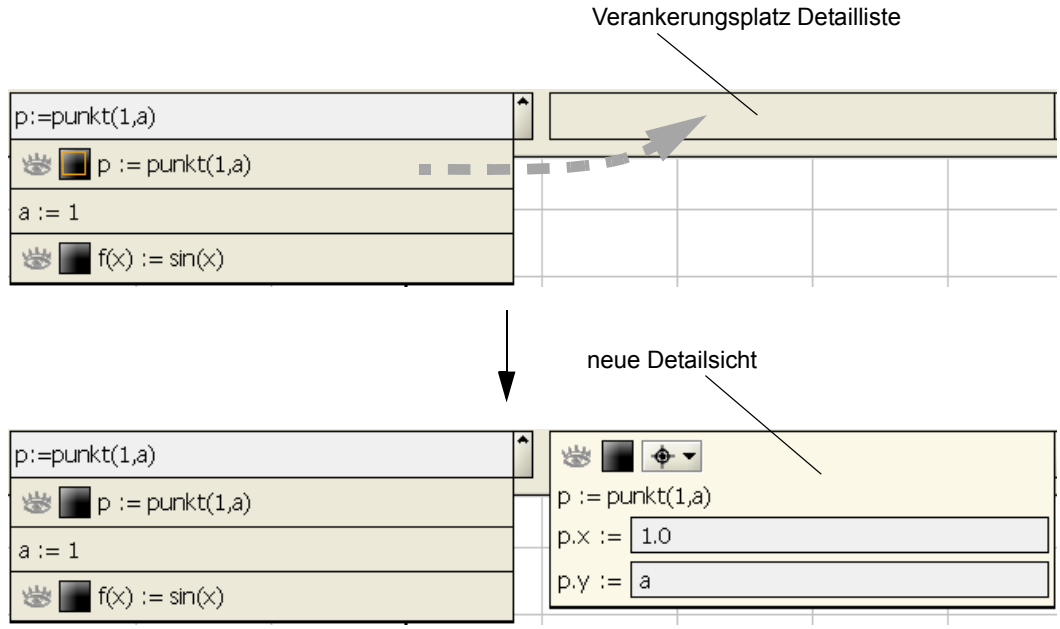
Für viele Objekte kann eine Detailsicht erzeugt werden. Eine Detailscht dient dazu, einzelne Eigenschaften eines Objekts zu beobachten oder zu verändern.

Die Form einer Detailsicht hängt vom Typ des zugehörigen Objekts ab. Mehr Informationen dazu finden Sie deshalb bei der Beschreibung der verschiedenen Objekt-Typen.

In der Detailliste können mehrere Detailsichten verwaltet werden. In der Regel wird man nur von den wenigen Objekten eine Detailsicht erzeugen, die aktuell von besonderem Interesse sind. Die nicht mehr benötigten Detailsichten können gelöscht weden, ohne dass davon das eigentliche Objekt betroffen ist.

Detailsicht erzeugen

Sie erzeugen eine Detailsicht eines Objekts, indem Sie das entsprechende Element der Definitionsliste mit gedrückter Maustaste nach rechts auf den Verankerungsplatz der Detailliste ziehen.



Maus-Detailsicht

Zum Verfolgen der aktuellen Maus-Koordinaten können Sie eine Detailsicht erzeugen, indem Sie im Menü der Steuerleiste (siehe 2.1.1) das Element "Mauskoordinaten anzeigen" auswählen.

Detailsicht löschen

Sie löschen die Detailsicht eines Objekts,

- indem Sie diese mit der Maus markieren und dann die ENTF-Taste drücken oder
- indem Sie die Sicht mit gedrücketer Maustaste in den math4u2-Papierkorb ziehen.

Achtung: Beim Löschen einer Detailsicht bleibt das eigentliche Objekt erhalten.

2.4 Objekte zeichnen und mit der Maus bewegen: Zeichenflächen

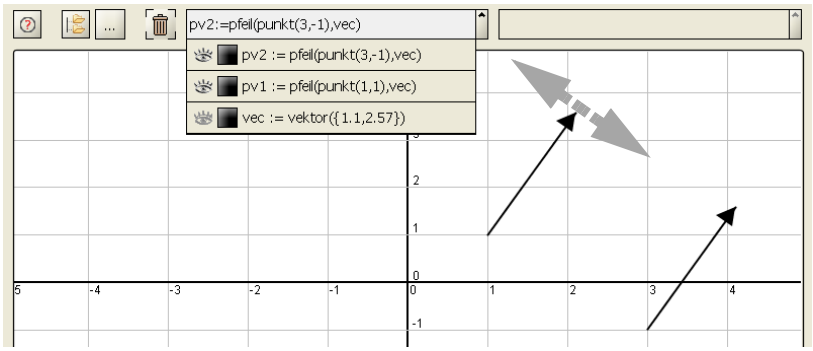
2.4.1 Objekte Zeichnen: drag and drop

Ein Objekt wird gezeichnet, indem man den entsprechenden Eintrag der Definitionsliste mit gedrückter linker Maustaste auf die gewünschte Zeichenfläche zieht und dort die Taste losläßt (drag and drop).

Wenn zu einem Objekt eine Detailsicht existiert, dann kann man auch diese Detailsicht auf die Zeichenfläche ziehen.

Je nach Typ des Objekts wird es jetzt auf der Zeichenfläche durch einen spezifischen Graphen dargestellt. Wichtige Beispiele sind:

- Zu einer einsteligen Funktion wie `f(x):=x*sin(x)` wird der Funktionsgraph gezeichnet.
- Ein Punkt `p:=punkt(1,1)` zeichet sich, er kann mit der Maus verschoben werden, seine Koordinaten ändern sich dann entsprechend.
- Die anderen affinen Objekte (`kreis`, `flaeche`, `kurve`, `bezier`, `kurvenzug` und `winkel`) zeichnen sich wie gewohnt.
- Eine Sonderstellung hat der zweidimensionale Vektor: Will man einen zweidimensionalen Vektor wie `vec:=vektor({1,2})` als Pfeil darstellen, so ist dieser Pfeil streng genommen nicht das Bild des Vektors, sondern das Bild eines Repräsentanten. Dies bildet man beim Zeichnen wir folgt nach:
  Zu `vec` definiert man z.B. mit `pv1:=pfeil(punkt(1,1),vec)` einen Repräsentanten, konkret mit dem Aufpunkt `punkt(1,1)`. Dieser Repräsentant `pv1` zeichnet sich nun auf der Zeichenfläche als Pfeil. Die Pfeilspitze kann mit der Maus bewegt werden, damit ändern sich die Koordinaten des Vekors `vec`. Wenn man mit `pv2:=pfeil(punkt(3,-1),vec)` einen weiteren Repräsentanten zu `vec` definiert und dann zeichnet, zeichnen sich `pv1` und `pv2` stets als parallelgleiche Pfeile.



Graph von Zeichenfläche löschen: Kontextmenü der Zeichenfläche öffnen (rechte Maustaste), über das Element "Lösche Graph" die gewünschte Auswahl treffen.

2.4.2 Zeichenfläche interaktiv manipulieren

Auf einer Zeichenfläche steht über die rechte Maustaste ein Menü mit verschiedenen Einstellmöglichkeiten zur Verfügung.

Ausschnitt einstellen: Zoom und Verschieben

Die ersten Elemente des Menüs bieten verschiedene Zoom-Modi.
Da diese sehr häufig benötigt werden, können sie auch ohne Umweg über das Menü durch entsprechende Tasten aktiviert werden:

Tipp: Zoom-Modus schnell über Tasten aktivieren!

Menüelement	Taste	Aktion
Zoom	z	Jetzt linke Maustaste drücken und Maus nach unten ziehen, gezeigter Ausschnitt wird größer (zoom-out), linke Maustaste drücken und Maus nach oben ziehen, Ausschnitt wird kleiner (zoom-in).
Zoom in x-Richtung	x	Zoom wirkt nur in x-Richtung.
Zoom in y-Richtung	y	Zoom wirkt nur in y-Richtung.
Box-Zoom	b	Linke Maustaste drücken, dann Rechteck aufziehen und so den gewünschten Ausschnitt festlegen.
Zoom 1:1	1	Stellt zusätzlich zu Zoom auf beiden Achsen gleiche Längeneinheiten her.
ständiger 1:1 Zoom		
Verschieben	v	Verschieben des Ausschnitts mit der Maus.
normaler Modus	Leertaste	Zoom-Funktion ausgeschaltet.

Einstellungen

Hier können Sie für die aktuelle Zeichenfläche folgende Einstellungen vornehmen:

Detail-Stufe: Bei Detail-Stufe 1 werden die Funktionswerte zur Darstellung eines Graphen horizontal in 1-Pixel-Abständen berechnet.

Bei sehr rechenintensiven Funktionen kann dies zu längeren Wartezeiten führen. Diese können durch eine Erhöhung der Detail-Stufe verkürzt werden. Bei Detail-Stufe 20 wird nur für jedes 20. Pixel ein Funktionswert gerechnet, die resultierenden Stützpunkte werden durch eine Kurve interpoliert. Kleine Details des Graphen können durch diese Interpolation verfälscht werden.

Durch Markieren von "grobes Detail beim Zoom" legen Sie fest, dass nur während eines Zoom-Vorgangs die Anzahl der berechneten Stützpunkte reduziert werden soll. Dies bewirkt, dass bei rechenintensiven Funktionen der Zoom-Vorgang flüssiger bleibt. Nach Beendigung des Zoom-Vorgangs wird die Anzahl der Stützpunkte automatisch wieder erhöht.

Koordinaten: Über die Eingabefelder können sie für das Koordinatensystem einen Ausschnitt festlegen, dieser wird durch Markieren von "Übernehmen" wirksam.

Graphenstärke: Stärke der Funktionsgraphen

Farbe: Farbe der Gitterlinien und Achsen

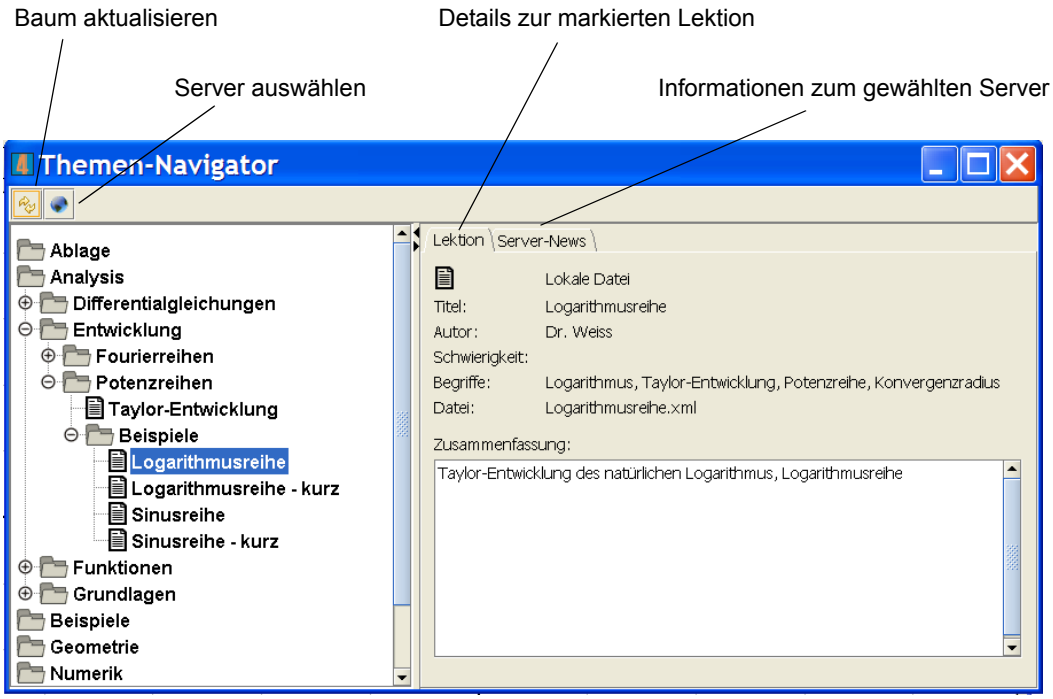
Hilfslinien: Abstand der Gitterlinien

Titel: Titel, der links oben im Fenster angezeigt wird.

2.5 Lektionen auswählen: Der Themen-Navigator

Mit dem Themen-Navigator

- verschaffen Sie sich einen Überblick über den aktuellen Bestand an math4u2-Lektionen,
- erhalten Sie Informationen über einzelne Lektionen,
- starten oder speichern Sie ausgewählte Lektionen und
- beziehen Sie aktuelle Informationen über das math4u2-Projekt.



Server auswählen

Wenn Ihr Rechner aktuell eine Netzwerkverbindung hat, können Sie sich direkt mit einem math4u2-Server verbinden. Dazu öffnen Sie die Server-Liste und wählen dort den gewünschten math4u2-Server aus. Im Themen-Baum werden jetzt angezeigt:

- lokal auf Ihrem Rechner gespeicherte Themen und Lektionen und
- auf dem ausgewählten Server verfügbare Themen und Lektionen.

Themen-Baum

Der Themen-Baum zeigt die Gliederung der Lektionen in größere Themenbereiche. Öffnen Sie die Tehmenbereiche durch einen Doppelklick.

Lektionen werden je nach Speicherort und Speicherdauer in verschiedenen Farben symbolisiert:

schwarz:
Die Lektion ist dauerhaft auf Ihrem Rechner gespeichert.

grau:
Die Lektion ist temporär auf Ihrem Rechner gespeichert. Wenn Sie math4u2 beenden, dann wird diese Lektion wieder gelöscht. Wenn Sie diese Lektion dauerhaft speichern wollen, dann wählen Sie aus dem Kontextmenü (rechte Maustaste) die Option "speichern".

blau:
Die Lektion ist auf dem aktuell ausgewählten Server verfügbar.

Lektionen-Information

Über den Reiter "Lektion" erhalten Sie nähere Informationen zur aktuell markierten Lektion.

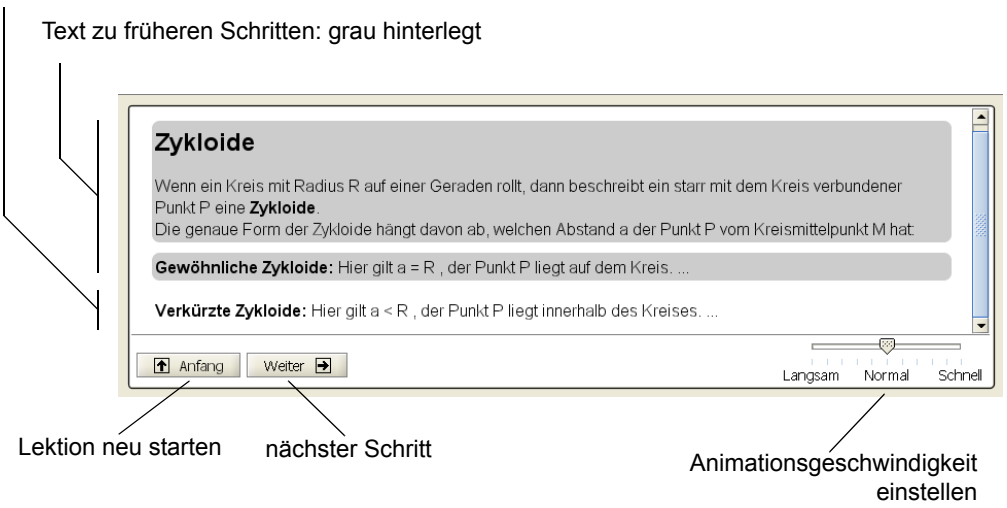
Lektion starten

In jedem Fall können Sie die gewünschte Lektion mit einem Doppelklick starten.

Server-News

Über den Reiter "Server-News" können Sie Neuigkeiten zum ausgewählten Server abrufen. Hier werden Sie z.B. über neue Lektionen oder math4u2-Versionen informiert.

2.6 Lektionen bearbeiten: Lektionen-Fenster

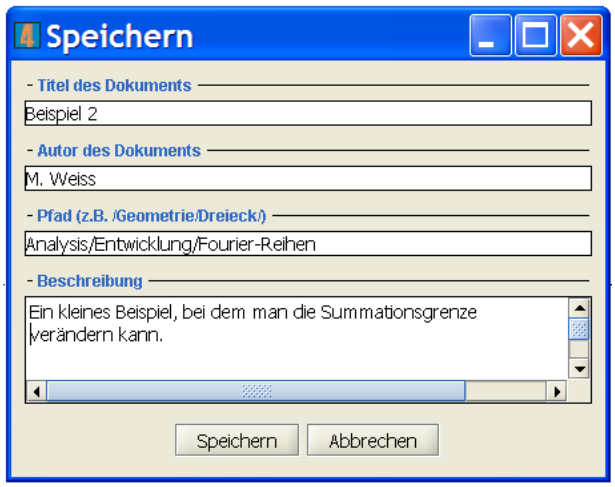


Über das Lektionen-Fenster steuern Sie den Ablauf einer Lektion. Die Steuerelemente werden erst eingeblendet, wenn Sie eine Lektion gestartet haben.

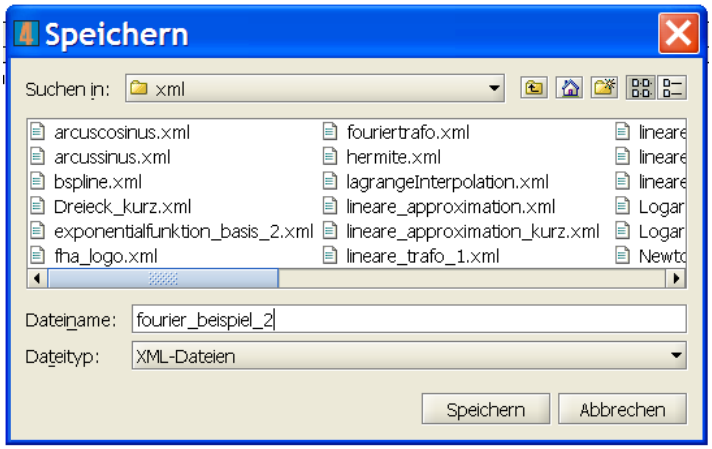
2.7 Arbeitsstand speichern

Durch die Auswahl des Elements "Speichern unter ..." im Menü der Steuerleiste leiten Sie das Speichern des aktuellen Arbeitsstands ein.
Gespeichert werden dabei die aktuell definierten Funktionen und Objekte mit ihren Beziehungen.
Nicht gespeichert werden: verschiedene Schritte einer Lektion und der Inhalt des Lektionen-Fensters.

Zunächst legen Sie logisch fest, mit welchem Titel und an welcher Stelle der Gliederungsstruktur der gespeicherte Arbeitsstand anschließend im Themen-Navigator angezeigt wird:



Nach der Auswahl "Speichern" legen Sie im anschließenden Dialog einen Namen für die Datei fest, in der die nötigen Daten gespeichert werden. Das dort voreingestellt Verzeichnis muss unverändert bleiben:



Nach nochmaliger Auswahl von "Speichern" wird die Datei auf der Festplatte abgelegt.

Der Arbeitsstand kann nun bei Bedarf durch Doppelklick auf den entsprechenden Eintrag im Themen-Navigator wieder geladen werden.

3 Daten, Funktionen, Objekte, Graphen

3.1 Bezeichner

Ein Element eines Modells wird wie
`punkt1 := punkt(1,2)`
definiert.
Sämtliche Modellelemente (Parameter, Funktionen, affine Objekte und Graphen) werden durch Bezeichner identifiziert.
Ein Bezeichner beginnt mit einem Buchstaben (keine Umlaute), es folgt eine beliebige Sequenz aus Buchstaben, Ziffern und Unterstrich _ . Gültige Bezeichner sind z.B. : `a`, `a1`, `punkt1`, `aber`, `zwei_sin`, `meinBez`.
Zwei verschiedene Modellelemente müssen durch zwei unterschiedliche Bezeichner identifiziert sein. Dabei wird zwischen Groß- und Kleinbuchstaben unterschieden, d.h. `bez1` und `Bez1` sind unterschiedliche Bezeichner.
Eine Reihe von Bezeichnern identifiziert vordefinierte Parameter, Funktionen oder Schlüsselwörter (z.B. `e`, `pi`, `sin`, `cos`, `punkt`, `vektor`). Diese Bezeichner können für selbstdefinierte Elemente nicht verwendet werden.

3.2 Datentypen

Der Arithmetik liegt die Gleitpunktarithmetik des Rechners zu Grunde. Dabei gilt:

Größter darstellbarer positiver Wert: $1,7976931348623157 \cdot 10^{308}$,

betragsmäßig kleinster darstellbarer Wert: $4,9 \cdot 10^{-324}$

3.2.1 Zahlen

Zahlen werden als Dezimalbrüche eingegeben und dargestellt.

- Ganze Zahlen: `-17` oder `0` oder `125 .`
- Das Trennzeichen in echten Dezimalbrüchen ist der Punkt: `-1.27` oder `0.023` .
- Für sehr kleine oder sehr große Zahlen kann die Exponential-Darstellung verwendet werden: `-0.273E-12` oder `12.3E7` .

Parameter definieren

Mit `a:=1.23` wird ein Parameter mit dem Namen `a` definiert, er hat zunächst den Wert `1.23` .

3.2.2 Vektoren (Spaltenvektoren)

Grundlagen

Definieren über eine Komponentenliste

Die Komponenten eines Vektors können Zahlen oder Terme aus Zahlen, Variablen und Parametern sein.

So definiert man durch `zv:=vektor({3,4,5})` den dreidimensionalen Spaltenvektor mit den Komponenten 3, 4 und 5.
Allgemein werden die Komponenten durch Kommata getrennt aufgezählt, die Aufzählung wird in Klammern `{}` eingeschlossen.

Wenn man zuvor mit `a:=2` einen Parameter definiert hat, dann erhält man mit
`pv:=vektor({1, a, a^2, a^3})`
einen vierdimensionalen Vektor. Die einzelnen Komponenten sind in unterschiedlicher Weise von `a` abhängig.

Definieren über eine Vorschrift

Wenn sich alle Komponenten eines Vektors durch einen Term beschreiben lassen, dann kann der Vektor mit Hilfe dieses Terms erzeugt werden:

```
zv1:=vektor(i,3,i+2)
```

An der ersten Stelle wird der Name eines lokalen Parameters vereinbart (hier: `i`), an der zweiten Stelle wird die Dimension dim festgelegt (hier: `3`), zuletzt der Initialisierungsterm (hier: `i+2`).

Bei der Erzeugung des Vektors druchläuft der lokale Parameter die Werte 1, 2, ..., dim und belegt die Komponenten mit den zugehörigen Instanzen des Initialisierungsterms. Im Beispiel hat der Spaltenvektor `zv1` die Komponenten `1+2`, `2+2` und `3+2`.

Vektor durch ASCII-Datei initialisieren

Dateiauswahl durch Pfadangabe

In der Definition `dv1:=vektor( finput:C:/work/temp/testvek.txt )` bewirkt das Schlüsselwort `finput` , dass die Komponenten des Vektors `dv1` aus der Datei mit dem Pfad `C:/work/temp/testvek.txt` eingelesen werden.

Das Schlüsselwort und die Pfadangabe werden dabei durch einen Doppelpunkt getrennt.

Dateiauswahl durch Benutzerdialog

Wenn wie bei `dv2:=vektor( finput )` keine Datei angegeben ist, wird der Pfad beim Laden der Definition über einen Benutzerdialog ermittelt.

Struktur der ASCII-Datei

Die Komponenten eines Vektors werden dabei in einer ASCII-Datei in einer Zeile durch Zahlen angegeben:

```
1.23  1.0e-4  -23.77      -17.23
```

Die einzelnen Komponenten werden durch Leerzeichen oder Tabulatoren voneinander getrennt.

Vektoren in ASCII-Datei exportieren

Bei nullstelligen vektorwertigen Funktionen wie `v:=vektor({1,2,3})` oder `w:=3/|v|*v` kann das aktuelle Ergebnis in eine ASCII-Datei exportiert werden. Dazu klickt man mit der rechten Maustaste auf das entsprechende Element der Definitions- oder Detailliste und startet über das Menüelement **Exportieren ...** den Speicherdialog.

Zunächst werden aus dem vorliegenden Term die aktuellen Werte der Komponenten berechnet. Diese werden als Gleitpunktzahlen, durch Tabulatoren getrennt, in der Datei abgelegt.

Vordefinierte Vektoren

Mit `e2x` ist der Vektor $\begin{bmatrix}1\\0\end{bmatrix}$ und mit `e2y` der Vektor $\begin{bmatrix}0\\1\end{bmatrix}$ vordefiniert.

Auf Komponenten zugreifen

Wie in `t:=pv[4]*sin(zv[1])` greift man mit `pv[4]` oder `zv[1]` auf die vierte bzw. erste Komponente des Vektors `pv` bzw. `zv` zu. Der Index der Komponente kann wie in `pv[a-1]` über einen Term gegeben sein. Der Term muss zum Zeitpunkt der Auswertung einen gültigen Index liefern.

Indexwerte werden abgerundet.

Bei einem zweidimensionalen Vektor `v2dim` kann man auch mit den Methoden `.x` und `.y` auf die beiden Komponenten zugreifen: `v2dim.x` ist gleichwertig mit `v2dim[1]`, analog `v2dim.y` zu `v2dim[2]`.

Entsprechendes gilt bei einem dreidimensionalen Vektor für die Methoden `.x`, `.y` und `.z`.

Methoden

- **Dimension**
  Für Vektoren wie `zv` oder `pv` von oben liefert die Methode `.dimension` die Anzahl der Zeilen, also die Dimension des Vektors.
  So liefert zu `pv:=vektor({1, a, a^2, a^3})` der Ausdruck `pv.dimension`

den Wert 4.

Diese Methode ist nur auf einen einzelnen Vektor anwendbar, nicht aber auf Vektor-Terme.

Wichtige Funktionen für Vektoren

Zu einem Vektor `v:=vektor({1,2,3})` berechnet man mit

- `|v|` seinen Betrag, mit
- `2*v` das zweifache von `v`, man berechnet mit
- `v/2` die Hälfte von `v`.
- Hat man zusätzlich `w:=vektor({-1,2,0})` definiert, so berechnet `v<*>w` das Skalarprodukt der beiden Vektoren.

Methoden, die Vektoren als Ergebnis liefern

Folgende Methoden anderer Objekte liefern als Ergebnis einen Vektor:

- `.r` den Ortsvektor eines Punktes wie bei `punkt1.r` ,
- `.richtung` den Richtungsvektor einer Geraden wie bei `gerade1.richtung` ,
- die Richtung der beiden Schenkel eines Winkels `winkel1` durch `winkel1.richtung1` und `winkel1.richtung2`.

Graphische Darstellung

- Standard-Graph
  Der Standard-Graph eines Vektors ist ein Balkendiagramm. Für einen Vektor der Dimension dim werden über den Stellen 1, 2, 3, ..., dim der x-Achse die entsprechenden Komponenten des Vektors als Balken aufgetragen.
- Balkendiagramm
  Das Balkendiagramm (vgl. 3.5.2) ist eine Erweiterung des Standard-Graphen. Hier werden wie in `b:=balken(vx,v)` die Stellen der x-Achse, über denen die Komponenten eines Vektors `v` aufgetragen werden, durch einen zweiten Vektor `vx` festgelegt.
- Pfeil
  Ein zweidimensionaler Vektor `v2d` kann wie bei `pf:=pfeil(punkt(1,2), v2d)` mit Hilfe eines Pfeils-Graphen (vgl. 3.4.11) dargestellt werden. Die Pfeilspitze kann mit der Maus bewegt werden, damit werden die Komponenten des Vektors verändert.

Fortgeschrittene Anwendungen

Dimension über einen Term festlegen

Wenn man mit `dd:=3` einen Parameter definiert, kann man anschließend einen Vektor erzeugen, dessen Dimension vom Wert dieses Parameters zum Zeitpuknt der Erzeugung abhängt:
`vd:=vektor(i,dd*dd,2*i)` erzeugt einen Spaltenvektor der Dimension 9. Allgemein wird zur Bestimmung der Dimension der aktuell gültige Wert des Dimensionsterms berechnet.

Vektorwertige Funktionen

Eine vektorwertige Funktionen entsteht, wenn der Definitionsterm auf der rechten Seite der Funktionsdefinition als Ergebnis einen Vektor liefert.

Im einfachsten Fall definiert man wie bei `cv:=vektor({7,-1,3,4})` eine nullstellige vektorwertige Funktion, kurz einen (konstanten) Vektor.
`pv(x):=vektor(i,4,x^(i-1))` definiert eine einstellige vektorwertige Funktion, die Komponenten des Vektors sind die x-Potenzen $x^{1-1}, x^{2-1}, x^{3-1}, x^{4-1}$.

Eine vektorwertige Funktion kann nun ihrerseits in einem Term überall da stehen, wo ein Vektor zugelassen ist. So definiert `poly(x):=cv<*>pv(x)` mit Hilfe des Skalarprodukts `<*>` jetzt die Polynomfunktion $7 \cdot x^0 - 1 \cdot x^1 + 3 \cdot x^2 + 4 \cdot x^4$.

Auf Komponenten vektorwertiger Funktionen zugreifen

Auf die Komponenten einer vektorwertigen Funktion wie `v(x):=vektor({x,x^2, x^3})` greift man zu, indem man zuerst durch Angabe eines Arguments einen Ergebnisvektor berechnet und anschließend durch Angabe eines Index die gewünschte Komponente auswählt:

`a:=v(3)[2]` berechnet mit dem Argument `(3)` zuerst den Ergebnisvektor $\begin{bmatrix} 3 \\ 9 \\ 27 \end{bmatrix}$, durch die Angabe des Index `[2]` wird davon die 2.Komponente, also konkret das Ergebnis 9 ausgewählt.

Entsprechend definiert `f(x):=v(sin(x))[3]` eine einstellige Funktion. Zu jedem Argument `x` berechnet `f(x)` insgesamt den Wert $(sin(x))^3$.

Vektoren als Funktionsargumente

Auch bei vom Benutzer definierten Funktionen können Vektoren als Argumente verwendet werden. Ein solches Argument muss durch den vorangestellten Typbezeichner `<vektor>` als vektorwertig gekennzeichnet werden. So kann man für zwei dreidimensionale Vektoren das Skalarprodukt `sp` und das Vektorprodukt `vp` wie folgt definieren:

```
sp(<vektor>a, <vektor>b):=a.x*b.x+a.y*b.y+a.z*b.z und
vp(<vektor>a, <vektor>b):=
         vektor({a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x})
```

Vektoren über eine Punktliste initialisieren

Hat man mit `pl:=<punkt>liste(i,10,punkt(i,1))` eine Liste von Punkten bereit gestellt, dann erzeugt man mit `xv:=vektor(i,10,pl[i].x)` einen Vektor, der die x-Koodinaten der Punkte enthält, entsprechend `yv:=vektor(i,10,pl[i].y)` die y-Koordinaten. Mit diesen beiden Vektoren erhält man dann z.B. mit `sp(x):=spline(splinecoeff(xv,yv), x)` die kubische Spline-Interpolation der Punkte.

3.2.3 Dualvektoren (Zeilenvektoren)

Zeilenvektoren werden wie `dv:=dualvektor({3,4,5})` mit dem Schlüsselwort `dualvektor` erzeugt.

Ansonsten gelten die im obigen Abschnitt für Spaltenvektoren beschriebenen Konventionen entsprechend.

3.2.4 Matrizen

Definieren über eine Zeilenliste

Die Elemente einer Matrix können Zahlen oder Terme aus Zahlen und Parametern sein. Wenn man zuvor mit `a:=2` einen Parameter definiert hat, dann erhält man mit
`zm := matrix({ {a*1, a*2, a*3 }, {a*4, a*5, a*6}, {a*7, a*8, a*9} })`
eine quadratische Matrix der Ordnung 3. Die erste Zeile enthält die Elemente (Terme)
`a*1`, `a*2` und `a*3`.
Die Definition `pm := matrix({ {1, 2, 3}, {0, 1, 2} })` erzeugt eine 2x3-Matrix mit den Elementen `1, 2, 3` in der ersten Zeile.
Allgemein werden die Zeilen durch Kommata getrennt aufgezählt, die Aufzählung wird in Klammern `{}` eingeschlossen. Jede Zeile ist ihrerseits wieder eine Liste von Elementen, diese werden wiederum in `{}` eingeschlossen.

Definieren über eine Vorschrift

Wenn sich alle Elemente einer Matrix durch einen allgemeinen Term beschreiben lassen, dann kann die Matrix mit Hilfe dieses Terms erzeugt werden wie
`zm1 := matrix( zei, 3 , spa, 4, (-1)^(zei+spa) ).`
Die Parameter haben der Reihe nach folgende Bedeutung:

- lokaler Zeilenindex, hier: `zei ,`
- Anzahl der Zeilen, hier `3 ,`
- lokaler Spaltenindex, hier `spa ,`
- Anzahl der Spalten, hier `4` und
- allgemeiner Term für die Initialisierung, hier `(-1)^(zei+spa) .`

So entsteht eine 3x4-Matrix mit den Elementen (Termen)
`(-1)^(1+1)`, `(-1)^(1+2)`, `(-1)^(1+3)` und `(-1)^(1+4)` in der ersten Zeile.

Matrix durch ASCII-Datei initialisieren

Dateiauswahl durch Pfadangabe

In der Definition `mat1:=vektor( finput:C:/work/temp/testmat.txt )` bewirkt das Schlüsselwort `finput` , dass die Komponenten der `mat1` aus der Datei mit dem Pfad `C:/work/temp/testmat.txt` eingelesen werden.
Das Schlüsselwort und die Pfadangabe werden dabei durch einen Doppelpunkt getrennt.

Dateiauswahl durch Benutzerdialog

Wenn wie bei `mat2:=vektor( finput )` keine Datei angegeben ist, wird der Pfad beim Laden der Definition über einen Benutzerdialog ermittelt.

Struktur der ASCII-Datei

Die Komponenten einer Matrix werden dabei in einer ASCII-Datei zeilenweise durch Zahlen angegeben. Folgende Daten erzeugen eine Matrix mit drei Zeilen und 4 Spalten:

```
1.23  1.0e-4  -23.77      -17.23
1  2  4  5
-123.555  0   1      1e-12
```

Die einzelnen Komponenten einer Zeile werden durch Leerzeichen oder Tabulatoren voneinander getrennt.

Matrizen in ASCII-Datei exportieren

Bei nullstelligen matrixwertigen Funktionen wie `m:=matrix({1,2},{1,3})` oder `w:=m*m/3` kann das aktuelle Ergebnis in eine ASCII-Datei exportiert werden. Dazu klickt man mit der rechten Maustaste auf das entsprechende Element der Definitions- oder Detailliste und startet über das Menüelement **Exportieren ...** den Speicherdialog.
Zunächst werden aus dem vorliegenden Term die aktuellen Werte der Komponenten berechnet. Diese werden zeilenweise als Gleitpunktzahlen, durch Tabulatoren getrennt, in der Datei abgelegt.

Matrixwertige Funktionen

Matrixwertige Funktionen werden ähnlich wie skalare Funktionen definiert:
`mm(x):=matrix({ {1, x, x^2, x^3}, { 1*x, 2*x, 3*x, 4*x} })` oder
`mm1(x):=matrix(i, 4, k, 4, x^(i+k))`.
Mehrstellige matrixwertige Funktionen sind möglich.

Auf Elemente zugreifen

- Zugriff auf Elemente einer Matrix:
  Wie in `t:=zm[1,2]` greift man auf ein Element einer Matrix `zm` durch Angabe eines Zeilenindex (hier: 1) und eines Spaltenindex (hier: 2) zu. Die beiden Indexangaben werden durch ein Komma getrennt. Beide Angaben können durch einen Term gegeben sein. Ein Term muss zum Zeitpunkt der Auswertung einen gültigen Index liefern. Indexwerte werden abgerundet.
- Zugriff auf Ergebnis-Elemente eines Matrix-Terms:
  Wie bei `(mat1*mat2-3*mat3)[1,3]` muss man den Term zuerst mit `()` klammern, dann erfolgt durch Angabe der entsprechenden Indices der Zugriff auf das gewünschte Element.
- Zugriff auf Elemente einer matrixwertigen Funktion:
  Auf die Elemente einer matrixwertigen Funktion wie `mm1(x)` oder
  `m2(x):=mm1(x)*mm1(x*x)` greift man zu, indem man zuerst durch Angabe eines Arguments eine Ergebnismatrix berechnet und anschließend durch Angabe eines Indexpaars das gewünschte Element auswählt, wie z.B. bei `a:=m2(1.7)[1,2]` oder
  `f(x):=m2(x*x)[2,3]` oder `g(x):=(mm1(x*x)+mm1(x)*mm1(x))[3,4]`.

Methoden

- **Zeilenzahl**
  Für Matrizen wie `zm`, `pm` und `zm1` von oben liefert die Methode `.zeilen` die Anzahl der Zeilen, also liefert z.B. zur 2x3-Matrix `pm := matrix({ {1, 2, 3}, {0, 1, 2} })` der Ausdruck `pm.zeilen` den Wert 2.
  Definiert man hier zusätzlich die 3x4-Matrix `rm:=matrix(i,3,k,4,i+k)` und das Produkt `pr:=pm*rm`, dann liefert `pr.zeilen` den Wert 2.
- **Spaltenzahl**
  Die Methode `.spalten` liefert analog die Anzahl der Spalten einer Matrix. Im obigen Beispiel liefert `pr.spalten` den Wert 4.

Diese Methoden sind nur auf einzelne Matrizen anwendbar, nicht aber auf Matrix-Terme.
Ungültig ist also ein Ausdruck wie `(pm*rm).zeilen` .

Wichtige Funktionen für Matrizen

Für zwei Matrizen `mat1` und `mat2` und eine quadratische Matrix `qmat` berechnet (bei passenden Zeilen- und Spaltendimensionen):

- `mat1 + mat2` : Summe
- `mat1 - mat2` : Differenz
- `mat1*mat2` : Matrixprodukt
- `| mat1 |` : Frobenius-Norm
- `det(qmat)` : Determinante
- `inverse(qmat)` **:** Inverse Matrix, falls $det(qmat) \neq 0$.
- `solvelin(qmat,inhom)` : Lösung des linearen Gleichungssystems mit der Koeffizientenmatrix `qmat` und der Inhomogenität `inhom`, falls $det(qmat) \neq 0$.

3.2.5 Listen

Grundlagen

Eine Liste dient zur Verwaltung einer Folge gleichartiger Elemente. Ein bestimmtes Element wird durch Angabe seines Index angesprochen. Beim Erzeugen einer Liste wird festgelegt, nach welcher Konvention die Indices verwendet werden:

`liste(...)` erzeugt eine Liste, bei der das erste Element mit dem Index 1 angesprochen wird,

`liste0(...)` erzeugt eine Liste, bei der das erste Element mit dem Index 0 angesprochen wird.

Damit kann man die Indizierung auf die im jeweiligen Problemfeld übliche Konvention abstimmen.

Elementtypen

Punkt-Listen

Listen von Punkten werden durch den Typbezeichner `<punkt>` gekennzeichnet. Durch `pl:=<punkt>liste({punkt(1,1), punkt(2,2)})` definiert man eine Punkt-Liste mit zwei Punkten, das erste Element der Liste wird mit `pl[1]` angesprochen.

Funktions-Listen

Listen von Funktionen werden durch den Typbezeichner `<funktion>` gekennzeichnet. Bei nullstelligen Funktionen genügt diese Angabe:

`nullstFktListe:=<funktion>liste({pi, 2*pi, 4*pi})` erzeugt eine Liste mit drei nullstelligen Funktionen. Nun eine Liste mit einstelligen Funktionen:

`einstFktL:=<funktion(x)>liste({sin(x), cos(x), x^2})` wird durch den Typbezeichner `<funktion(x)>` vereinbart, dass die Liste einstellige Funktionen enthält und dass das Argument mit `x` bezeichnet wird. Die Liste einstFktListe enthält dann konkret die drei Elemente `sin(x)`, `cos(x)` und `x^2`. Mit `einstFktL[2](x*x)` wird der Wert des zweiten Elements dieser Liste an der Stelle `x*x` (also `cos(x*x)`) berechnet.

Definieren über eine Elementliste

Wie oben bei `einstFktL:=<funktion(x)>liste({sin(x),cos(x),x^2})`
kann man eine Liste definieren, indem man ihre Elemente, durch Kommata getrennt, aufzählt. Die Aufzählung wird in Klammern `{}` eingeschlossen.

Definieren über eine Vorschrift

Wenn lange Listen erzeugt werden sollen, bietet sich die Definition über eine Vorschrift an.

`pl2:=<punkt>liste(i,11,punkt(i-6,1))` definiert eine Liste mit 11 Punkten, bei sämtlichen Punkten hat die y-Koordinate den Wert `1`, die x-Koordinaten haben der Reihe nach die Werte `1-6`, `2-6`, ..., `11-6`.

Allgemein gilt bei der Definition über eine Vorschrift: An der ersten Stelle wird der Name eines lokalen Parameters vereinbart (hier: `i`), an der zweiten Stelle wird die Länge der Liste (hier: `11`) festgelegt, zuletzt Initialisierungsvorschrift (hier: `punkt(i-6,1)`).

Bei der Erzeugung der Liste mit Schlüsselwort `liste` druchläuft der lokale Parameter dann konkret die Werte 1, 2, ..., 11 und belegt die Elemente mit den zugehörigen Ergebnissen aus der Initialisierungsvorschrift.

Entsprechend erzeugt man mit `fl2:=<funktion(x)>liste(i,10,sin(i*x))` eine Liste aus einstelligen Funktionen.

Mit `fl3:=<funktion(x)>liste0(i,10,x^i)` wird ebenfalls eine Liste mit 10 Elementen erzeugt. Beim Erzeugen dieser Liste mit Schlüsselwort `liste0` durchläuft der lokale Parameter die 10 Werte 0,1,2,...,9 .

Auf Elemente zugreifen

Je nach Indizierungskonvention (`liste` bzw. `liste0`) sind die Elemente einer Liste der Reihe nach durch die Zahlen 1, 2, 3, ... bzw. 0, 1, 2,... identifiziert. Durch Angabe eines Index greift man auf das gewünsche Element zu: `pl2[1]` liefert das erste Element aus der oben definierten Punkt-Liste, `pl2[2]` das zweite, ... Definiert man einen Parameter `a:=1`, so ist jetzt durch `s:=strecke(pl2[a], pl2[a+1])` zunächst die Verbindungsstrecke zwischen den ersten beiden Punkten der Punktliste gemeint. Ändert man jetzt den Parameter zu `a:=2`, dann verbindet s die Punkte mit dem Index 2 und 3 der Punktliste `pl2`.

Bei einer Funktionsliste wählt man zuerst durch Angabe eines Index die gewünscht Funktion aus, anschließend wird diese auf ein Argument angewendet: `einstFktL[2](2*3)` .

Methoden einer Liste

- **Länge einer Liste**
  `.laenge` liefert die Anzahl der Elemente einer Liste.
  Konkret liefern `fl2.laenge` und `fl3.laenge` beide den Wert 10.
- **Index des ersten Elements**
  `.startindex` liefert den Index des ersten Elements, also `fl2.startindex` den Wert 1, `fl3.startindex` den Wert 0.
- **Index des letzten Elements**
  `.endeindex` liefert den Index des letzten Elements einer Liste, also `fl2.endeindex` den Wert 10, `fl3.endeindex` den Wert 9.

Mit `f(x):=sum(i,fl3.startindex,fl3.endeindex,fl3[i](x))` kann man jetzt eine Summation über sämtliche Elemente der Liste `fl3` ausführen.

3.3 Funktionen

3.3.1 Überblick

Ein-, mehr- und nullstellige Funktionen werden wie gewohnt definiert. Neben den üblichen Standardfunktionen (siehe 3.3.3) können zur Definition auch Methoden arithmetischer oder affiner Objekte verwendet werden. In `sin( strecke1.laenge )` wird zu einer Strecke `strecke1` zunächst über den Methodenaufruf `.laenge` die Länge der Strecke berechnet, der resultierende Wert ist dann das Argument für die Standardfunktion `sin()` .

Die verfügbaren Methoden sind bei der Beschreibung der verschiedenen Objekt-Typen aufgelistet.

3.3.2 Funktionen definieren

Eine Funktion wird im einfachsten Fall wie `funkt1(x):=x*sin(x)` definiert.

Auf der linken Seite des Definitionssymbols `:=` wird mit `funkt1` der **Name** der Funktion vereinbart und eine reelle **Variable** mit dem Namen `x`. Beide werden nach den Regeln für Bezeichner (vgl. 3.1) gebildet.

Der Name der Funktion muss eindeutig sein, d.h. Sie können keine andere Funktion (und kein anderes Objekt) mit dem gleichen Namen vereinbaren.

Der Name der Variable gilt nur lokal in der aktuellen Definition. Sie können also in mehreren (in allen) Definitionen die Variable mit `x` bezeichnen.

Die Zuordnungsvorschrift wird durch den **Funktionsterm** auf der rechten Seite von `:=` festgelegt.

Anzahl der Variablen, Stelligkeit

Neben den typischen einstelligen Funktionen können auch Funktionen mit einer anderen Anzahl von Variablen (d.h. einer anderen Stelligkeit) definiert werden.

Mit `mult(x,y):=x*y` wird eine zweistellige Funktion, mit
`minmax(x,y,z):=max(min(x,y), z)` wird eine dreistellige Funktion definiert.

Beim Aufruf einer mehrsteligen Funktion erfolgt die Zuordnung der Argumente über die Position:

`minmax(3,7,5)` berechnet zunächst `min(3,7)=3` und dann `max(3,5)=5` als Endergebnis.

Bei der Definition einer nullstelligen Funktion wie `a:=pi*sin(3)` kann die Angabe der leeren Klammern `()` entfallen.

Typ der Variablen

Häufig werden Variablen verwendet, die einen reellen Zahlenwert (Skalar) vertreten. Bei solchen Variablen wird kein Typ angegeben.

Bei einer Variable, die einen Vektor (Spaltenvektor), Dualvektor (Zeilenvektor) oder eine Matrix vertritt, muss der entsprechende Typ angegeben werden.

In `vv(<vektor>x,<vektor>y,a) := sum(i, 1, a, v1[i]*v2[i])`

werden zwei Vektor-wertige Variable `x` und `y` vereinbart und als dritte Variable eine Zahlenvariable `a`.

Die Typangaben sind:

`<vektor>`	für einen Vektor (Spaltenvektor)
`<dualvektor>`	für einen Dualvektor (Zeilenvektor) und
`<matrix>`	für eine Matrix.

Funktionsterm

Zum Aufbau des Funktionsterms können Sie verwenden:

- Zahlen, Vektoren und Matrizen,
- vordefinierte Operatoren und Funktionen,
  insbesondere die Operatoren `+`, `-`, `*`, `/` und `^` ,
- Funktionen, die Sie vorher schon definiert haben und
- Methodenaufrufe für Objekte, die Sie vorher schon definiert haben,
  z.B. erhalten Sie zu einem schon definierten Punkt `p` mit dem Methodenaufruf `p.x` dessen aktuelle x-Koordinate.

Ergebnistyp

Der Ergebnistyp einer Funktion ist durch den Funktionsterm festgelegt. Definiert man z.B.

`f1(<vektor>x, <vektor>y):=2*x<*>y` und

`f2(<vektor>x, <vektor>y):=x[1]*y`, dann gilt:

`f1(<vektor>x, <vektor>y)` liefert als Ergebnis das zweifache Skalarprodukt der Vektoren `x` und `y`, also eine Zahl (Skalar),

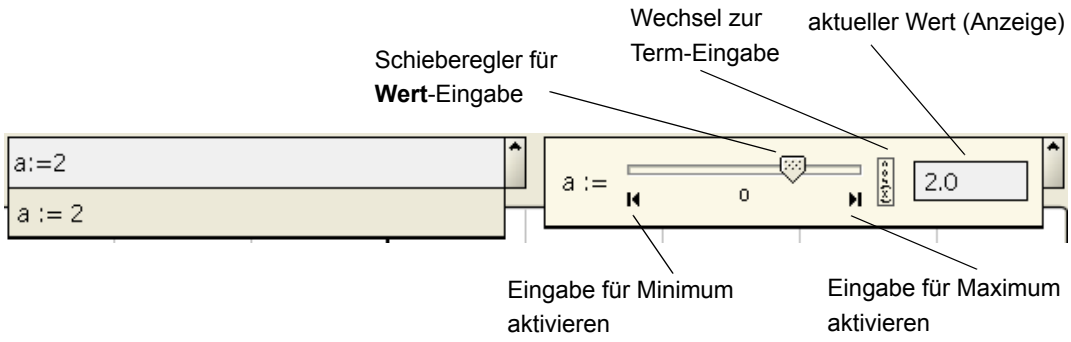
`f2(<vektor>x, <vektor>y)` multipliziert die erste Komponente des Vektors `x` (das ist eine Zahl) mit dem Vektor `y`, das Ergebnis ist ein Vektor.

Nullstellige reelle Funktionen: Parameter

Nullstellige reelle Funktionen werden oft bereitgestellt, um sie später als Parameter in anderen Funktionen zu verwenden. Sie werden im Folgenden deshalb auch kurz als Parameter bezeichnet.

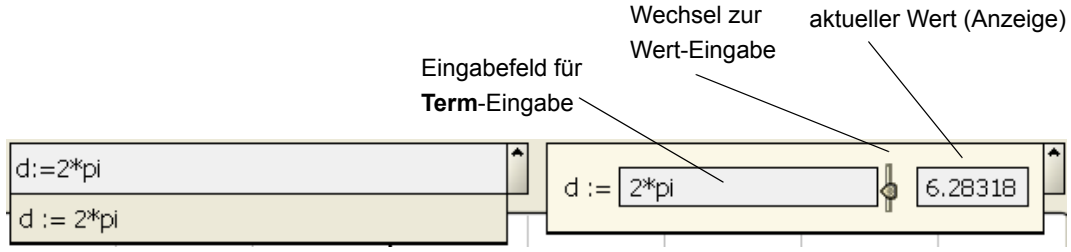
Definiert man z.B. zunächst `a:=2`, `b:=1`, `c:=1` durch Zahlenwerte und dann das Polynom `p(x):=a*x^2+b*x+c` , dann kann man leicht experimentell verfogen, welchen Einfluß diese Parameter auf die Eigenschaften des Polynoms haben.

Wenn Sie das Element `a:=2` der Definitionsliste nach rechs ziehen, erhalten Sie folgende Detailsicht:



Sie können den Wert jetzt bequem mit dem Schieberegler verändern. Wenn der Regelbereich nicht ausreicht, aktivieren Sie die Eingabe für Minimum bzw. Maximum und ändern ihn.

An einer anderen Stelle kann es sinnvoll sein, den Wert eines Parameters durch einen Term festzulegen. Definiert man d:=2*pi, dann wird beim Erzeugen der Detailsicht der Definitionsterm angezeigt:

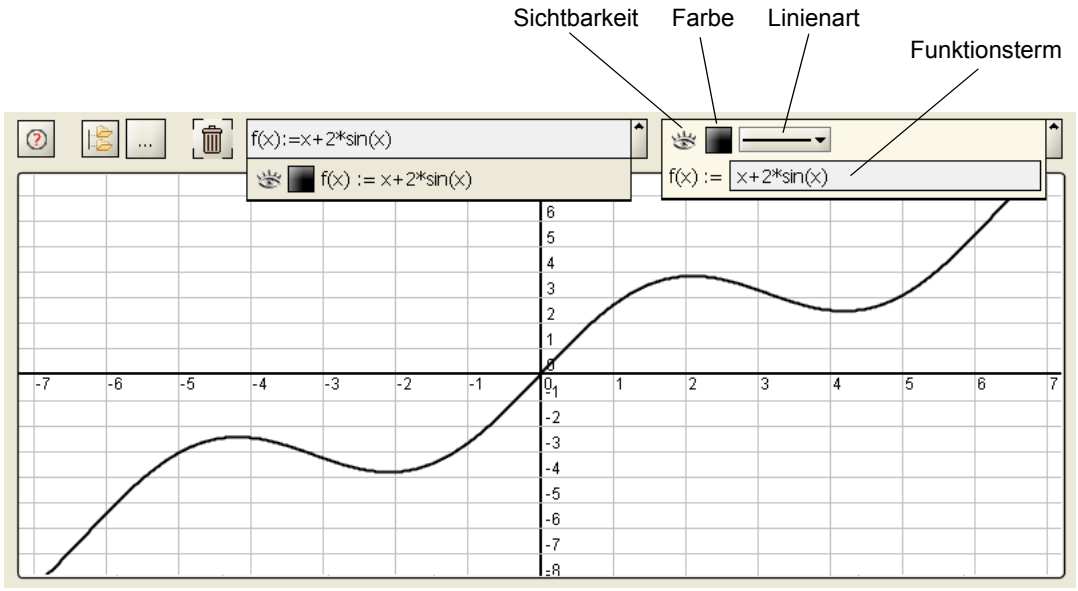


Bei Bedarf kann man in der Detailsicht durch Klicken auf das Alternativ-Symbol zwischen Schieberegler für Wert-Eingabe und Feld für Term-Eingabe wechseln.

Einstellige reelle Funktionen einer reellen Variable

Wenn Sie die Definition `f(x):=x+2*sin(x)` aus der Definitionsliste nach rechs ziehen, erhalten Sie die unten dargestellte Detailsicht.

Sie können eine einstellige reelle Funktion zeichnen (genauer: einen zugehörigen Graphen), indem Sie das entsprechende Element der Definitionsliste mit gedrückter Maustaste auf eine Zeichenfläche ziehen.



Sie können über die Detailsicht den Funktionsterm einer Funktion redefinieren, d.h. durch einen anderen Term ersetzen.

Wenn Sie zu einer Funktion bereits eine Detailsicht erzeugt haben, dann können Sie einen Graphen auch erzeugen, indem Sie diese Detailsicht auf eine Zeichenfläche ziehen.

3.3.3 Standardfunktionen

Folgende Konstanten und Standardfunktionen sind vordefiniert:

Zahlen-Konstanten

Bezeichner	Bedeutung
pi	Kreiszahl π
e	Eulersche Zahl

Vordefinierte Vektoren

Bezeichner	Bedeutung
e2x	Einheitsvektor in x-Richtung: $\begin{bmatrix}1\\0\end{bmatrix}$
e2y	Einheitsvektor in y-Richtung: $\begin{bmatrix}0\\1\end{bmatrix}$

Einstellige Funktionen

Bezeichner	Bedeutung
|x| oder abs(x)	Betrag, dabei x: reelle Zahl
arccos(x)	Arcus-Cosinus
arcsin(x)	Arcus-Sinus
arctan(x)	Arcus-Tangens
ceil(x)	aufrunden auf die nächst größere ganze Zahl
cos(x)	Cosinus zum Bogenmaß x
exp(x)	Exponentialfunktion zur Basis e
floor(x)	abrunden auf die nächst kleinere ganze Zahl
ln(x)	natürlicher Logarithmus
maxelem(MatVec)	Wert des größten Element einer Matrix oder eines Vektors MatVec
minelem(MatVec)	Wert des kleinsten Element einer Matrix oder eines Vektors MatVec
signum(x)	Vorzeichen-Funktion: $signum(x) = \begin{cases} -1 & \text{falls } x < 0 \\ 0 & \text{falls } x = 0 \\ 1 & \text{falls } x > 0 \end{cases}$
sin(x)	Sinus zum Bogenmaß x

Bezeichner	Bedeutung
sqrt(x)	Quadratwurzel
tan(x)	Tangens zum Bogenmaß x
x!	Fakultät, dabei x: 0,1,2,3,...

Zweistellige Funktionen mit Operatorschreibweise

Es gilt die übliche Bindungsstärke "Exponentiation vor Punkt vor Strich".
Die Assoziativität ist wie angegeben.

Term	Bedeutung	Assoziativität
x+y	Summe von zwei reellen Zahlen (d.h. Skalaren), zwei Vektoren, zwei Dualvektoren oder zwei Matrizen	linksassoziativ
x-y	Differenz von zwei reellen Zahlen (d.h. Skalaren), zwei Vektoren, zwei Dualvektoren oder zwei Matrizen	linksassoziativ
x*y	Produkt von zwei reellen Zahlen x, y, von zwei Matrizen x, y oder einer Zahl x mit einer Matrix (einem Vektor, einem Dualvektor) y oder einer Matrix (eines Vektors, eines Dualvektors) x mit einer Zahl y	linksassoziativ
x/y	Quotient von zwei Zahlen x, y , Quotient einer Matrix (eines Vektors, eines Dualvektors) x durch eine Zahl y	linksassoziativ
x <*> y	Skalarprodukt der beiden Vektoren x, y	-
x^y	Exponentiation der Zahl x mit dem Exponenten y	rechtsassoziativ

Zweistellige Funktionen

Bezeichner	Bedeutung
max(x,y)	Maximum der beiden Zahlen x, y
min(x,y)	Minimum der beiden Zahlen x, y
mod(x,y)	Rest nach Division von x durch y. Das Ergebnis hat dasselbe Vorzeichen wie x.
polynom(cv,x)	Wert des Polynoms mit den Koeffizienten des Koeffizientenvektors cv an der Stelle x. Berechnet wird je nach Dimension des Vektors cv der Wert $cv[1] + cv[2] \cdot x + cv[3] \cdot x^2 + cv[4] \cdot x^3 + \ldots$

Transformationen

Bezeichner	Bedeutung
splinecoeff(xv, yv)	Spline-Koeffizienten Berechnet zu einem Vektor xv von Stützstellen und einem passenden Vektor yv von Stützwerten die Koeffizienten einer kubischen Spline-Interpolation. Die Koeffizienten werden als Matrix zurückgegeben. MIt der Funktoin spline(sc,x) wird die eigentliche Interpolation erzeugt.
spline(sc, x)	Kubische Spline-Interpolation Wert einer Spline-Interpolatoin an der Stelle x. Die Koeffizienten sc der Interpolation werden mit der Funkton splinecoeff(xv, yv) berechnet.

Lineare Algebra

Bezeichner	Bedeutung
det(A)	Wert der Determinante der quadratischen Matrix A.
transpose(A)	Transposition der Matrix A
inverse(A)	Zur quadratischen Matrix A inverse Matrix, falls $det(A) \neq 0$
solvelin(A,b)	Berechnet den Lösungsvektor *x* eines linearen Gleichungssystems $A \cdot x = b$. A ist dabei die quadratischen Matrix mit $det(A) \neq 0$, b ein von der Dimension her passender Vektor (rechte Seite, Inhomogenität).

3.3.4 Summen- und Produktiterator

Struktur	Bedeutung

`sum(index,unten,oben,term)`
 	
wie z.B.
 	
`f(x):=sum(i,0,3,x^i)`
	Summe
	`index`: Bezeichner für den Summationsindex
	`unten`: Term für die untere Summationsgrenze mit ganzzahligem Wert
	`oben`: Term für die obere Summatioonsgrenze mit ganzzahligem Wert
	`term`: Term für den Summanden

`prod(index,unten,oben,term)`
 	
wie z.B.
 	
`fak(i):=prod(ind,1,i,i)`
	Produkt
	`index`: Bezeichner für den Summationsindex
	`unten`: Term für die untere Summationsgrenze mit ganzzahligem Wert
	`oben`: Term für die obere Summatioonsgrenze mit ganzzahligem Wert
	`term`: Term für den Faktor

Zum Zeitpunkt der Auswertung müssen die Terme für die obere und die untere Summationsgrenze eine ganze Zahl als Wert liefern. Falls die Summationsgrenze über einen Parameter `a` verändert werden soll, kann man die Ganzzahligkeit z.B. durch `floor(a)` oder `ceil(a)` sicherstellen.

Wenn der Wert der oberen Grenze kleiner ist als der Wert der unteren Grenze, liefert `sum` das Ergebnis 0, `prod` das Ergebnis 1.

Beispiel

Definiert man zunächst `n:=3`, so kann man mit

`tayCos(x):=sum(i,0,n,(-1)^i*x^(2*i)/(2*i)! )` das allgemeine Taylor-Polynom für die cos-Funktion mit Entwicklungspunkt c=0 definieren. Über den Wert von `n` kann man dann die Ordnung des Polynoms verändern.

3.3.5 Ableitung

Ableitung reeller Funktionen

Nach der Definition einer Funktion wie `g(x):=x*sin(x)` können zu dieser die Ableitungen berechnet werden. Bei

`g1(x):=derive(g(x))` wird `g1(x)` als die erste Ableitung der Funktion `g(x)` definiert.

Höhere Ableitungen werden durch Angabe der Ordnung der Ableitung spezifiziert:
`g3(x):=derive(g(x),3)` .

Wenn vorher ein Parameter `ord:=4` definiert wurde, kann man auch über diesen die Ordnung der Ableitung steuern: `gabl(x):=derive(g(x),ord)` .

Der Parameter `ord` darf dann natürlich nur Werte 0, 1, 2, 3, ... annehmen.

Ableitungen in einem Funktionsterm verwenden

Ableitungen können auch direkt in einem Funktionsterm verwendet werden:

`h(x):=g(x)+x*derive(g(x))+x^2*derive(g(x),2).`

Wert einer Ableitung an einer bestimmten Stelle

Wenn in einem Funktionsterm Ableitungen einer Funktion an einer bestimmten Stelle benötigt werden, dann kann diese Stelle wie bei `derive(g(x))(1)` als Funktionsargument `(1)` angegeben werden:

`tay2(x) := g(1)+derive(g(x))(1)*(x-1)+derive(g(x),2)(1)/2*(x-1)^2`

berechnet zu `g(x)` das Taylor-Polynom vom Grad 2 zum Entwicklungspunkt `1,` darin werden die erste und zweite Ableitung an der Stelle 1 benötigt.

Wenn die zweite Ableitung gebildet, aber an der Stelle `x*x` ausgewertet werden soll, so geschieht dies durch `derive(g(x),2)(x*x)` .

Neuberechnung der Ableitung

Eine Ableitung wie `gabl(x):=derive(g(x),ord)` wird in folgenden Fällen neu berechnet:

- Die Definition der abzuleitenden Funktion `g(x)` hat sich geändert oder
- die Ordnung `ord` der Ableitung hat sich geändert.

3.3.6 Partielle Ableitung

Mehrstellige Funktionen können partiell abgeleitet werden.

Mit `d(x,y):=pderive(x*y*sin(y), x, y, y)` wird `d(x,y)` als partielle Ableitung dritter Ordnung der Funktion `x*y*sin(y)` definiert, dabei wird zuerst nach der Variablen `x` differenziert, anschließend zweimal nach der Variablen `y`.

Wenn eine partielle Ableitung im Rahmen der Definition einer weiteren Funktion verwendet werden soll, so geschieht dies wie bei

`g(z):=z+pderive(vars(x,y), y*sin(x*y), y, y, x)(z, 2).`

Die einzelnen Bestandteile bedeuten:

- Mit `vars(x, y)` bedeutet, dass `x` die erste Variable und `y` die zweite Variable ist. Der Ableitung liegen diese beiden Variablen zugrunde.
- Der Term `y*sin(x*y)` ist der Funktionsterm der abzuleitenden Funktion.
- Differenziert wird zuerst zweimal nach `y`, dann einmal nach `x`.

- In der entstehenden Funktion wie die (erste) Variable `x` durch die Variable der Funktion `g(z)` substituiert, die (zweite) Variable `y` durch die Zahl `2`.

3.4 Affine Objekte

Überblick

Die folgende Tabelle gibt einen Überblick über die affinen Objekte.
Anschließend sind sie in der gleichen Reihenfolge ausführlich beschrieben.

Objekt	Eigenschaften
punkt(xf,yf)	Punkt mit den Koordeinatenfunktionen xf und yf. Die beiden Koordinatenfunktionen xf und yf sind nullstellige Funktionen. Methoden: .x : x-Koordinate (nullstellige Funktion) .y : y-Koordinate (nullstellige Funktion) .r : Ortsvektor (zweidimensionaler Vektor) Interaktion: Der Punkt kann mit der Maus bewegt werden. Dadurch werden die Koordinaten des Punkts verändert.
marker(xf,yf)	Markierung mit den Koordinatenfunktionen xf und yf. Anders als beim Punkt wird der Name nicht angezeigt, eine Markierung kann nicht mit der Maus bewegt werden.
strecke(p1,p2)	Verbindungsstrecke der Punkte p1 und p2. Methoden: .begin : Startpunkt der Strecke .ende : Endpunkt der Strecke
gerade(p,v)	Gerade durch den Punkt p mit dem Richtungsvektor v. v ist eine nullstellige Funktion, die als Ergebnis einen zweidimensionalen Vektor liefert. Methoden: .start : Punkt p auf der Gerade .richtung : Richtungsvektor v der Gerade
kreis(p,r)	Kreis mit Mittelpunkt p und Radius r. r ist eine nullstellige Funktion, die als Ergebnis eine Zahl liefert. Methoden: .mitte : Mittelpunkt .radius : Radius .flaeche : Flächeninhalt
winkel(ps, v1, v2, r)	Winkel mit Scheitelpunkt ps und den beiden Richtungen v1 und v2. r legt den Radius fest, mit dem der Winkel gezeichnet wird. Methoden: .scheitelpunkt : Scheitelpunkt des Winkels
kurve(t, s, e, fx(t), fy(t))	Parametrisierte Kurve mit Kurvenparameter t. s und e sind zwei nullstellige Funktionen, sie liefern Start- und Endwert für das Parameterintervall. fx(t) und fy(t) sind die Koordinatenfunktionen für die x- bzw. y-Koordinate. Methoden: .min : Untere Grenze des Parameterintervalls .max : Obere Grenze des Parameterintervalls .x : x-Koordinatenfunktion (einstellige Funktion) .y : y-Koordinatenfunktion (einstellige Funktion)

Objekt	Eigenschaften
bezier(p1,p2,p3,p4)	Kubische Bezier-Kurve mit den Steuerpunkten p1, p2, p3, p4. Startet bei p1 mit Richtung auf p2 zu, endet bei p4, kommt dort mit der Richtung von p3 her an. Methoden: .p1 : Startpunkt .p2 : Steuerpunkt für Startrichtung .p3 : Steuerpunkt für Endrichtung .p4: Endpunkt
kurvenzug({e1, e2, ... })	Kurvenzug, der entsteht, wenn die Elemente e1, e2, ... durch Strecken verbunden werden. Die Elemente e1, e2, ... können sein: Punkte, Strecken, parametrisierte Kurven oder Bezier-Kurven.
flaeche({e1, e2, ... })	Fläche, deren Begrenzung durch eine Liste {e1, e2, ...} von Elementen gegeben ist. Die Elemente e1, e2, ... können sein: Punkte, Strecken, parametrisierte Kurven oder Bezier-Kurven sein.
pfeil(p,v)	Repräsentant des Vektors v mit Aufpunkt p. v ist dabei ein zweidimensionaler Vektor bzw. eine Funktion, die als Ergebnis einen zweidimensionalen Vektor liefert. Methoden: .start : Startpunkt des Pfeils .richtung : Richtungsvektor des Pfeils Interaktion: Die Pfeilspitze kann mit der Maus bewegt werden. Dadurch werden die Koordinaten des Vektors v verändert.

Maus-Position als vordefinierter Punkt

Die aktuelle Maus-Position ist als Punkt mit dem Namen `mouse` vordefiniert.

Durch Auswahl des Elements "Mauskoordinaten anzeigen" im Menü der Steuerleiste kann eine Detailsicht erzeugt werden, die immer die aktuellen Koordinaten der Maus anzeigt.

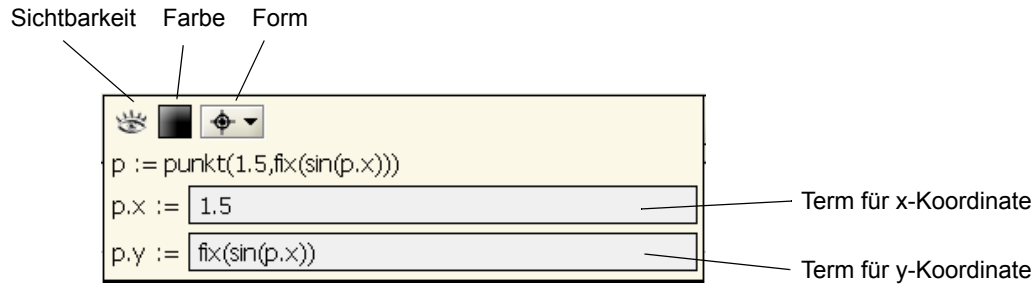
3.4.1 Punkt

Punkt definieren

Mit `p1:=punkt(1,1)` wird ein Punkt mit den Koordinaten `(1,1)` erzeugt.

Der Punkt kann mit der Maus bewegt werden, dadurch ändern sich seine Koordinaten.

Detailsicht



Punkt bewegen, Zwangsbedingungen

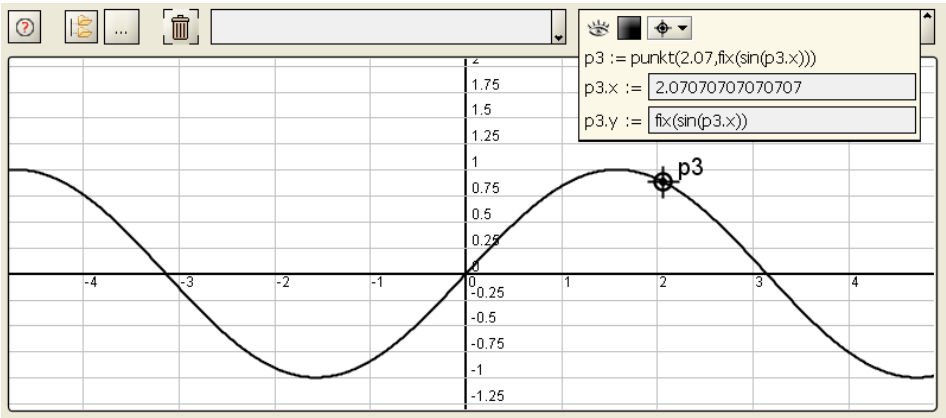
Eine Punktkoordinate mit Zwangsbedingung

Man legt für eine Punktkoordinate eine Zwangsbedingung fest, indem man die entsprechende Koordinatenfunktion in die Funktion `fix()` einschließt.

Im einfachsten Fall kann man damit z.B. beim Erzeugen eines Punktes `p2` festlegen, dass sich seine x-Koordinate nicht ändern darf. Dies erreicht man durch `p2:=punkt(fix(2),2)` . Der Punkt hat nach der Definition die Koordinaten `(2,2)`, er kann aber mit der Maus nur noch in y-Richtung bewegt werden, seine x-Koordinaten wird beim Wert `2` festgehalten.

Man kann mit einer Zwangsbedingung auch festlegen, dass eine Koordinate über eine bestimmte Funktion von der anderen Koordinate abhängt. Will man z.B. einen Punkt `p3` erzeugen, der sich nur auf dem Graphen der sin-Funktion bewegen läßt, so geht man vor wie folgt:

Mit `p3:=punkt(1,1)` definiert man zunächst den gewünschten Punkt, anschließend definiert man mit `p3.y:=fix(sin(p3.x))` seine Koordinatenfunktion für die y-Koordinate neu.



Beide Punktkoordinaten mit Zwangsbedingung

Mit `p4:=punkt(fix(2),fix(3))` erzeugt man einen Punkt, der sich mit der Maus nicht bewegen läßt, da beide Koordinaten gebunden sind.

Hat man einen Parameter `a:=0`, so erzeugt man mit `p5:=punkt(fix(cos(a)),fix(sin(a)))` einen Punkt, der sich zwar mit der Maus nicht bewegen läßt, der sich aber mit Veränderung des Wertes von `a` auf dem Einheitskreis bewegt.

Zwangsbedingung aufheben

Eine Zwangsbedingung wird aufgehoben, indem man die entsprechende Koordinatenfunktion neu definiert. Durch `p3.y:=3` wir der oben definierte Punkt `p3` bei der y-Koordinate `3` positioniert und ist dann mit der Maus auch in y-Richtung frei bewegbar.

Methoden eines Punktes

Zu einem Punkt `p1` liefert

- `p1.x` die (nullstellige) Koordinatenfunktion der **x-Koordinate**
- `p1.y` die (nullstellige) Koordinatenfunktion der **y-Koordinate** und
- `p1.r` den zweidimensionalen **Ortsvektor** des Punktes.

Den Abstand des Punktes `p1` vom Ursprung kann man also mit dem Term `sqrt(p1.x^2+p1.y^2)` berechen, einfacher allerdings als Betrag des Ortsvektors mit dem Term `|p1.r|`.

Methoden, die Punkte als Ergebnis liefern

Folgende Methoden anderer Objekte liefern insbesondere als Ergebnis einen Punkt:

- `.mitte` liefert wie bei `kreis1.mitte` den Mittelpunkt eines Kreises.
- `.start` liefert wie bei `strecke1.start` den "Start"-Punkt einer Strecke,
- `.ende` liefert wie bei `strecke1.ende` den "End"-Punkt einer Strecke.
- `.punkt` liefert wie bei `gerade1.punkt` einen Punkt einer Gerade.

Weitere Objekte, die über Methoden Punkte als Ergebnis liefern sind: Ein Winkel liefert seinen Scheitelpunkt, eine Bezier-Kurve liefert ihre 4 Steuerpunkte.

Die Maus als vordefinierter Punkt

Mit dem Bezeichner `mouse` wird der Punkt angesprochen, an dem sich der Mauszeiger aktuell befindet. So liefert `ma:=|mouse.r|` den Abstand des Mauszeigers vom Ursprung,

`so:=strecke(mouse, punkt(0,0))` die Verbindungsstrecken von Maus-Position und Ursprung.

3.4.2 Markierung

Die Markierung ist eine optisch und funktional reduzierte Variante des Punktes.

Wenn eine bestimmte Stelle der Zeichenfläche mit einem Text beschrieben werden soll, dann steht hierfür das Text-Element (siehe 3.5.5) zur Verfügung.

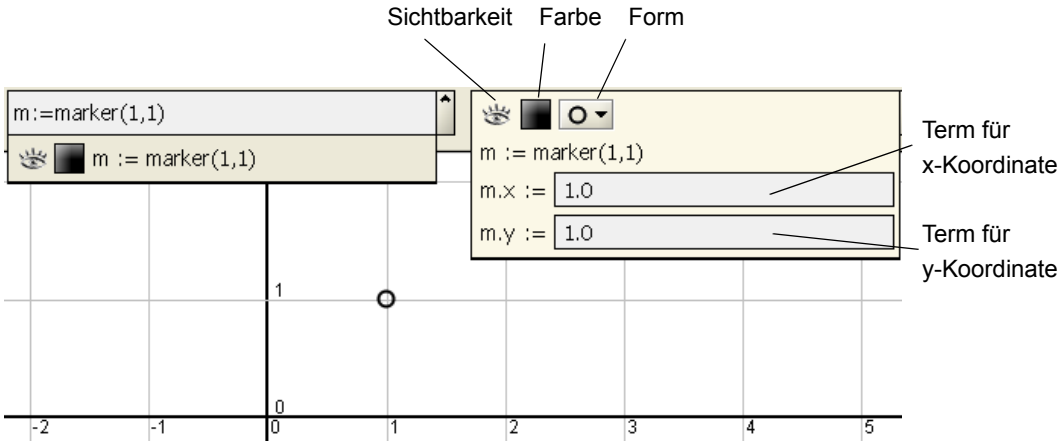
Markierung definieren

Mit `m1:=marker(1,1)` wird eine Markierung mit den Koordinaten `(1,1)` erzeugt.

Eine Markierung kann nicht mit der Maus bewegt werden.

Der Name einer Markierung wird auf der Zeichenfläche nicht angezeigt.

Detailsicht



3.4.3 Strecke

Strecke definieren

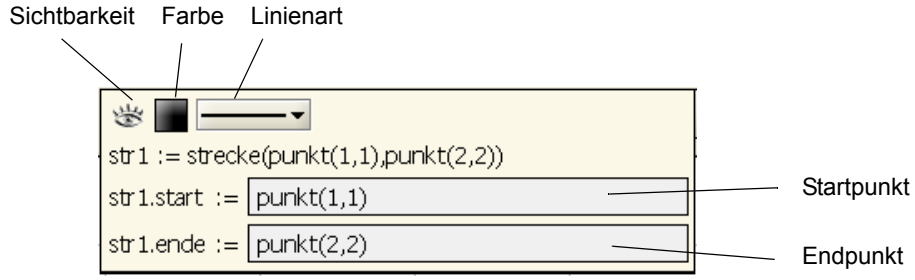
Eine Strecke wird durch Angabe zweier Punkte definiert:
`str1:=strecke(punkt(1,1),punkt(2,2))` .

Wenn vorher schon Punkte `p1` und `p2` definiert wurden, dann können diese mit ihrem Namen angebeben werden: `str2:=strecke(p1,p2)`.

Für zwei Kreise `k1` und `k2` erhält man mit `str3:=strecke(k1.mitte, k2.mitte)` die Strecke, die die Mittelpunkte der beiden Kreise verbindet.

Definiert man mit `pl:=<punkt>liste(i,10,punkt(i,i))` eine Liste von Punkten, zusätzlich einen Parameter `a:=1`, dann ist `str4:=strecke(pl[a],pl[a+1])` zunächst die Verbindungsstrecke des ersten und zweiten Punktes aus der Liste `pl`. Wenn man jetzt den Wert von `a` durch `a:=2` verändert, dann ist `str4` die Verbindungstrecke des zweiten und dritten Punktes der Liste `pl`.

Detailsicht

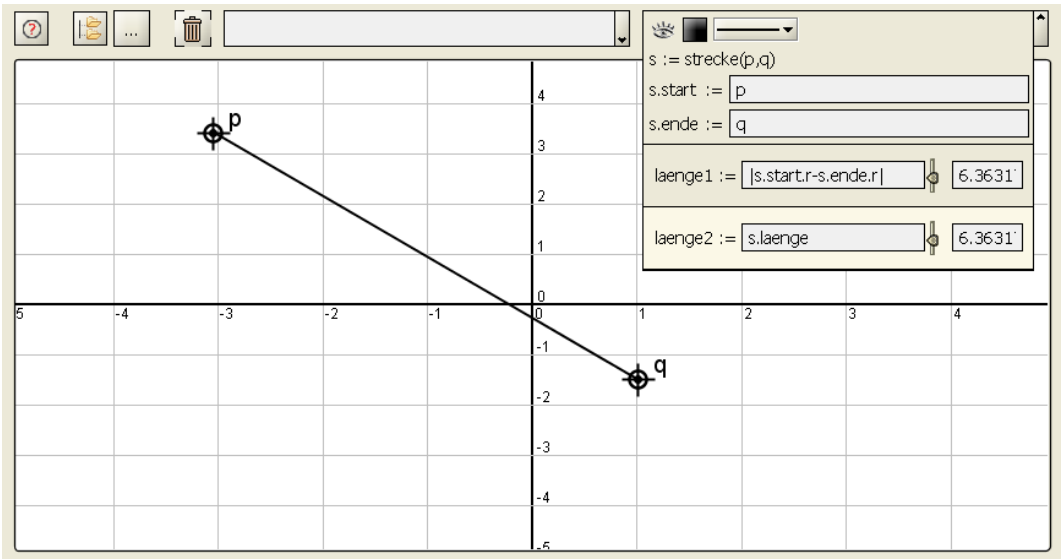


Methoden einer Strecke

Zu einer Strecke `str` liefert

`str.start` den Startpunkt der Strecke und
`str.ende` den Endpunkt der Strecke.
`str.laenge` die Länge der Strecke.

Beispiel: Zwei Möglichkeiten, die Länge einer Strecke zu ermitteln:



3.4.4 Gerade

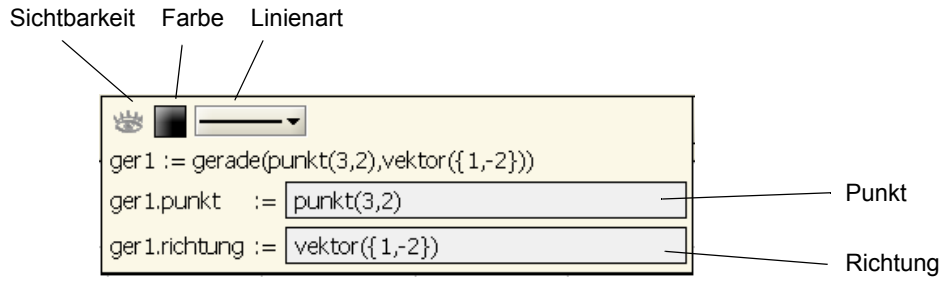
Gerade definieren

Eine Gerade wird durch Angabe eines Punktes und eines zweidimensionalen Richtungsvektors definiert: `ger1:=gerade(punkt(3,2), vektor({1,-2}))`.

An der ersten Stelle kann auch ein vorher schon definierter Punkt angegeben werden, an der zweiten Stelle ein Term, der als Ergebnis einen zweidimensionalen Vektor liefert.

Definiert man zuerst `p:= punkt(-2, 2)` und `k:=kreis(punkt(1,2), 2)`, dann erhält man mit `ger2:=gerade(p, k.mitte.r-p.r)` die Gerade, die `p` und den Mittelpunkt des Kreises `k` verbindet.

Detailsicht



Methoden einer Gerade

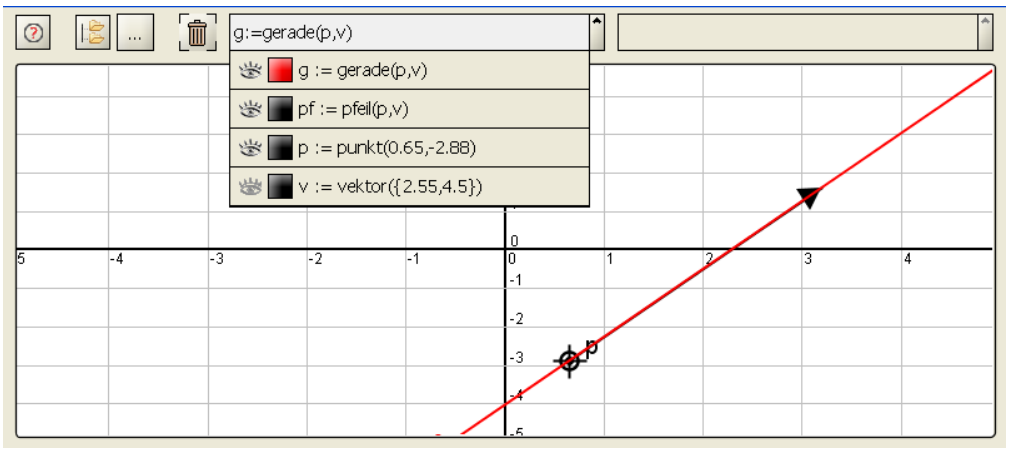
Zu einer Gerade `g` erhält man mit

`g.punkt` den Startpunkt der Gerade und mit

`g.richtung` den Richtungsvektor der Gerade.

Mit `g2:=gerade(g.punkt, vektor({g.richtung.y, -g.richtung.x}))` erhält man eine Gerade `g2`, die `g` in deren Startpunkt immer senkrecht schneidet.

Beispiel: Gerade `g` über Punkt `p` und Richtungspfeil `pf` manipulieren



3.4.5 Kreis

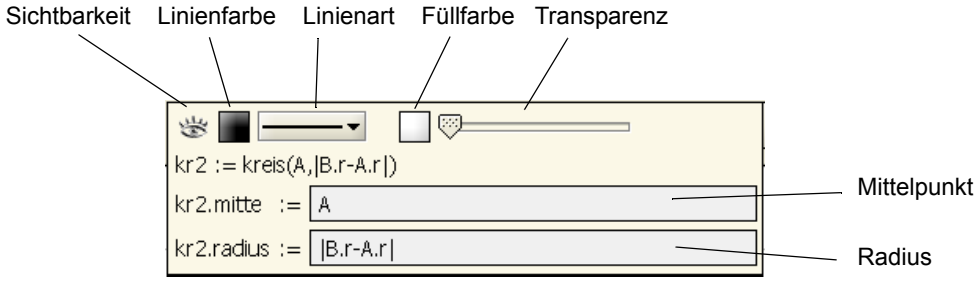
Kreis definieren

Ein Kreis wird durch Angabe von Mittelpunkt und Radius definiert:

`kr1:=kreis(punkt(3,5), 3).`

An der ersten Stelle kann ein vorher definierter Punkt verwendet werden, an der zweiten Stelle ein Funktionsterm, der eine Zahl liefert. Sind `A` und `B` Punkte, dann entsteht mit `kr2:=kreis(A, |B.r-A.r|)` ein Kreis mit dem Mittelpunkt `A`. Der Term für den Radius ist so eingerichtet, dass die Kreislinie stets den Punkt `B` enthält.

Detailsicht

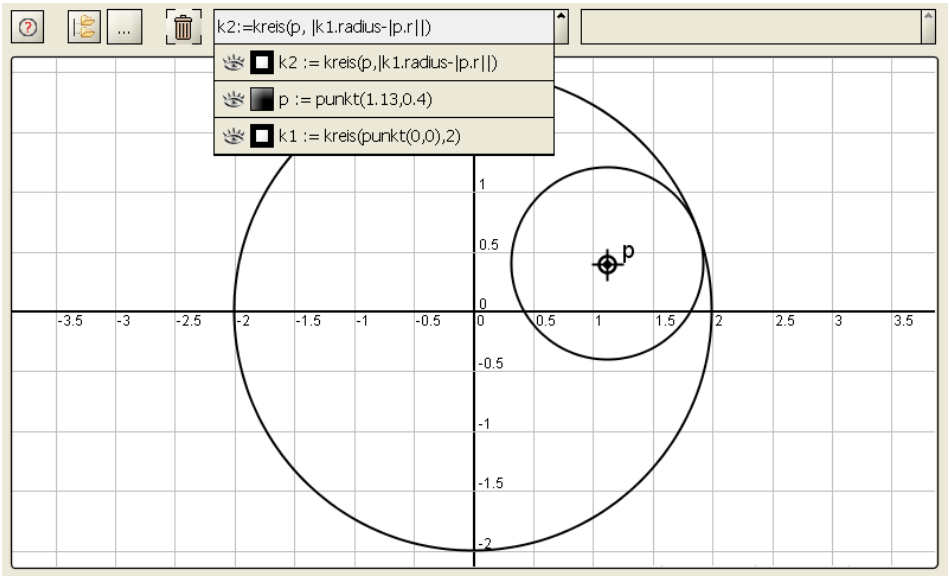


Methoden eines Kreises

Zu einem Kreis `k` liefert

`k.mitte`	den Mittelpunkt,
`k.radius`	den Radius und
`k.flaeche`	den Flächeninhalt.

Beispiel: Der Kreis `k2` berührt stets den Kreis `k1`



Hinweis

Auf der Zeichenfläche erscheinen Kreise nur dann als Kreise, wenn für beide Achsen der gleiche Maßstab gewählt ist. Dies erreicht man am schnellsten durch die Auswahl `zoom 1:1` (Taste: `1`) oder durch die Festlegung `ständiger 1:1 Zoom` im Menü der Zeichenfläche.

3.4.6 Winkel

Winkel definieren

Ein Winkel wird definiert wie `w1:=winkel(punkt(1,1), e2x, vektor({1,1}), 1)`, die Argumente sind der Reihe nach:

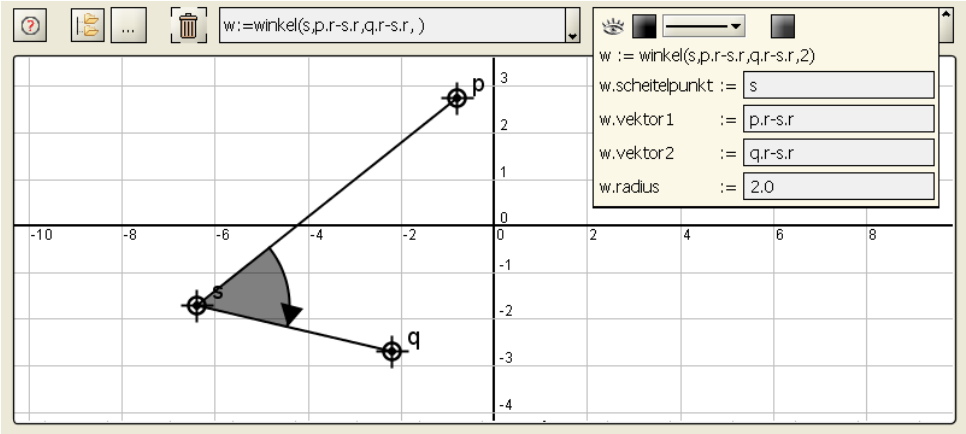
- Scheitelpunkt des Winkels
- Richtungsvektor des ersten Schenkels
- Richtungsvektor des zweiten Schenkels
- Radius, mit dem der Winkel gezeichnet wird.

Methoden eines Winkels

Zu einem Winkel `w` liefert

`w.scheitelpunkt`	den Scheitelpunkt,
`w.vektor1`	den ersten Richtungsvektor,
`w.vektor2`	den zweiten Richtungsvektor und
`w.radius`	den Radius.

Beispiel: Winkel `w` über Punkte `s`, `p` und `q` manipulieren



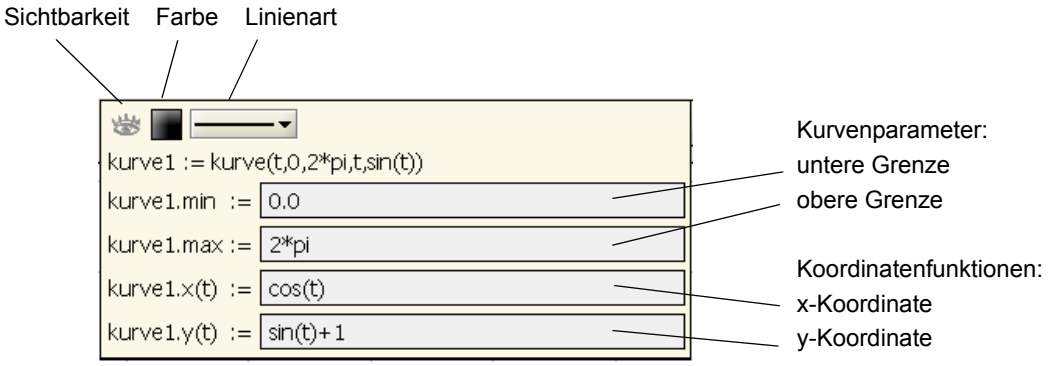
3.4.7 Parametrisierte Kurve

Parametrisierte Kurve definieren

Eine Kurve wird durch Angabe von 5 Parametern wie bei
`kurve1:=kurve(t,0,2*pi, cos(t), sin(t)+1)` definiert.

Dabei ist der erste Parameter (hier: `t`, allgemein: ein Bezeichner) der sogenannte Kurvenparameter, er wird oft als Zeit interpretiert. Die beiden folgenden Parameter legen das Intervall fest, das der Kurvenparameter durchläuf. Hier legen die beiden Parameter `0` und `2*pi` das Intervall $[0, 2\pi]$ fest. Zuletzt folgen die beiden Koordinatenfunktionen, zuerst für die x-Koordinate, dann für die y-Koordinate. Sie sind einstellige Funktionen, ihre Funktionsvariable ist der vereinbarte Kurvenparameter. Hier sind als Koordinatenfunktoinen die beiden Funktionen `cos(t)` und `sin(t)+1` vereinbart.
`kurve1` definiert die Kreislinie eines Kreises mit Radius 1 und Mittelpunkt bei (0,1).

Detailsicht

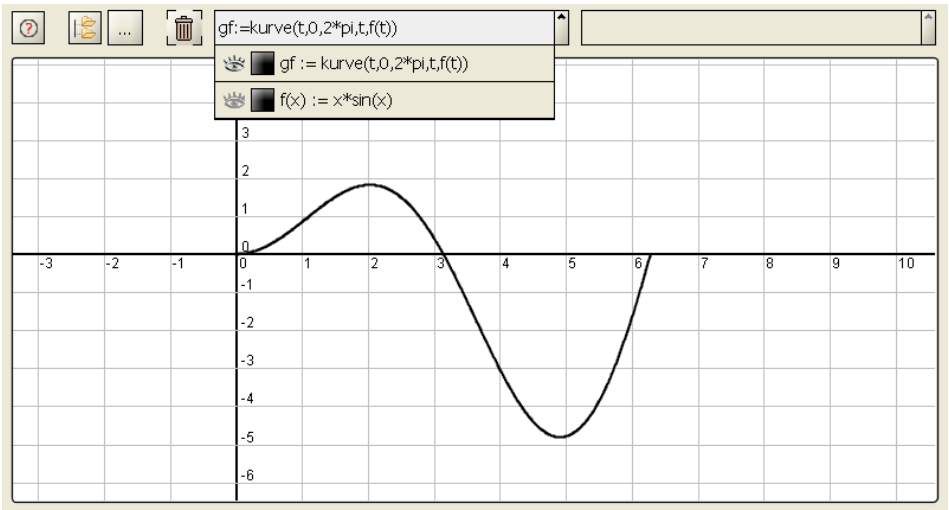


Methoden einer Kurve

Zu einer Kurve `k` liefert

`k.min`	die untere Grenze des Parameterintervalls,
`k.max`	die obere Grenze des Parameterintervalls,
`k.x`	die Koordinatenfunktion der x-Koordinate (einstellige Funktion) und
`k.y`	die Koordinatenfunktion der y-Koordinate (einstellige Funktion).

Beispiel: Ausschnitt aus einem Funktionsgraphen als parametrisierte Kurve



3.4.8 Bezier-Kurve

Eine Bezier-Kurve ist ein Kurvenstück, dessen Verlauf durch vier Steuerpunkte festgelegt wird.

Bezier-Kurve definieren

Definiert man zunächst vier Punkte `pkt1:=punkt(1,2)`, `pkt2:=punkt(2,0)`, `pkt3:=punkt(3,3)` und `pkt4:=punkt(4,2)`, dann erhält man mit `bez1:=bezier(pkt1,pkt2,pkt3,pkt4)` die Bezier-Kurve, die beim Punkt `pk1` startet, sich zunächst auf den Punkt `pk2` zubewegt, die beim Punkt `pk4` endet, dabei kommt sie aus der Richtung, die durch `pk3` festgelegt ist.

Methoden einer Bezier-Kurve

Zu einer Bezier-Kurve `bez` liefern `bez.p1`, `bez.p2`, `bez.p3` und `bez.p4` der Reihe nach die vier Steuerpunkte der Kurve.

Beispiel

Bei vielen Zeichenwerkzeugen werden Kurven stückweise aus Bezier-Kurven aufgebaut. Zum Steuern des Verlaufs hat man dort meist

- den Startpunkt `st`,
- einen Pfeil `st_pf`, mit dem man einstellen kann, in welcher Richtung die Kurve den Startpunkt verläßt,
- den Endpunkt `ed` und
- einen Pfeil `ed_pf`, mit dem man einstellen kann, in welcher Richtung die Kurve am Endpunkt ankommt.

Eine vergleichbare Situation erhalten wir wie folgt:

Wir definieren zwei Punkte `st:=punkt(1,1)` und `ed:=punkt(3,4)`, die beiden Vektoren `st_v:=vektor({1,1})` und `ed_v:=vektor({-1,2})` und Pfeile zur graphischen Darstellung der Vektoren `st_pf:=pfeil(start, start_v)` und `ed_pf:=pfeil(end, end_v)`.

Dann können wir die Kurve

```
bez2:=bezier(st,punkt(st.x+st_v.x,ed.x+ed_v.x),
                punkt(ed.x-ed_v.x, ed.y-ed_v.y),ed)
```

wie gewohnt mit der Maus manipulieren (wenn wir alle Elemente zeichnen).

3.4.9 Kurvenzug

Ein Kurvenzug wird definiert durch eine Liste von Punkten, Strecken und Kurven (parametrisierte Kurven und Bezier-Kurven). Beim Zeichnen wird das erste Element der Liste gezeichnet, dessen Endpunkt mit dem Startpunkt des zweiten Elements durch eine Strecke verbunden, anschließend wird das zweite Element gezeichnet usw. . Ein Punkt ist dabei Start- und Endpunkt zugleich.

Kurvenzug definieren

Zunächst ein paar Punkte definieren:
`p:=<punkt>liste(i,20,punkt(cos(pi/10*i), sin(pi/10*i)))`. Dann erzeugt

```
kvz1:=kurvenzug({  bezier(p[1],p[2],p[3],p[4]),
                   strecke(p[7],p[8]),
                   p[15]  })
```

einen offenen Kurvenzug. Wenn wir den Startpunkt `p1[1]` der Bezier-Kurve als letztes Element in der Liste nochmals aufführen, entsteht ein geschlossener Kurvenzug:

```
kvz1:=kurvenzug({  bezier(p[1],p[2],p[3],p[4]),
                   strecke(p[7],p[8]),
                   p[15],
                   p[1]  }).
```

Wenn man mit `b:=bezier(p[1],p[2],p[3],p[4])` die Bezier-Kurve schon definiert hat, dann erhält man zu dieser mit `b.p1` ihren Startpunkt und könnte also mit `kvz2:=kurvenzug({b,b.p1})` einen geschlossenen Kurvenzug erhalten.

3.4.10 Fläche

Die Regeln bei der Definition einer Fläche sind zunächst die gleichen wie die bei einem Kurvenzug.

Zusätzlich wird der Endpunkt des letzten Elements mit dem Startpunkt des ersten Elements durch eine Strecke verbunden (falls diese Punkte nicht identisch sind). Beim Zeichnen wird die so eingeschlossene Fläche mit der gewählten Farbe ausgefüllt.

Fläche definieren

```
dreieck:=flaeche({ punkt(0,0), punkt(4,-1), punkt(3,7) })
```

definiert ein Dreieck.

Ansonsten gelten die gleichen Regeln wie bei einem Kurvenzug.

3.4.11 Pfeil

Ein Pfeil ist ein Repräsentant eines zweidimensionalen Vektors `v`. Neben diesem Vektor ist zu seiner Definition noch ein Aufpunkt erforderlich.

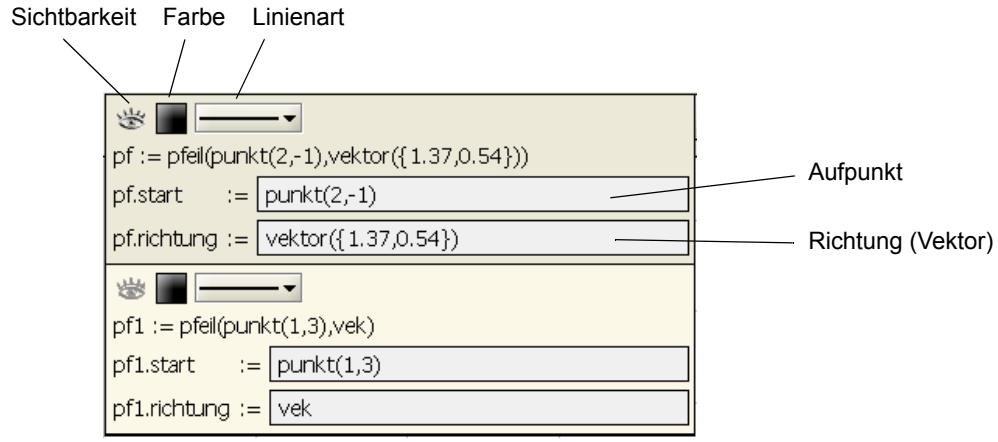
Pfeil definieren

Ein Pfeil wird definiert wie:

`pf:=pfeil(punkt(2,-1),vektor({1.5,1})).`

Hat man einen Vektor `vek:=vektor({1,-3})` definiert, so erhält man mit `pf1:=pfeil(punkt(1,1),vek)` und `pf2:=pfeil(punkt(1,3),vek)` zwei Repräsentanten des einen Vektors `vek`, also zwei parallelgleiche Pfeile.

Detailsicht



Pfeil mit der Maus verändern

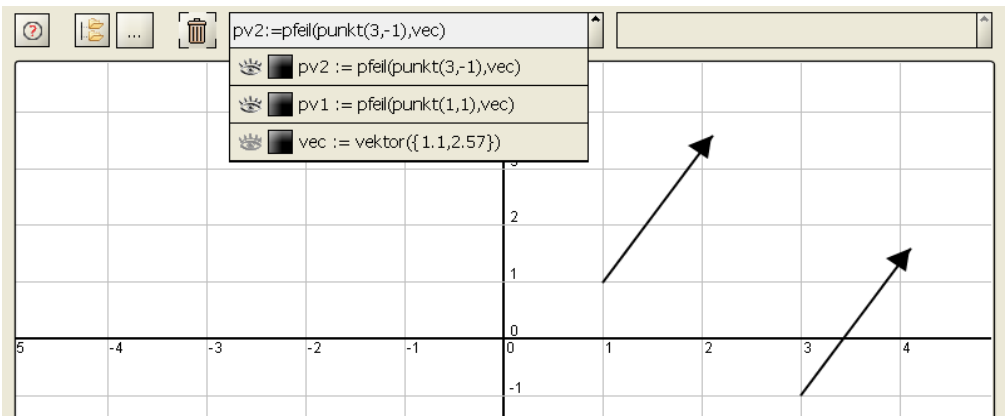
Wenn ein Pfeil `pf1` auf der Zeichenebene dargestellt wird, dann kann man seine Spitze mit der Maus bewegen und so die Komponenten des zugehörigen Vektors `vek` verändern.
Alle anderen Repräsentanten des Vektors `vek` verändern sich dann parallelgleich.

Methoden eines Pfeils

Zu einem Pfeil `pf1` liefert

`pf1.start`	den Aufpunkt und
`pf1.richtung`	die Richtung des Pfeils, also den Vektor, den der Pfeil repräsentiert.

Beispiel: Vektor mit zwei Repräsentanten



3.5 Diagramme und Textelemente

Diagramme

Bezeichner	Bedeutung
punkte(vx,vy,radius)	Diskretes Punkt-Diagramm. Der Vektor vx enthält die x-Werte für die Position der Punkte, der Vektor vy die y-Werte für die Position der Punkte. Die Funktion radius legt fest, wie gross die einzelnen Punkte markiert werden.
balken(vx,vy,breite)	Balkendiagramm. Der Vektor vx enthält die x-Werte für die Position der Balken, der Vektor vy die y-Werte für die Höhe der Balken. Die Funktion breite regelt die Breite der Balken.
karte(f,konfvek)	Farbliche Darstellung der zweistelligen Funktion mit dem Namen f. Der dargestellte Wertebereich und die Lage von Höhenlinien wird durch den Vektor konfvek festgelegt.
feld(f, p, distx, disty, cutoff) feld(f, p, distx, disty, cutoff, layout)	Darstellung der zweisteligen vektorwertigen Funktion mit dem Namen f als Vektorfeld. Der Punkt p und die Abstände distx und disty legen ein Puntgitter fest. An den Punkten dieses Gitters wird der Wert der Funktion f als Pfeil dargstellt, sofern der Betrag des Vektors nicht größer als der Wert von cutoff ist. Bei Bedarf kann die Art der Darstellung durch verschiedene Werte von layout angepasst werden.

Text-Elemente

Bezeichner	Bedeutung
text(p1, "meintext", POSTIT)	Text-Element am Verankerungspunkt p1 mit Text `meintext`, Aussehen je nach Layout-Konstante: Notizzettel: `POSTIT`. **Bubble (**Sprechblase**):** Sie wird durch die Himmensrichtung festgelegt, in der sie von Verankerungspunkt aus erscheinen: `B_SE` (SouthEast), `B_SW`, `B_NW`, `B_NE`. **Arrow** (Pfeil): Er wird festgelegt durch die Himmelsrichtung, in die er zeigt: `A_N`, `A_E`, `A_S`, `A_W`. Pfeile für die **K**oordinatenachsen: `K_N`, `K_E`, `K_S`, `K_W`. Markierung für Koordinatenachsen in den Himmelsrichtungen: `C_N`, `C_E`, `C_S`, `C_W`.

3.5.1 Diskretes Punkt-Diagramm

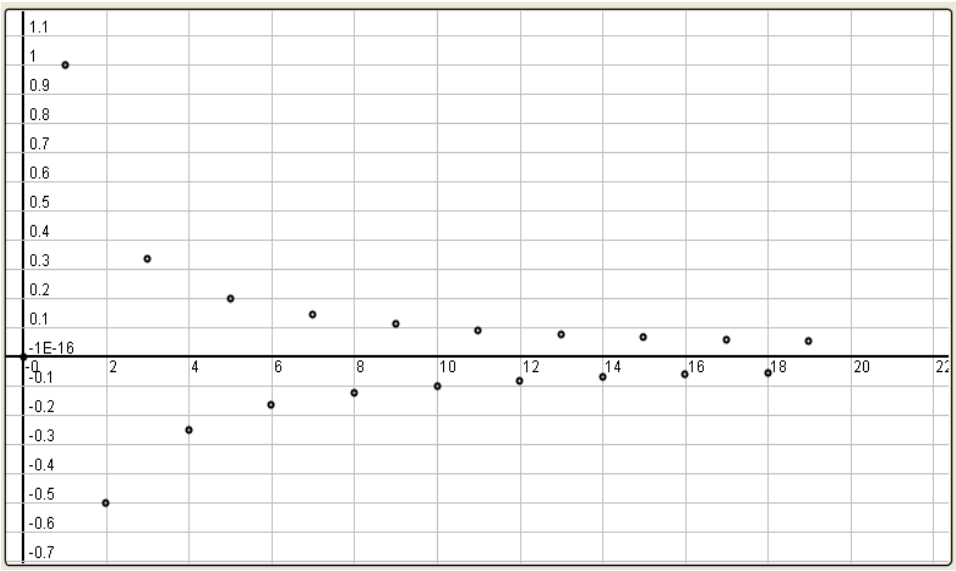
Punkt-Diagramm definieren

Ein Punktdiagramm wird wie `disk1:=punkte(vx,vy,radius)` definiert durch die Angabe der Namen von zwei Vektoren: Die Elemente des Vektors `xv` geben die x-Positionen der Punkte an, die Elemente des Vektors `yv` bestimmen die entsprechenden y-Positionen. Jeder Punkt wird durch einen Kreis markiert. Die Funktion `radius` legt den Radius dieser Kreise in Bildpunkten (Pixel) fest.

Beispiel

Koeffizienten des Taylor-Polynoms der Funktion ln(x) zum Entwicklungspunkt $c = 1$:

Definiert man den Vektor `nv:=vektor(i,20,i-1)` mit den Komponenten 0, 1, ..., 19 und den Vektor `cv:=vektor(i,20,derive(vars(x), ln(x), i-1)(1)/(i-1)!)`, dann zeigt die durch `cg := punkte( nv,cv,2 )` erzeugte Graphik die Abhängigkeit der Koeffizienten des Taylor-Polynoms vom Index `i`:



3.5.2 Balkendiagramm

Balkendiagramm definieren

Ein Balkendiagramm wird wie `balk1:=balken(vx,vy,breite)` definiert durch die Angabe von zwei Vektoren und einer Breite. Die Elemente des Vektors `xv` geben die Positionen der Balken an, die Elemente des Vektors `yv` bestimmen die Höhe der Balken.

Die Breite der Balken wird über die nullstellige Funktion `breite` festgelegt:

- Wenn `breite` einen positiven Wert liefert, werden die Balken mit dieser Breite gezeichnet.
- Wenn `breite` den Wert 0 liefert, erscheinen die Balken als Striche.
- Wenn `breite` einen negativen Wert liefert, dann wird die Breite der Balken automatisch so eingestellt, dass sich benachbarte Balken nicht überlappen.

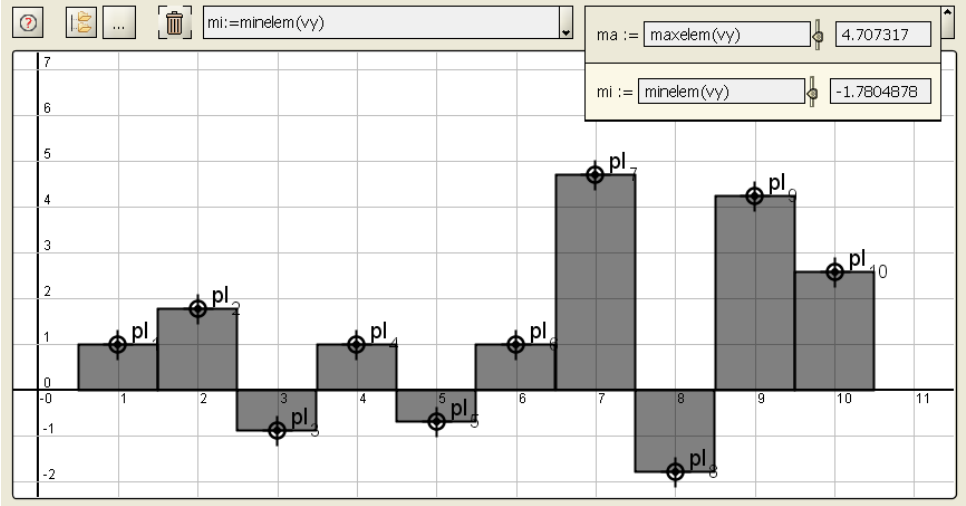
Beispiel

Ein Balkendiagramm, bei dem man die Höhe der Balken interaktiv mit der Maus einstellen kann, erzeugt man wie folgt:
Das Einstellen erfolgt über (Steuer-)Punkte, eine entsprechende Anzahl wird in einer Liste erzeugt: `pl:=<punkt>liste(i,10,punkt(fix(i), 2))`. Die x-Koordinaten der Punkte sind mit `fix(i)` festgelegt (Zwangsbedingung), die Punkte können mit der Maus nur in y-Richtung bewegt werden. Mit `vx:=vektor(i,10,pl[i].x)` sammeln wir die x-Koordinaten der Punkte, in

`vy:=vektor(i,10,pl[i].y)` die y-Koordinaten.
Jetzt erzeugen wir mit `b:=balken(vx,vy,-1)` das Balkendiagramm.
Die Breite soll dabei automatisch so eingestellt werden, dass sich die Balken nicht überlappen. Deshalb wir als Breite der Wert `-1` eingetragen.

Wenn wir `pl` und `b` zeichnen, können wir die Höhe der Balken über die erzeugten Steuerpunkte verändern.

Zusätzlich wird hier über die nullstelligen Funktionen `ma` und `mi` der maximale und der minimale Wert aus `vy` berechnet und angezeigt.



3.5.3 Karte

Eine Karte `karte(f,konfvek)` stellt die Werte einer zweistelligen Funktion `f(x,y)` farblich dar.

Darin ist `f` der Name der Funktion, die dargestellt werden soll.

Durch die Elemente des Vektors `konfvek` wie `konfvek:=vektor({-2,-1,0,1,2,2})` werden die Details der Darstellung wie folgt festgelegt:

Erste und letzte Komponente:

- Die erste Komponente (hier: `-2`) von `konfvek` legt den kleinsten Funktionswert dar, der farbig dargestellt wird.
- Die letzte Komponente (hier: `2`) legt den größten Funktionswert fest, der farbig dargestellt wird.

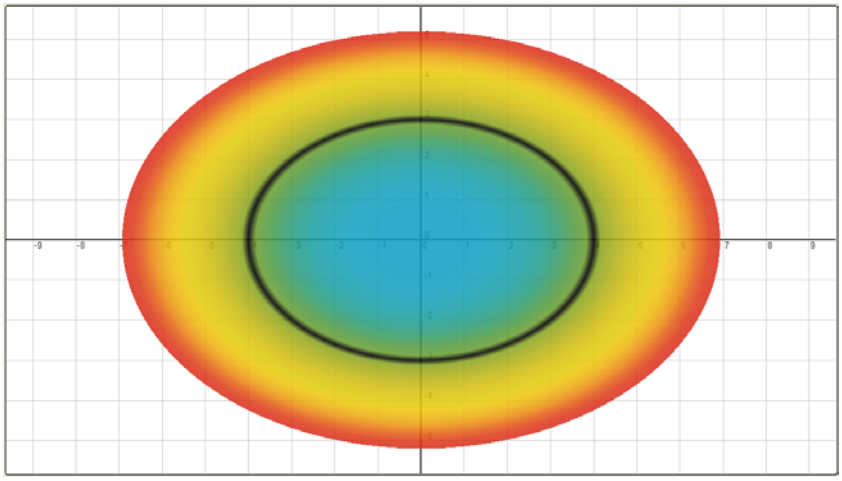
Wenn der Wert der ersten Komponente größer ist als der Wert der letzten Komponente, dann wird keine farbige Darstellung gezeichnet.

Weitere Komponenten:

Weitere Komponenten (hier: `-1,0,1,2`) legen Werte fest, an denen "Höhenlinien" gezeichnet werden: Eine "Höhenlinie" zum Wert `w` entsteht dadurch, dass alle Bildpunkte (y,x) mit $f(x,y) \cong w$ schwarz eingefärbt werden.

Beispiel: Implizite Definition einer Ellipse

Definiert man die zweistellige Funktion `F(x,y):=x^2/16+y^2/9-1` , dann ist durch die Menge der Punkte, deren Koordinaten die Gleichung $F(x,y) = 0$ erfüllen, eine Ellipse mit den Halbachsen `4` auf der x-Achse und `3` auf der y-Achse gegeben.
Die Karte `k:=karte(F, vektor({-2,0,2}))` veranschaulicht die Funktionswerte, insbesondere sind Stellen mit $F(x,y) = 0$ schwarz hervorgehoben.



Beispiel: Potential eines Dipols

Eine Ladung am Punkt mit den Koordinaten `xp` und `yp` erzeugt an einem Punkt mit den Koordinaten `x` und `y` (bis auf Konstanten) ein Potential

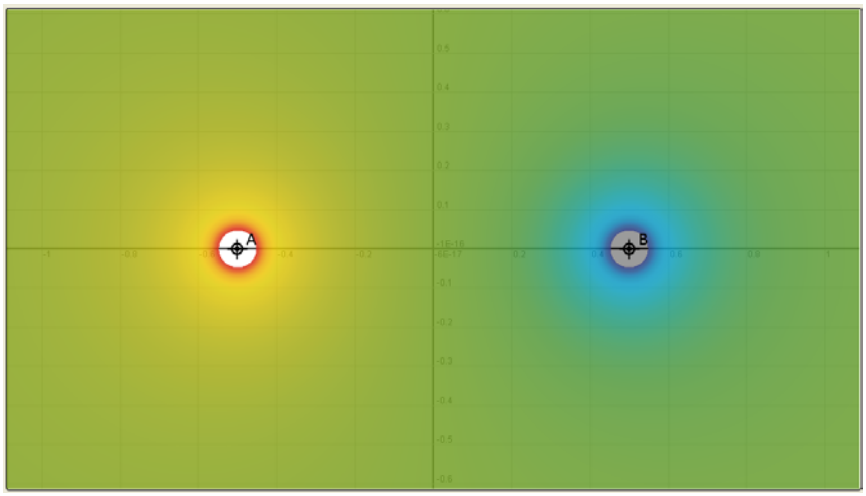
`punktpot(xp,yp,x,y) := 1/(sqrt((xp-x)^2+(yp-y)^2)).`

Hat man am Punkt `A := punkt(-0.5,0)` eine positive Einheitsladung und bei `B := punkt(0.5,0)` eine negative Einheitsladung, so ergibt

`pot(x,y) := punktpot(A.x, A.y, x, y)-punktpot(B.x, B.y, x, y)`

das Gesamtpotential dieser Ladungsanordnung (elektrischer Dipol).

`k := karte(pot,vektor({-20,20}))` zeichnet eine farbige Darstellung der Funktionswerte im Bereich `-20` bis `20`.



3.5.4 Vektorfeld

Vektorfeld definieren

Zu einer zweistelligen Funktion `vek(x,y)`, die als Ergebnis einen zweidimensionalen Vektor liefert, definiert man durch `vekgraph:=feld(vek, aufpunkt, xdist, ydist, cutoff)` einen Graphen. Dieser wird im folgenden als Vektorfeld bezeichnet.

`vek` ist darin der Name der Funktion, die dargestellt werden soll.

Der Punkt `aufpunkt` und die nullstelligen Funktionen `xdist` und `ydist` definieren ein Gitter:

`aufpunkt` ist ein Gitterpunkt und legt so die Lage des Gitters fest, `xdist` bestimmt den Abstand der Gitterpunkte in x-Richtung, `ydist` den Abstand in y-Richtung.

Die nullstellige Funktion `cutoff` legt den größten darzustellenden Betrag fest.

An den Gitterpunkten wird der Wert der Funktion `vek` als Pfeil dargestellt, sofern der Betrag nicht größer als der Wert von `cutoff` ist.

Beispiel:

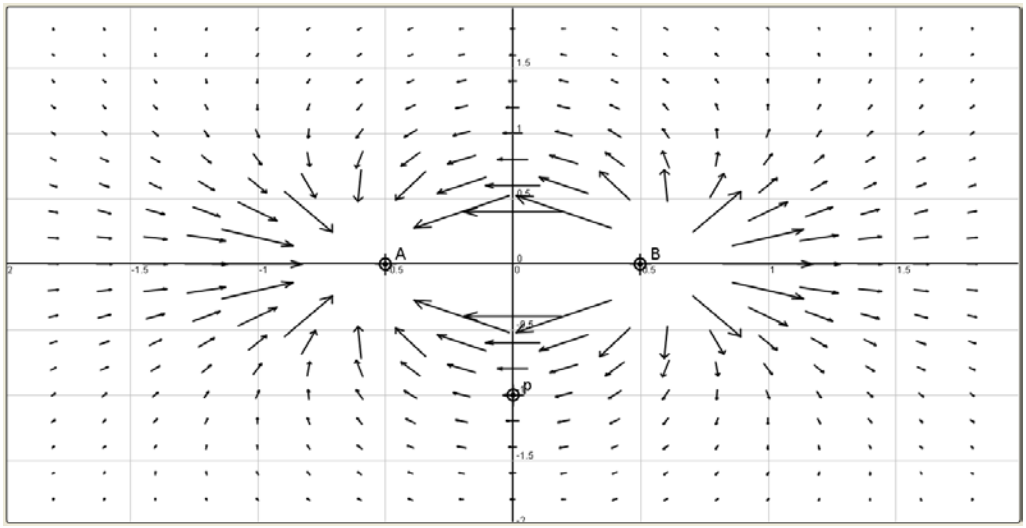
Zu einem Potential `pot(x,y)` (d.h. zu einer zweistelligen skalaren Funktion) erhält man das Kraftfeld als Vektor der partiellen Ableitungen:

```
v(x,y):=vektor({pderive(pot(x,y), x), pderive(pot(x,y),y)})
```

Um die Darstellung dieses Feldes gezielt anpassen zu können, definiert man

- einen Aufpunkt `p:=punkt(0,-1)`,
- einen Parameter, der die Gitterkonstante für die Darstellung festlegt: `d:=0.2` und
- einen Faktor für die Darstellung der Feldvektoren: `s:=0.1 ,`
- damit das darzustellende Feld als `dv(x,y):=s*v(x,y).`

Nach diesen Vorarbeiten zeichnet `g:=feld(dv,p,d,d,0.5)` ein Bild des Kraftfeldes.



Die Vektoren des Feldes werden an den Stellen nicht gezeichnet, an denen ihr Betrag größer als der eingestellte Cuttof `0.5` ist. Die ist nahe bei den Punkten `A` und `B` der Fall.

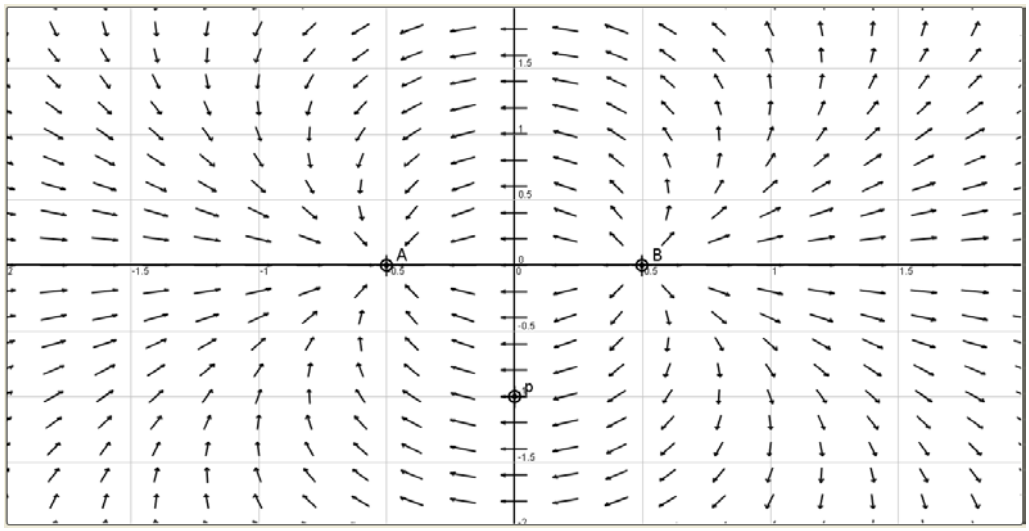
Durch Verschieben des Punktes `p` und Variieren der Parameter `d` und `s` kann man die Darstellung verändern.

Will man nur die Richtung des Kraftfeldes darstellen, so kann man die Vektoren normieren:

```
nv(x,y):=s*v(x,y)/|v(x,y)|.
```

mit `ng:=feld(nv,p,d,d,0.5 )` erhält man jetzt eine übersichtliche Darstellung

der Kraftrichtung.



Punktgitter am Linienraster des Fensters ausrichten

Ein Zeichenfenster `da` liefert mit der Methode `da.xDist` den Abstand der Koordinatenlinien in x-Richtung und mit `da.yDist` den Abstand in y-Richtung.

Definiert man

```
vekgraph1:=feld(vek, punkt(0,0), da.xDist/2, da.yDist/2, cutoff)
```

und zeichnet dieses Vektorfeld im Zeichenfenster `da`, dann ist in beiden Richtungen der Abstand der Gitterpunkte zum Zeichnen des Feldes stets halb so groß wie der Abstand der Koordinatenlinien. Dieser Zusammenhang bleibt auch beim Zoomen erhalten.

Layout festlegen

Bei Bedarf kann die Darstellung wie bei

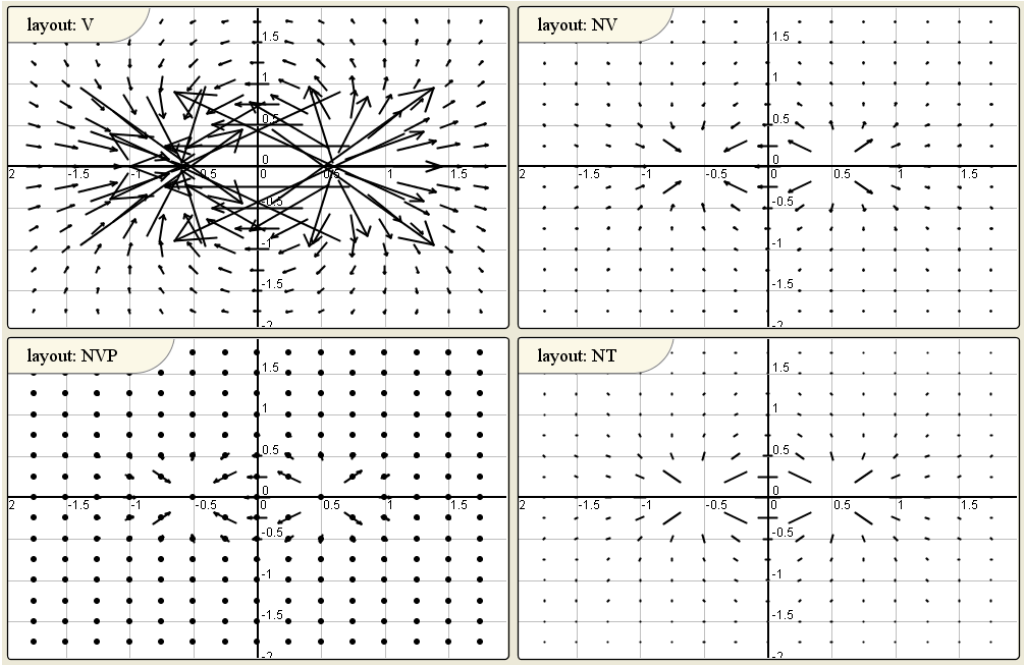
```
nvekgraph:=feld(vek, aufpunkt, xdist, ydist, cutoff, NV)
```

durch eine Layout-Konstante (6. Parameter) beeinflusst werden.

Wenn diese Angabe fehlt, wird der Wert `V` als Voreinstellung ergänzt.

Folgende Layout-Konstanten sind möglich:

`V`	Betragstreue Pfeile (Voreinstellung),
`VP`	mit Darstellung des Aufpunkts.
`NV`	Normierte Pfeile: Die Länge der Pfeile wird an den Abstand der Gitterpunkte angepasst,
`NVP`	mit Darstellung des Aufpunkts.
`T`	Betragstreue Tangenten-Strecken (ohne Pfeilspitze),
`TP`	mit Darstellung des Aufpunkts.
`NT`	Normierte Tangenten-Strecken (ohne Pfeilspitze): Die Länge der Strecken wird an den Abstand der Gitterpunkte angepasst,
`NTP`	mit Darstellung des Aufpunkts.


layout: V
layout: NV
layout: NVP
layout: NT

3.5.5 Text-Element

Text-Element definieren

Man definiert ein Text-Element wie bei `text1:=text(punkt(1,2), "Wichtig!", B_SW)` durch Angabe eines Verankerungspunktes (hier: `punkt(1,2)`), des gewünschten Texts (in "" eingeschlossen) und einer Layout-Konstante (hier: `B_SW`).

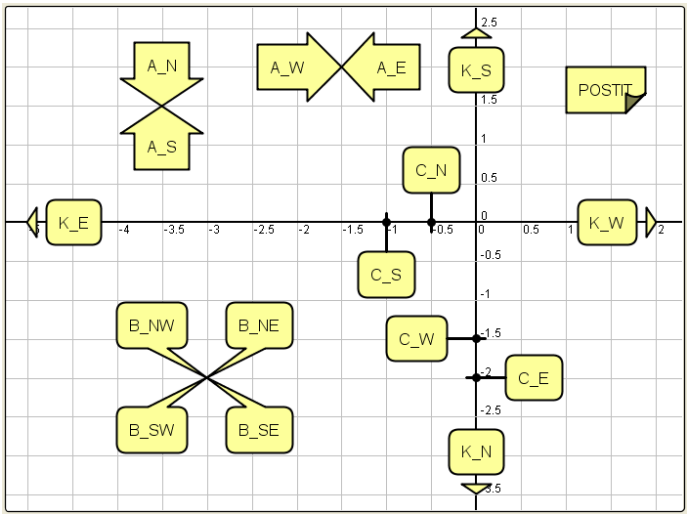
Beispiel

Ein Textelement, das immer der Maus folgt, kann man erstellen mit

```
maustext:=text(mouse, "Maus", B_NE) .
```

Layout

Folgende Graphik zeigt die verschiedenen Layouts. Als Text ist jeweils die zugehörige Layout-Konstante (Typ und Orientierung) eingetragen.



Der Verankerungspunkt liegt an der Pfeilspitze, bei `POSTIT` in der linken oberen Ecke.

Sind bei einem Typ mehrere Orientierungen möglich, dann beschreibt die Orientierung die Himmelsrichtung, in der vom Verankerungspunkt aus gesehen das Textelement plaziert ist.

Pfeil (Arrow)

Der Verankerungspunkt liegt an der Pfeilspitze, die Orientierung ist durch die Himmelsrichtungen (N, S, E, W) gegeben.

Sprechblase (Bubble)

Der Verankerungspunkt liegt an der Pfeilspitze, die Orientierung ist durch die Himmelsrichtungen (NW, NE, SW, SE) gegeben.

Pfeile für die Koordinatenachsen (Koordinatenachsen)

Der Verankerungspunkt liegt an der Pfeilspitze, Die Orientierung ist durch die Himmelsrichtungen (N, S, E, W) gegeben.

Markierung für Koordinatenachsen (C)

Der Verankerungspunkt liegt beim hervorgehobenen Punkt, die Orientierung ist durch die Himmelsrichtungen gegeben.

Notizzettel (POSTIT)

Der Verankerungspunkt liegt in der linken oberen Ecke.

4 Lektions-Skripte

math4u2-Lektionen werden durch XML-Skripte gesteuert. Der Aufbau dieser Skripte ist im folgenden beschrieben.

Die Skripte können prinzipiell mit jedem ASCII-Editor erstellt und verändert werden. Hilfreich ist allerdings ein spezieller XML-Editor, der den Benutzer bei der Einhaltung der korrekten Syntax unterstützt.

4.1 Überblick: Skript einer leeren Lektion

Folgende Zeilen zeigen die notwendige Grundstruktur eines Lektions-Skripts. Durch dieses Skript wird eine "leere" Lektion erzeugt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE math4u2 PUBLIC "-//FH Augsburg//DTD Math4u2 V2.1//DE"
  "http://www.math4u2.de/xml/dtd/math4u2_V2_1DTD.dtd">

<math4u2>
   <head>
      <Lektion_nur_in_math4u2_Version_2.1_ausfuehrbar/>
      <version>2.1</version>
      <module id="main" class="ShowMeGraph"/>
   </head>
   <body>
      <location>Test/Lektionen</location>
      <title>Leere Lektion</title>
      <step></step>
   </body>
</math4u2>
```

Die ersten drei Zeilen vereinbaren das zugrunde liegende XML-Schema und dürfen nicht verändert werden.

Die eigentliche Lektion ist in das `<math4u2>`-Tag eingeschlossen.

Zunächst werden im `<head>`-Tag grundsätzliche Vereinbarungen über Version und System getroffen, auch diese werden unverändert übernommen.

Zuletzt folgt im `<body>`-Tag der Inhalt der Lektion. Er gliedert sich in

- Allgemeine Dokumentinformationen

  Hier **muss** in `<location>` der Pfad angegeben werden, unter dem die Lektion im Themen-Navigator zu finden ist und in `<title>` der Titel der Lektion.

- Die Schritte der Lektion

  Mit dem leeren `<step>`-Tag wird der erste Schritt der Lektion angelegt.

Im Folgenden werden diese beiden Abschnitte näher beschrieben.

4.2 Allgemeine Dokumentinformation

Die ersten Tags legen die allgemeinen Dokumentinformationen fest.

Beispiel:

```
<body>
   <author>Dr. Weiss</author>
   <location>Test/Analysis/Entwicklung/Potenzreihen</location>
   <title>Logarithmusreihe</title>
   <summary>
   Taylor-Entwicklung des natürlichen Logarithmus, Logarithmusreihe
   </summary>
   <keywords>
      <keyword>Logarithmus</keyword>
      <keyword>Taylor-Entwicklung</keyword>
      <keyword>Potenzreihe</keyword>
      <keyword>Konvergenzradius</keyword>
      <keyword>Konvergenzradius2</keyword>
   </keywords>
   <step>
   </step>
</body>
```

Darin bedeutet:

- `<author>(optional)`: Autor der Lektion.
- `<location>`: Pfad, unter dem die Lektion im Themennavigator angezeigt wird. Die einzelnen Teile des Pfads werden durch das Zeichen `/` getrennt.
- `<title>`: Titel der Lektion.
- `<summary>(optional)`**:** Kurzbeschreibung. Wird im Themennavigator rechts angezeigt.
- `<keywords>(optional)`**:** Schließt ein oder mehrere `<keyword>`-Tags ein. Jedes `<keyword>`-Tag definiert einen Schlüsselbegriff.

Die Tags werden in der angegebenen Reihenfolge angegeben, allerdings sind lediglich `<location>` und `<title>` verpflichtend, die übrigen können entfallen.

4.3 Die Schritte einer Lektion

In einem oder mehreren <step>-Tags (*Schritte*) wird der Ablauf der Lektion festgelegt. Beim Start der Lektion wird der Inhalt des ersten Schritts abgearbeitet, beim Klick auf den WEITER-Button jeweils der Inhalt des folgenden Schritts.

Beispiel:

```
<step>
   <title>
      Fourier-Summe der Rechteckfunktion, Gibbsches Phänomen
   </title>
   <description>
      <title2>
         Die Koordinatensysteme
      </title2>
      Links sehen Sie das gesamte Rechteck,
      rechts den kritischen Ausschnitt
   </description>
   <layout>
      <row perc="100">
        <col perc="60" name="gesamt"/>
        <col perc="40" name="detail"/>
      </row>
   </layout>
   <seq>
   </seq>
</step>
```

Zu Beginn eines Schritts wird mit <title>(optional) die Überschrift für den Textbereich festgelegt.

<description>(optional) definiert den erläuternden Text. Näheres unten.

<layout>(optional) definiert die Aufteilung der Zeichenfläche in einzelne Fenster. Näheres unten.

<seq>(optional) definiert die Aktionen zum Aufbau und Ablauf des Schritts. Näheres unten.

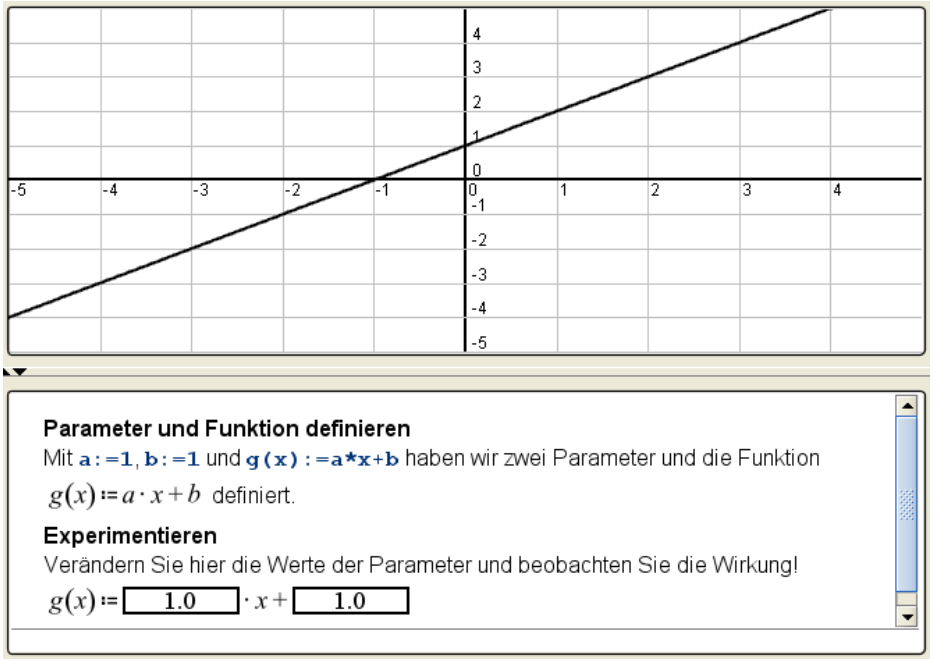
4.4 <description>: Text und Formelsatz

Jeder Schritt kann durch ein <description>-Tag mit einer Beschreibung erläutert werden.

Darin wechseln sich Überschriften (<title2>-Tag), einfache Textpassagen, hervorgehobene Passagen (z.B. <command>-Tag), Tabellen (<table>-Tag) und Formeln (<f>-Tag) in beliebiger Reihenfolge ab.

Formeln können durch ein ergänzendes <conf>-Tag um Ein-/Ausgabemöglichkeiten ergänzt werden.

Beispiel:




```
<step>
  <description>
     <title2>Parameter und Funktion definieren</title2>
      Wie mit <command>a:=1</command>, <command>b:=1</command>
      und <command>f(x):=a*x+b</command>
      haben wir zwei Parameter und die Funktion
      <f>$fd(g)</f> definiert.
      <title2>Experimentieren</title2>
       Verändern Sie hier die Werte der Parameter und
       beobachten Sie die Wirkung!
       <f><conf>a(style=text;width=6);
                b(style=text;width=6)</conf>$fd(g)</f>
   </description>
   <seq>
      <script>
          newObject( def = a:=1;  );
          newObject( def = b:=1; );
          newObject( def = g(x):=a*x+b; target = da );
      </script>
   </seq>
</step>
```

4.4.1 Text strukturieren

Überschriften

Einzelne Abschnitte eines Textbereichs können wie bei

```
<title2>Die Koordinatensysteme</title2>
```

mit einer Überschrift versehen werden.

Text hervorheben

- Tabulator: `<tab/>`
- Zeilenumbruch: `<br/>`
- *kursiv:* `<em>kursiv</em>`
- **fett**: `<strong>fett</strong>`
- `typewriter`: `<code>typewriter</code>`
- command: `<command>command</command>`

Aufzählungen und Tabellen

- **Punkt-Aufzählung**: `<ul>...</ul>`
  Die einzelnen Abschnitte der Aufzählung werden in ein `<li>...</li>`-Tag eingeschlossen:

```
<ul>
  <li>erster Abschnitt</li>
  <li>zweiter Abschnitt</li>
  ...
  <li>letzter Abschnitt</li>
</ul>
```

- **Tabelle**: `<table>...</table>`

  Pro Zeile der Tabelle wird ein `<tr>...</tr>`-Tag angeführt, in diesem für jede Spalte ein `<td>...</td>`-Tag, insgesamt entsteht eine Struktur wie:

```
<table>
   <tr>
     <td>Erste Zeile, erste Spalte</td>
     <td>Erste Zeile, zweite Spalte</td>
  </tr>
  <tr>
     <td>Zweite Zeile, erste Spalte</td>
     <td>Zweite Zeile, zweite Spalte</td>
  </tr>
   ...
</table>
```

Jede Zelle kann für sich strukturierten Text mit Formeln enthalten.

4.4.2 <f>: Formeln und Ein-/Ausgabe

Mit <f>-Tags werden wie bei <f>h(x,y):=x*sin(y)</f> Formeln gesetzt.

Formeln zitieren

Formeln von Objekten, die in einem der vorhergehenden Schritte oder im aktuellen Schritt definiert werden (wie z.B. die Funktion g(x) durch
`newObject( def = g(x):=a*x+b; target = da );`),
können im Text zitiert werden:

```
<step>
  <description>
    <title2>Formel einer definierten Funktion anzeigen</title2>
    Oben sehen Sie die Graphen der Funktionen <f>$fd(g)</f> und
    <f>$fd(h)</f> .
  </description>
  <seq>
    <script>
      newObject( def = a:=1 );
      newObject( def = b:=1 );
      newObject( def = g(x):=a*x+b; target = da );
      newObject( def = h(x):=sqrt(|g(x)|); target = da)  ;
    </script>
  </seq>
</step>
```

Hier werden im aktuellen Schritt die Funktionen g(x) und h(x) definiert. Ihre Definitionen werden durch <f>$fd(g)</f> bzw. <f>$fd(h</f> in den Text eingefügt.

math4u2-Anzeige:

Formel einer definierten Funktion anzeigen

Oben sehen Sie die Graphen der Funktionen $g(x) := a \cdot x + b$ und $h(x) := \sqrt{|g(x)|}$.

Allgemein gilt:
Ist die Funktion wie `h(x):=sqrt(|g(x)|)` definiert, so kann man die Bestandteile dieser Definition zitieren als:

Anweisung		Beispiel	Ergebnis
$fd(...)	Vollständige Definition	<f>$fd(h)</f>	$h(x) := \sqrt{\lvert g(x)\rvert}$
$fh(...)	Kopf (head) der Definition	<f>$fh(h)</f>	$h(x)$
$fb(...)	Rumpf (body) der Definition	<f>$fb(h)</f>	$\sqrt{\lvert g(x)\rvert}$

In der Klammer wird wie bei `$fd(h)` lediglich der Name des zitierten Objekts angegeben.
Eine zitierte Formel kann auch ein Teil einer umfangreicheren Formel sein. Hat man die zweistellige Funktion `F(x,y):=y+x/2` definiert, so ergibt

```
... <f>$fh(F)</f> auf den Hauptnenner: <f>$fb(F)=(2*y+x)/2</f> ...
```

die folgende Textpassage mit den beiden Formeln:

... $F(x,y)$ auf den Hauptnenner: $y+\frac{x}{2} = \frac{2\cdot y+x}{2}$ ...

Formeln setzen

Grundlage des Formelsatzes ist die auch bei der Definition von Objekten verwendetet Termschreibweise.

- **Zahlen**: `1.2, 1.3E-3, -22.0E-12 .`
- **Bezeichner**:
  Ein Bezeichner beginnt mit einem Buchstaben (keine Umlaute), es folgt eine beliebige Sequenz aus Buchstaben und Ziffern. Diese Sequenz kann einmal das Zeichen Unterstrich _ oder das Zeichen Tilde ~ enthalten.
  Gültige Bezeichner sind z.B. : `a`, `a1`, `punkt1`, `aber`, `zwei_sin`, `meinBez`.

  Bei der Darstellung von Bezeichnern im Formelsatz gelten folgende besonderen Konventionen:

  Griechische Zeichen:
  Die Bezeichner `alpha`, `beta`, `gamma`, `delta,` ... und `Alpha`, `Beta`, `Gamma`, `Delta,` ... werden durch die Zeichen $\alpha, \beta, \gamma, \delta, \ldots$ bzw. $A, B, \Gamma, \Delta, \ldots$ des griechischen Alphabets dargestellt.

  Bezeichner, die einen Unterstrich _ enthalten:
  Wie bei `t_start` oder `x_alpha` wird der Bestandteil nach dem Unterstrich als tiefgestellter Index gesetzt: t_{start} und x_{α}.

  Bezeichner, die eine Tilde ~ enthalten:
  Wie bei `t~neu` oder `alpha~beta` wird der Bestandteil nach der Tilde als hochgestellter Index gesetzt: t^{neu} und α^{β}.

  Hier ergeben folgende Bestandteile eine besondere Dekoration der Bezeichner:
  `x~quer, x~dach, x~tilde, x~stern` ergibt $\bar{x}, \hat{x}, \tilde{x}, x^{*}$.

- **Indices**:
  `a[1]` erzeugt a_1, `b[n+1,k]` erzeugt $b_{n+1,k}$ und `alpha[0]` erzeugt α_0 .

- **Operatoren**:
  Binäre Infix-Operatoren =, +, −, *, /, ^, unärer Postfix-Operator `!` .

- **Aufzählungen**:
  Das Komma "," trennt die Elemente einer Aufzählung.
  Die Eingabe `sin(x),cos(x),tan(x)` ergibt die Aufzählung $sin(x), cos(x), tan(x)$.

- **Leerer Bezeichner und Fortsetzungspunkte:**
  Ein Bezeichner kann ersetzt werden durch den leerer Bezeichner `$void` oder durch Fortsetzungspunkte `$dots`.
  Die Formel `<f>a, b, c, $dots</f>` ergibt $a,b,c,\ldots$,
  die Formel `<f>$void = 0</f>` ergibt $= 0$.

- **Klammern**: (und) , sie müssen paarweise auftreten.

- **Funktionen**:
  Ein- und mehrstellinge Funktionen werden allgemein geschrieben wie `f(x,y)` und `f(2*x, sin(y))`.

  Folgende Funktionen werden in der üblichen Spezialform dargestellt:

Einstellige Funktionen mit spezieller Darstellung

`sqrt(x)`	$\sqrt{x}$
`exp(x)`	e^{x}
`|x|`	$\|x\|$
`inverse(M)`	M^{-1}
`transpose(M)`	M^{T}

Zweistellige Funktionen mit spezieller Darstellung

`log(a,b)`	$log\ _{a}b$
`root(a+x,3)`	$\sqrt[3]{a+x}$
`pow(x,2+n)`	x^{2+n}

Ableitung

`derive(f(x),2)`	$f''(x)$
`derive(x+2,2)`	$(x+2)''$
`derive(f(x), n+1)`	$f^{(n+1)}(x)$
`derive(f(x), 2)(c)`	$f''(c)$
`derive(vars(x), x*sin(x), 1)(y^2)`	$(xsin(x))'(y^2)$

Summe, Produkt und Integral

Summe: `sum(i,3,n,f(x))`	$\sum_{i=3}^{n} f(x)$
Produkt: `prod(i,3,n,f(x))`	$\prod_{i=3}^{n} f(x)$
Bestimmtes Integral: `int(x,a,b,f(x))`	$\int_{a}^{b} f(x)dx$
Unbestimmtes Integral: `int(x,f(x))`	$\int f(x)dx$

Grenzwert

Grenzwert: `lim(vars(x),x^2/(x+1),$infin)`	$\lim_{x \to \infty} \frac{x^2}{x+1}$
rechtsseitiger Grenzwert: `lim+(vars(x),x^2/(x+1),-1)`	$\lim_{x \to -1^+} \frac{x^2}{x+1}$
linksseitiger Grenzwert: `lim-(vars(x),x^2/(x+1),-1)`	$\lim_{x \to -1^-} \frac{x^2}{x+1}$

- **Spezielle Symbole**

  `$infin` wird dargestellt als ∞ .

- **Zweistellige Relationen und Operatoren**

  arithmetische Operatoren `+`, `-`, `*`, `/`, `^` , weitere Operatoren und Relationen durch **:**

`a = b`	$a = b$	`x $isin R`	$x \in R$	`C $nsub E`	$C \not\subset E$
`a $ne b`	$a \neq b$	`x $notin R`	$x \notin R$	`a $equiv B`	$a \equiv b$
`a $lt b`	$a < b$	`M $ni x`	$M \ni x$	`a $cong b`	$a \cong b$
`a $le b`	$a \leq b$	`A $sub B`	$A \subset B$	`a $prop b`	$a \propto b$
`a $gt b`	$a > b$	`B $sup A`	$B \supset A$	`a $sim b`	$a \sim b$
`a $ge b`	$a \geq b$	`A $supe B`	$A \subseteq B$	`a $perp b`	$a \perp b$
`a $asymp b`	$a \approx b$				

- **Vektoren und Matrizen**

  Vektoren werden als Spalten gesetzt: `<f>vektor({1,x,x^2})</f>` erscheint wie $\begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$,

  Dualvektoren als Zeilen: `<f>dualvektor({1,x,x^2})</f>` erscheint wie $\begin{bmatrix} 1 & x & x^2 \end{bmatrix}$,

  Matrizen wie üblich: `<f>matrix({{1,2,3},{2,2,2}})</f>` erscheint wie $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 2 \end{bmatrix}$.

Ein-/Ausgabeelemente:

Parameter (nullstellige skalare Funktionen) können in Formeln durch Ein-/Ausgabeelemente ersetzt werden.

So kann die Formel $y = sin(\omega \cdot t)$ so konfiguriert werden, dass der Wert für die Kreisfrequenz ω in der Formel angezeit wird und auch dort verändert werden kann.

Dazu wird im `<f>`-Tag als erstes Element ein `<conf>`-(Konfigurations-)Tag eingefügt. Darin wird festgelegt, welcher Parameter in welcher Form (`style`) dargestellt werden soll.

Als Formen (`style`) stehen zur Verfügung:

`text`	Ein-/Ausgabefeld
`dtext`	Ein-Ausgabefeld mit Namen des Parameters
`slider`	Schieberegler
`dslider`	Schieberegler mit Namen des Parameters

Ein-/Ausgabefeld: `text`

Jedes Vorkommen des Parameters `omega` wird durch ein Ein-/Ausgabefeld der Breite 8 ersetzt:

```
<description>Ändern Sie hier die Kreisfrequenz <f>omega</f>:
<br/><f><conf>omega(style=text;width=8)</conf>y=sin(omega*t)</f>
</description>
```

math4u2-Anzeige:

Ändern Sie hier die Kreisfrequenz ω:

$y = \sin(\boxed{1.0} \cdot t)$

Ein-Ausgabefeld mit Namen des Parameters: `dtext`

```
<description>Ändern Sie hier die Kreisfrequenz <f>omega</f>:
<br/><f><conf>omega(style=dtext;width=8)</conf>y=sin((omega)*t)</f>
</description>
```

math4u2-Anzeige:

Ändern Sie hier die Kreisfrequenz ω:

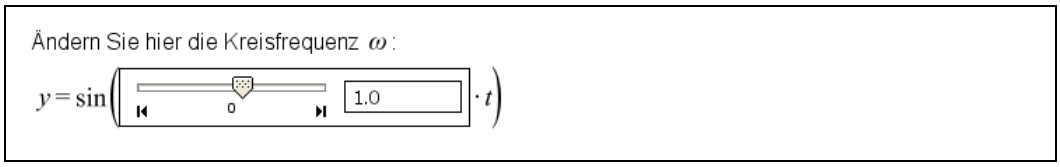
$y = \sin((\omega := \boxed{1.0}) \cdot t)$

Schieberegler: `slider`

Jedes Vorkommen des Parameters `omega` wird durch einen Schieberegler ersetzt. Der Regelbereich wird durch die Angaben `min = -10` und `max = 10` festgelegt:

```
<description>Ändern Sie hier die Kreisfrequenz <f>omega</f>:
<br/><f><conf>omega(style=slider;min = -10;max = 10)</conf>
              y=sin(omega*t)
     </f>
</description>
```

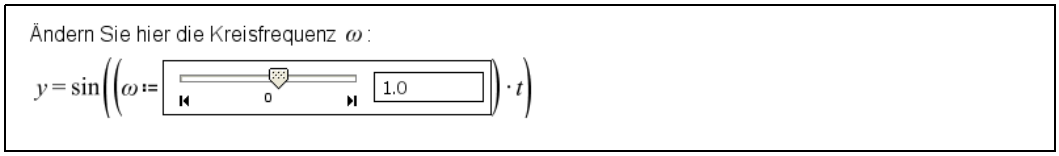
math4u2-Anzeige:



Schieberegler mit Namen des Parameters: `dslider`

```
<description>Ändern Sie hier die Kreisfrequenz <f>omega</f>:
<br/><f><conf>omega(style=dslider;min=-10;max=10)</conf>
        y=sin((omega)*t)
     </f>
</description>
```

math4u2-Anzeige:



Wenn in einer Formel mehrere Paramter durch Ein-/Ausgabeelemente dargestellt werden sollen, so werden die entsprechenden Angaben im `<conf>`-Tag jeweils durch ein Semikolon (;) getrennt.

Werden z.B. im aktuellen Schritt die Parameter und Funktionen

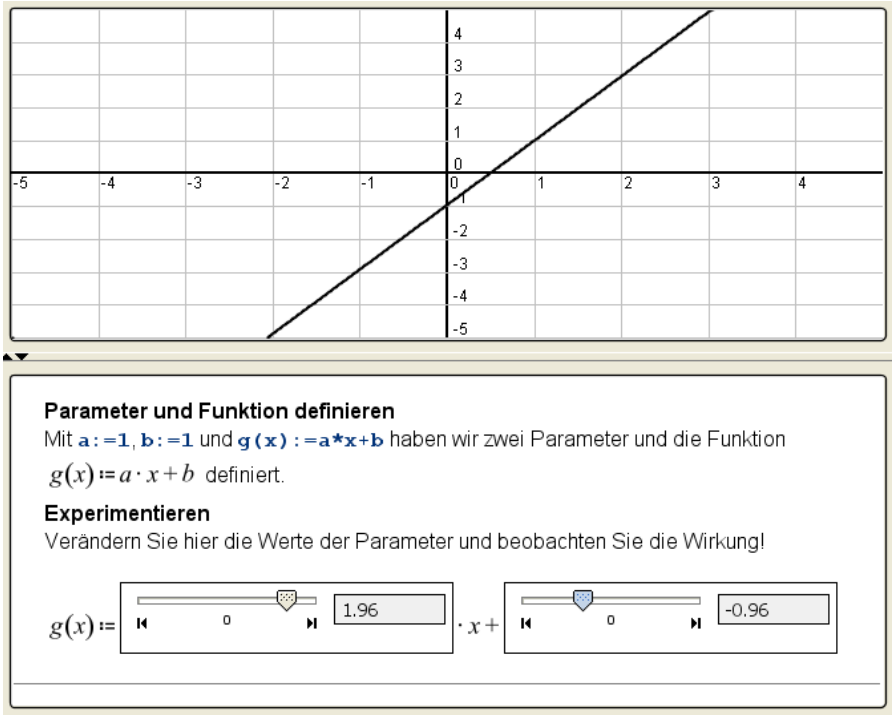
`a:= 1`,

`b:= 2` und

`g(x):=a*sin(b*x)` definiert,

so kann man wie folgt eine Formeldarstelliung erzeugen, in der beide Parameter durch Ein-/Ausgabefelder dargestellt werden:

```
<step>
<description>
  <title2>Parameter und Funktion definieren</title2>
      Mit <code>a:=1</code>, <code>b:=1</code>
      und <code>g(x):=a*x+b</code>
      haben wir zwei Parameter und die Funktion
      <f>$fd(g)</f> definiert.
  <title2>Experimentieren</title2>
      Verändern Sie hier die Werte der Parameter und
      beobachten Sie die Wirkung!
      <f>
          <conf>  a(style=slider; min = -3; max = 3);
                  b(style=slider; min = -3; max = 3)
          </conf>
          $fd(g)
      </f>
</description>
<seq>
  <script>
         newObject( def = a:=1;  );
         newObject( def = b:=1; );
         newObject( def = g(x):=a*x+b; target = da );
   </script>
</seq>
</step>
```



Formatierung

Bei Bedarf kann die Form der Ausgabe bei `text` und `dtext` durch ein Formatmuster (den Parameter `pattern`) festgelegt werden. Das Muster besteht aus folgenden Zeichen:

0	Eine Ziffer
#	Eine Ziffer, eine 0 wird hier nicht angezeigt
.	Der Dezimalpunkt
E	Beginn der Musters für den Exponenten (zur Basis 10)

Beispiele für Format-Muster:
Stets gilt: Wenn die Ausgabe in der durch **`width`** festgelegten Breite nicht Platz hat, dann erfolgt keine Ausgabe.

Es werden immer mindestens 3 Ziffern vor dem Dezimalpunkt angezeigt, 8 Ziffern nach dem Dezimalpunkt:
conf-Parameter: **`style=deval;width=13;pattern=000.00000000`**

$a1=$ 002.71828183 , $a2=$ 2718.28182846 , $a3=$ 37000.00000000

$b1=$ -002.71828183 , $b2=$ -2718.28182846 , $b3=$ -37000.00000000

Es wird immer mindestens eine Ziffern vor und nach dem Dezimalpunkt angezeigt:
conf-Parameter: **`style=deval;width=13;pattern=##0.0#######`**

$a1=$ 2.71828183 , $a2=$ 2718.28182846 , $a3=$ 37000.0

$b1=$ -2.71828183 , $b2=$ -2718.28182846 , $b3=$ -37000.0

Exponantialdarstellung, stets eine Ziffer vor dem Dezimalpunkt
conf-Parameter: **`style=deval;width=13;pattern=0.#######E0`**

$a1=$ 2.7182818E0 , $a2=$ 2.7182818E3 , $a3=$ 3.7E4

$b1=$ -2.7182818E0 , $b2=$ -2.7182818E3 , $b3=$ -3.7E4

Exponentialdarstellung, der Exponent ist stets ein Vielfaches von 3 (wissenschaftliche Darstellung):
conf-Parameter: **`style=deval;width=13;pattern=##0.#####E0`**

$a1=$ 2.7182818E0 , $a2=$ 2.7182818E3 , $a3=$ 37E3

$b1=$ -2.7182818E0 , $b2=$ -2.7182818E3 , $b3=$ -37E3

Anzeige des aktuellen Ergebnisses:

Nullstellige Funktionen mit dem Ergebnis Skalar, Vektor, Dualvektor oder Matrix können durch eine Anzeige ihres aktuellen Ergebnisses ersetzt werden.

Anders als bei den oben beschriebenen Ein-/Ausgabeelementen kann hier der Wert nicht verändert werden.

Als Formen (`style`) stehen zur Verfügung:

`eval`	Ergebnis anzeigen
`deval`	Ergebnis mit Namen anzeigen

Hier wird zu einer Matrix `m1` die Inverse `m2` berechnet und beide Matrizen werden angezeigt:

```
<step>
 <description>
  <f>
   <conf> m1(style=eval;width=4);m2(style=eval;width=4)</conf>
    B:=m1*m2
  </f>
 </description>
 <seq>
  <script>
       newObject( def = m1:= matrix({{1,0,0},{0,2,0},{0,0,1}}));
       newObject( def = m2:= inverse(m1));
   </script>
 </seq>
</step>
```

math4u2-Anzeige:

$$B := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Zur gezielten Formatierung kann auch hier ein Formatierungs-Muster angegeben werden.

4.5 <layout>: Zeichenfläche in Zeichenfenster aufteilen

Standardmäßig wird die Zeichenfläche von einem Zeichenfenster mit dem Namen `da` (drawarea) ausgefüllt. Bei Bedarf kann die Zeichenfläche in mehrere Zeichenfenster aufgeteilt werden.

Zeichenfläche in Zeichenfenster aufteilen, interne Namen vergeben

Die Zeichenfläche wird durch eine `<layout>`-Tag aufgeteilt:

```
<layout>
   <row perc = "60">
     <col perc="100" name ="fenster1"/>
   </row>
   <row perc = "40">
     <col perc="50" name ="cosinusF"/>
     <col perc="50" name ="sinusF"/>
   </row>
</layout>
```

Hier wird die gesamte Zeichenfläche in zwei horizontale Zeilen (`row`) eingeteilt. Die erste Zeile nimmt dabei stets 60% der verfügbaren Höhe ein, die zweite Zeile 40%.

Die gesamte Breite (100%) der ersten Zeile wird von einem einzigen Zeichenfenster eingenommen, für dieses wird der interne Name `fenster1` vereinbart.

Die zweite Zeile wird zu gleichen Teilen (50% zu 50%) in zwei Spalten (`col`umn) aufgeteilt. Von links nach rechts werden für diese Zeichenfenster die internen Namen `cosinusF` und `sinusF` vereinbart.

Die vereinbarten internen Namen (`fenster1`, ...) werden zur Manipulation der Fenster verwendet. Näheres dazu beim `<seq>`-Tag.

Aufteilung der Zeichenfläche ändern

Wenn die Zeichenfläche in einem Schritt anders als im vorhergehenden Schritt aufgeteilt werden soll, so geschieht dies durch ein neues `<layout>`-tag:

```
<layout>
   <row perc = "50">
     <col perc="50" name ="cosinusF"/>
     <col perc="50" name ="sinusF"/>
   </row>
<row perc = "50">
     <col perc="50" name ="amplitude"/>
     <col perc="50" name ="phase"/>
   </row>
</layout>
```

Zeichenfenster, die schon vorher eingerichtet waren (wie `cosinusF` und `sinusF`), bleiben mit ihren Graphen erhalten, sie werden in ihrer Größe an das neue Layout angepasst.

Zeichenfenster zeitweise ausblenden

Wenn ein Zeichenfenster zeitweise nicht mehr benötigt wird, dann wird es in der neuen layout-Anweisung nicht mehr aufgeführt (z.B. das Zeichenfenster `fenster1` in der layout-Anweisung vorher). Es bleibt aber logisch (mit seinem gesamten Inhalt) vorhanden, es wird lediglich nicht mehr gezeichnet. Soll es in einem späteren Schritt wieder gezeichnet werden, dann muss es lediglich wieder mit seinem Namen in einer layout-Anweisung aufgeführt werden.

4.6 <seq>: Aktionen eines Schritts festlegen

Die Aktionen eines Schritts werden durch **ein** <seq>-Tag (optional) beschrieben.

Beispiel:

```
<step>
   <description>
      Punkt p1:=punkt(a,b) erzeugen, dann seine Koordinaten a und b
      parallel animieren.
   </description>
   <seq>
      <script>
         deleteAll();
         da.coordinateSystem(xMin=-10; xMax=10; yMin=-3; yMax=1.5);
         newObject( def = a:=0 );
         newObject( def = b:= -2 );
         newObject( def = p1:=punkt( a, b );
      </script>
      <par>
         <script>
            animate( name = a; to = 8; end = 5 );
            animate( name = b; to = 1; end = 10 );
         </script>
      </par>
   </seq>
</step>
```

Dabei werden elementare Aktionen/Prozesse durch <script>-Tags zu Gruppen zusammengefasst. Aus diesen Gruppen wird mit <seq>-Tags (sequentiell) und <par>-Tags (parallel) ein Gesamtprozess von sequentiellen und parallelen Abläufen definiert.

4.6.1 Übersicht: Elementare Anweisungen

Die elementaren Anweisungen beschreiben folgende grundlegenden Aktionen und Prozesse (eine ausführliche Beschreibung folgt unten) :

Benutzeroberfläche konfigurieren:

`defaults()`	Voreinstellungen setzen
`expand()`	Listen ein/ausklappen

Objekte erzeugen, verändern, löschen:

`newObject()`	Objekt erzeugen
`modifyObject()`	Eigenschaften eines Objekts ändern
`deleteObject()`	Objekte löschen
`deleteAll()`	Alle Objekte löschen

Eigenschaften eines Zeichenfensters festlegen:

Die Methoden richten sich an ein Fenster mit dem Namen *fenstername* :

fenstername`.setTitle()`	Titel setzen
fenstername`.addGraph()`	Graphen hinzufügen
fenstername`.deleteGraph()`	Graphen löschen

fenstername.coordinateSystem()	Koordinatensystem einrichten
fenstername.zoom1To1()	1To1-Zoom ein/ausschalten

Hinweis: Standardmäßig ist das Zeichenfenster mit dem Namen da eingerichtet.
Weitere Zeichenfenster können durch ein <layout>-Tag erzeugt werden

Animation:

animate()	Animation

4.6.2 <skript>-Tag: Elementare Anweisungen gruppieren

- Ein <script>-Tag schließt eine oder mehrere elementare Anweisungen ein.
- Jede Anweisung wird durch ein Semikolon ; abgeschlossen.
- Die Anweisungen können durch Kommentare erläutert werden.
- Das umgebende Tag (<seq> oder <par>) regelt, ob die Anweisungen sequentiell (nacheinander) oder parallel ausgeführt werden.

Kommentare

Block-Kommentare

Sie werden eingeleitet durch die Zeichenfolge /* und beendet durch die Zeichenfolge */ .

Zeilenende-Kommentare

Sie beginnen mit der Zeichenfolge // und erstrecken sich bis zum Ende der Zeile.

Beispiel:

```
<script>
  /***********************************
   * ein auffaelliger Blockkommentar  *
   ***********************************/

  newObject(def = n:=1);  //Anweisung, durch Semikolon abgeschlossen
  newObject(def = f(x):=x*x);

  /*  ein
      dezenter
      Blockkommentar  */
</script>
```

4.6.3 <seq>-Tag: sequentuelle Ausführung

Ein <seq>-Tag beschreibt die sequentielle Ausführung von Aktionen und Prozessen. Es enthält eine Folge von:

<script>-Tag	Beschreibt eine Folge von Elementar-Aktionen und Elementar-Prozessen, die schrittweise nacheinander abgearbeitet werden.
<par>-Tag	Beschreibt einen Schritt, der aus der parallelen Ausführung von Teilprozessen besteht.

4.6.4 <par>-Tag: parallele Ausführung

Ein <par>-Tag beschreibt die parallele Ausführung mehrerer Prozesse. Es enthält eine beliebige Sequenz von:

<script>-Tag	Enthält ein oder mehrere Elementarprozesse.
<seq>-Tag	Beschreibt einen zusammengesetzten Prozess.

Dadurch ist eine Menge von Prozessen definiert. Diese Prozesse werden parallel ausgeführt.

4.6.5 Beispiele

Die folgenden Beispiele zeigen einige Grundmuster.

Mehrere Anweisungen nacheinander ausführen

Häufig werden in einem Schritt einer Lektion der Reihe nach einige elementare Aktionen durchgeführt: Nacheinander wird das Koordinatensystem da angepasst, der Parameter a und dann der Punkt p1 erzeugt.

```
<seq>
  <script>
        // Koordinatensystem da anpassen
     da.coordinateSystem(xMin=-9.0;xMax=9.0;yMin=-5.0;yMax=5.0;);
        // Parameter a erzeugen.
     newObject( def = a:= 3 );
        /* Punkt p1 haengt von Parameter a ab.
           er wird durch target = da Fenster mit Namen da
         (Standard-Fenster) gezeichnet. */
     newObject( def = p1:=punkt(a,8); target = da );
  </script>
</seq>
```

Elementare Animation

Bewegungen auf dem Bildschirm entstehen oft durch Animation eines einzigen Parameters.

```
<seq>
    <script>
        // a erzeugen.
      newObject( def = a:= 0 );
        // p1 haengt von a ab.
      newObject( def = p1:=punkt(a,sin(a)); target = da );
        // a animieren, p1 bewegt sich auf sin-Kurve
      animate( name = a; to = 6.24; end = 10 );
   </script>
</seq>
```

Mehrere Parameter parallel animieren

Kompliziertere Bewegungsmuster entstehen durch parallele Animation mehrerer Parameter:

```
<seq>
    <script>
        // a erzeugen.
      newObject( def = a:= 3 );
```

```
            // b erzeugen.
         newObject( def = b := 1 );
            // p1 haengt von a und b ab.
         newObject( def = p1:=punkt(a,sin(b)); target = da );
      </script>
      <par>
         <script>
               /* ***********************************
                * a und b werden parallel verändert,
                * a bis zum Wert -3 in 10 Sekunden,
                * b bis zum Wert 30 in 20 Sekunden
                ************************************/
            animate( name = a; to = -3; end = 10 );
            animate( name = b; to = 30; end = 20 );
         </script>
      </par>
</seq>
```

Sequenz mit Wechsel zwischen Animation und Objekterzeugung

```
<seq>
     <script>
        newObject( def = a:= 3 );
        newObject( def = p1:=punkt(a,a); target = da );
        animate( name = a; to = -3; end = 10 ); // p1 bewegt sich
        // p1 steht still
        newObject( def = b:= 0 );
        newObject( def = p2:=punkt(sin(b),cos(b)); target = da );
        animate( name = b; to = 20; end = 10 ); // p2 bewegt sich
        // p2 steht still
      </script>
</seq>
```

4.7 Elementare Anweisungen

Die Wirkung der elementaren Anweisungen wird durch entsprechende Attribute gesteuert.
Jedes Attribut wird in der Form *ATTRIBUTNAME* = *WERT* angegeben.
Die Reihenfolge, in der die Attribute angegeben werden, ist beliebig.
Die Angabe von Attributen, für die eine Voreinstellung angegeben ist, kann entfallen.

4.7.1 Benutzeroberfläche konfigurieren

defaults(): Voreinstellungen setzen

Mit der `defaults()`-Anweisung werden global gültige Voreinstellungen gesetzt.
Die Anweisung `defaults( hide = true )` bewirkt, dass bei allen folgenden Objektdefinitionen wie z.B. `newObject( def = f(x):=x )` für die Sichtbarkeit die Voreinstellung `hide=true` gilt.

- Voreinstellung für die Sichtbarkeit von Definitionen
  `hide = true` oder `false` Voreinstellung: `false`

expand(): Listen ein/ausklappen

Mit der `expand()`-Anweisung wird festgelegt, in welchem Zustand sich die beiden Listen der Steuerleiste (links: Definitionsliste, rechts: Detailliste) befinden.

Mit `expand( left = one; right = all )` legt man fest, dass von der linken Liste (Definitionsliste) ein Element sichtbar ist (bei der Definitionsliste immer das oberste) und dass von der rechten Liste (Detailliste) alle Elemente sichtbar sind.

Mögliche Werte sind für die Attribute `left` und `right` sind:

`all`	Alle Elemente sichtbar (Voreinstellung).
`one`	Ein Element sichtbar. Bei der Definitionsliste: das oberste, bei Detailliste: das ausgewählte.
`none`	Kein Element sichtbar.

4.7.2 Objekte erzeugen, verändern, löschen

newObject() : Objekt erzeugen

`newObjekt( def = kreis1 := kreis( punkt(1,1), 2 ) )` erzeugt ein neues Objekt.

- Als Wert des Attributs `def` wird die gewünschte Definition angegeben, hier die Definition `kreis1 := kreis( punkt(1,1), 2 ).`

  Die weiteren Eigenschaften des Objekts werden durch folgende Attribute und zugehörige Werte festgelegt:

- Sichtbarkeit der Definition

  `hide = true` oder `false`

  Dieses Attribut legt fest, ob die Definition des Objekts in der Definitionsliste angezeigt wird (`hide = false`) oder nicht (`hide = true`).

  Wenn dieses Attribut nicht angegeben ist, dann wird der aktuell gültige default-Wert übernommen, Dieser wird durch die `defaults`-Anweisung eingestellt.

- Darstellung des Objekts in der Detailliste

  `select = true` oder `false` Voreinstellung: `false`

  Dieses Attribut legt fest, ob das Objekt in der Detailliste angezeigt wird (`select = true`) oder nicht (`select = false`).

- Sichtbarkeit der Graphen

  `visible = true` oder `false` Voreinstellung: `true`

- Namen von Punkten einer Punktliste

  `showNames = true` oder `false` Voreinstellung: `true`

  Legt fest, ob die Punkte einer Punktliste mit Namen (`true`) oder ohne Namen (`false`) auf einer Zeichenfläche dargestellt werden.

- Linienart

  `linestyle = solid` oder `dot` oder `dash` oder `dot-dash` ; Voreinstellung: `solid`

- Farbe
  `color = Farbangabe` Voreinstellung: `black`
  `fillcolor = Farbangabe` Voreinstellung: weiss transparent
  `bordercolor = Farbangabe` Voreinstellung: `black`

  Farbangabe kann sein:

  - Eine der folgenden Fabkonstanten:

    Besonders geeignet für Linien:
    `black, darkgray, gray, lightgray, white, darkred, red, orange, lightorange, yellow, lightgreen, green, cyan, blue, magenta, pink.`

    Besonders geeignet als Füllfarben (mit Transparenz):
    `fillgray, fillwhite, fillred, fillorange, fillyellow, fillgreen, fillcyan, fillblue, fillmagenta.`

  - Eine RGB-Angabe wie `100.0.255` . Darin steht allgemein für jede der drei Farben Rot, Grün und Blau ein ganzzahliger Wert `0 ... 255` .

  - Eine RGB-Angabe mit alpha-Wert wie `100.0.255.120` .
    Der zusätzliche alpha-Wert (hier: `120`) beschreibt die Transparenz: `0` bedeutet vollständig transparant, `255` bedeutet vollkommen deckend.

- Darstellende Zeichenfläche
  `target = Fenstername` Voreinstellung: keine

  Im Zeichenfenster mit dem Namen *Fenstername* wird ein Graph des Objekts erzeugt und gezeichnet (falls das Objekt einen entsprechenden Graphen hat und das Attribut `visible` den Wert `true` hat).
  Wenn ein Objekt in mehreren Fenstern gezeichnet werden soll, so können diese wie bei `target = da1,da2,da3` durch Kommata getrennt angeführt werden.

deleteObject() : Objekt löschen

`deleteObject( name = obj1 )` löscht das Objekt mit Namen `obj` .
Wenn mehrere Objekte nacheinander gelöscht werden sollen, dann können diese wie bei `deleteObject( name = obj2, obj3, obj4 )` in einer Anweisung durch Kommata getrennt aufgeführt werden.

deleteAll() : Alle Objekte löschen

`deleteAll()` löscht sämtliche Objekte. Wenn mehrere Fenster eingerichtet wurden, so werden auch diese gelöscht, anschließend wird das Standardfenster `da` wieder eingerichtet.

modifyObject() : Eigenschaften eines Objekts ändern

Mit dieser Anweisung können die Eigenschaften eines Objekts geändert werden.
Das betroffene Objekt wir durch das Attribut `name=Objektname` angegeben.
Eigenschaften sind dabei die oben bei `newObject()` angegebenen Attribute und ihre Werte.

Beispiel:
Hat man mit `newObject(def = k:=kreis(punkt(1,1), 2 ); target = da1 )` den Kreis `k` erzeugt, dann wird der Kreis zunächst im Fenster `da1` gezeichnet, per Voreinstellung ist die Kreislinie schwarz und die Fläche wird nicht gefüllt.
Mit `modifyObject(name=k;bordercolor=red;fillcolor=blue;target=da2,da3)` wird die Farbe für die Umrandung (Kreislinie) und für die Fläche verändert. Ausserdem wird festgelegt, dass der Kreis **zusätzlich** in den Fenstern `da2` und `da3` gezeichnet werden soll.

Hinweis: Wenn der Graph des Kreises `k` aus dem Fenster `da1` entfernt werden soll, dann geschieht dies durch die Anweisung `da1.deleteGraph( name = k )` .

4.7.3 Animation

animate(): Animation

Eine Animation wird im allgemeinsten Fall durch eine `animate`-Anweisung mit 5 Attributen beschrieben: Durch

```
animate( name = par1; from = -7; to = 5; start = 3; end = 10 )
```

wird vereinbart, dass der Wert des Parameters (der nullstelligen Funktion) `par1` animiert wird. Der Wert des Parameters wird dabei von `-7` nach `5` linear verändert. Die Veränderung beginnt `3` Zeiteinheiten (Sekunden) nach dem Aufruf der Anweisung und endet `10` Zeiteinheiten (Sekunden) nach dem Aufruf der Anweisung.

Als Zeiteinheit für die Animation ist zunächst eine Sekunde eingestellt.
Der Benutzer kann diese Zeiteinheit über den Schieberegler *Langsam/Normal/Schnell* rechts unten in der Benutzeroberfläche verändern.

Die Attribute der Anweisung bedeuten im Detail:

`name`	Name des Parameters, der animiert werden soll.
`from (optional)`	Zahlenwert, der zu Beginn der Animation eingestellt wird. Wenn dieses Atribut nicht angegeben ist, wird zu Beginn der Animation der aktuell vorliegende Wert des Parameters eingestellt.
`to`	Zahlenwert, der am Ende der Animation erreicht wird.
`start (optional)`	Zeitpunkt (Zahl), zu dem die eigentliche Animation beginnt. Wenn dieses Attribut nicht angegeben ist, wird der Standard-Wert `start=0` verwendet.
`end`	Zeitpunkt (Zahl), zu dem die Animation endet.

4.7.4 Eigenschaften eines Zeichenfensters festlegen

Standardmäßig ist das Zeichenfenster mit dem Namen `da` eingerichtet.

Andere Zeichenfenster werden bei der Erzeugung über ein `<layout>`-Tag mit Namen (z.B. `fenster1`) versehen.

Im Folgenden ist beispielhaft beschrieben, wie die Eigenschaften des Fensters `fenster1` verändert werden können.

Titel setzen

`fenster1.setTitle(title="text mit Leerzeichen")` legt den Text fest, der rechts oben im Fenster `fenster1` angezeigt wird. Der Text selbst muss in Anführungszeichen `""` eingeschlossen werden.

Graphen hinzufügen

`fenster1.addGraph(name = kreis1)` fügt im Fenster `fenster1` einen Graphen des Objekts mit dem Namen `kreis1` ein, d.h. das Bild von `kreis1` wird jetzt dort gezeichnet.

Wenn mehrere Graphen hinzugefügt werden sollen, dann können die Namen der Objekte, durch Kommata getrennt, aufgeführt werden:

```
fenster1.addGraph(name = kreis1, kreis2, strecke1, f).
```

Wenn ein neu erzeugtes Objekt sofort gezeichnet werden soll, so kann über das Attribut `target` das Fenster benannt werden, in dem der Graph eingefügt werden soll:

```
newObject( def = g(x):= x*sin(x); target = fenster1 ) .
```

Graphen löschen

`fenster1.deleteGraph( name = kreis1, kreis2, g )` löscht im Fenster `fenster1` die Graphen der Objekte `kreis1`, `kreis2` und `g`.
Die Objekte selbst und die Graphen in anderen Fenstern bleiben erhalten.

Koordinatensystem mit Zahlen statisch einrichten

`fenster1.coordinateSystem(xMin=-4; xMax=50; yMin=-11; yMax=12)` richtet im Fenster `fenster1` das Koordinatensystem mit den angegebenen Bereichsgrenzen ein.
Die Grenzen werden durch Zahlenwerte angegeben.

Koordinatensystem mit Termen dynamisch festlegen

Die Bereichsgrenzen des Koordinatensystems werden für jedes Fenster durch die nullstelligen Funktionen `xMin`, `xMax`, `yMin`, `yMax` definiert. Wenn ein Koordinatensystem statisch eingerichtet wurde, dann haben diese Funktionen zunächst die so festgelegten Zahlenwerte.
Die Definition dieser Funktionen kann durch beliebige Terme überschrieben werden.

```
newObject( def = a:= 12 );
newObject( def = fenster1.xMin := fix( -a/6 ) );
newObject( def = fenster1.xMax := fix( a*5/6 ) );
```

Nach diesen Anweisungen zeigt das Koordinatensystem im Fenster `fenster1` in x-Richtung die Werte von $-\frac{a}{6} = -2$ bis $a \cdot \frac{5}{6} = 10$. Die zusätzliche Funktion `fix( ... )` in den Funktionstermen bewirkt, dass diese Festlegungen nicht durch manuelles Zoomen überschrieben werden, Zoomen in x-Richtung ist dann in diesem Fenster nicht möglich.

Jetzt kann das Koordinatensystem über den Parameter a eingestellt werden:
`newObject( def = a:=24 )` bewirkt, dass der x-Ausschnitt des Koordinatensystems doppelt so groß wird, die y-Achse behält im Fenster ihre graphische Position.

Mit `animate( name = a; to = 120; end = 10 )` wird der x-Ausschnitt jetzt kontinuierlich verändert.

Koordinaten-Grenzen in Termen verwenden

Die Funktionen `xMin`, `xMax`, `yMin` und `yMax` eines Fensters können selbst als Bestandteil beliebiger anderer Terme verwendet werden.

Automatisches Zoomen in einem zweiten Fenster

Hat man z.B. zwei Fenster `fenster1` und `fenster2`, so bewirkt

```
newObject( def = fenster2.xMin := fix( fenster1.xMin ) );
newObject( def = fenster2.xMax := fix( fenster1.xMax ) );
newObject( def = fenster2.yMin := fix( fenster1.yMin ) );
newObject( def = fenster2.yMax := fix( fenster1.yMax ) ); ,
```

dass sich beim Zoomen in `fenster1` automatisch in `fenster2` der gleiche Ausschnitt einstellt.
Wegen der `fix`-Funktion ist ein direktes Zoomen in `fenster2` nicht möglich.

Punkt in der Mitte eines Fensters einrichten

Definiert man zu einem Fenster `f1`

```
newObject( def=p:=punkt(  fix( (f1.xMin+f1.xMax)/2 ),
                          fix( (f1.yMin+f1.yMax)/2 )     ) ;
           target = f1); ,
```

dann liegt der Punkt `p` immer (beim Zoomen, Verschieben ,,,) in der graphischen Mitte des Fensters `f1`. Wegen der `fix`-Funktionen kann er nicht mit der Maus verschoben werden.

1To1-Zoom ein/ausschalten

Das 1To1-Zoom stellt auf beiden Achsen gleiche Längeneinheiten her und gewährleistet so eine verzerrungsfreie Darstelltung geometrischer Objekte, z.B. von Kreisen.

`fenster1.zoom1To1(activate=true)` schaltet im Fenster `fenster1` das 1To1-Zoom ein, `fenster1.zoom1To1(activate=false)` schaltet es aus.