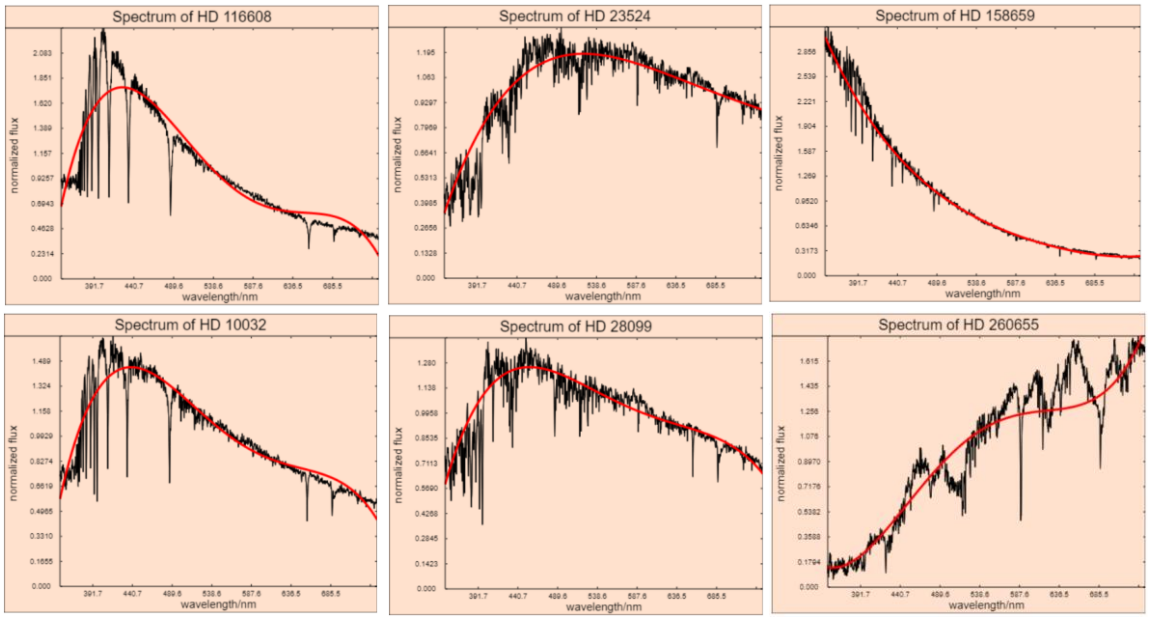
Eckart Modrow

Maschinelles Lernen mit dem Snap! -DataSprite



emodrow@informatik.uni-goettingen.de

Die DataSpriteLibrary und dieses Skript können geladen werden von

http://emu-online.de/DataSpriteLibrary.zip bzw.
http://emu-online.de/MaschinellesLernenMitSnap.pdf

Prof. Dr. Modrow, Eckart:
Maschinelles Lernen mit dem *Snap!* - DataSprite

Wenn Sie mit diesem Buch zufrieden sind und ihre Anerkennung in Form einer Spende zeigen möchten, dann können Sie das auf folgendem PayPal-Konto tun:

emodrow@emu-online.de
Verwendungszweck: ML-Buch

Vorwort

Das vorliegende Skript beschreibt die *DataSpriteLibrary* mit *Snap!*-Blöcken, die für die (relativ) schnelle Verarbeitung großer Datenmengen gedacht ist. „Groß“ sind Datenmengen in Schule und Anfangsausbildung der Universitäten fast nie – weil sie noch vor einiger Zeit kaum frei zur Verfügung standen, und Geld ist im Ausbildungsbereich nun mal knapp. Inzwischen gibt es große Datenmengen aber zuhauf, sei es als Datensammlung im Netz oder eben als Bilddateien, denn die sind auch „groß“. Die Ausbildung hat damit die Chance, mit relevanten Daten umzugehen und damit zahlreiche Berührungspunkte zum Bereich „Informatik und Gesellschaft“ zu finden. Die sind auf lange Sicht unter Allgemeinbildungsaspekten ja wichtiger als irgendwelche Programmiertricks.

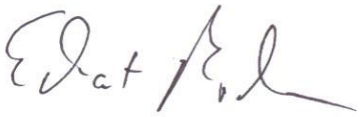
Gerade für Anfänger ist es wichtig, zu „sehen“, was sie mit ihren Programmierversuchen anrichten. Die fantastischen Visualisierungsmöglichkeiten von *Snap!* werden durch die *DataSpriteLibrary* um Bibliotheksfunktionen für Grafiken und Bilder ergänzt, die ebenso wie die *Snap!*-Tabellen schnell das Ergebnis von Operationen anzeigen. Geschwindigkeit ist in diesem Bereich wichtig, weil sie experimentelles Arbeiten im trial-and error-Stil unterstützt. Muss man zu lange warten, dann probiert man nicht so viel aus. Die *DataSpriteLibrary* unterstützt dieses Vorgehen dadurch, dass die meisten zeitkritischen Funktionen in JavaScript implementiert sind. Nebenbei zeigen diese Blöcke auch, wie textbasiertes Programmieren sinnvoll in eine grafische Entwicklungsumgebung integriert werden kann.

Die *DataSpriteLibrary* enthält Blöcke aus dem Bereich der Visualisierung von Daten und dem Umgang mit Tabellen, der durch die Einführung des Datentyps *table* unterstützt wird. Zusätzlich sind Funktionen der linearen Algebra mit den Datentypen *vektor* und *matrix*, die Lösung linearer Gleichungssysteme und das Interpolieren durch Polynome vorhanden. Bildoperationen können über Kernels sowie durch die Vektor- und Matrixoperationen schnell ausgeführt werden. Die Beispiele zeigen, wie das geschehen kann. Sie zeigen aber immer nur einen Weg – erfinden Sie für sich andere und bessere!

Ich bedanke mich sehr bei Jens Mönig und Rick Hessman für ihre Unterstützung und die zahlreichen Diskussionen.

Ansonsten wünsche ich viel Freude bei der Arbeit mit *Snap!* und der *DataSpriteLibrary!*

Göttingen, am 6.8.2019



Inhalt

1 Künstliche Intelligenz und Schule

Der Begriff „Künstliche Intelligenz“ ist derzeit und auf absehbare Zeit mehr als aktuell. In Deutschland wurde das Wissenschaftsjahr 2019 zum „Jahr der künstlichen Intelligenz“ ausgerufen. Der Begriff prägt im Bereich der „Digitalisierung“ die Diskussion in Medien, Wirtschaft und Politik. Zunehmend finden sich zum Thema auch informatische fachdidaktische Beiträge.

In der Schulinformatik ist das Thema eigentlich nicht neu. Schon seit drei bis vier Jahrzehnten gibt es schulgeeignete Beispiele für z. B. zeichenerkennende neuronale Netze (NN), die von den Schülerinnen und Schüler selbst entwickelt und trainiert werden [Baumann] [Modrow1]. Solche Netze sind übersichtlich und gut verständlich, regen zu selbstständigem Arbeiten und danach zu auf fachlichen Erfahrungen beruhenden Diskussionen z. B. über die philosophischen Implikationen an [Modrow2]. Vor allem aber sind sie klein. Genau darin liegt der Unterschied zu den aktuellen NNs: die sind groß. Laut Ian Goodfellow [Deep Learning], einem der führenden Entwickler auf diesem Gebiet, hat sich grundsätzlich nichts gegenüber den alten kleinen Netzen geändert. Struktur und Methoden sind (fast) gleichgeblieben, aber natürlich verbessert worden. Geändert haben sich die Leistungsfähigkeit der Computer, auf denen die NNs laufen, und die Menge der vorhandenen Daten, mit denen sie trainiert werden können. Damit bleiben aber ältere Erkenntnisse gültig wie z. B. das aus 1967 von Marvin Minsky [Minsky] über die Äquivalenz von NNs und endlichen Automaten. Das ist auch kein Wunder, denn das Modell der endlichen Automaten hat seine Wurzeln eben in den ersten NNs.

Die Vorschläge zur Behandlung großer NNs im Unterricht bestehen oft darin, dass fertige NNs mithilfe fertiger Trainingsdaten trainiert werden. Die Schülerinnen und Schüler können dann zusehen, wie das Netz lernt, also langsam seine Ergebnisse verbessert. Eigentlich benötigt man für diese Erfahrung kein reales NN, ein Video genügte auch. Man kann ja nicht sehen, dass das Netz groß ist, und aus dem Zusehen auch nicht erschließen, weshalb diese Größe von Bedeutung ist. Man sieht nur, dass sich die Ergebnisse verbessern. Über NNs lernt man aus dieser Erfahrung allein nichts. Eine Diskussion der Auswirkungen der NNs beruht dann auf der Information, dass es sie gibt und dass sie lernen können. Weitere fachliche Grundlagen fehlen, sodass diese Diskussion ebenso gut in anderen Fächern stattfinden könnte.

Vergleichen wir die Situation einmal mit einem Beispiel aus der Physik. Das relativ neue Bild eines schwarzen Lochs [SZ] zeigt, dass es schwarze Löcher gibt und dass sie anscheinend Materie „verschlucken“. Diese Information allein integriert das Thema aber nicht in den Fachunterricht Physik, denn eine fachliche Behandlung schwarzen Löcher liegt weitgehend außerhalb der Möglichkeiten der Schule. Innerhalb eines Themenbereichs „Gravitation“, der zahlreiche Aktivitäten, historische und gesellschaftliche Bezüge, fachliche Probleme der Schulphysik usw. enthält, verknüpft das Bild aber die Schulphysik mit der „Wissenschaft nach der Schule“, zeigt Wege zu einer tiefgreifenderen Beschäftigung damit und regt z. B. zum Nachdenken darüber an, ob die Lernenden eine persönliche Perspektive auf diesem Gebiet sehen – oder eben nicht.

Was lernen wir daraus?

Das reine Vorstellen neuer Technologien hat in der Schule eigentlich nichts zu suchen – für Shows gibt es andere Kanäle. Die reine Information, dass es solche Technologien gibt, genügt ebenfalls nicht, das Thema einem bestimmten Fach zuzuordnen. Im Gegenteil: beschränkt man sich darauf, dann wäre es besser Fächern anzusiedeln, in denen z. B. die gesellschaftlichen oder philosophischen Auswirkungen diskutiert werden und das Thema so mit anderen Aspekten vernetzt wird. Erst die fachdidaktische Reduktion einer Fragestellung auf eine Komplexitätsebene, auf der in der Schule möglichst selbstständig und ideenreich von den Lernenden gearbeitet werden kann, macht das Thema pädagogisch fruchtbar.

> *Im Bereich der Künstlichen Intelligenz gehört damit nicht das passive Beobachten des Lernens der Netze in die Schule, sondern das aktive Fördern des Verständnisses der menschlichen Lernenden für die Grundlagen und Implikationen dieses Prozesses.*

Dazu noch eine Anmerkung: Konzentriert sich die Schule auf das Vermitteln von Fakten und Daten sowie das Einüben der Anwendung von Kalkülen, dann setzt das m. E. die Vorstellung voraus, dass die Lernenden nicht in der Lage sind, Zusammenhänge und Hintergründe selbst zu entdecken und zu verstehen. Das Vorgehen fördert also die Unmündigkeit. Und schlimmer noch: es hält die Lernenden in der Unmündigkeit, weil sie eines ganz sicher dabei lernen: dass ihnen eigenständiges Denken nicht zugetraut wird.

2 Maschinelles Lernen

Der Begriff „Maschinelles Lernen“ wird oft als Synonym für „Künstliche Intelligenz“ oder „Neuronale Netze“ benutzt. Diese Einschränkung stimmt aber nicht. Präziser ist z. B. die Definition, die sich auf der SAP-Seite [SAP] findet:

> *Die Technologie des maschinellen Lernens lehrt Computern die Ausführung von Aufgaben durch Lernen aus Daten, anstatt für die Aufgaben programmiert zu werden.*

Das „Lernen aus Daten“ kann dabei als Anpassung der Parameter einer Funktion verstanden werden. Einer Maschine wird ein Datensatz (Bild, Tabelle, Zeichenfolge, …) als Eingabevektor $\boldsymbol{E}$ präsentiert. Sie berechnet daraus einen Ausgabewert k, der die Eingabe einer Kategorie („Es handelt sich um eine Katze“, „Merkmal vorhanden“ (oder nicht), „das Wort ‚Auto‘“, …) zuordnet.

$$f(\boldsymbol{E}) = k$$

Diese Zuordnung kann auf sehr unterschiedliche Weise erfolgen. Man kann z. B. die Parameter eines Polynoms anpassen, ähnliche Eingabewerte suchen („k-nächste-Nachbarn“), mit Entscheidungsbäumen arbeiten, Bayes-Filter einsetzen, … – oder eben ein NN trainieren. Alle diese Verfahren haben die Gemeinsamkeit, dass die „Maschine“ einen Satz von Parametern enthält, die änderbar sind. Die Maschine „lernt aus Daten“, indem sie wiederholt einen Datensatz einliest, aus diesem mithilfe des aktuellen Parametersatzes den Ausgabewert berechnet und danach nach irgendeinem Verfahren diese Ausgabe mit dem „gewünschten“ Ausgabewert vergleicht. Gibt es da eine Abweichung, dann ändert sie die Parameter so, dass die Ausgabe dem „gewünschten“ Wert zumindest näherkommt. „Gewünschte“ Werte können schon vorher bekannt sein („Das Bild ist ein Katzenbild“), von außen z. B. von einem „Trainer“ kommen („überwachtes Lernen“) oder von der Maschine selbst generiert werden („nicht überwachtes Lernen“), z. B. indem sie Merkmale aus vielen Trainingsdaten extrahiert („Clustering“). In allen Fällen „lernt“ die Maschine gar nichts, sondern sie passt Parameter nach einem vorgegebenen Verfahren an.

Auch dieses Vorgehen ist seit langem in den Schulen verbreitet. „Lernende“ Nimm-Spiele usw. finden sich schon in den ersten Informatik-Schulbüchern. Neu ist wiederum der Umfang der erforderlichen Trainingsdaten. Ein großes NN kann über Milliarden von Parametern verfügen, die trainiert werden müssen – und dafür sind dann auch „sehr viele“ Trainingsdaten vonnöten. Neu ist auch, dass diese Daten im Netz zur Verfügung stehen. Wenn also „umsonst“ zur Verfügung stehende Anwendungen „mit Daten“ bezahlt werden, dann wissen wir jetzt auch, wie und weshalb das geschieht.

Sichtet man gängige Lehrbücher zum maschinellen Lernen [Grus] [Albon], dann findet man dort folgerichtet gar nicht so viel über NNs, aber sehr viel über den Umgang mit Daten. Diese müssen z. B. normalisiert werden, um die vielen Eingabedaten, die aus sehr verschiedenen Quellen stammen können, kompatibel zu machen. Fotografieren wir z. B. viele Hunde mit einer älteren Digitalkamera und viele Katzen mit einer neueren, dann würde ein NN aus diesen Bildern sehr wahrscheinlich lernen, dass Hundebilder kleiner als Katzenbilder sind.

Die Aufbereitung von Daten ist nun eine sehr handwerkliche Tätigkeit. Sie kann schrittweise erfolgen, erprobt und danach mit einfachen Algorithmen automatisiert werden. Die Erprobung wird stark erleichtert, wenn die Struktur der Daten leicht zu visualisieren, also z. B. in Tabellen oder als Graph darstellbar ist. Und Algorithmen sind einfach, wenn sie über eine übersichtliche Struktur verfügen, also z. B. nach einigen Vorbereitungsschritten aus einer Schleife bestehen, in der einige Alternativen mit den entsprechenden Anweisungen aufgezählt werden. Die Mächtigkeit der entwickelten Skripte hängt dann nicht so sehr von der algorithmischen Struktur wie von der Mächtigkeit der zur Verfügung stehenden Befehle ab. Oder umgekehrt: wenn man über genügend mächtige Befehle verfügt, dann kann man auch mit einfachen Programmen viel ausrichten. Die Anpassung der Parameter kann danach auf eine der gängigen Arten erfolgen. Stehen also entsprechende Werkzeuge zur Verfügung, dann ist die Aufbereitung von Daten ein sehr schulgeeignetes Thema. Das *DataSprite* ist als solch ein Werkzeug gedacht.

3 Der Aufbau des DataSprites

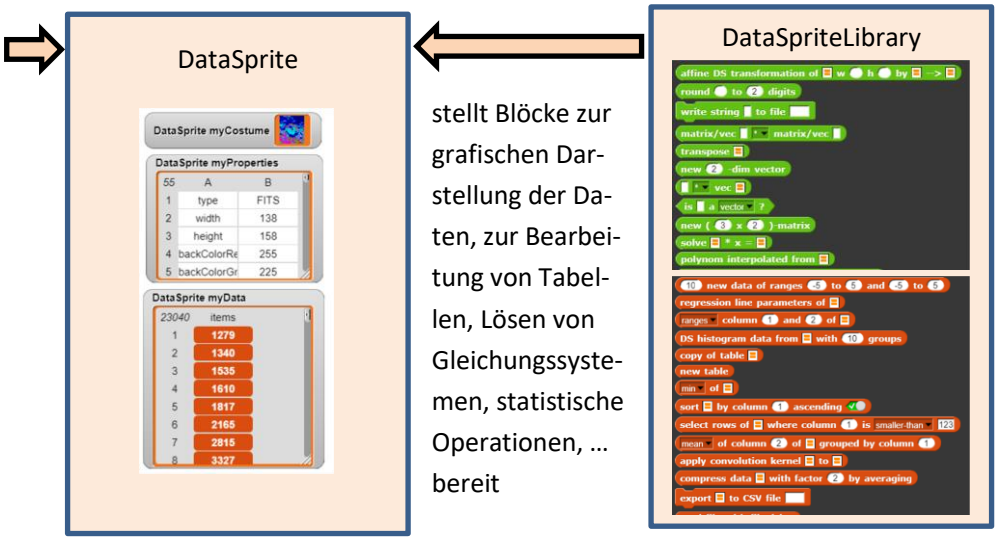
Der Aufbau des *DataSprites* orientiert sich an der Idee von dokumentierten Datensätzen, die aus zwei Teilen bestehen: den *Metadaten*, die die Struktur und den inhaltlichen Kontext der Daten beschreibt (z. B. Zahlenformat, Bilddimensionen, Aufnahmegerät, Aufnahmedatum, ...) und den dazugehörigen reinen *Datensegmenten*. Metadaten bestehen gewöhnlich aus Dictionaries - Namen mit zugewiesenen Werten (z. B. "Aufnahmedatum: 24.12.2018"). Beispiele für diese Struktur sind FITS-Dateien [FITS], die in der Astrophysik Standard sind, aber auch in der Vatikanischen Bibliothek Verwendung finden, oder JPEG Bilder vom Handy. Auch hier gibt es Metadaten (Bildgröße, Kompressionsgrad, Aufnahmedatum, oft auch GPS Koordinaten), ohne die eine Bilderzeugung nicht möglich wäre. Wichtig ist, dass die Bilderzeugung die Originaldaten nicht verändert.

Wir adaptieren diese Struktur, indem wir einem *DataSprite* drei lokale Variable verpassen, die die Daten (*myData*), die Datenbeschreibung (*myProperties*) und das aktuelle Kostüm (*myCostume*) enthalten. Diese Variablen können einerseits durch den Import von Daten aus unterschiedlichen Quellen (SQL-Abfragen, Textdatei, CVS-Datei, JSON-Datei, FITS-Datei, direkte Zuweisung, …) gefüllt werden, wobei die Eigenschaften *myProperties* den jeweiligen Daten anzupassen sind. Mithilfe dieser Eigenschaften können Daten in grafische Darstellungen (Graph, Datenplot, Histogramm, Bild, …) umgesetzt werden, wobei als Quelle entweder *myData* oder eine andere geeignete Tabelle gewählt wird. Weil Tabellen in *Snap!* sehr schön dargestellt werden können, ist diese Darstellungsform nicht zusätzlich implementiert. Dafür ist der Datentyp *table* mit zahlreichen der im Bereich von *Data Science* üblichen Operationen (Tabellenoperationen, Korrelationsberechnung, affine Transformationen, Lösen linearer Gleichungssysteme, …) implementiert, der ausreichend schnell auch mit größeren Datenmengen umgehen kann.

Insgesamt ergibt sich die folgende Struktur:

importiert Daten aus …

- FITS-Dateien
- Text-Dateien
- SQL-Abfragen
- JSON-Dateien
- CSV-Dateien
- …

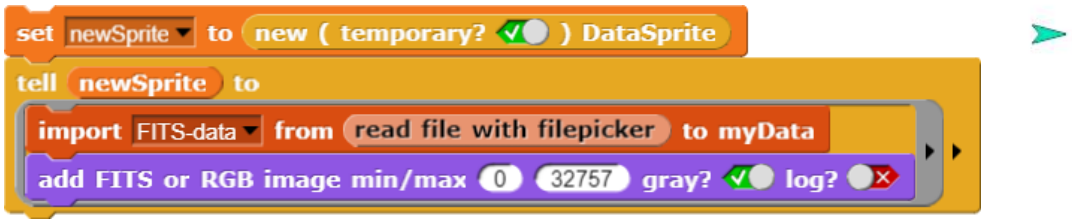


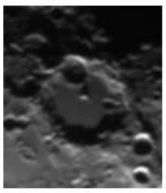
4 Arbeiten mit der DataSpriteLibrary

4.1 Erzeugen von DataSprites

Wenn wir mit Daten und Grafiken arbeiten, bietet es sich an, mehrere *DataSprites* gleichzeitig zu verwenden. Haben wir schon eins, dann können wir durch Klonen oder Kopieren weitere erzeugen. Wir kopieren damit natürlich auch die Überreste der vorhergehenden Arbeiten (Variable, neue Blöcke, …), und vielleicht fehlen die drei lokalen Variablen des Sprites, sind anders benannt, … Es gibt viele Fehlermöglichkeiten. Um die zu vermeiden, gibt es einen neuen Reporter-Block *new (temporary?) DataSprite* in der Commands-Palette. Er erzeugt ein neues Sprite, entweder dauerhaft (im Sprite-Coral sichtbar) oder temporär (es wird automatisch gelöscht beim Drücken des roten Buttons oder beim Schließen von *Snap!*). Der Block ist ein Reporter, weil sehr oft auf das neue Sprite von außen zugegriffen werden muss. Man sollte sich eine Referenz darauf in einer Variablen speichern.

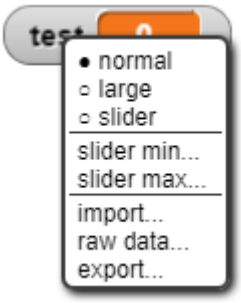
Beispiel: Eine FITS-Datei (Quelle: [HOU]) wird eingelesen und auf einem neuen DataSprite dargestellt.





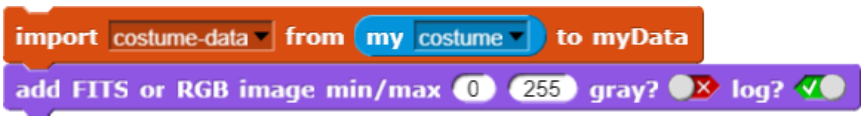
4.2 Importieren von Daten

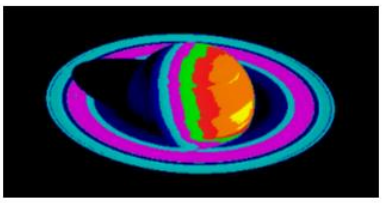
Snap! kann eine Reihe von Datenformaten direkt importieren. Das kann geschehen, indem man entsprechende Dateien auf das *Snap!*-Fenster „fallen“ lässt oder sie durch Rechtsklick auf einen Variablen-Watcher importiert. Beides klappt gut mit Text-, CSV- und JSON-Dateien. Andere Text-Dateiformate wie FITS kann man ebenfalls so importieren, wobei nachgefragt wird, ob man es ernst meint. Das Exportieren funktioniert auf die gleiche Art. Will man dasselbe programmgesteuert machen, dann benutzt man den Reporter-Block *read file with filepicker*. Es erscheint ein Dateimanager-Fenster, in dem man die Datei wie üblich auswählt. Danach werden die Daten importiert.



Als wesentliche Aufgabe bleibt anschließend, diese Daten der *myData*-Variablen zuzuweisen und die entsprechenden Eigenschaften in *myProperties* zu setzen. Das wird von dem folgenden Block erledigt, der Daten von außen in den *myData*-Bereich importiert. Dabei kann es sich um FITS-Daten, Tabellendaten oder die Daten des aktuellen Kostüms handeln. Dieses wird als Tabelle von RGB-Werten gespeichert.

Beispiel: Ein Bild (Quelle: [NASA]) wird gespeichert und mit Falschfarben neu dargestellt.





Beispiel: Knapp 600000 Datensätze aus einer CSV-Datei werden in etwa 10 Sekunden eingelesen. Die Eigenschaften werden gesetzt.

DataSprite myProperties

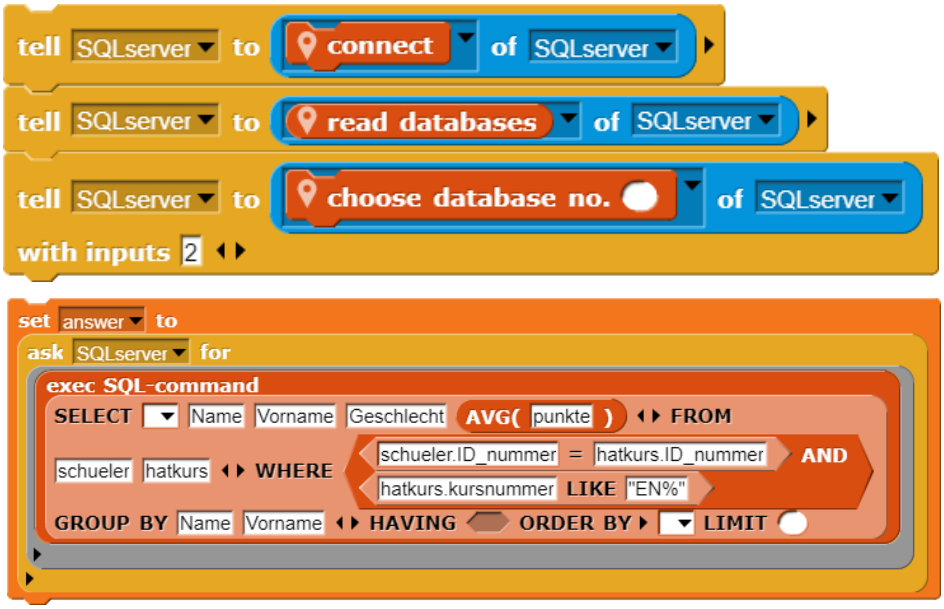
34	A	B
1	type	table
2	width	0
3	height	0
4	backColorRe	255
5	backColorGr	225

DataSprite myData

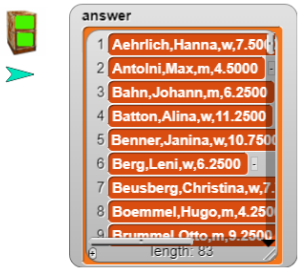
577704	A	B
1	tripduration	starttime
2	695	2013-06-01 (201
3	693	2013-06-01 (201
4	2059	2013-06-01 (201
5	123	2013-06-01 (201

Beispiel: SQL-Import

Haben wir Zugang zu einem SQL-Server, dann können wir auch von dort Daten einlesen. In unserem Fall importieren wir ein *SQL-Server-Sprite* sowie die Bibliothek mit den SQL-Blöcken [SQL] in ein *DataSprite*. Danach bitten wir den SQL-Server, die Verbindung zu einer Schuldatenbank aufzubauen.



Aus dieser möchten wir die Namen, das Geschlecht und die mittlere Englischnote abfragen. Damit erhalten wir eine Liste von Zeichenketten, die diese Daten enthalten – durch Kommas getrennt. Wir müssen die Listeneinträge also noch in Teillisten umformen und in eine Tabelle eintragen. Diese können wir dann im Sprite importieren.



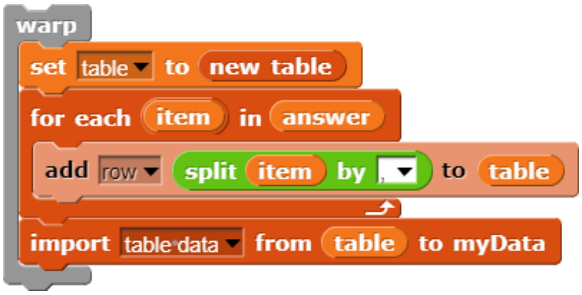
answer

1	Aehrlich,Hanna,w,7.500
2	Antolni,Max,m,4.5000
3	Bahn,Johann,m,6.2500
4	Batton,Alina,w,11.2500
5	Benner,Janina,w,10.750
6	Berg,Leni,w,6.2500
7	Beusberg,Christina,w,7.
8	Boemmel,Hugo,m,4.250
9	Brummel,Otto,m,9.2500

length: 83

table

83	A	B	C	D
1	Aehrlich	Hanna	w	7.5000
2	Antolni	Max	m	4.5000
3	Bahn	Johann	m	6.2500
4	Batton	Alina	w	11.2500
5	Benner	Janina	w	10.7500
6	Berg	Leni	w	6.2500
7	Beusberg	Christina	w	7.7500
8	Boemmel	Hugo	m	4.2500
9	Brummel	Otto	m	9.2500
10	Brummer	Steffi	w	6.0000



DataSprite myData

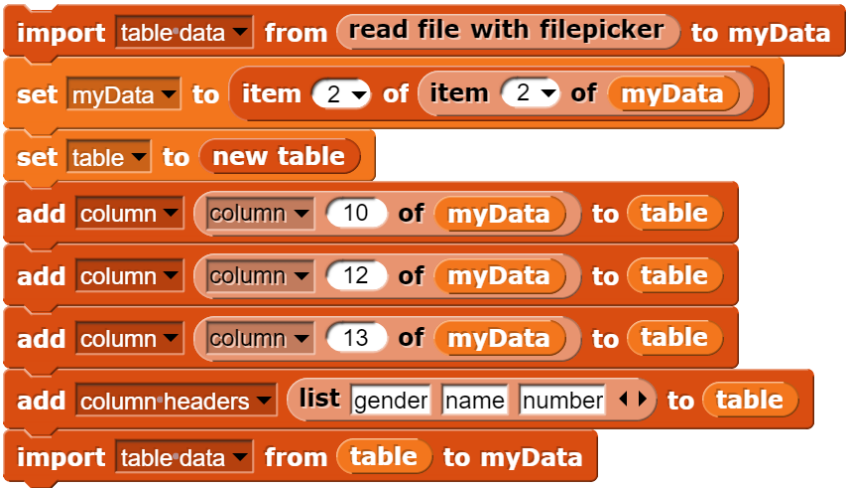
83	A	B	C	D
1	Aehrlich	Hanna	w	7.5000
2	Antolni	Max	m	4.5000
3	Bahn	Johann	m	6.2500
4	Batton	Alina	w	11.2500
5	Benner	Janina	w	10.7500
6	Berg	Leni	w	6.2500
7	Beusberg	Christina	w	7.7500
8	Boemmel	Hugo	m	4.2500

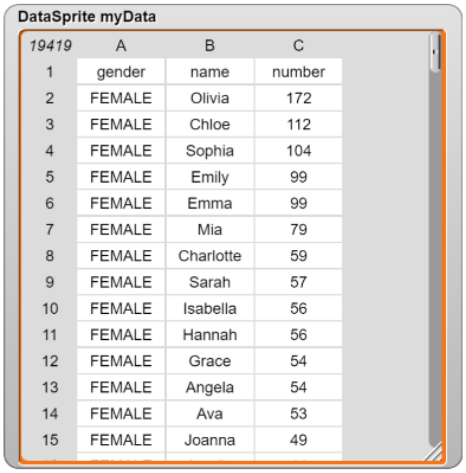
DataSprite myProperties

34	A	B
28	scalesPrecis	3
29	scalesTexthe	12
30	scalesNumb	10
31	minValue	not set
32	maxValue	not set
33	columns	4
34	rows	83

Beispiel: JSON-Import

Der einfachste Weg ist auch hier, ein JSON-Datei einfach ins *Snap!*-Fenster „fallen“ zu lassen. Es geht aber auch automatisiert. Zuerst einmal suchen wir uns interessante JSON-Daten und wählen dafür natürlich die Statistik der Baby-Namen in New York City – was sonst. Der geeignete Block dafür ist wieder *import <table data> from <read file with filepicker> to myData*. Das Ergebnis ist eine List mit zwei Spalten und zwei Reihen, den Metadaten und den eigentlichen Daten. Weil wir uns für die interessieren, ersetzen wir die Originaldaten durch das Element (2|2) der Tabelle. Von den vielen Spalten kopieren wir die drei interessanten in eine neue Tabelle, fügen Spaltenüberschriften hinzu und importieren das Ergebnis wieder in *myData*.




DataSprite myData

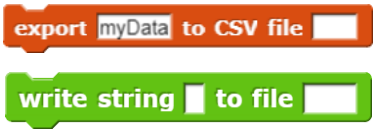
19419

	A	B	C
1	gender	name	number
2	FEMALE	Olivia	172
3	FEMALE	Chloe	112
4	FEMALE	Sophia	104
5	FEMALE	Emily	99
6	FEMALE	Emma	99
7	FEMALE	Mia	79
8	FEMALE	Charlotte	59
9	FEMALE	Sarah	57
10	FEMALE	Isabella	56
11	FEMALE	Hannah	56
12	FEMALE	Grace	54
13	FEMALE	Angela	54
14	FEMALE	Ava	53
15	FEMALE	Joanna	49

Das Ergebnis: 19419 Babynamen

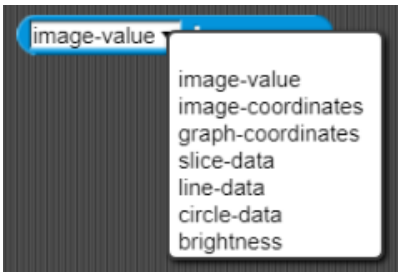
Wer hätte das gedacht!

4.3 Exportieren von Daten



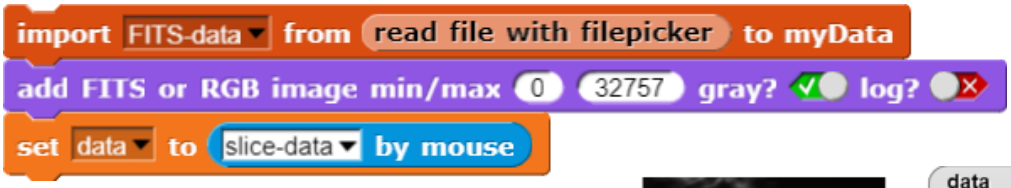
Der Export kann wiederum direkt aus einem Variablen-Watcher geschehen. Für Skripte gibt es zwei neue Blöcke *export <table> to CSV file <filename>* sowie *write string <string> to file <filename>*. Die Ergebnisse landen wie in *Snap!* üblich jeweils im Download-Ordner des Browsers. Die beiden Blöcke gestatten es, den Datenaustausch mit Tabellenkalkulationsprogrammen bzw. über Textdateien zu automatisieren, beispielsweise um Ergebnisse der Datenverarbeitung zu sichern.

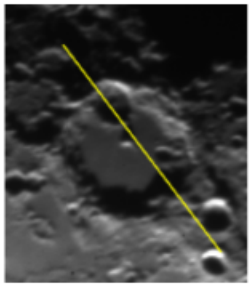
4.4 Datenimport über die Maus



In vielen Fällen ist es vorteilhaft, Daten mithilfe der Maus einzulesen. Dafür steht in der Sensing-Palette der Block *<…> by mouse* zur Verfügung, mit dem Bildwerte, Bildkoordinaten, Koordinaten im benutzen Koordinatensystem für Graphen und/oder Datenpunkte, die Daten auf einem Schnitt durchs Bild, Anfangs- und Endpunkt einer Linie, Mittelpunkt und Radius eines Kreises und die summierten Bildschirmwerte zusammen mit deren Zahl in einem Kreis bestimmt werden können. Als Beispiel wird ein Schnitt durch ein Mondbild gezeigt.

Beispiel: Slice durch ein Bild (Quelle: [HOU])

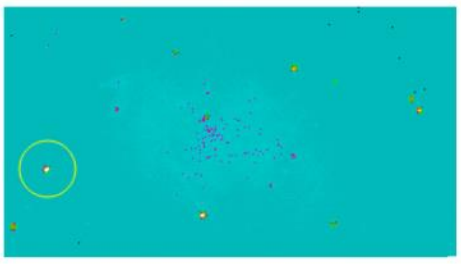


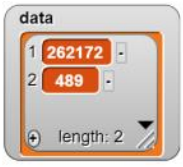


data

149	A	B
1	0	2047
2	1	1877
3	2	2303
4	3	1791
5	4	2901
6	5	3182

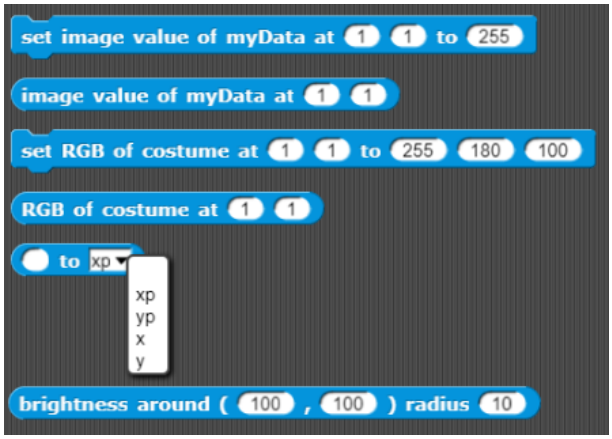
Beispiel: Messen der Bildwertsumme im Umkreis eines Pixels ein Bild (Quelle: [HOU])





4.5 Weitere Blöcke in der Sensing-Palette

Mit den folgenden Blöcken kann der Bildwert an einer Position bzw. der entsprechende Pixelwert des Kostüms gelesen oder gesetzt werden. Mit dem vorletzten Block erfolgen Umrechnungen zwischen den Koordinatensystemen. Die Helligkeit im Umkreis eines Punktes kann mit dem letzten Block ermittelt werden.

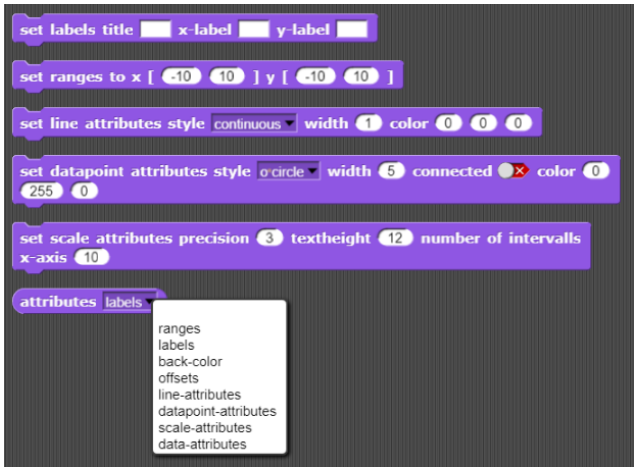


4.6 Die Properties des DataSprites

In der Looks-Palette finden sich einige Blöcke zur Darstellung von Daten auf dem *DataSprite*. Sie benutzen dabei die Einstellungen von *myProperties* über Wertebereiche, Farben, Größen und Linienarten, Beschriftung der Achsen usw. Diese können mit dem *set properties*-Block auf Anfangswerte gesetzt und als *Snap!*-Tabelle direkt angezeigt werden. Ist man mit diesen zufrieden (wie in den bisherigen Beispielen), dann können Grafiken direkt auf das *DataSprite* gezeichnet werden. Sonst muss man die Einstellungen ändern. Das kann direkt mit den beiden Blöcken zum Lesen von Property-Werten *property <eintrag>* und zum Schreiben *set property <eintrag> to <wert>* geschehen. Ein typischer Anwendungsfall wäre das Eintragen des Bereichs der Bildwerte – wenn das nicht schon automatisch geschehen ist.

DataSprite myProperties

34	A	B
1	type	empty
2	width	0
3	height	0
4	backColorRed	255
5	backColorGreen	225
6	backColorBlue	205
7	leftOffset	60
8	upperOffset	0
9	lowerOffset	20
10	title	
11	xLabel	
12	yLabel	
13	xLeft	-10
14	xRight	10
15	yLower	-10
16	yUpper	10
17	lineStyle	continuous

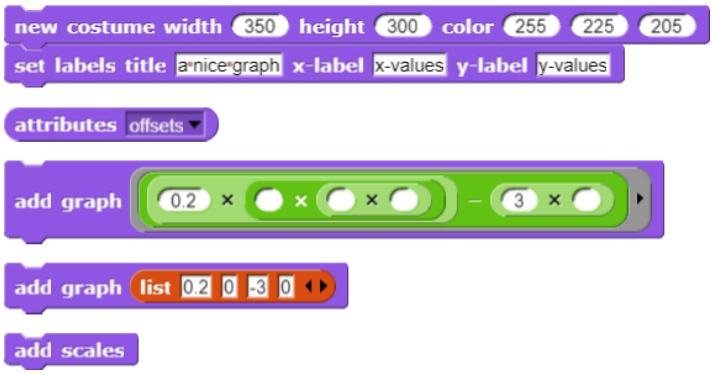


Etwas komfortabler kann das mit den Blöcken geschehen, die Gruppen von Eigenschaften zusammenfassen – zum Setzen oder Lesen. Das zweite vereinfacht den Aufruf von JavaScript-Funktionen, weil die Zahl der Parameter etwas begrenzt wird.

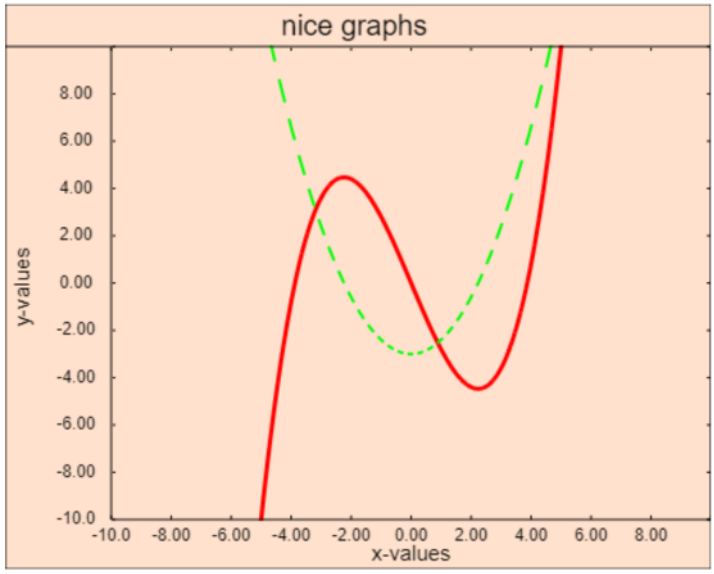
4.7 Grafiken auf dem DataSprite

Grafiken können auf *DataSprites* „überlagernd“ gezeichnet werden. Als Beispiel kann ein Datenplot dienen, für den man eine Näherungsfunktion sucht. Mit der kann man solange experimentieren, bis man zufrieden mit dem Ergebnis ist. Ist das Bild „voll“, zeichnet man einfach neu.

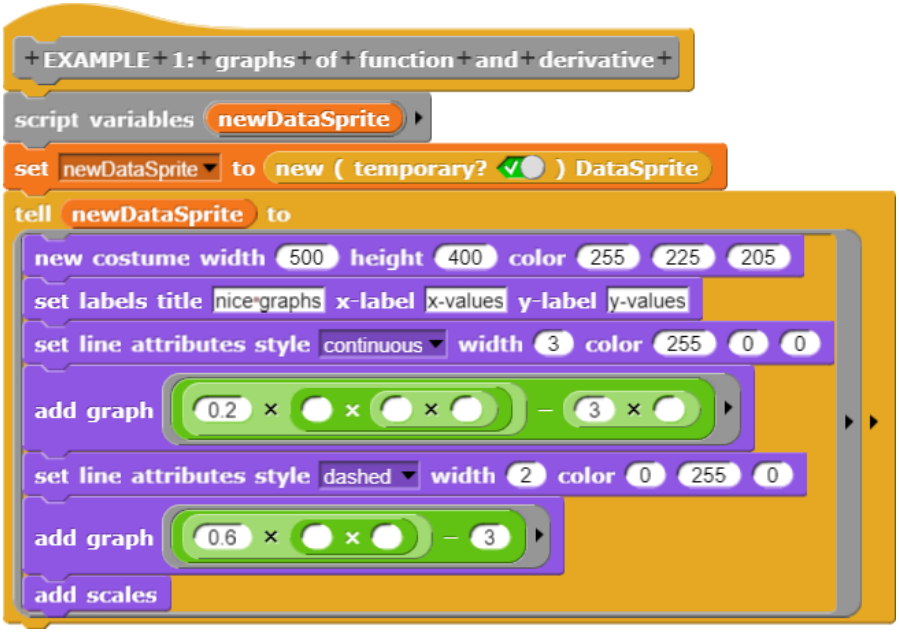
Zuerst einmal benötigt man ein *DataSprite*. Größe und Hintergrundfarbe kann man einstellen, damit alle zufrieden sind. Wenn wir schon mal dabei sind, geben wir auch einen Diagrammtitel sowie die Achsenbeschriftungen vor. Da diese Platz auf dem Diagramm benötigen, werden die *offsets* so eingestellt, dass die reine Diagrammfläche etwas reduziert wird. Bei Bedarf können wir sie uns auch mit *attributes <offsets>* anzeigen lassen.



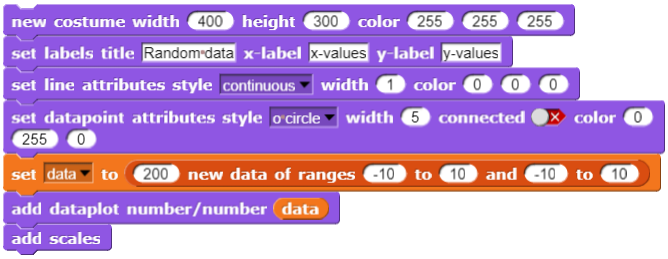
Als nächstes benutzen wir den Block *add graph <term>* zum Zeichnen eines Funktionsgraphen. Als Term können wir entweder einen *Snap!*-Operator (*ringified*, damit er nicht vor dem Aufruf ausgeführt wird!) oder die Koeffizientenliste eines Polynoms übergeben. Weitere Graphen – hier: die Ableitung – können folgen. Das Zeichnen der Achsen und der Beschriftungen übernimmt der Block *add scales*.

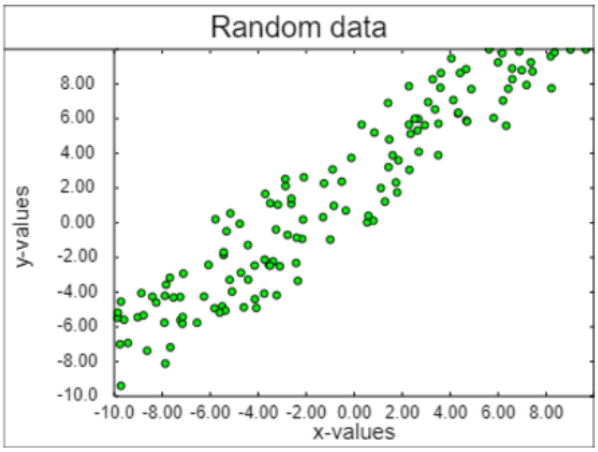


Natürlich können wir stattdessen auch ein neues *DataSprite* erzeugen und diesem mitteilen, was es zeichnen soll.

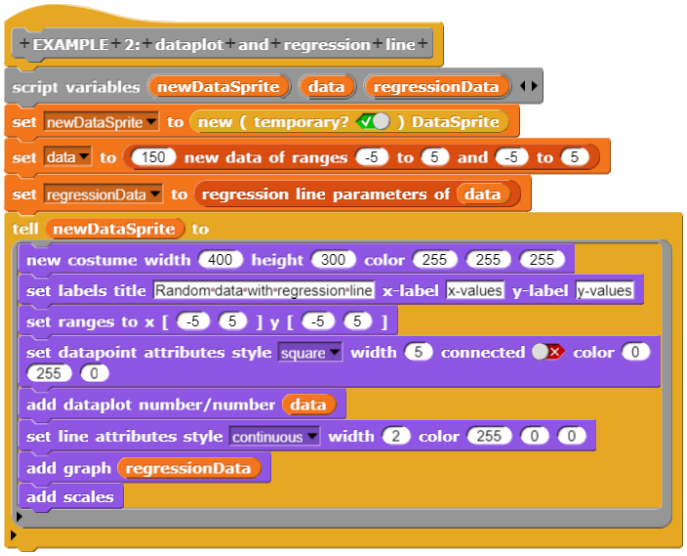


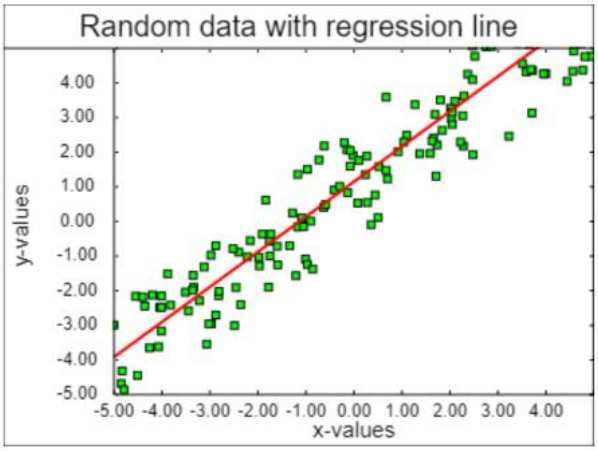
Wenn wir die Inhalte einer Datentabelle grafisch darstellen wollen, kann das mit dem Block *add dataplot number/number* geschehen. Skalen und Beschriftung werden wieder mit dem Block *add scales* ergänzt. Die Art der Darstellung kann mit dem Block *set datapoint attributes …* sehr genau eingestellt werden.





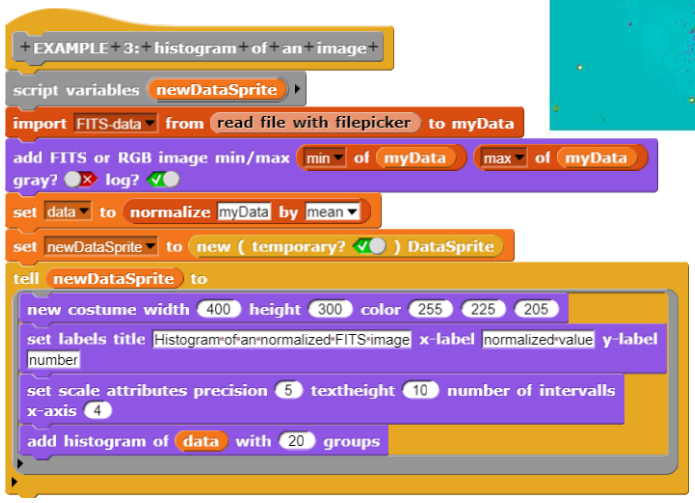
Beispiel: Ein Datensatz wird mit der zugehörigen Regressionsgerade in einem *DataSprite* dargestellt.

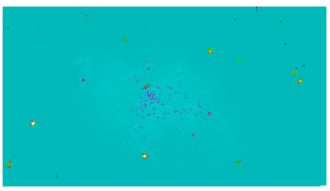


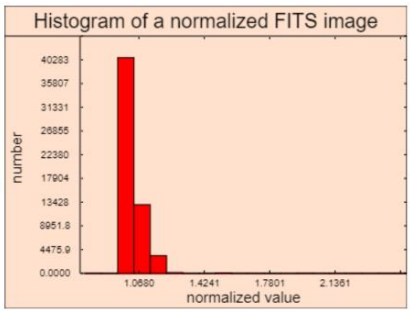


Histogramme lassen sich direkt aus Datenquellen erzeugen und darstellen.

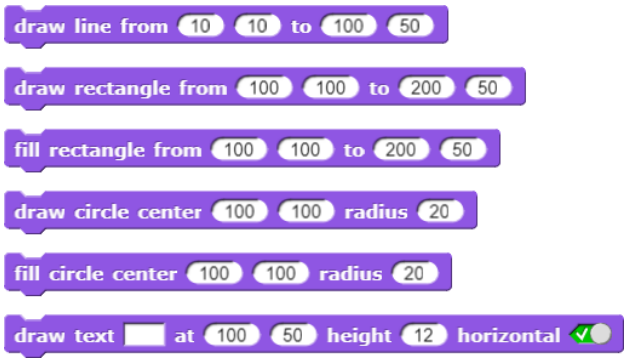
Beispiel: Ein FITS-Bild wird geladen, die normalisierte Verteilung der Bildwerte wird als Histogramm auf einem neuen *DataSprite* dargestellt.





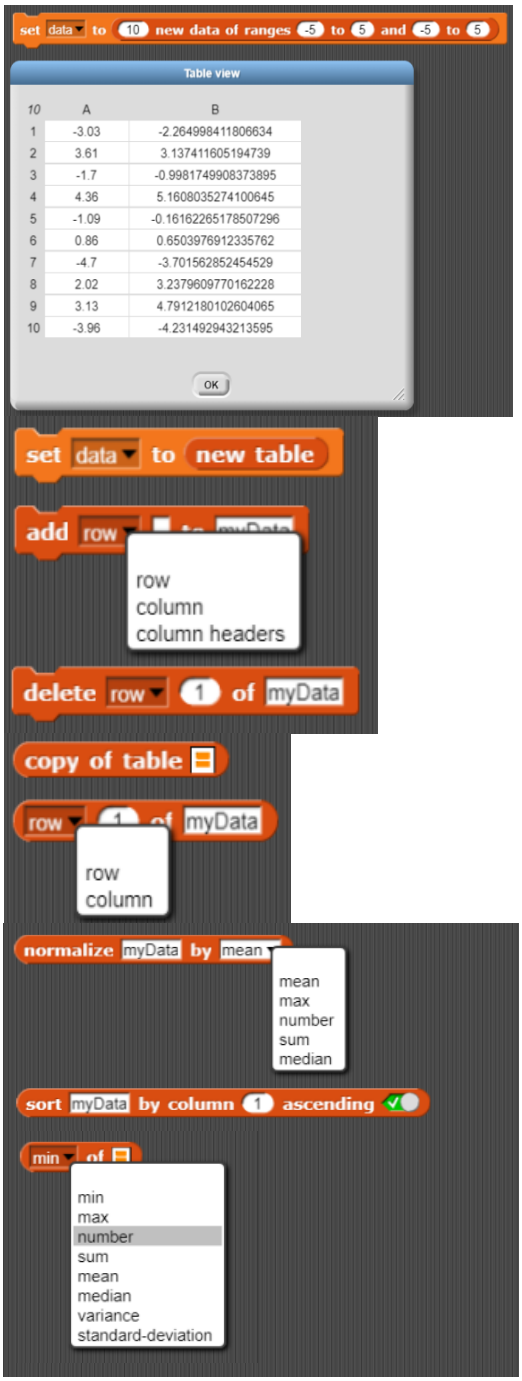


Zusätzlich finden sich in der Looks-Palette fünf Blöcke für das Zeichnen von Grundstrukturen auf der Sprite-Oberfläche (nicht auf der Bühne).



4.8 Arbeiten mit DataSprite-Tabellen

Weil *Snap!* Tabellen (Listen von Listen) so gut darstellen kann (*Rechtsklick auf den Variable-Watcher, open in dialog, Spaltenbreite durch Ziehen bei gedrückter Maustaste in der Spaltenüberschrift anpassen*), lassen sich importierte, veränderte, anders erzeugte, … Tabellen sofort anzeigen. Die Ergebnisse der Arbeit mit Tabellen können sofort überprüft werden – eine sehr wichtige Eigenschaft für das interaktive Arbeiten mit Daten. Das *DataSprite* enthält deshalb einen Datentyp *table*, der einer solchen zweidimensionalen Tabelle entspricht.

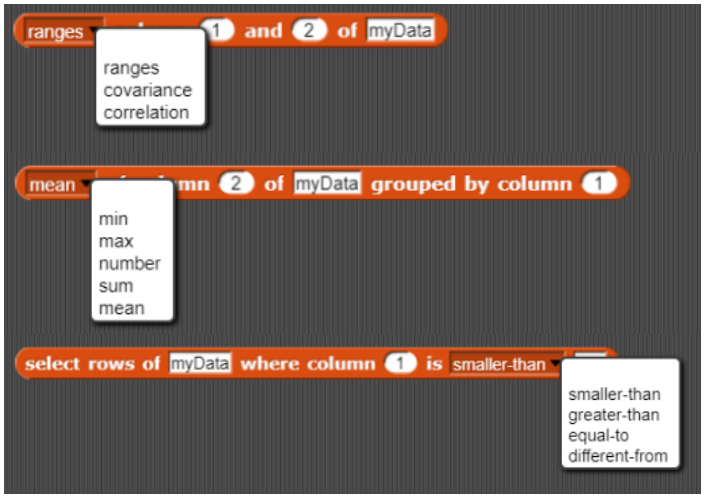


Eine neue leere Tabelle ist einfach eine leere Liste. Dieser können Zeilen, Spalten und Spaltenüberschriften hinzugefügt werden, und man kann diese auch wieder löschen. Bei Bedarf kann man eine zweispaltige Tabelle aus Zufallszahlen direkt erzeugen (s.o.). In vielen Fällen ist *myData* als zu bearbeitende Tabelle voreingestellt. Diesen Eintrag kann man durch Einfügen einer Tabellenvariable überschreiben.

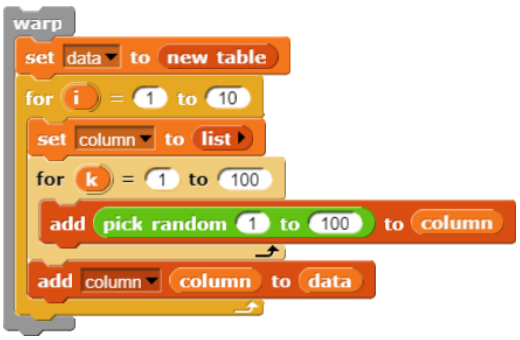
Wenn man eine echte Kopie, also Daten ohne Referenzen auf andere Daten, benötigt, dann kann man das durch *copy of table <tablename>* erreichen. Ebenso kann man einzelne Zeilen oder Spalten einer Tabelle auslesen.

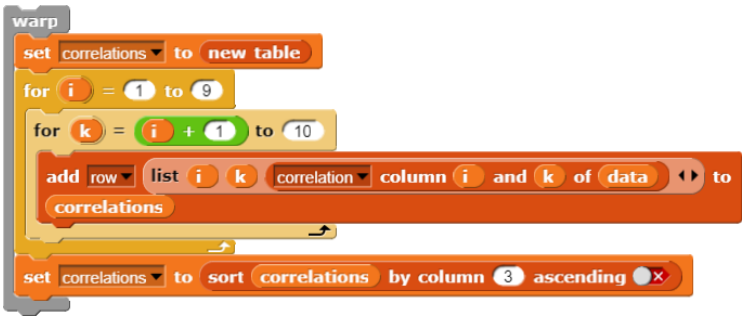
Mit einzelnen Tabellenspalten kann man auch einiges anstellen: sie können z. B. normalisiert werden, indem man z. B. alle Einträge durch den Mittelwert teilt, man kann sie aufsteigend oder absteigend sortieren sowie Minimum, Maximum, Anzahl, Summe, Mittelwert, Median, Varianz und Standardabweichung der Tabellenwerte schnell berechnen. Benötigt man Zufallspaare, die um einen vorgegebenen Funktionsgraphen streuen, dann kann man den Block *<n> random points near <operator> between <xmin> and <xmax>* benutzen.

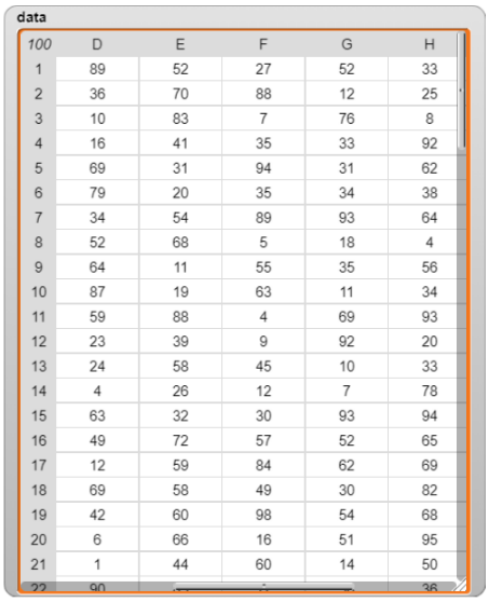
Die Bezüge zwischen mehreren Tabellenspalten sind natürlich auch von Interesse. Für zwei Tabellenspalten lassen sich die Wertebereiche, die Kovarianz und die Korrelation berechnen. Einige Standardwerte lassen sich nach den Einträgen einer anderen Spalte gruppieren und Tabellenzeilen mit vorgegebenen Eigenschaften lassen sich auswählen.



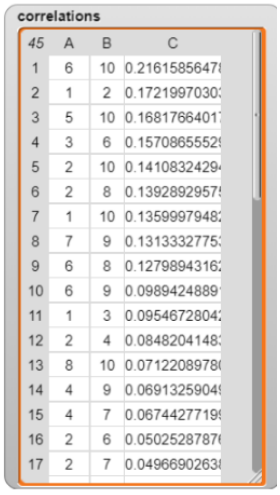
Beispiel: Eine Tabelle mit 100 Reihen und 10 Spalten wird mit Zufallszahlen gefüllt. Anschließend soll festgestellt werden, zwischen welchen Spalten die höchste Korrelation besteht.






data

100	D	E	F	G	H
1	89	52	27	52	33
2	36	70	88	12	25
3	10	83	7	76	8
4	16	41	35	33	92
5	69	31	94	31	62
6	79	20	35	34	38
7	34	54	89	93	64
8	52	68	5	18	4
9	64	11	55	35	56
10	87	19	63	11	34
11	59	88	4	69	93
12	23	39	9	92	20
13	24	58	45	10	33
14	4	26	12	7	78
15	63	32	30	93	94
16	49	72	57	52	65
17	12	59	84	62	69
18	69	58	49	30	82
19	42	60	98	54	68
20	6	66	16	51	95
21	1	44	60	14	50
22	90				

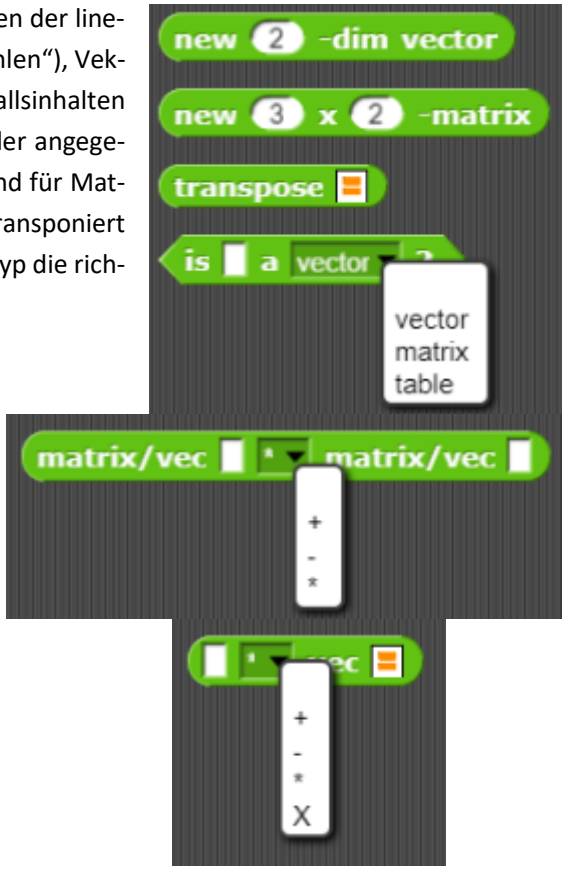

correlations

45	A	B	C
1	6	10	0.2161585647
2	1	2	0.1721997030
3	5	10	0.1681766401
4	3	6	0.1570865552
5	2	10	0.1410832429
6	2	8	0.1392892957
7	1	10	0.1359997948
8	7	9	0.1313332775
9	6	8	0.1279894316
10	6	9	0.0989424889
11	1	3	0.0954672804
12	2	4	0.0848204148
13	8	10	0.0712208978
14	4	9	0.0691325904
15	4	7	0.0674427719
16	2	6	0.0502528787
17	2	7	0.0496690263

Bei Bedarf können Faltungen z. B. auf Bilddaten mithilfe von Kernels angewandt, Datenreihen komprimiert und die nächsten Nachbarn eines neuen Datentupels bestimmt werden (kNN).

4.9 Arbeiten mit DataSprite-Operatoren

Maschinelles Lernen erfordert häufig den Einsatz von Methoden der linearen Algebra. Zu deren Grundelementen gehören Skalare („Zahlen“), Vektoren und Matrizen. Die letzten beiden lassen sich mit Zufallsinhalten schnell erzeugen: *new <n>-dim vector* liefert einen Vektor der angegebenen Länge und *new <n>x<m>-matrix* arbeitet entsprechend für Matrizen. Mit *transpose <data>* können Vektoren und Matrizen transponiert werden, *is <data> a …* überprüft, ob der angegebene Datentyp die richtige Struktur hat.



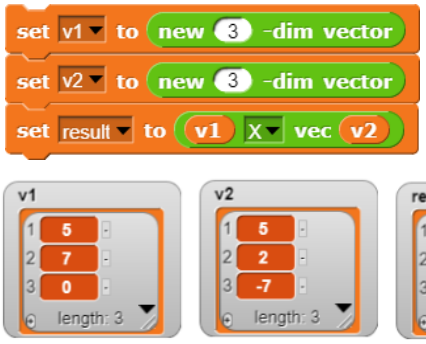
Operationen zwischen Skalaren, Vektoren und Matrizen werden durch die beiden Blöcke

<operand> <operator> vec <data> bzw.

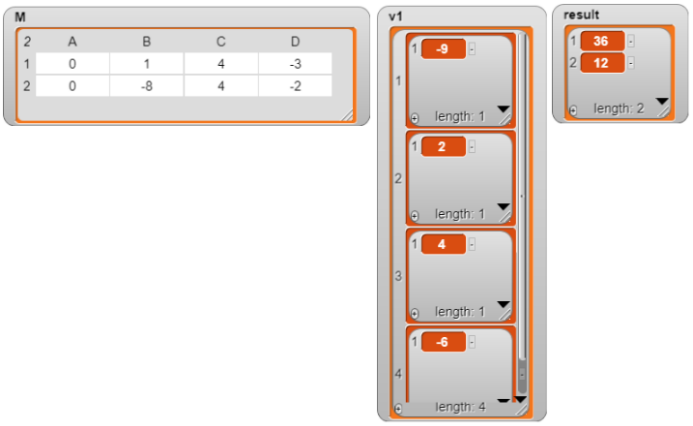
matrix/vec <op1><operator>matrix/vec <op2>

ausgeführt – wenn möglich.

Beispiel: Kreuzprodukt zwischen Vektoren

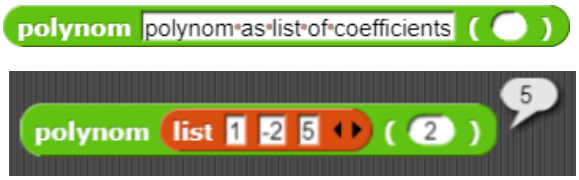


Beispiel: Produkt aus Matrix und Vektor



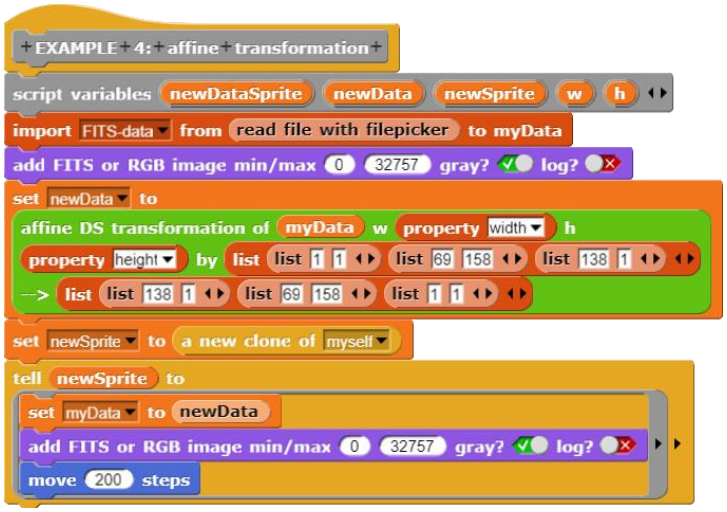
Etwas spezieller sind die folgenden Operatoren:

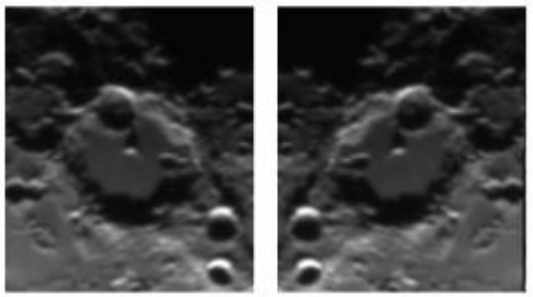
polynom<poly>(<value>) berechnet den Wert eines Polynoms für den angegebenen Wert nach dem Horner-Schema, wobei ein Polynom durch eine Liste seiner Koeffizienten beschrieben wird, beginnend mit dem höchsten. Das Polynom $x^2 - 2 * x + 5$ würde also durch die Liste  dargestellt.



Affine Transformationen mit Bildern werden mit dem Block
affine transformation of <imagedata><width><height> by <points1>→ <points2>
durchgeführt.

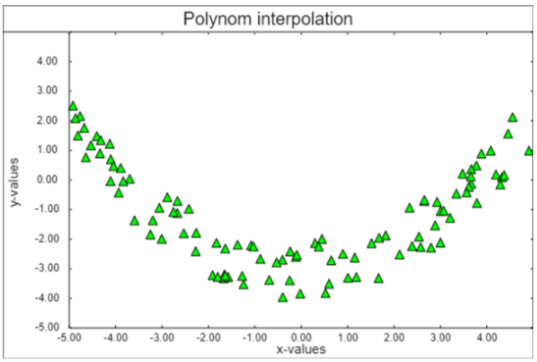
Beispiel: Spiegelung eines Bildes an der Vertikalen

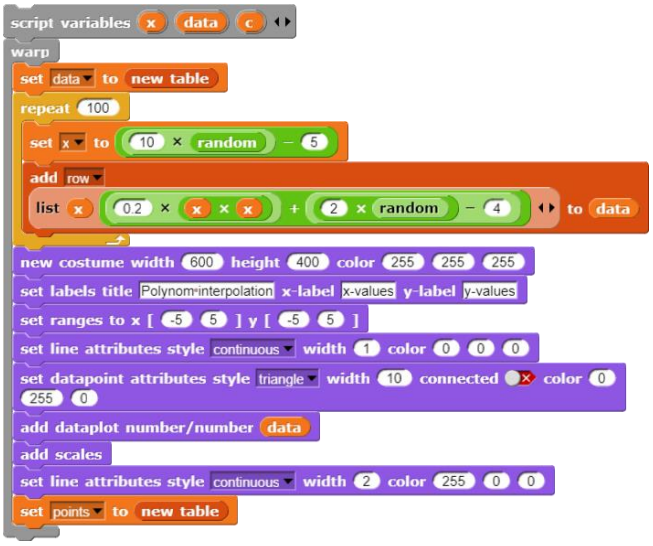




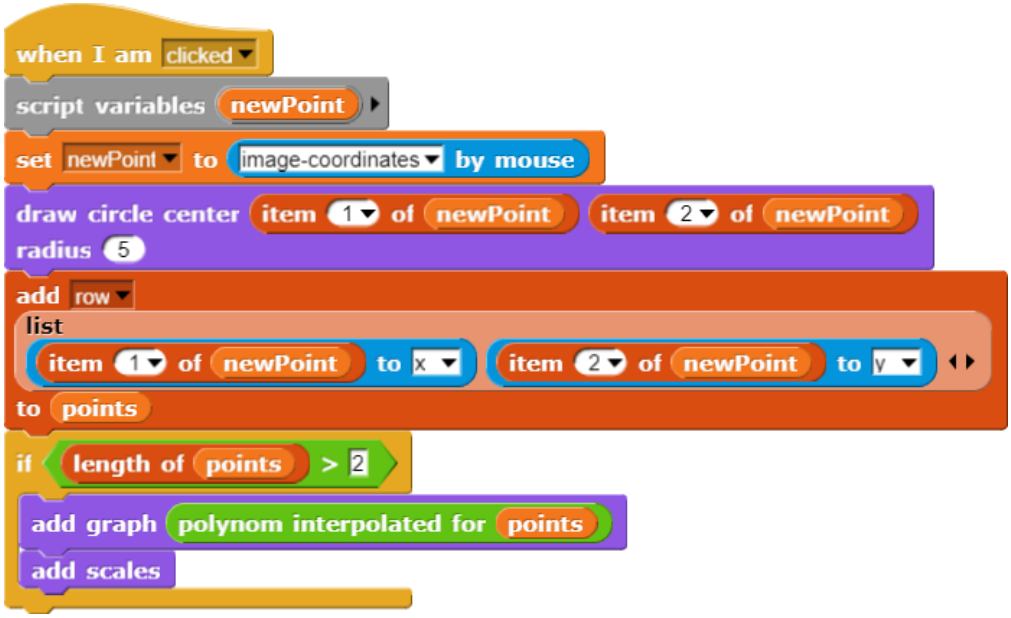
Lineare Gleichungssysteme können mit *solve <matrix>*x=<vector>* gelöst werden.
Eine Anwendung dafür ist die Polynominterpolation
polynom interpolation for <table>,
in der ein Polynom durch n Punkte berechnet wird.

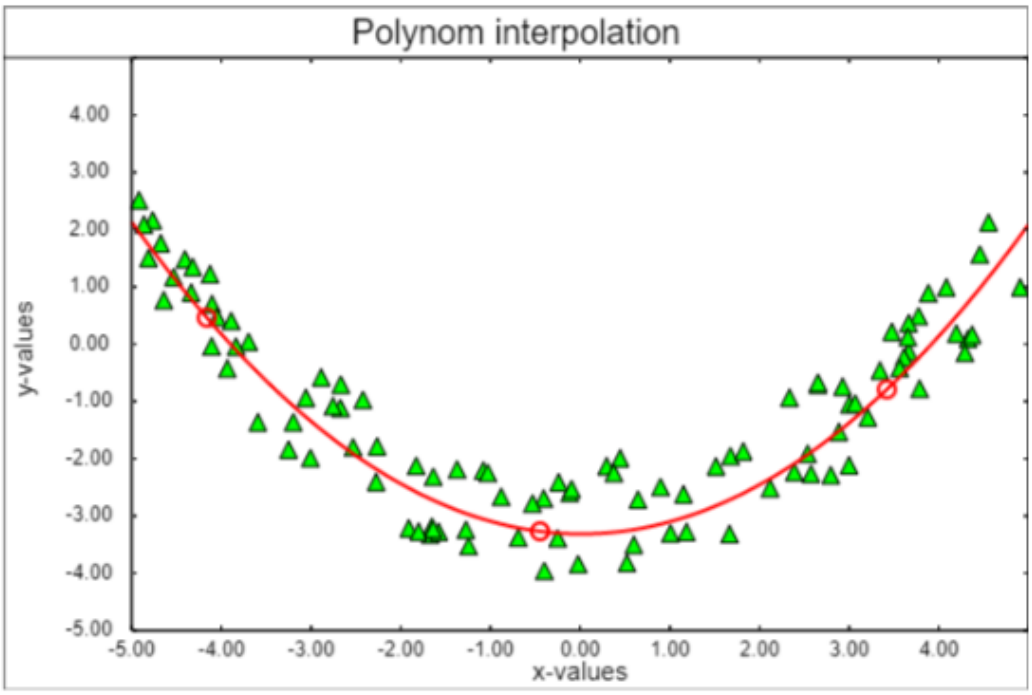
Beispiel: Kurve durch n Punkte
Wir erzeugen uns einige Zufallspunkte, die um eine Parabel herum verteilt sind und stellen das Ergebnis dar.





Da wir offensichtlich mit einer Regressionsgerade nicht weiterkommen, klicken wir drei Punkte in dieser Punktemenge an. Nach dem dritten Punkt berechnen wir ein Interpolationspolynom und lassen es ins Diagramm zeichnen. Dabei sind zwei Koordinatentransformationen erforderlich.





Die letzten Blöcke der Operatoren-Palette runden eine Zahl auf die angegebene Stellenzahl, was z. B. bei einer Verbesserung der Diagramm-Darstellung hilfreich wäre, und liefern Zufallszahlen zwischen 0 und 1 in voller Länge. Der letzte Block wandelt eine Liste aus Texten in eine Zeichenkette mit vorgegebenen Teillängen um. Damit können z. B. die Spaltenüberschriften einer Tabelle schnell in Achsenbeschriftungen eines Diagramms verwandelt werden.

5 Anwendungen des DataSprites

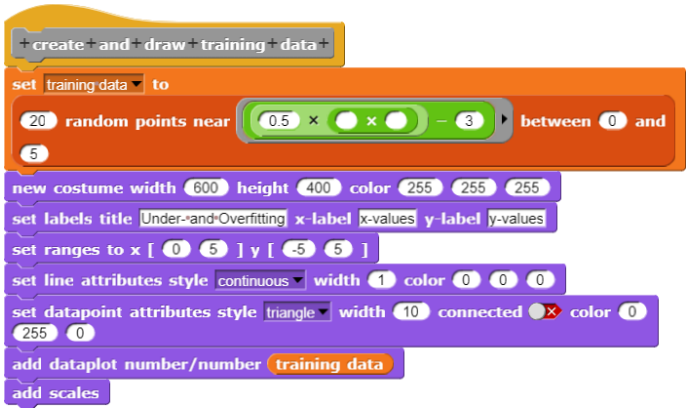
5.1 Under- and Overfitting

Maschinelles Lernen passt die Parameter einer Funktion mithilfe von Trainingsdaten so an, dass andere Werte gut prognostiziert werden – wenn alles klappt. Man baut also ein Prognoseinstrument, so eine Art „Fernrohr“ für Daten.

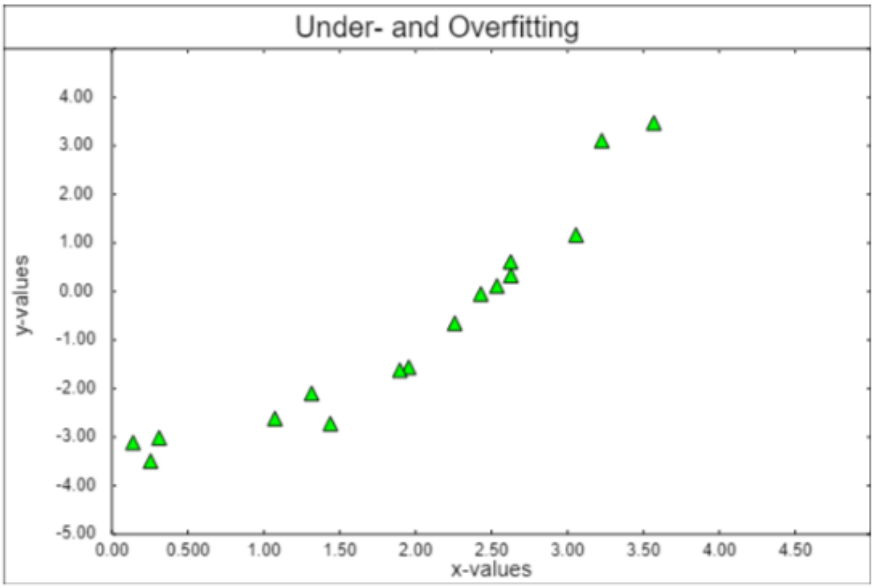
Wenn die Trainingsdaten von der Funktion gut reproduziert werden, dann heißt das noch lange nicht, dass das auch für andere Daten gilt. Es hängt sehr von der Art der Funktion ab, die erzeugt wird. Als Anwendung wählen wir das letzte Beispiel: die Polynominterpolation.

Die Aufgabe lautet: *Mithilfe von Trainingsdaten werden die Koeffizienten eines Polynoms so angepasst, dass ANDERE Daten möglichst gut prognostiziert werden.*

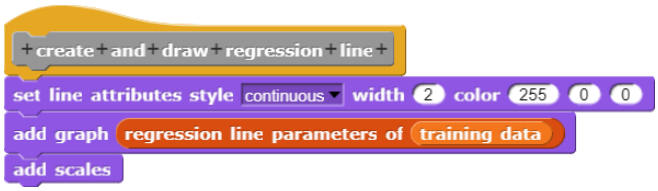
Zuerst einmal wollen wir einige Daten erzeugen, mit deren Hilfe ein Interpolations-Polynom berechnet wird. Um nicht immer wieder neu anfangen zu müssen, schreiben wir einen Reporter, der eine Tabelle mit n Daten einer beliebigen Funktion berechnet.

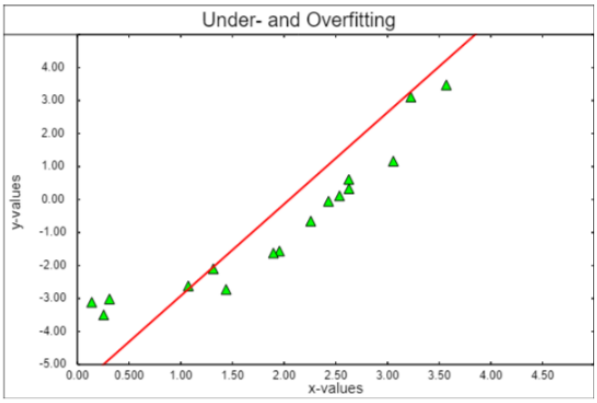


Diesen benutzen wir zur Erzeugung der Trainingsdaten, die wir auch gleich darstellen.



Wir versuchen es zuerst einmal mit einer Regressionsgeraden.

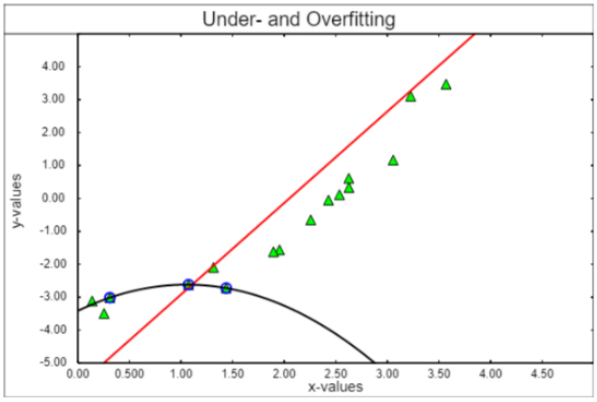


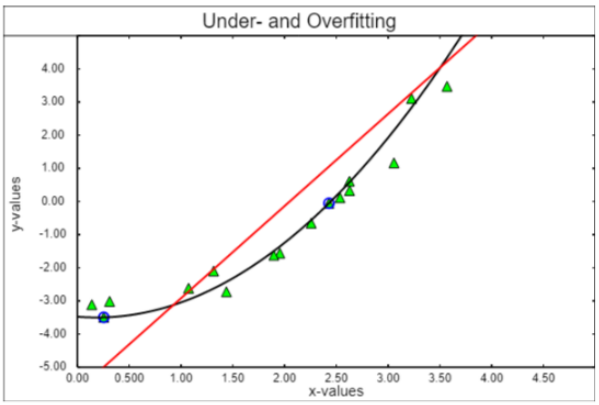


Das sieht eigentlich ganz nett aus, aber an den Seiten passt es nicht so recht.

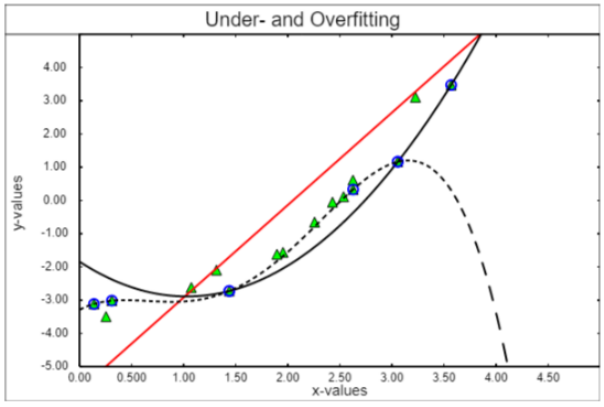
Wir probieren es also mit einer Polynom-Interpolation.

Zuerst einmal wählen wir drei Zufallspaare aus den Trainingsdaten, bestimmen daraus das Interpolationspolynom und zeichnen es. Weil wir weiter experimentieren wollen, verallgemeinern wir die Lösung gleich zu einem Polynom durch n Punkte. Dabei hoffen wir, dass bei der Auswahl alles gut geht! Die Ergebnisse hängen davon ab, welche Punkte erwischt wurden. Anbei ein schlechtes und ein ganz gutes Ergebnis.

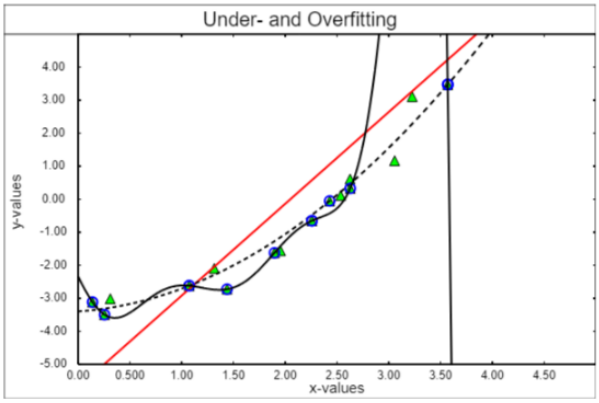




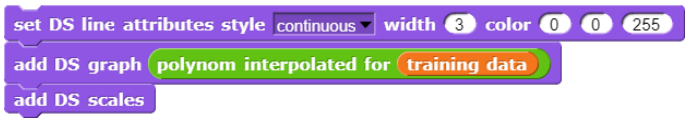
Jetzt werden wir mutig! Statt drei Punkten wählen wir 5. Wir wollen ja schließlich gute Arbeit abliefern! Das klappt prima bis zum rechten Rand, und dann – upps!

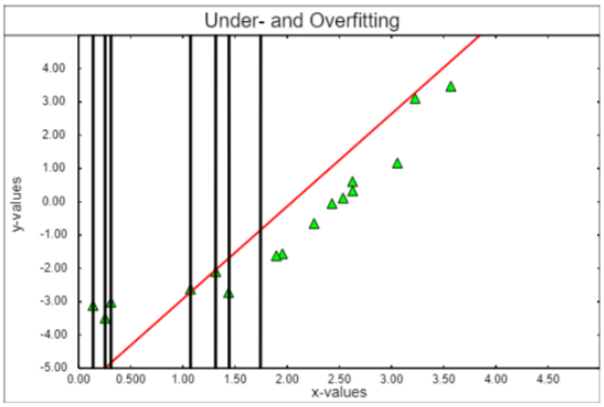


Vielleicht müssen wir ja einfach mehr Punkte nehmen. Versuchen wir es mit 10. Das Polynom läuft zwar durch mehr Punkte, aber an den Rändern „haut es ab“.



Na, dann mit allen Punkten!





Man sieht, dass mit zunehmendem Grad des Polynoms zwar mehr Trainingsdaten direkt auf dem Graph liegen, dass aber dazwischen durch die wilden Oszillationen des Polynoms nur noch unsinnige Werte „prognostiziert“ werden.

Die Qualität des Gelernten hängt also sehr davon ab, wie wir mit Abweichungen umgehen. Wir müssen entscheiden, welche Ungenauigkeiten im Detail tolerierbar sind, damit die Prognose insgesamt zuverlässig wird. Ist der Grad des Polynoms zu klein, dann spricht man von *Underfitting*, ist er zu hoch, von *Overfitting*.

Aufgaben:

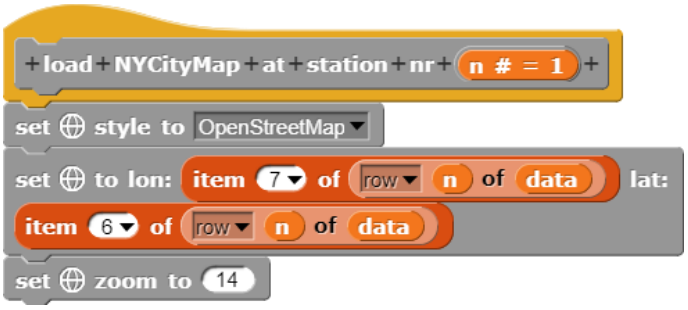
1. Diskutieren Sie unterschiedliche Möglichkeiten, einen „guten“ Grad des Interpolationspolynom (also seine höchste Potenz) festzulegen.
2. Formulieren Sie ihre Ergebnisse so präzise, dass sie sich als Skripte realisieren lassen.
3. Texten Sie die Skripte an unterschiedlichen Datensätzen.

5.2 New York Citibike Tripdata [NYcitibike]

Selbst in New York ist das Fahrradfahren inzwischen „hip“ geworden und die Entleihdaten können als CSV-Dateien geladen werden. Wir tun das und laden die knapp 600 000 Datensätze vom 30. Juni 2013 in eine Tabelle. Dabei spalten wir gleich die Spaltenüberschriften ab, um eine reine Datentabelle zu erhalten.

Was haben wir da eigentlich gefunden?

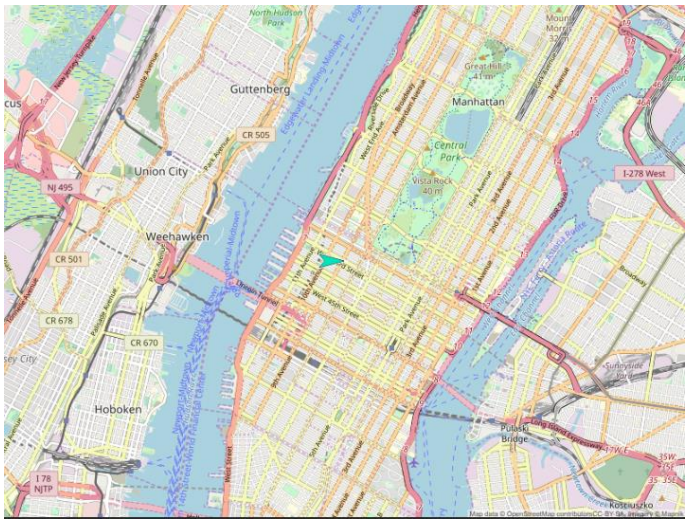
Die Datenlegende liefert die Interpretationsvorschrift für die Daten: *Trip Duration (seconds), Start Time and Date, Stop Time and Date, Start Station Name, End Station Name, Station ID, Station Lat/Long, Bike ID, User Type (Customer = 24-hour pass or 3-day pass user; Subscriber = Annual Member), Gender (Zero=unknown; 1=male; 2=female), Year of Birth*



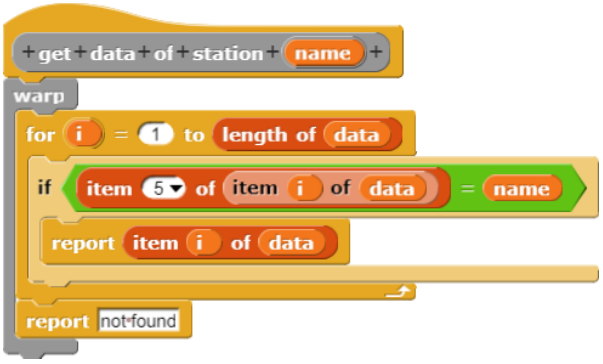
Da geografische Länge und Breite der Entleihstationen angegeben sind, bietet es sich an, die *Word Map Library* von *Snap!* einzusetzen. Wir schreiben dafür einen kleinen Block, der die Umgebung einer Entleihstation als Karte darstellt.

Wir wollen doch mal sehen, wo man Fahrräder entleihen kann. Für die Übersicht extrahieren wir die Entleihstationen aus der Gesamtliste, z. B. indem wir sie nach dem Namen der Startstation (Spalte 5) gruppieren lassen und nur diese Spalte als Ergebnis wählen.

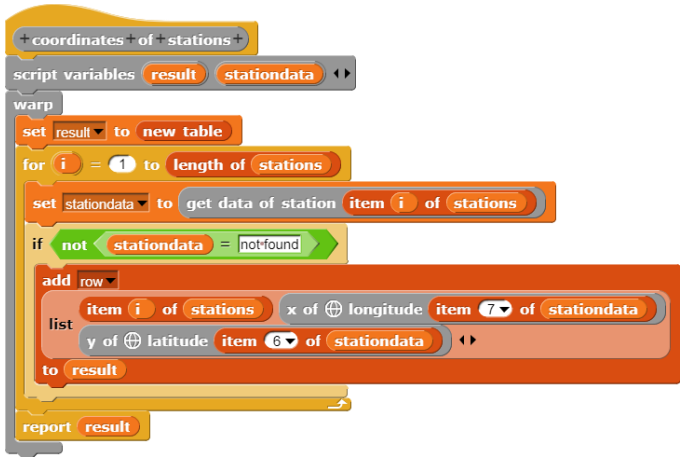
Wir erhalten immerhin 337 Stationen.



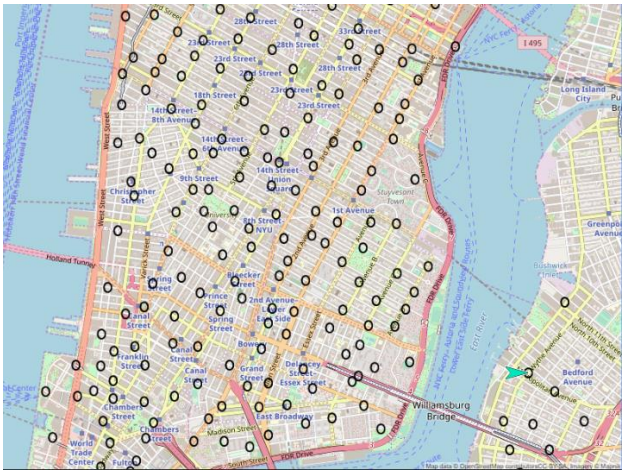
Danach suchen wir die Daten einer Station zusammen …



… und stellen so die Koordinatenliste der Stationen auf.



Mit diesen Daten können wir das Sprite zu den einzelnen Positionen schicken und dort mit dem *stamp*-Block zum Hinterlassen von Kreisen auffordern.



```
+show+all+citibike+stations+on+map+
warp
clear
set stations to
  column 1 of mean of column 1 of data grouped by column 5
delete 1 of stations
set coordinates of stations to coordinates of stations
set size to 70 %
switch to costume circle
for i = 1 to length of coordinates of stations
  go to x: item 2 of row i of coordinates of stations y:
    item 3 of row i of coordinates of stations
  stamp
set size to 100 %
switch to costume Turtle
```

Zumindest in Midtown Manhattan müssen wir uns wohl keine Sorge darum machen, ob wir eine Entleihstation finden!

Wir wollen uns jetzt einmal die Verleihstation Broadway – Ecke 41 Street (Nr. 55) genauer ansehen. Dazu ziehen wir alle Datensätze aus der Gesamtliste, die an dieser Station starten oder enden. Das sind an diesem Tag 5005 Vorgänge. Zeiten sind in dieser Liste zusammen mit dem immer gleichen Datum eingetragen. Das können wir herauswerfen (Abtrennen mit *split* mit „ „) und auf die Stunde reduzieren (*split* mit „:“). Wir haben danach eine numerische Skala mit der Einheit „Stunde“.

```
+connection+to+or+from+station+ name +
report
  keep items
    item 5 of = name or item 9 of = name
  from data
```

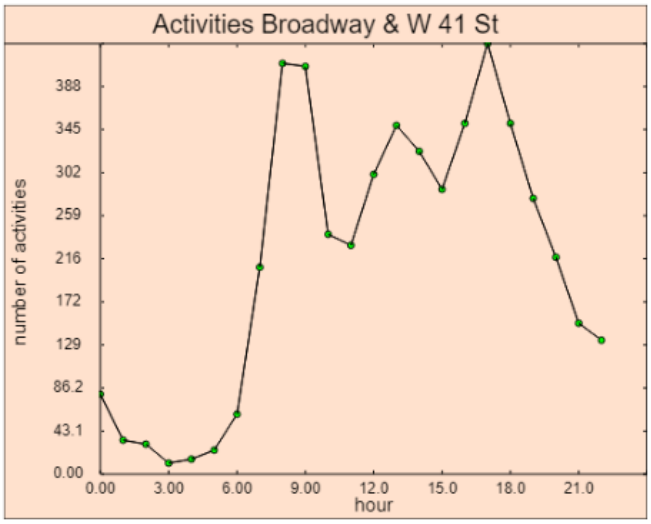
```
set borrowing data to
  connection to or from station item 55 of stations
```

```
set borrowing data to reduce time columns of borrowing data to hours
```

Jetzt können wir sehen, was in den einzelnen Stunden des Tages an der Station los ist.

```
set graph data to
  number of column 1 of borrowing data grouped by column 2
```

Und das können wir wie üblich grafisch darstellen.

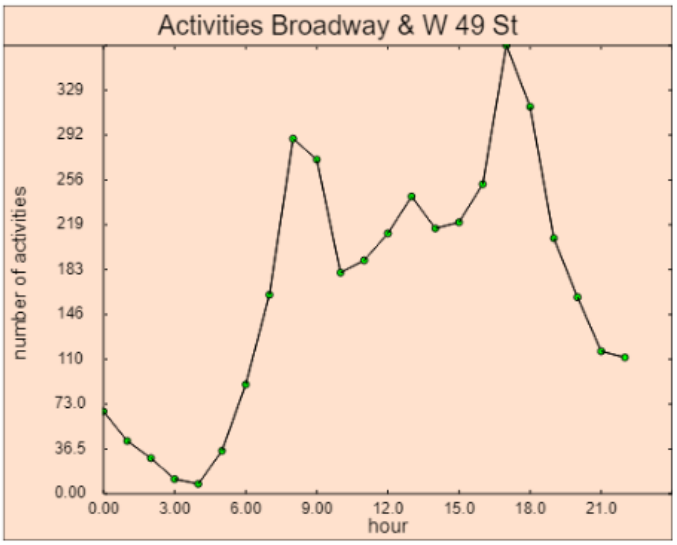



```
+reduce+time+columns+of+ table +to+hours+
script variables result
warp
set result to new table
add column column 1 of table to result
add column
  map item 1 of split item 2 of split  by  by  over
    column 2 of table
  to result
add column
  map item 1 of split item 2 of split  by  by  over
    column 3 of table
  to result
for i = 4 to 15
  add column column i of table to result
report result
```

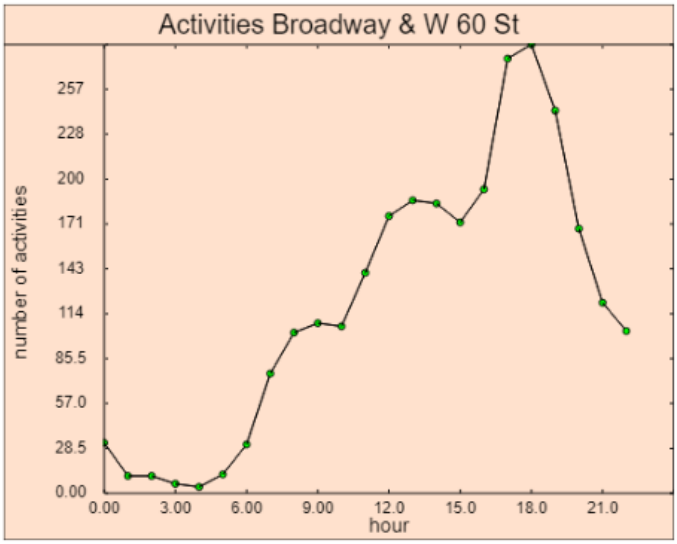
```
new costume width 500 height 400 color 255 225 205
import table-data from graph data to myData
set labels title join Activities item 55 of stations x-label hour
  y-label number-of-activities
set datapoint attributes style o-circle width 5 connected color 0
  255 0
set scale attributes precision 3 textheight 12 number of intervalls
  x-axis 8
set ranges to x [ 0 24 ] y [ 0
  max of column 2 of graph data ]
add dataplot number/number myData
add scales
```

Ein paar Straßen weiter sieht es ganz ähnlich aus.

Ist das ein allgemeines Muster?



Nun gut, am Central Park stehen die Leute später auf und die Touristen sind noch nicht da. Dafür schließen die Museen immer zur gleichen Zeit.



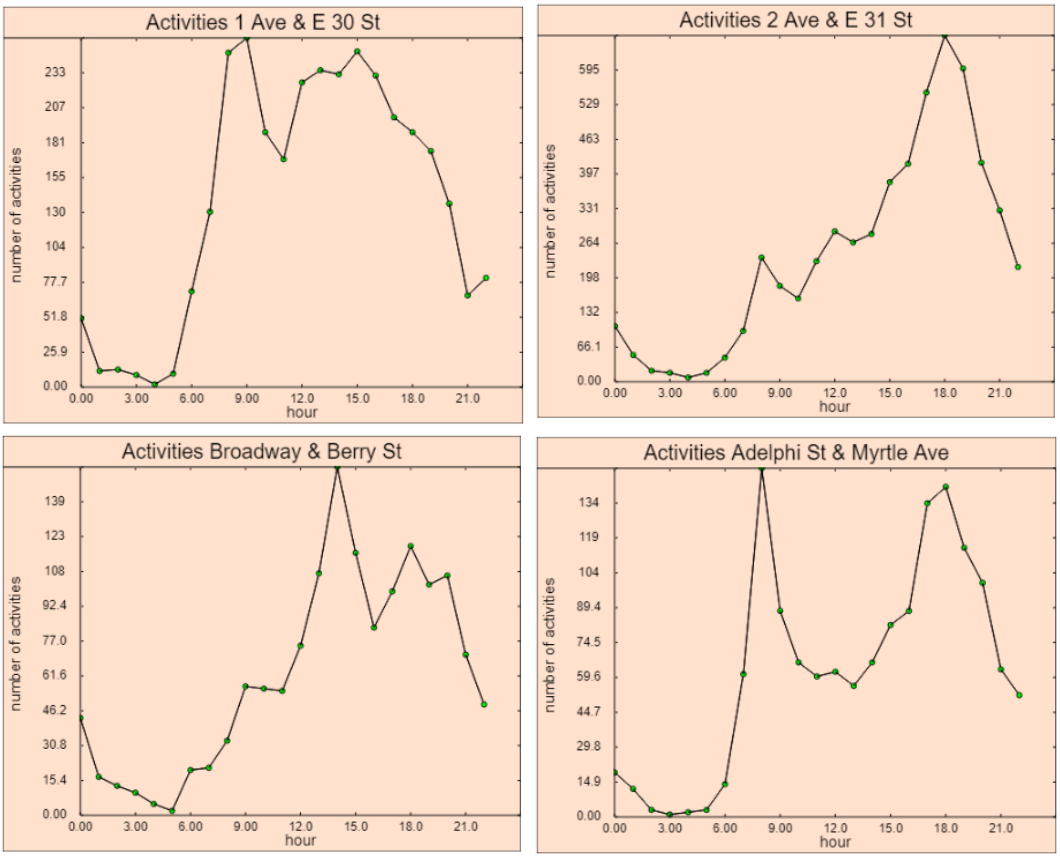
Was können unsere Programme nun aus diesen Daten lernen?

- *Wir könnten z. B. aus den üblichen Ab- und Zugängen sowie dem Istbestand voraussagen, ob an einer Station noch rechtzeitig genügend Fahrräder zurückgegeben werden oder ob es besser wäre, einige dorthin zu transportieren.*
- *Wir könnten aus den mittleren Weglängen ermitteln, welche Akkus für eBikes gebraucht werden.*
- *Wir könnten feststellen, ob eher Frauen oder Männer zu einer bestimmten Tageszeit die Räder entleihen und dann dafür sorgen, dass das Angebot stimmt. Das Entsprechende könnten wir für das Alter der Entleihenden machen.*
- *Wir könnten die Entleihdaten pro Rad ermitteln und voraussagen, wann Reparaturen fällig sein werden. Wir könnten das z. B. auch in Abhängigkeit von der Lage der Stationen machen.*
- *Wir könnten versuchen, Verteilungen von einigen Stationen so zu verallgemeinern, dass sich Prognosen für andere daraus ableiten lassen. Wenn also am Central Park die Museen schließen, kann das Programm aus den alten Daten „lernen“, in welchen Bezirken die Räder wann vermutlich abgegeben werden, und warnen, falls dort nicht genug freie Slots zur Verfügung stehen.*

usw.

Aufgaben:

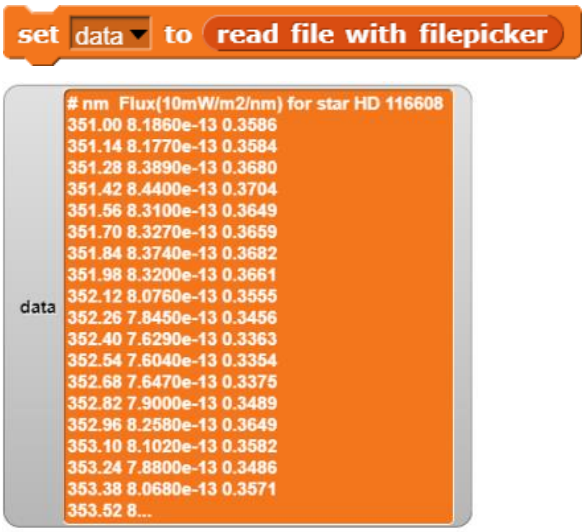
1. Gliedern Sie die Aktivitäten der Stationen nach Zu- und Abgängen auf.
2. Schreiben Sie eine Prognosefunktion, die warnt, wenn in den nächsten Stunden ein Radmangel an einer Station droht.
3. Stellen Sie für bestimmte Stationen durch direkte Linien die Verbindungen zu den meist gewählten Abgabestationen graphisch auf der Karte dar. Wählen Sie die Liniendicke entsprechend der Anzahl der Entleihvorgängen und die Farben je nach Station. Bilden sich Cluster?
4. Stellen Sie mithilfe von Korrelationen fest, ob es Zusammenhänge im Entleihverhalten (z. B. bzgl. der Tageszeiten, dem Ort, …) mit dem Geschlecht, dem Alter, dem Status der Entleihenden gibt. Ggf. müssen Sie die Daten vorher durch numerische Daten ersetzen – ähnlich wie bei den Zeiten. Diskutieren Sie mögliche Konsequenzen.
5. Ermitteln Sie für einen kleinen Abschnitt von Midtown (dort, wo alles schön rechteckig ist) die Koordinaten der Straßenecken. Entwickeln Sie dann einen Router, der den kürzesten Weg zur nächsten Citibike-Station anzeigt.
6. Die Entleihzahlen abhängig von der Tageszeit zeigen in unterschiedlichen Bereichen Manhattans ziemliche Unterschiede. Untersuchen Sie Gemeinsamkeiten und Unterschiede systematisch und versuchen Sie, die Ergebnisse zu erklären.



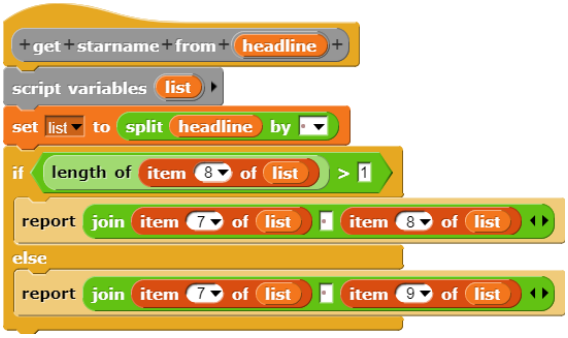
5.3 Sternspektren [UniGOE]

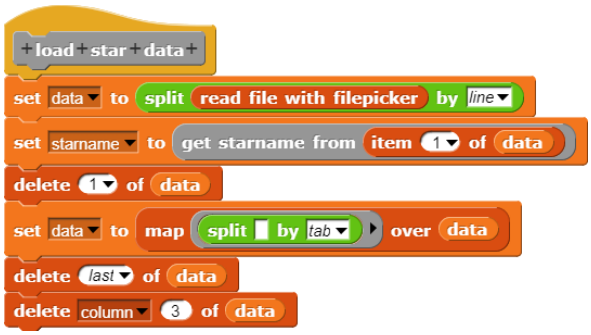
Sterne leuchten in unterschiedlichen Farben, weil sie unterschiedliche Temperaturen haben. Zusätzlich unterscheiden sich die Spektren in ihren Absorptionslinien. Das wollen wir etwas genauer untersuchen.

Wir besorgen uns einige Sternspektren (Quelle: [UniGOE]) und speichern sie als Textdatei. Solch eine lesen wir ein. In der ersten Zeile steht nach den Spaltenbeschriftungen der Sternname. Den isolieren wir und speichern ihn in der Variablen *star name*.

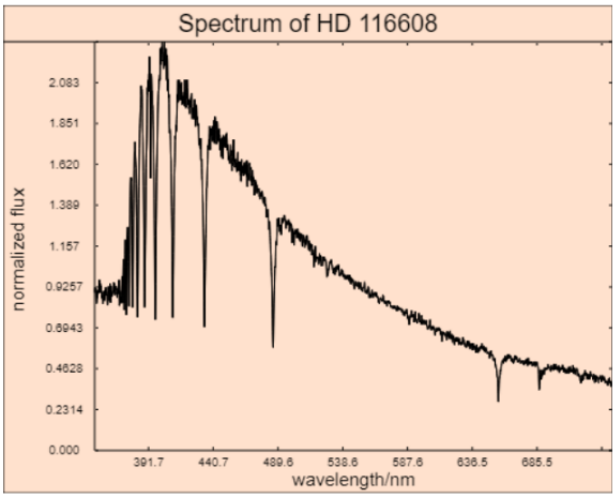


Wir wissen jetzt, wie der Stern heißt. Wenn Sie im Internet danach suchen, finden Sie eine Fülle von Informationen darüber. Damit wir den Ladevorgang mit anderen Daten wiederholen können, kapseln wir ihn in einem eigenen Block. Nach dessen Ausführung liegen die eigentlichen Sterndaten als Tabelle vor.



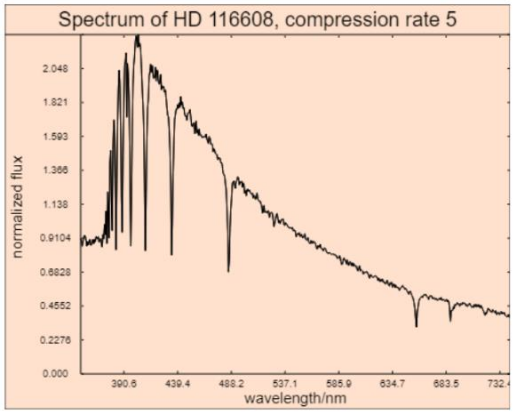


Mit diesen Daten lässt sich schnell ein Diagramm erstellen.

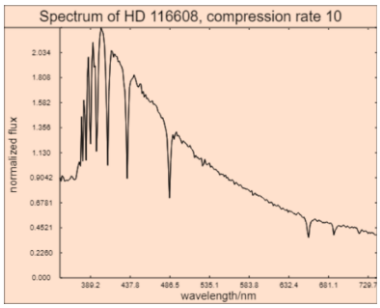


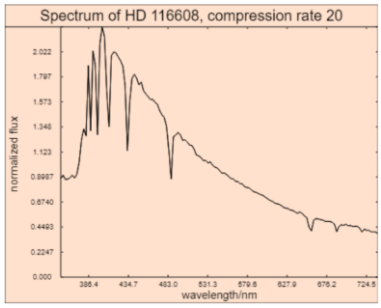
Man erkennt gut den abfallenden Verlauf mit einigen markanten Absorptionslinien. Aber braucht man für diese Erkenntnis überhaupt alle Spektraldaten?

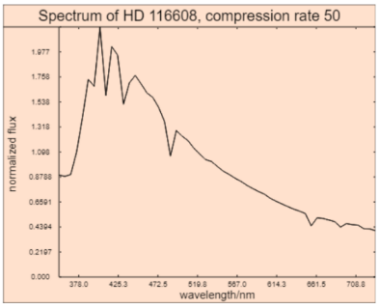
Vielleicht genügt es ja, durch Mittelwertbildung die Datenmenge zu reduzieren. Wir führen einen Kompressionsfaktor *compression rate* ein und ergänzen das Skript vor der Diagrammerstellung.

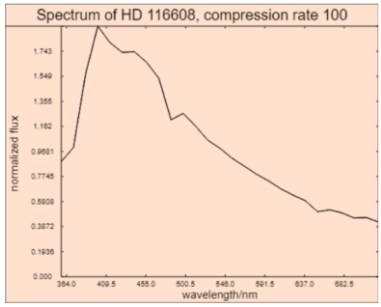


Der Faktor 5 ändert nicht viel. Probieren wir also weiter.

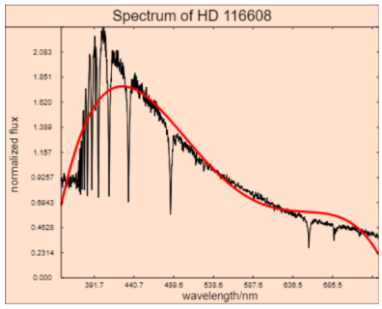


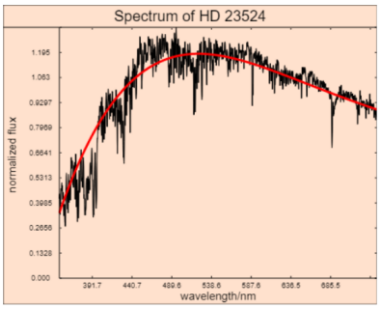


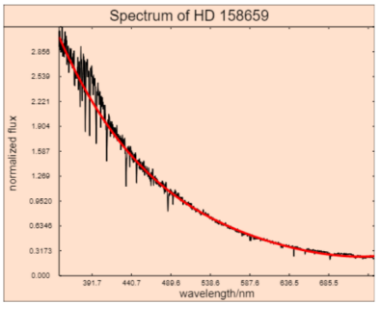


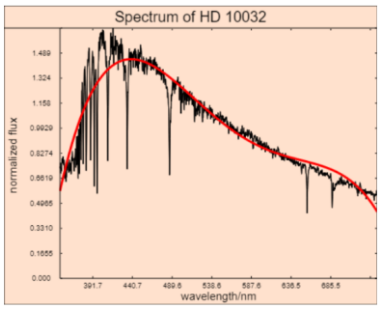


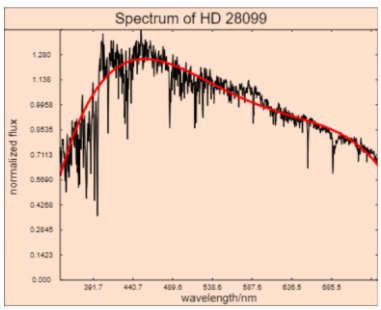
Man sieht, dass der temperaturabhängige Verlauf des Spektrums kaum verändert wird. Nur die Absorptionslinien gehen verloren. Somit sollte sich der Typ des Spektrums durch ein Interpolationspolynom z. B. 4. Grades beschreiben lassen.

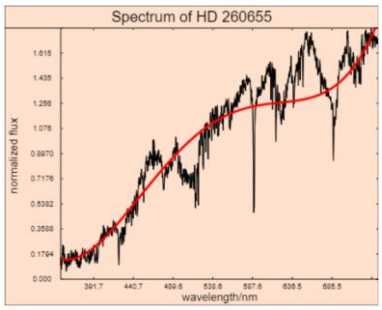












Das funktioniert auch hervorragend! Protokollieren wir die Polynomparameter bei der Untersuchung gleich mit, dann können wir anhand der Parameterbereiche die Sterntypen leicht unterscheiden.

7	A	B	C	D	E	F
1	star name	a4	a3	a2	a1	a0
2	HD 116608	-1.2493580327340172e-9	0.000002886821087800814	-0.002459579857425176	0.9103088865090065	0.9103088865090065
3	HD 158659	1.565259017017166e-10	-3.963032080178107e-7	0.0003879661846290463	-0.17622312866994078	-0.17622312866994078
4	HD 10032	-7.27005023271818e-10	0.00000169492984799 1264	-0.001462425925103779	0.5500801501694278	0.5500801501694278
5	HD 28099	-4.0018935572381893e-10	9.399457129604694e-7	-0.0008209889485783107	0.3141072191721327	0.3141072191721327
6	HD 23524	-8.18301248511472e-11	2.3253458278204257e-7	-0.0002461580054487696 5	0.11348374829256708	0.11348374829256708
7	HD 260655	6.248027476637483e-10	-0.000001337322548726115	0.001045033368386972 3	-0.3486709605339992	-0.3486709605339992

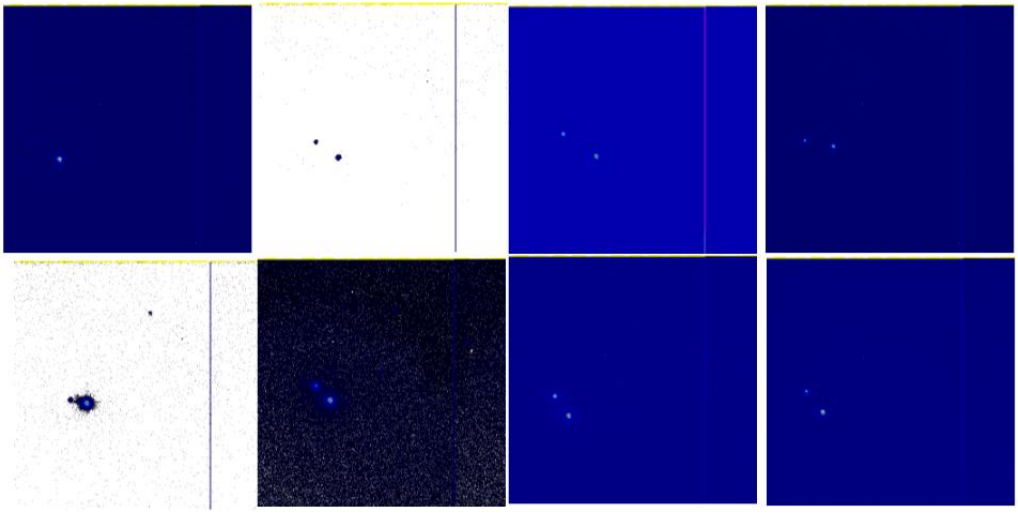
Das Programm kann anhand der alten Daten also „lernen“ welche Parameterintervalle zu welchen Sternklassen gehören. Gibt man die Daten eines neuen Sterns ein, dann ermittelt es die Koeffizienten des Polynoms und gibt danach eine gut begründete Prognose ab, um welche Art von Stern es sich handeln könnte.

Aufgaben:

1. Stellen Sie für die nicht komprimierten Spektrumsdaten jeweils ein Interpolationspolynom möglichst niedrigen Grades auf. Welche Punkte sollten dafür gewählt werden? Ergeben sich Unterschiede zwischen diesen Polynomen und den Ergebnissen des oben gezeigten Verfahrens?
2. Entwickeln Sie ein Skript, das ein unbekanntes Spektrum einem der bisher auftretenden Typen zuordnet.
3. Entwickeln Sie ein Verfahren, um die markantesten Absorptionslinien genauer zu untersuchen. Stellen Sie diese für Sterne der gleichen Klasse vergrößert dar und versuchen Sie, Unterschiede „automatisch“ zu bestimmen. Diskutieren Sie Ihre Ideen vor der Realisierung.

5.4 Periodendauer eines Cepheiden [HOU]

Cepheiden sind Sterne, deren Helligkeit periodisch schwankt. Da ihre Leuchtkraft direkt von der Periodendauer der Schwankung abhängt, dienen sie als „Standardkerzen“ zur Entfernungsmessung im All. Wir wollen diese Periodendauer messen. Dazu besorgen wir uns einige Bilder eines Cepheiden, die an unterschiedlichen Tagen aufgenommen wurden (Quelle: [HOU]). Das Beispiel dient zur Erläuterung von *DataSprite*-Bildoperationen.

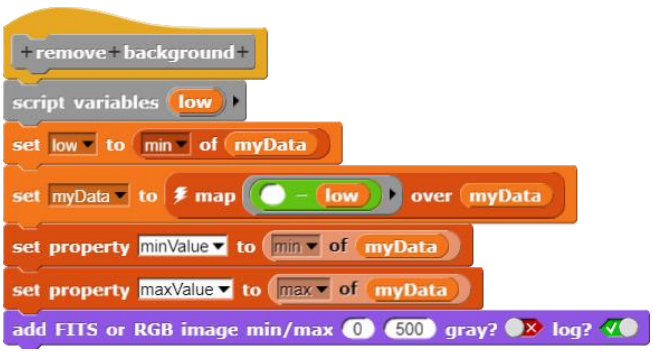


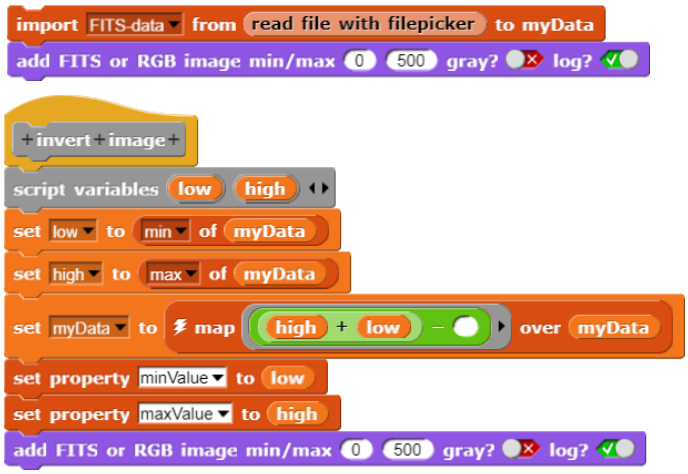
Man sieht, dass die Bilder sehr unterschiedliche Qualität haben. Auf einigen ist der Cepheide, der Stern neben dem größeren Stern, kaum zu sehen. Mal ist der Hintergrund heller, mal dunkler. Wir haben also einige Aufgaben zu lösen:

- Laden der Bilder, Einstellen einer Ansicht, die beide Sterne gut zeigt.
- Abziehen der Hintergrundstrahlung des Bildes, ggf. Invertierung.
- Messen der Helligkeit eines Sterns.
- Darstellung der Ergebnisse und Bestimmung der Periodendauer.

Und wie bestimmt man die Helligkeit eines Cepheiden? Man berechnet sie relativ zu einem unveränderlichen Stern in der Nähe. In jedem Bild müssen also zwei Helligkeitswerte gemessen und durcheinander geteilt werden.

Ein Bild wird wie in den anderen Beispielen ins *DataSprite* geladen und angezeigt. Minimum und Maximum der angezeigten Werte werden experimentell so ermittelt, dass beide Sterne gut zu sehen sind. (Zur Erinnerung: die eigentlichen Bildwerte werden dadurch nicht verändert.) Bei Bedarf wird das Bild invertiert.

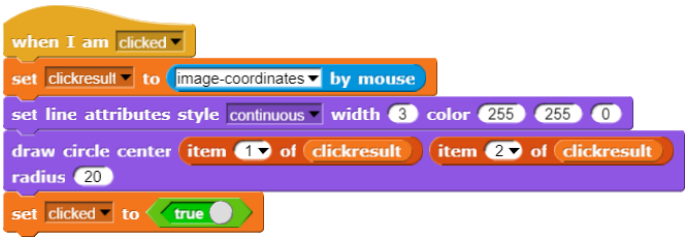




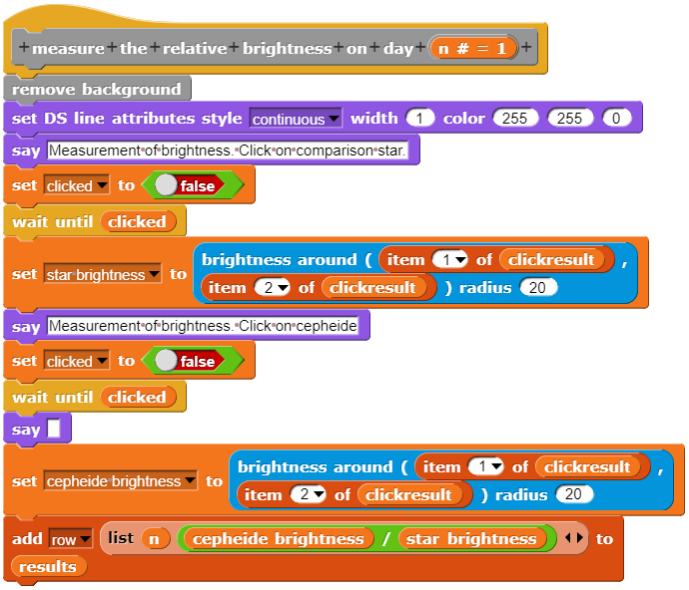
Der Strahlungshintergrund wird abgezogen, indem einfach der kleinste Bildwert von allen abgezogen wird.

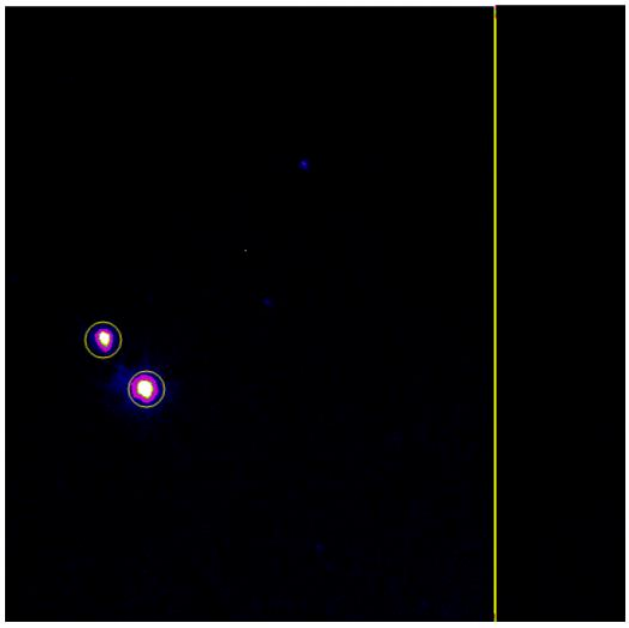
Wir verarbeiten die Mausklicks zur Ermittlung der Helligkeit, indem wir eine Variable *clicked* auf *false* setzen. Wird das Bild angeklickt, dann erhält sie den Wert *true* und die angeklickten Bildkoordinaten werden der Variablen *clickresult* zugewiesen. Im Skript wartet das Programm mithilfe des Befehls 

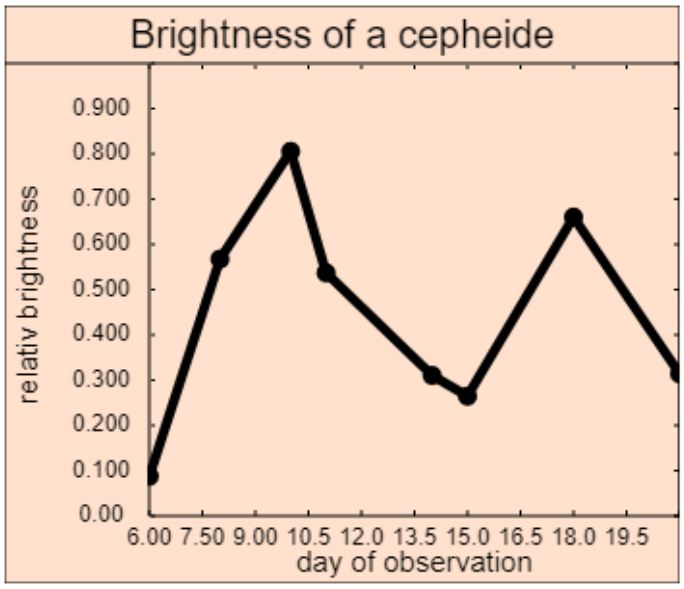
, bis Daten vorliegen.



Mit diesen Hilfen lässt sich der Messprozess in einem eigenen Block zusammenfassen. Die Ergebnisse werden einer Variablen *result* zugewiesen. Diese wird später ausgewertet.







8	A	B
1	6	0.0876221753506
2	8	0.5674336949167
3	10	0.8054445068099
4	11	0.5370600093037
5	14	0.3104731277173
6	15	0.2646507209664
7	18	0.6603668797582
8	21	0.3140171254899

Die Periodendauer wird also bei etwa 9 Tagen liegen.

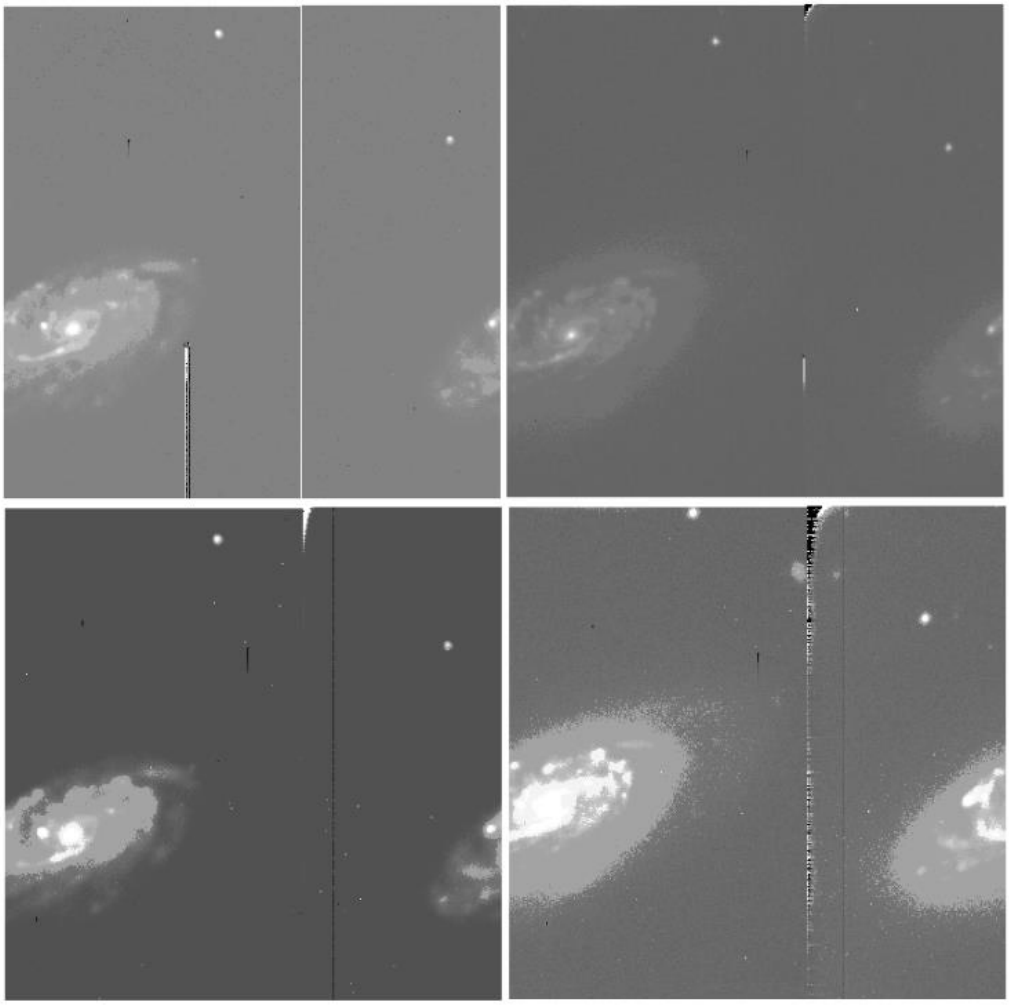
Aufgaben:

1. Legen Sie eine schöne glatte Sinuskurve (oder Ähnliches) durch die Datenpunkte. Experimentieren Sie dafür etwas herum. Wird die Periodendauer dadurch genauer ermittelt?
2. Entwickeln Sie ein Skript, das es gestattet, Linien auf dem Bild zu zeichnen, die der Periodendauer entsprechen sollen. Aus der mittleren Länge dieser Linien wird dann die Periodendauer bestimmt.
3. Auf manchen Bildern ist der Cepheide kaum zu sehen. Entwickeln Sie ein Skript, das anzeigt, wo sich Sterne auf dem Bild befinden, vielleicht durch ein Kreuz. Was ist überhaupt ein (abgebildeter) Stern? Der senkrechte Strich auf den Bildern ja wohl nicht!

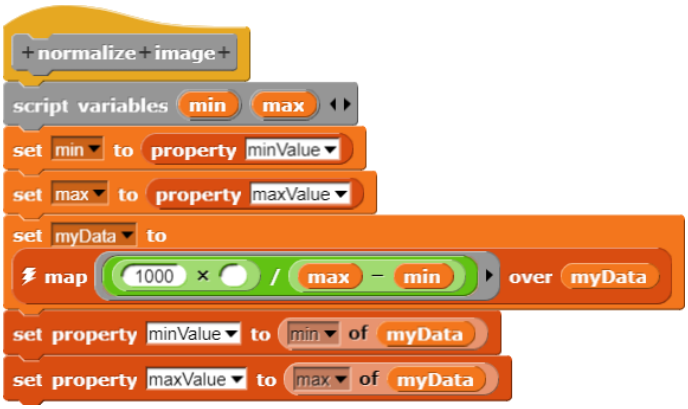
5.5 Suchen nach einer Supernova [HOU]

Eine neue Supernova ist gar nicht so leicht zwischen den vielen anderen Sternen zu entdecken. Man kann aber Kandidaten dafür identifizieren, indem man alte Bilder eines Himmelbereichs von neuen abzieht. Bleibt ein heller (oder dunkler) Fleck übrig, dann sollte man sich diesen Bereich etwas genauer ansehen (Quelle: [HOU]). Das Beispiel dient wieder zur Erläuterung der Anwendung von *DataSprite*-Bildoperationen.

Wir besorgen uns also vier Bilder, die an unterschiedlichen Tagen aufgenommen, natürlich unterschiedlich belichtet wurden, unterschiedliche Hintergrundstrahlung aufweisen, … und auch sonst einige Macken haben.

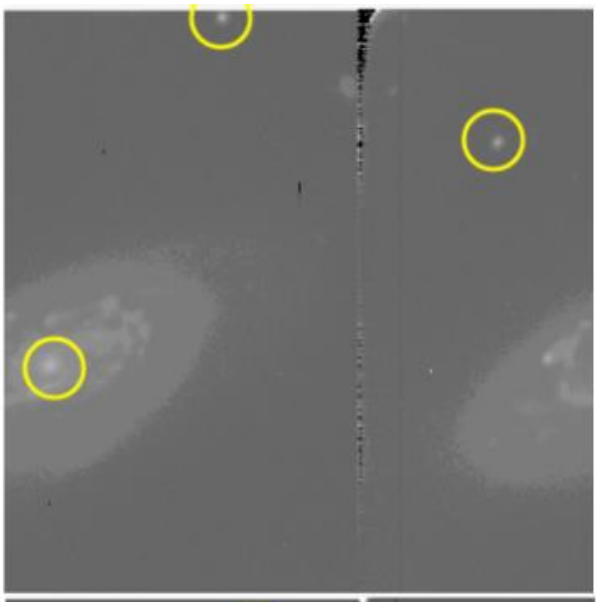


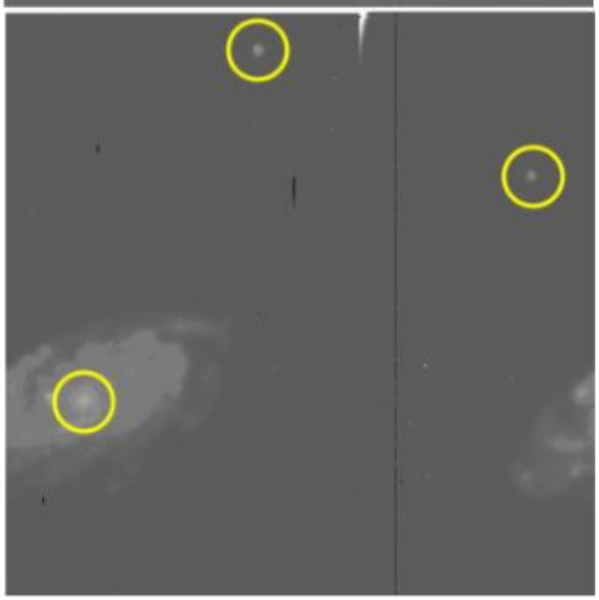
Damit wir überhaupt etwas auf den Bildern sehen, wurden sie zuerst „normalisiert, d.h. der Hintergrund wurde abgezogen und der Rest der Bildwerte auf den Bereich von 0 bis 1000 abgebildet. Damit haben sie aber noch lange nicht die gleiche Gesamthelligkeit, weil das Ergebnis stark von den größten Bildwerten abhängt.

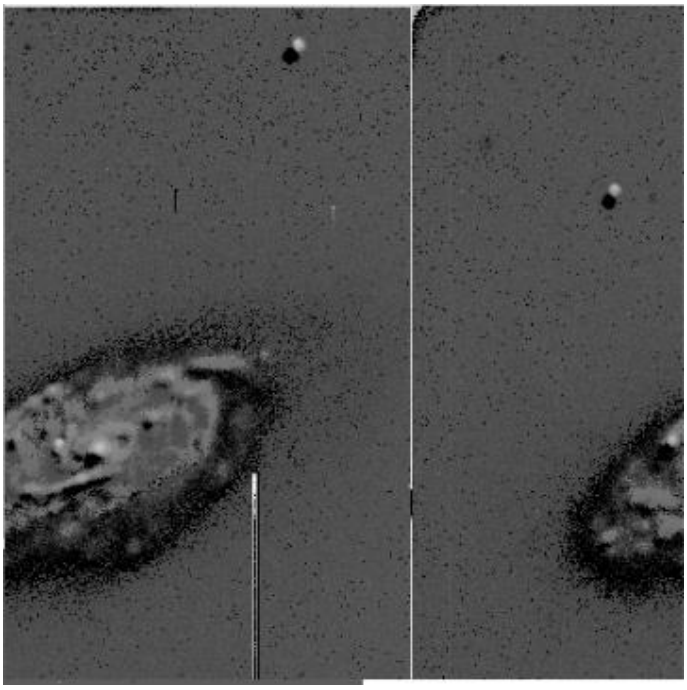


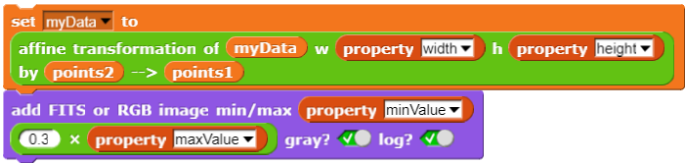
Ziehen wir jetzt einfach ein Bild vom anderen ab, dann erhalten wir ein plastisch wirkendes Ergebnis mit einigen „Ying-Yang-Effekten“ dort, wo helle Bildbereiche nicht ganz aufeinander passten. Wir können für diese Operation den Block für Vektorarithmetik benutzen, weil FITS-Daten ja eine einfache Liste darstellen.

Deshalb führen wir eine affine Transformation mit einem Bild aus, indem wir zuerst auf einem, dann auf dem anderen Bild jeweils drei sich entsprechende Punkte anklicken und speichern.

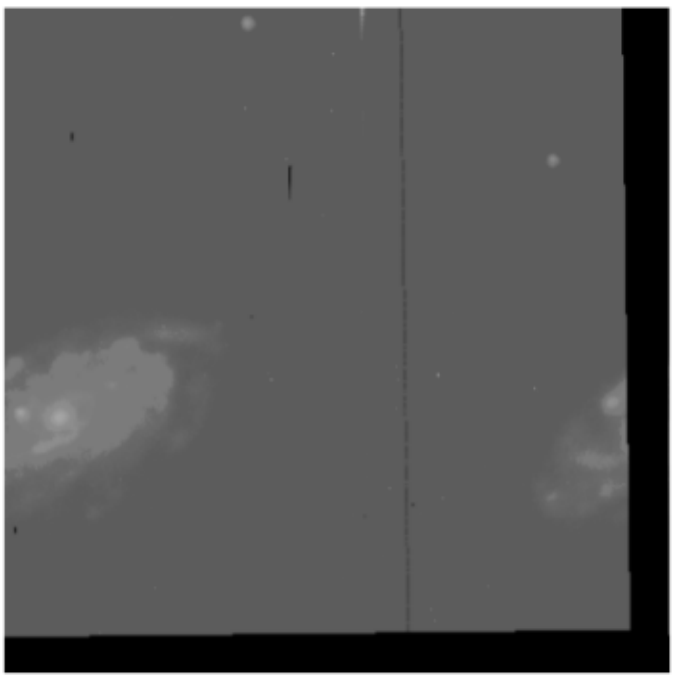








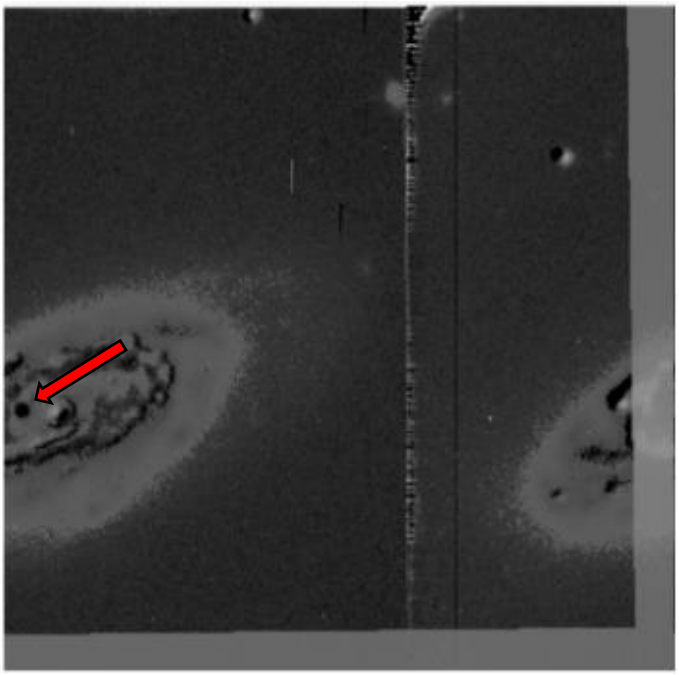
Das Ergebnis:



Dieses Bild ziehen wir vom ersten ab.

Na ja, es geht auch besser!

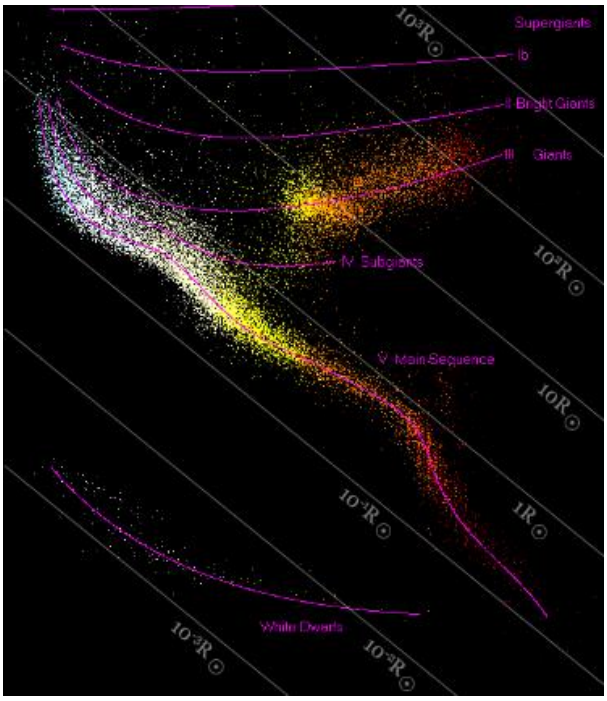
Aber immerhin haben wir in der Galaxie einen neuen leuchtenden Fleck gefunden! (Der ist hier schwarz, weil wir das Bild mit dem Supernova-Kandidaten abgezogen haben. Aber das wussten wir ja vorher nicht. 😉)



Aufgaben:

1. Suchen Sie Aufnahmen des gleichen Himmelsbereichs mit und ohne Nova und verarbeiten Sie diese wie gezeigt.
2. Automatisieren Sie Teile des Suchvorgangs – soweit möglich. Diskutieren Sie die Schwierigkeiten.
3. Kann man die Genauigkeit der „Handarbeit“ bei der Supernovasuche verbessern, etwa indem man die angeklickten Sterne besser fokussiert? Versuchen Sie es!

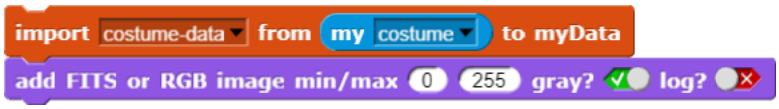
5.6 Klassifizierung von Sternen nach dem kNN-Verfahren



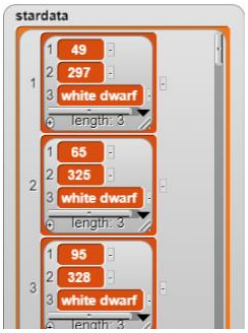
Im Hertzsprung-Russel-Diagramm (s. Wikipedia) wird die Leuchtkraft von Sternen über ihrer Sternklasse aufgetragen. Es ergibt sich einen Art Linie von links-oben-nach rechts-unten, die „Hauptreihe“. Auf dieser halten sich Sterne wie die Sonne überwiegend auf. Recht-oben über der Hauptreihe finden wir die Roten Riesen, links-unten unter der Hauptreihe die Weißen Zwerge. Das recht erstmal. (Bildquelle: [HR])

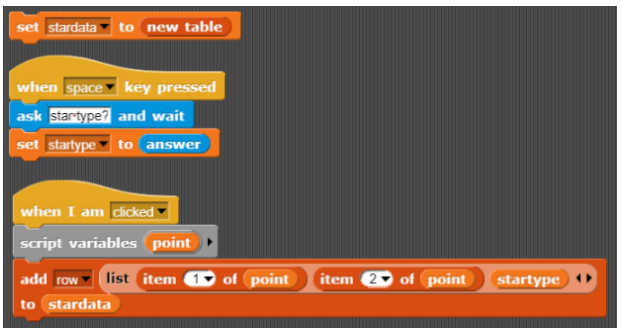
Wir wollen neue Sterne in diesem Diagramm klassifizieren, indem wir das Verfahren der k-nächsten-Nachbarn (kNN) verwenden: Wir erzeugen als Trainingsdaten eine Liste von Sternen mit deren Koordinaten (einfach als Bildkoordinaten im Diagramm) und deren Typ. Wollen wir einen neuen Sternklassifizieren, dann bestimmen wir seine Position im Diagramm und suchen die nächsten k (z. B. k=5) Nachbarn. Danach bestimmen wir den am häufigsten auftauchenden Sterntyp in dieser Liste. Den weisen wir dem neuen Stern zu.

Zuerst einmal benötigen wir ein Bild des Hertzsprung-Russel-Diagramms ([HR]). Dieses importieren wir in Snap! als Kostüm und erzeugen daraus die benötigten Daten.



Die Trainingsdaten erzeugen wir, indem wir einen Sterntyp vorgeben und danach einige Punkte im Diagramm anklicken, die diesem Typ entsprechen.



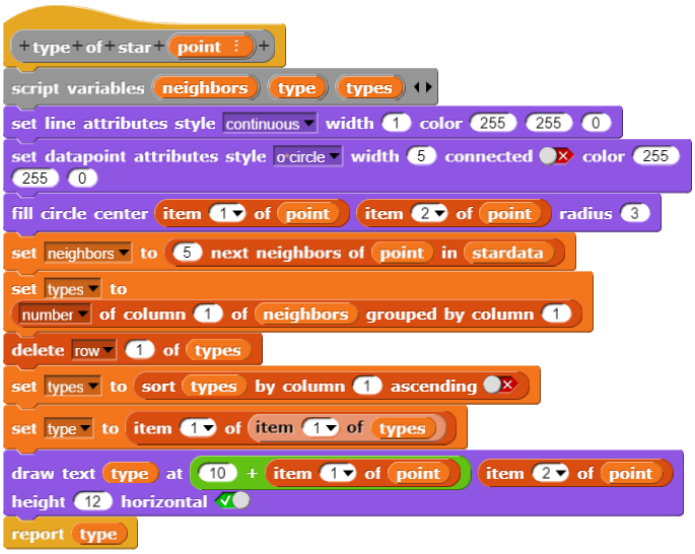


Danach können wir neue Sterne klassifizieren, indem wir sie (hier) anklicken und beschriften.

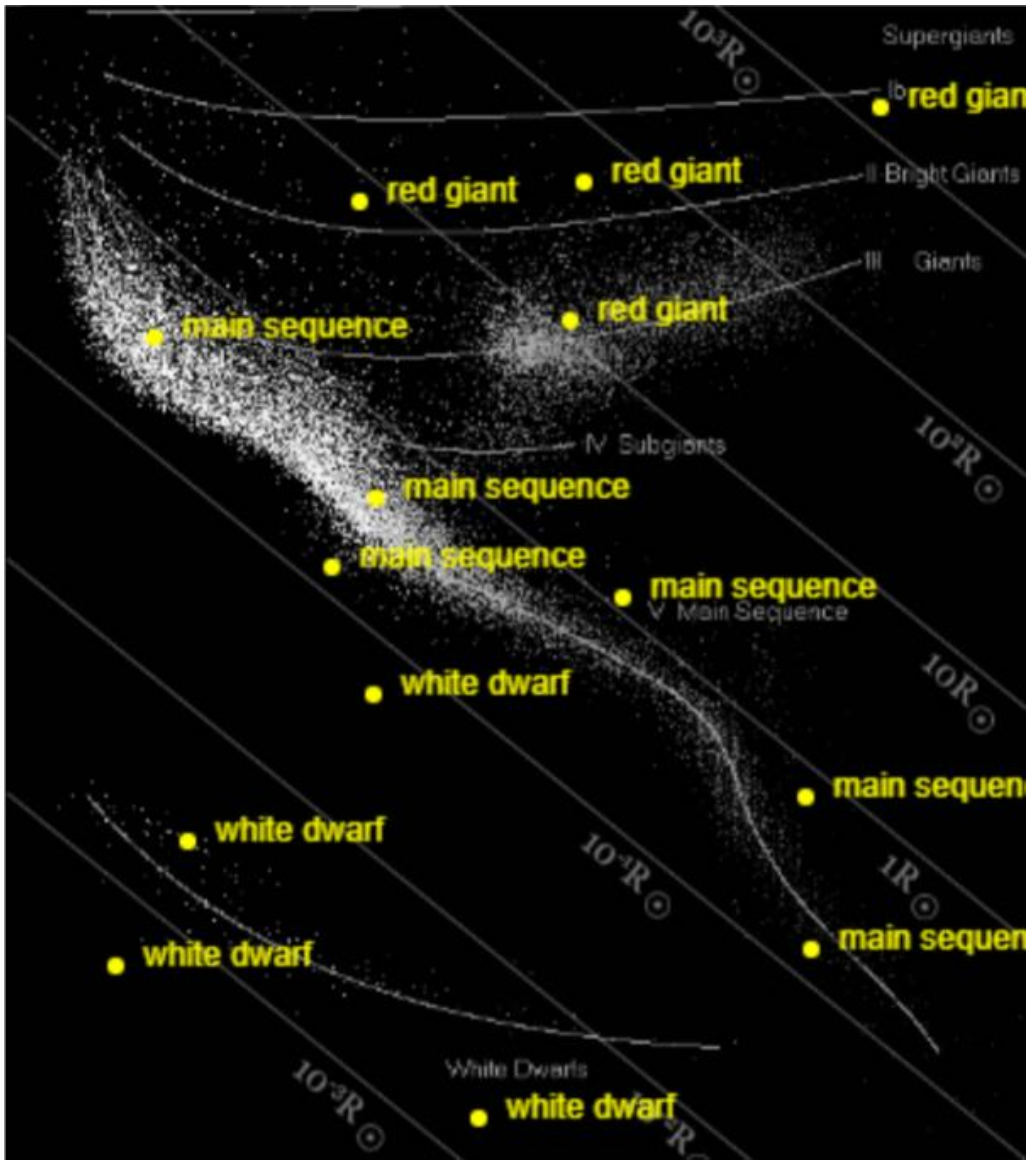
Wir stellen einige Properties für die Darstellung ein, …

… und zeichnen einen Kreis am Ort des Sterns.

Danach bestimmen wir die fünf nächsten Nachbarn und die Anzahlen des Auftretens ihres Typs. Im Ergebnis löschen wir die Überschriften und sortieren die Liste absteigend. Der Typ des neuen Sterns steht dann als erstes Element in der ersten Zeile. Diesen schreiben wir neben den Stern.



Das Ergebnis:



6 Hinweise

Die Beispiele wurden weitgehend aus der Astrophysik gewählt, weil die *DataSpriteLibrary* im nächsten Semester in diesem Bereich eingesetzt wird. Aber die Bibliotheks-Operationen sind natürlich nicht auf diesen Bereich beschränkt.

Maschinelles Lernen besteht zu einem großen Teil aus der Aufbereitung von Daten – egal, ob es sich um Tabellendaten oder Bilder handelt. Die eigentlichen Lernvorgänge der Maschinen bestehen dann aus den Parameteranpassungen, die sich aus den Daten ergeben. Da beides gut visualisierbar ist, bietet sich ein breites Feld für Programmieranfänger mit vielen Übergängen zum Bereich „Informatik und Gesellschaft“.

Beispiele für die Anwendung der Operationen der *DataSpriteLibrary*, insbesondere der Faltungen mithilfe eines Kernels, findet man reichlich in [DBV].

Literaturhinweise und Quellen

[Albon] Albon, Chris: Machine Learning Kochbuch, O'Reilly, 2919

[Baumann] Baumann, Rüdeger: Didaktik der Informatik, Klett, 1990

[DBV] Burger, W., Burge, M.-J-: Digitale Bildverarbeitung – Eine Einführung mit Java und ImageJ, Springer 2006

[FITS] de.wikipedia.org/wiki/Flexible_Image_Transport_System

[Goodfellow] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning, MIT Press, 2016

[Grus] Grus, Joel: Einführung in Data Science, O'Reilly, 2016

[HOU] Hands-On Universe: handsonuniverse.org/

[HR] https://studylibde.com/doc/2985884/hertzsprung-russell--und-farb-helligkeits

[JSON] Popular Baby Names: https://catalog.data.gov/dataset/most-popular-baby-names-by-sex-and-mothers-ethnic-group-new-york-city-8c742

[NYcitibike] https://www.citibikenyc.com/system-data

[Minsky] Minsky, Marvin: Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, New York, 1967

[Modrow1] Modrow, Eckart: Technische Informatik mit Delphi, emu-online, 2004

[Modrow2] Modrow, Eckart: Zur Didaktik des Informatikunterrichts – Band 2, Dümmler, 1992

[Rojas] Rojas, Rau´l: Neural Networks - A Systematic Introduction, Springer Berlin, 1996

[SAP] www.sap.com/germany/products/leonardo/machine-learning.html

[SQL] Modrow, Eckart: Informatik mit Snap!, http://ddi-mod.uni-goettingen.de/InformatikMitSnap.pdf

[SZ] Süddeutsche Zeitung: 10. April 2019 www.sueddeutsche.de/wissen/schwarzes-loch-bild-1.4404130

[UniGOE] Institut für Astrophysik, Universitaet Goettingen