Julius-Maximilians-Universität Würzburg

Bachelor Thesis

Simulation von Fermi-LAT Blazar Lichtkurven mit Zufallsprozessen durch Optimierung der Leistungsdichtespektren

Author:
Niklas Richter

Supervisor:
Prof. Dr. Karl Mannheim

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science

ZUSAMMENFASSUNG

Die im Rahmen dieser Arbeit entwickelte Software zeigt, dass es möglich ist, künstliche Lichtkurven mithilfe von Normalverteilungen zu erzeugen, die nicht nur optisch ein plausibles Erscheinungsbild abgeben, sondern auch in Leistungsdichtespektren (auch PSDs) eine für Blazare charakteristische Eigenschaft erfüllen. Nach (Abdo, Ackermann, Ajello et al., 2010) beträgt die Steigung der ausgleichenden Geraden dort im Mittel für die 9 hellsten FSRQs -1.4 ± 0.1 und die 6 hellsten BL Lacs -1.7 ± 0.3. Die Vergleichsdatensatz stammt vom Fermi Gamma-ray Space Telescope und umfasst Zeitreihen der Energieflüsse von 2278 Blazaren. Um eine große Vielfalt an Lichtkurven bei der Generierung gewährleisten zu können, greift das Programm oft auf zufällig errechnete Werte zurück und berechnet anschließend aus diesen die Strahlungswerte. Die Arbeit legt ausführlich dar, welche Kriterien dabei zu beachten sind und beschreibt detailliert ihre Umsetzung. Eine tiefgründige Analyse der künstlichen Lichtkurven in den PSDs stellt heraus, dass diese dort große Ähnlichkeiten mit denen der realen Quellen aufweisen und eine mittlere Steigung von $mean_{gen} = -1.616 \pm 0.176$ erreicht werden kann. In der Folge werden mehrere Parameter, die direkten Einfluss auf die Erzeugung der Lichtkurven besitzen, näher untersucht und nach ihrer Anpassung die Wirkung im PSD bestimmt. Dabei stellt sich heraus, dass die Modifikation mehrerer Variablen (u.a. Anzahl der verwendeten Zufallswerte als Basis für eine Normalverteilung, Anzahl der integrierten Strahlungsausbrüche, Amplitude eines Ausbruchs) eine Änderung in der mittleren Steigung der Ausgleichsgeraden zur Folge hat. Während beispielsweise eine Eingrenzung auf maximal drei Ausbrüche pro Lichtkurve zu einem Anstieg des Durchschnitts auf $mean_{1-3} = -1.725 \pm 0.180$ führt, sorgt ein Herabsetzen der verwendeten Zufallswerte für eine Normalverteilung auf $n = 50$ für einen Abfall auf $mean_{50} = -1.382 \pm 0.149$. Es existieren demnach mehrere Möglichkeiten, um das Verhalten der Lichtkurven in den resultierenden PSDs zu beeinflussen, was die Software zu einem interessanten Werkzeug im Bereich der Generierung künstlicher Datenreihen macht.

INHALTSVERZEICHNIS

1. EINLEITUNG

Im Jahre 2008 wurde das Fermi Gamma-ray Space Telescope gestartet, um im Bereich der Gammastrahlen-Astronomie neben Pulsaren, stellaren Schwarzen Löchern und solaren Strahlungsausbrüchen auch aktive Galaxienkerne, insbesondere die Untergruppe der Blazare, zu untersuchen und deren Strahlung regelmäßig zu messen (Atwood et al., 2009). Für Astronomen ist die Variabilität einer Quelle von großem Interesse, da sich aus ihr vielerlei Eigenschaften über den Himmelskörper ableiten lassen. Eine nähere Analyse lässt Rückschlüsse auf die Entstehung der Strahlung zu, woraus Erkenntnisse über Vorgänge im Inneren der Quelle gewonnen werden können.

Mit steigender Beobachtungsdauer, wachsen auch die generierten Datenmengen an, die manuell ausgewertet werden müssen. Eine gegebenenfalls künftig automatisierte Auswertung der Energieflüsse mithilfe künstlicher Intelligenz benötigt bestenfalls eine größtmögliche Anzahl an Trainingsdaten, um zuverlässige Ergebnisse erzielen zu können.

Im Rahmen dieser Arbeit wird eine Software entwickelt, die große Mengen imitierender Energieflüsse generieren kann. Die resultierenden Daten ahmen, auf einer fingierten Zeitachse aufgetragen, sogenannte Lichtkurven nach. Um den Realitätsbezug der dadurch entstandenen Datenreihen zu wahren, sollen diese, abgesehen von einem optisch angemessenen Erscheinungsbild, auch vergleichbare Eigenschaften in der Analyse aufweisen. Eine charakteristische Eigenschaft der hellsten Blazare ist demnach die mittlere Steigung der Datenpunkte in einem Leistungsdichtespektrum (Abdo, Ackermann, Ajello et al., 2010). Es wird also untersucht, ob die generierten Lichtkurven in dem auch genannten „Power spectral density (PSD)“ vergleichbare Ergebnisse erzielen können. Die Anpassung programminterner Parameter prüft, ob hierdurch gegebenenfalls auftretende Abweichung kompensiert werden können.

Nach einer kurzen Einführung in die für die Arbeit relevanten Komponenten, wird die entwickelte Software vorgestellt und näher erläutert. Es wird gezeigt, welche Bibliotheken Anwendung finden und welche Möglichkeiten die Benutzeroberfläche zur Lichtkurvengenerierung bietet. Anschließend wird vorgestellt, welche Überlegungen hinter der Erzeugung der optischen Struktur stecken und welche speziellen Merkmale dabei aufgegriffen wurden, um eine angemessene Umsetzung zu erzielen. Nachfolgend wird detaillierter auf das Leistungsdichtespektrum einge-

gangen und ihre Eigenschaften in Bezug auf Blazare verdeutlicht. Hierbei werden erstmals Gemeinsamkeiten und Unterschiede zwischen den realen und generierten Lichtkurven herausgestellt. Abschließend soll der Einfluss der einzelnen Parameter auf die Eigenschaften der Lichtkurve im PSD analysiert, und weiter untersucht werden, ob und inwiefern eine Annäherung an die charakteristischen Werte durch Anpassung der Parameter erreicht werden kann.

2. WISSENSCHAFTLICHER HINTERGRUND

Unter dem Begriff der Variabilität versteht man in der Astronomie die Veränderung eines Objekts hinsichtlich ihrer Eigenschaften. Im Folgenden zielt der Terminus speziell darauf ab, die Helligkeitsänderung eines astrophysikalischen Objekts zu beschreiben. Herbeigeführt werden kann diese beispielsweise durch einen Stern in einem Doppelsternsystem, der sich vor seinen Begleiter schiebt und dabei dessen Licht teilweise vor der Erde abschirmt. Auch die Ausbrüche von (Super-)Novae oder die Annäherung eines Himmelskörpers an ein Supermassives Schwarzes Loch (SMSL) können Variabilitäten eines Objekts hervorrufen. Derartige Helligkeitsänderung reichen von einem tausendstel einer Magnitude bis hin zu zwanzig Magnituden.

Aber nicht nur die Variabilität einzelner Sterne, sondern auch die ganzer Sternregionen und -systeme, wie zum Beispiel den aktiven Galaxienkernen (AGN), sind von besonderem Interesse. Mit $10^{42}\,\frac{erg}{s} \leq L \leq 10^{48}\,\frac{erg}{s}$ zählen sie zu den leuchtkräftigsten Objekten im gesamten Universum (Krolik, 1999). BL Lac Objekte, eine Untergruppe der Blazare, können innerhalb von Minuten stärkste Helligkeitsänderungen aufweisen (Albert et al., 2007). Ihre Emissionen scheinen hauptsächlich dem relativistischen Jet zu entspringen, der unter einem kleinen Winkel beobachtet wird. Durch die Beobachtung dieser Variabilitäten in verschiedensten Spektralbereichen lassen sich Informationen über die physikalischen Prozesse im Inneren des Blazars, wie der Teilchenbeschleunigung, dem relativistische Beaming oder dem Ursprung eines Ausbruchs gewinnen (Abdo, Ackermann, Ajello et al., 2010).

2.1 Blazare

Jede Galaxie besitzt in ihrem Zentrum ein SMSL, das eine Masse von $10^7 \leq \frac{M}{M_\odot} \leq 10^{10}$ erreichen kann (Vestergaard & Peterson, 2006). Für das statische schwarze Loch ist die Größe durch den Schwarzschildradius $r_s = \frac{2\,G\,M}{c^2}$ gegeben (Jovanović & Popović, 2009), die ungefähr der Ausdehnung unseres Sonnensystems entspricht. Dessen enorme Gravitation sorgt dafür, dass Materie aus der Umgebung auf das SMSL akkretiert wird. Man nimmt an, dass diese wiederum durch magnetohydrodynamische Prozesse beschleunigt und kollimiert wird, wodurch der bekannte rela-

tivistisch Jet entsteht (Krolik, 1999). Dieses Verhalten kann unter anderem durch den Blandford-Payne-Prozess beschrieben werden (Blandford & Payne, 1982). Dieser Materiestrom tritt bipolar auf und steht senkrecht zur Akkretionsscheibe. Lässt sich der kosmische Jet von der Erde aus einem nur kleinen Winkel erfassen, spricht man von einem Blazar.

Blazare zählen zu den hellsten und energiereichsten Objekten im uns bekannten Universum. Sie gehören zur Gruppe der aktiven Galaktischen Kerne und lassen sich weiter in BL Lacerta (BL Lac) Objekte und flat-spectrum radio quasars (FSRQs) unterteilen. Beide weisen ein nicht-thermisches Spektrum auf und besitzen im optischen Bereich eine Polarisation von wenigen Prozent. BL Lacs enthalten in diesem Spektralbereich nur sehr schmale bis gar keine Emissionslinien, im Gegensatz zu FSRQs, die verstärkt breitere Resonanzlinien zeigen und zudem leuchtstärker sind (Sambruna, 1997). Dennoch lassen sich zu Zeiten geringer Leuchtkraft auch bei den BL Lac Objekten Emissionslinien beobachten (Giommi et al., 2008), was vermuten lässt, dass der gewaltige Synchrotonjet diese einfach überstrahlt. Da der Jet des Blazars unter einem kleinen Winkel betrachtet wird, lässt sich aufgrund relativistischer Effekte eine scheinbare Überlichtgeschwindigkeit des Jets und eine erhöhte Leuchtkraft feststellen. Charakteristisch für beide Blazar-Arten ist die Doppelhöckerstruktur in der spektralen Energieverteilung, die auf die Synchrotonstrahlung und inverse Comptonstrahlung zurückzuführen sind (Giommi et al., 2008) (Figur 2.2).

2.2 Fermi Gamma-ray Space Telescope

Aktive Galaxienkerne strahlen über das gesamte Spektrum verteilt hohe Mengen an Energie aus. Für die vollständige Ergründung dieser Wellenlängen sind die Teleskope auf der Erde nicht ausreichend, da die Erdatmosphäre fast ausschließlich optische Wellen und Radiowellen passieren lässt (Burke, Graham-Smith & Wilkinson, 2019). Für diesen Zweck wurde das Fermi Gamma-ray Space Telescope (FGST) entwickelt und in die Erdumlaufbahn geschickt. In 565 km Höhe umkreist der Satellit innerhalb von 95 Minuten einmal die Erde und tastet mit seinem Hauptinstrument, dem Large Area Telescope (LAT), den Weltraum nach Gammastrahlung ab. Es detektiert Photonen im Energiebereich von 20 MeV bis 300 GeV, wobei sein Gesichtsfeld 20% des Himmels abdeckt (Atwood et al., 2009). Für eine Aufnahme des gesamten Himmels benötigt das Teleskop drei Stunden. Ergänzt um den Gamma-ray Burst Monitor (GBM), der zu jedem Zeitpunkt die Erkennung von Gammablitzen im Bereich von 8 keV bis 40 MeV am gesamten Himmel ermöglicht, überdecken beide Instrumente zusammengenommen einen Energiebereich, wie er bislang bei keiner Raumflugmission erreicht wurde (Meegan et al., 2009).

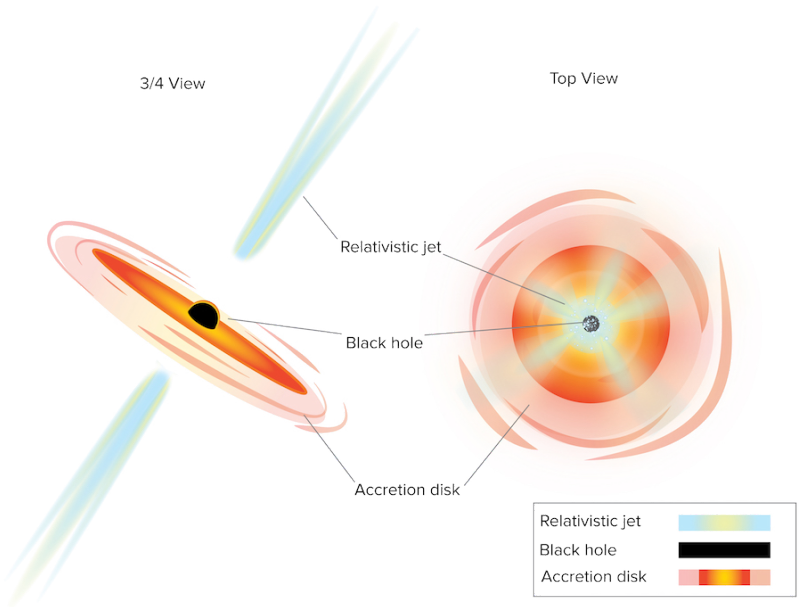

Fig. 2.1: Aufbau eines Blazars (Dagnello, o. J.)

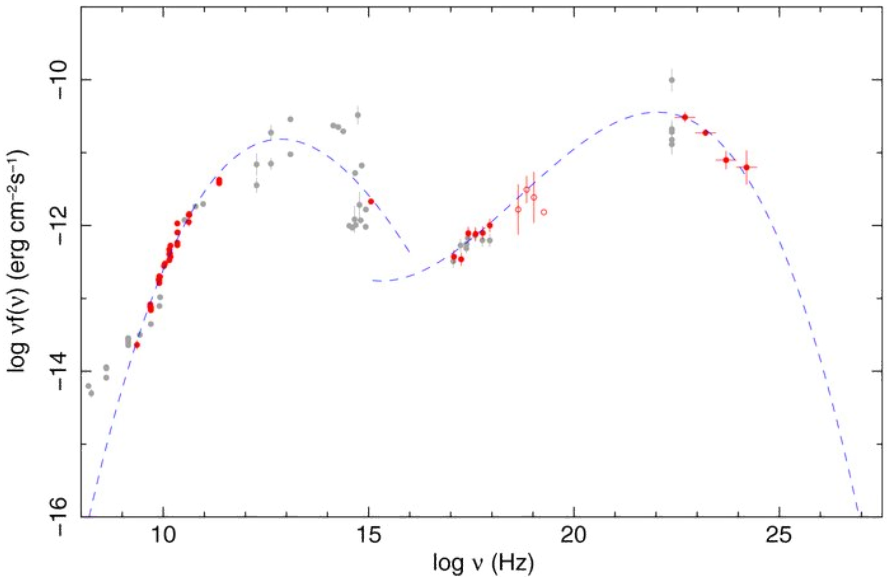

Fig. 2.2: Spektrale Energieverteilung des Blazars 4C29.45 (Abdo, Ackermann, Agudo et al., 2010).

Primäre Aufgabe des Teleskops ist die die Erforschung der Ursprünge energiereicher Gammastrahlung. Dazu gehören aktive Galaxienkerne, Pulsare, stellare Schwarzen Löcher, Gammablitze und Flares der Sonne und von Sternen (Atwood et al., 2009). Seit seinem ersten wissenschaftlichen Einsatz zwei Monate nach dem Start, hat das FGST mit seinem Hauptinstrument bis Mitte 2018 mehr als 5000 individuelle Gammastrahlenquellen ausfindig gemacht und mit dem GBM über 2300 extraterrestrische und 5000 terrestrische Gammablitze in Erdatmosphäre entdeckt. Durch ihn waren wir bisher in der Lage, den uns noch relativ unerforschten Wellenlängenbereich und seine relativistische Teilchenbeschleunigung besser zu verstehen und erklären zu können (Kazmierczak, 2018).
Die mir zur Verfügung gestellten Energieflüsse, die dieser Arbeit zugrunde liegen und als Richtlinie dienen, sind Messungen des Fermi Gamma-ray Space Teleskops. Sie liegen als Datei im FITS-Format vor.

2.3 Lichtkurven

Fermi misst die Anzahl der auf die Instrumente auftreffenden Photonen, woraus sich beispielsweise auf die Energieflüsse schließen lassen. Lichtkurven eignen sich an dieser Stelle hervorragend, um die zeitliche Änderung der Strahlungsintensität für eine Quelle darzustellen. Es handelt sich hierbei um eine zweidimensionale Abbildung, bei der der Fluss (Y-Achse) auf eine Zeitachse (X-Achse) aufgetragen wird. Da Blazare in Zeiträumen von Minuten bis Monaten hoch signifikante Variabilitäten aufweisen können, lassen sich diese hierin deutlich hervorheben (Figur 2.3). Gegebenenfalls vorhandene Regelmäßigkeiten wie Anstiegs-/Abfallverhalten, Formen der Ausbrüche oder Verhältnisse zwischen Breite und Höhe der Peaks können hierbei gut erkannt und analysiert werden.

Des Weiteren spielt das Ergründen der Variabilität in der Gammaastronomie eine sehr große Rolle, da sich aus den Energieflüssen Rückschlüsse auf die Eigenschaften der beobachteten Himmelskörper ziehen lassen. Auch die Möglichkeit der Unterscheidung zwischen tatsächlichen Punktquellen und hintergründigen Fluktuationen wird so verbessert, was die Lokalisierung schwach strahlender Quellen vereinfacht (Abdo, Ackermann, Ajello et al., 2010).

Auch werden Lichtkurven gerne herangezogen, um aus ihnen sogenannte Leistungsdichtespektren zu erzeugen. Diese zeigen, wie die Leistung eines Signals über die verschiedenen Frequenzen verteilt ist. Charakteristische Perioden in der Lichtkurve werden dann als Peak im Leistungsdichtespektrum ersichtlich. Dies ermöglicht es den Astronomen, die physikalischen Prozesse hinter der entstandenen Strahlung erkennen zu können.

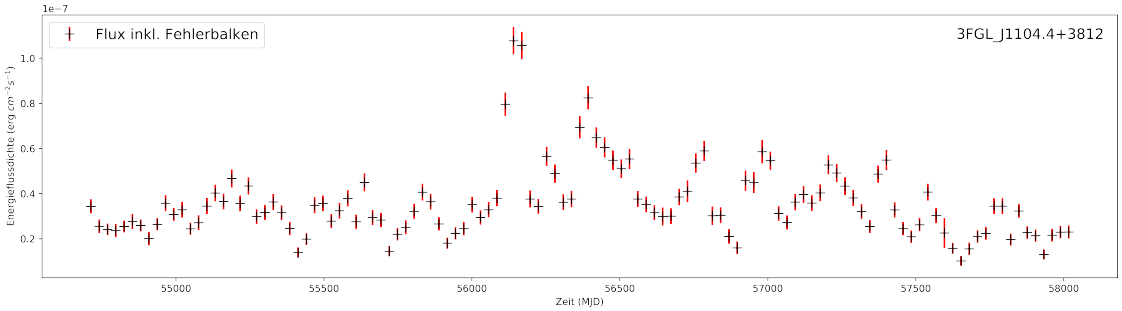

Fig. 2.3: Lichtkurve des Blazars Markarjan 421

3. SOFTWARE

Das Programm zielt auf die Generierung großer Mengen künstlicher Daten ab. Dabei setzt die Software mehr auf die effektive Berechnung von Zahlwerten als auf ein umfangreiche grafische Benutzeroberfläche. Es wird versucht, Eigenschaften der Lichtkurven mit Parametern der Simulation zu vergleichen, um eine bestmögliche Repräsentation der realen Datenreihen zu erhalten. Andere Variablen sollen wiederum möglichst zufällig gewählt werden, um eine ausreichende Diversität an Lichtkurven zu sichern. Im Rahmen dieser Arbeit sollen die Parameter so angepasst werden, dass die Eigenschaften der resultierenden Datenreihen im Leistungsdichtespektrum denen der realen Energieflüsse möglichst nahe kommen.

3.1 Verwendete Komponenten

3.1.1 Die Programmiersprache

Da sich das Programm bei seiner Verwendung hauptsächlich der Grundrechenarten bedienen soll und keine allzu spezifischen, plattformabhängigen Anforderungen stellt, habe ich mich für die Programmiersprache Python entschieden (Python Software Foundation, 2018).
Aufgrund ihrer Einfachheit lassen sich in Python üblicherweise Programme bedeutend schneller entwickeln als in anderen Programmiersprachen. Sie ist durch ihre vorgegebene Struktur sehr leicht lesbar und weist eine reichhaltige Standardbibliothek auf, die für die gängigen Betriebssysteme frei verfügbar ist. Da Python in anderen Programmiersprachen ebenfalls einen sehr guten Ruf genießt, lässt sich der Code problemlos und ohne Verlust einzelner Funktionalitäten in viele andere Systeme einbetten und integrieren. Derzeit erfährt Python auch im Bereich der künstlichen Intelligenz eine große Unterstützung, wodurch auch hier noch einmal an Kompatibilität gewonnen wird (*Python*, o. J.; Kantel-Chaos-Team, 2010).

3.1.2 Die Bibliotheken

Die wohl am häufigsten verwendete Bibliothek in diesem Programm ist NumPy. Dieses Paket ist darauf ausgelegt, das Arbeiten mit mehrdimensionalen Feldern (Arrays) zu vereinfachen und effizient numerische Berechnungen durchzuführen

(Oliphant, 2006). So wird NumPy hier meist verwendet, um normalverteilte Werte zu berechnen und Daten strukturiert abzuspeichern.

Trotz ihrer Unkompliziertheit bietet Python mit der Standardbibliothek tkinter eine flexible Möglichkeit der Implementierung einer grafischen Oberfläche, die auf allen gängigen Betriebssystemen problemlos dargestellt werden kann. Auch wenn tkinter nur eine kleine Dokumentation mit sich bringt und eine veraltete Oberfläche aufweist, gilt es als sehr schnell und und einfach umsetzbar.

Die Bibliotheken Matplotlib (Hunter, 2007), SciPy (Jones E et al., 2001–) und Astropy (Astropy Collaboration et al., 2013; Price-Whelan et al., 2018) sind in der ausgelieferten Software nicht enthalten, wurden aber zur Entwicklung des Programms und Analysen danach eingesetzt.

3.2 Benutzerschnittstelle

Um die volle Funktionalität des Programms nutzen zu können, sollte die in englischer Sprache implementierte Benutzeroberfläche verwendet werden. Diese wird automatisch nach Ausführung der Datei **lightcurve-generator.py** als Python-Code gestartet. Das sich öffnende Fenster bietet eine übersichtliche Zusammenstellung an Auswahl- und Parametrisierungsmöglichkeiten, die folgend näher erläutert werden sollen. Das NumPy-Array, das alle relevanten Informationen über die generierten Lichtkurven enthält, soll zur korrekten Auswertung und Interpretation ebenfalls näher betrachtet werden.

3.2.1 Parametrisierung

Das Programm bietet grundsätzlich zwei Varianten zur Daten-Generierung an: Einerseits lassen sich vorgegebene Parameter, die Intensität und Form der auftretenden Ausbrüche betreffen, manuell setzen, um den Aufbau der Lichtkurven einzugrenzen. Alternativ kann man diese Attribute auch von der Software zufällig generieren lassen, wodurch jede Lichtkurve unterschiedlich strukturiert wird und somit ein umfangreiches Spektrum an Lichtkurven entsteht. Nach Programmstart muss zunächst zwischen diesen Optionen entschieden werden, wobei die Auswahl zu jeder Zeit angepasst werden kann.

Bei Selektion des Punktes **specific data** (siehe Figur 3.1) können die aufgeführten Parameter für die Generierung individuell festgelegt werden. Es handelt sich hierbei um drei Attribute, die die Form der Strahlungsausbrüche in einer Lichtkurve definieren. Die Menge der Eingaben pro Attribut entscheidet darüber, wie viele Peaks die zu erzeugende Datenreihe aufweisen wird, weshalb die Anzahl der eingetragenen Werte in den Feldern Flux Positions, Flux Factors und Flux Widths identisch sein muss.

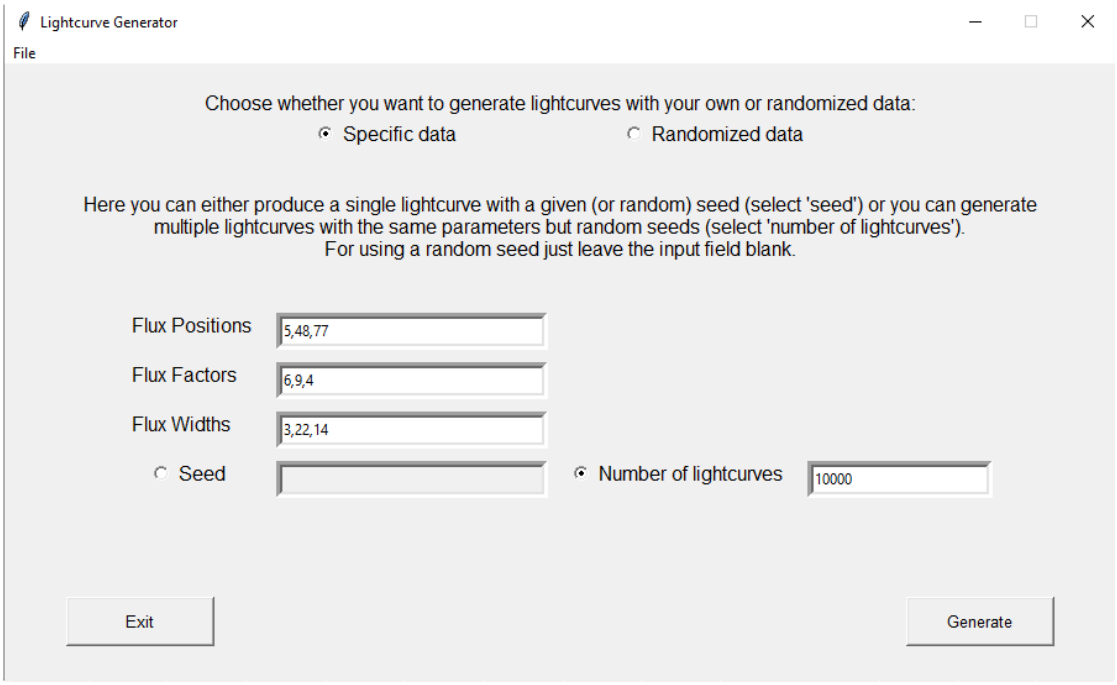

Fig. 3.1: GUI des Lightcurve-Generators. Dieses Schaubild zeigt eine Konfiguration zur Generierung von 10000 Lichtkurven. Jede einzelne von ihnen wird an den Indizes 5, 48 und 77 Strahlungsausbrüche aufweisen, wobei der Peak des Ersten 6-fach, des Zweiten 9-fach und des Dritten 4-fach größer als das Grundrauschen sein wird. Des weiteren werden diese Fluxes Halbwertsbreiten von 3, 22 und 14 Datenpunkten besitzen.

In dem Feld **Flux Positions** wird der Index vermerkt, an dem ein Ausbruch sein Maximum besitzt. Analog zu den mir zur Verfügung stehenden realen Lichtkurven können hier Indizes von 0 bis 118 vergeben werden.
Die Eingabe **Flux Factors** bestimmt die Intensität des jeweiligen Ausbruchs. Der übermittelte Wert stellt jedoch nicht die Amplitude direkt dar, sondern repräsentiert einen Faktor, mit dem das Grundrauschen einer Quelle multipliziert wird, um einen Ausschlag zu provozieren.
Über das Feld **Flux Widths** lässt sich die Breite des Peaks regulieren. Der übergebene Wert entspricht der Halbwertsbreite (engl. Full Width at Half Maximum) des Ausbruchs.
Da die Erzeugung einer Lichtkurve zum großen Teil auf Zufallszahlen basiert, werden sogenannte Seeds verwendet. Es handelt sich dabei um eine natürliche Zahl, mit der ein Zufallsgenerator initialisiert wird. Eine Folge von Zufallszahlen ist unter Angabe eines gleichen Seed-Keys also immer identisch (The SciPy community, 2019), wodurch sich eine zufällig generierte Lichtkurve anhand des abgespeicherten Seeds stets reproduzieren lässt. Wählt man in der Software die Option **Seed** aus, muss im nebenliegenden Feld eine entsprechende Zahl ($0 \leq Seed \leq 2^{32} - 1$) eingetragen werden, um eine einzelne Lichtkurve mit den angegebenen Parametern zu erzeugen.
Möchte man mehrere Lichtkurven mit den zuvor deklarierten Werten erzeugen, ist die Option **Number of lightcurves** zu wählen. Jeder Quelle wird ein zufälliger Seed-Key zugewiesen und obwohl die Positionen, Faktoren und Breiten der Fluxes bei jeder Lichtkurve identisch sind, variieren die simulierten Daten aufgrund der unterschidlichen Seeds.

Wählt man stattdessen im Kopfbereich der Software den Punkt **randomized data** aus, wird die Vergabe jeglicher in der vorigen Option noch auswählbaren Parameter vom Generator selbst übernommen und für jede einzelne Lichtkurve unterschiedlich gesetzt. Der Seed erhält einen zufälligen Wert diskreter Gleichverteilung aus dem oben aufgeführten Bereich. Auf gleiche Weise wird die Anzahl der Fluxes, die auf maximal 10 begrenzt ist, die Halbwertsbreite, die höchstens 100 Datenpunkte beträgt und die Flux-Position, die zwischen 0 und 118 liegen kann, berechnet. Der Faktor eines Fluxes entstammen jedoch einer Normalverteilung mit Erwartungswert $\mu = 10$ und $\sigma = 5$, wobei $factor > 1$ erfüllt sein muss. Einzig die Anzahl der zu generierenden Lichtkurven kann (und muss) bei dieser Option manuell über das Eingabefeld angepasst werden.

Nach Start der Generierung wird der Benutzer nach einem Speicherort und Dateinamen gefragt. Nach Abschluss des gesamten Prozesses wird man über eine erfolgreiche Durchführung informiert.

3.2.2 Programmausgabe

Die vom Generator geschaffenen Datenreihen werden in Form eines NumPy-Arrays abgespeichert. Dies ist sinnvoll, da eine große Menge an Informationen (in Zahlenform) sinnvoll strukturiert vorbehalten werden soll. Auch wenn die 119 generierten Datenpunkte von primärem Interesse sind, werden zur näheren Analyse die vom Anwender verwendeten Parameter benötigt. Aus diesem Grund beinhaltet das resultierende NumPy-Array zusätzlich zu den erzeugten Datenpunkten Informationen über Position, Faktor und Breite der Fluxes. Des Weiteren enthält das Array den Seed-Key, so wie den Wert des Grundrauschens und die Anzahl der generierten Zufallswerte pro Normalverteilung. Eine beliebige Lichtkurve kann anhand dieser Informationen vollständig reproduziert werden. Zusätzlich ermöglicht die Mitführung der Eingabeparameter im Bereich der neuronalen Netzwerke den Einsatz des supervised learnings.

Eine konkrete, beispielhafte Auswertung des Arrays ist in der Software unter dem Punkt **Evaluation** zu finden.

4. PHYSIKALISCHE RELEVANZ

Im vorigen Kapitel wurden die Möglichkeiten aufgezeigt, die ein Nutzer hat, um die Form einer Lichtkurve anhand von Parametern zu beeinflussen. Dennoch soll das Programm gewisse Freiheiten besitzen, um eine Diversität unter den Lichtkurven aufzuzeigen und diese nicht zu statisch zu belassen. Da die Datenreihen die emittierte Strahlung unterschiedlicher Quellen darstellen sollen, ist der Randomisierungsprozess einzelner Datenwerte in der Software ein Hauptbestandteil. Was macht also eine Liste zufällig generierter Datenpunkte zu einer realistischen Lichtkurve? Im Folgenden wird gezeigt, wie die Software mithilfe der Normalverteilung versucht, reale Quellen zu imitieren. Des weiteren soll diskutiert werden, ob die daraus resultierenden Daten in einem Leistungsdichtespektrum ebenfalls vergleichbare Eigenschaften aufweisen können.

4.1 Optischer Abgleich

Im ersten Schritt sollen die optische Merkmale und Auffälligkeiten ausfindig gemacht und analysiert werden. Der Vergleichsdatensatz liegt im FITS-Format vor und umfasst 2278 Lichtkurven mit einem Energiebereicht $> 1GeV$, die für den Zeitraum 54698,775 bis 58030,775 MJD monatlich gebinnt wurden. Daraus resultieren 119 Strahlungswerte, die jeweils die Energieflüsse eines Monats Beobachtung repräsentieren. Da jede Messung auch einen individuellen Messfehler mit sich bringt, existiert zu jedem Datenpunkt eine sogenannte Teststatistik (TS). Diese prüft, wie wahrscheinlich es ist, dass der Wert tatsächlich einer gemessenen Quelle entspricht. In alle folgenden Analysen und Berechnungen wurden ausschließlich Datenpunkte einbezogen, die eine TS > 9 besitzen.

4.1.1 Basismodell

Bei erster Betrachtung der Lichtkurven aus dem vorliegenden Datensatz (Figur 2.3, Figur 4.1) lassen zunächst keine regelmäßigen Muster erkennen. Gelegentliche Anstiege der Energieflüsse, die zumeist von einem nachfolgenden Abfall begleitet werden, zeigen unterschiedlich starke Intensitäten und Breiten. Ausreißerische Werte, die sich in kleineren Bereichen aufeinanderfolgender Datenpunkte deutlich abgrenzen, erschweren das Erkennen einer zusammenhängenden Struktur.

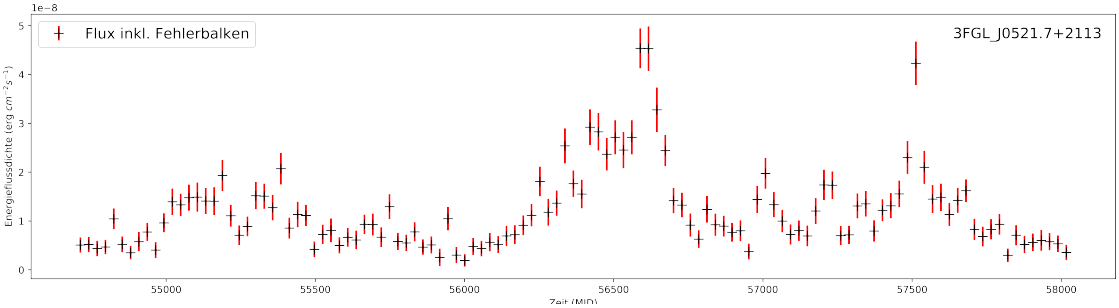

Fig. 4.1: Lichtkurve der Quelle TXS 0518+211: Um die Zeit 56000 baut sich der Peak allmählich auf und fällt extrem schnell ab, während bei 57500 sowohl Anstieg als auch Abfall rasant sind.

Fokussiert man sich stattdessen auf die größeren Ausbrüche, lassen sich hier in vielen Fällen im mathematischen Sinne exponentielle Anstiege bzw. Abfälle beobachten. Jedoch trifft dies keineswegs auf alle Lichtkurven zu. Das Programm muss in der Lage sein, auch asymmetrische Ausbrüche darstellen zu können und sogar innerhalb eines einzelnen Fluxes weitere Aktivität zu zeigen (siehe Figur 4.1). Da dieser Ansatz ein zu stark vereinfachtes Basismodell darstellen würde und in der Folge mehr Aufwand und Komplexität in der Umsetzung spezieller Verhaltensweisen bedeutete, betrachtete man das Ganze noch einmal aus einem anderen Blickwinkel.

Ein weiteres Konzept entstammt aus dem Bereich der Statistik. In der wissenschaftlichen Arbeit **Studies in astronomical time series analysis. VI. Bayesian block representations** (Scargle, Norris, Jackson & Chiang, 2013) hat man sich zum Ziel genommen, astronomische Zeitreihendaten mithilfe Bayesscher Blöcke zu repräsentieren, um signifikante Variabilitäten zu identifizieren und charakterisieren. Vorliegende Lichtkurven werden dabei in mehrere Intervalle gegliedert, die jeweils durch eine Verteilung nachgestellt werden, wobei die Höhe der Blöcke die Energieflüsse darstellt und der Grad der Anpassung an die reale Lichtkurve vom gewählten Wert σ abhängt. Inspiriert von der Idee der Verwendung von Wahrscheinlichkeitsverteilungen wurde nun untersucht, inwiefern ein solcher Ansatz die Generierung künstlicher Lichtkurven unterstützen kann. Der Einsatz einer Normalverteilung, dessen Form relativ leicht über das σ modifiziert werden kann, scheint dabei eine geeignete Möglichkeit darzustellen. Die Beobachtung, dass beim Großteil der Lichtkurven die maximale Intensität eines Ausbruchs nur eine kurze Zeit währt und die Energie nach dem Abfall um einem geringeren Gleichgewichtswert herumschwankt, lässt sich hervorragend in einer Gaußverteilung widerspiegeln. Gegenüber dem vorigen Gedanken besteht hierin der Vorteil, dass die Steigungen des Anstiegs bzw. des Gefälles über die Größe σ angepasst werden können und nicht zwingendermaßen eine exponentielle Form aufweisen müssen.

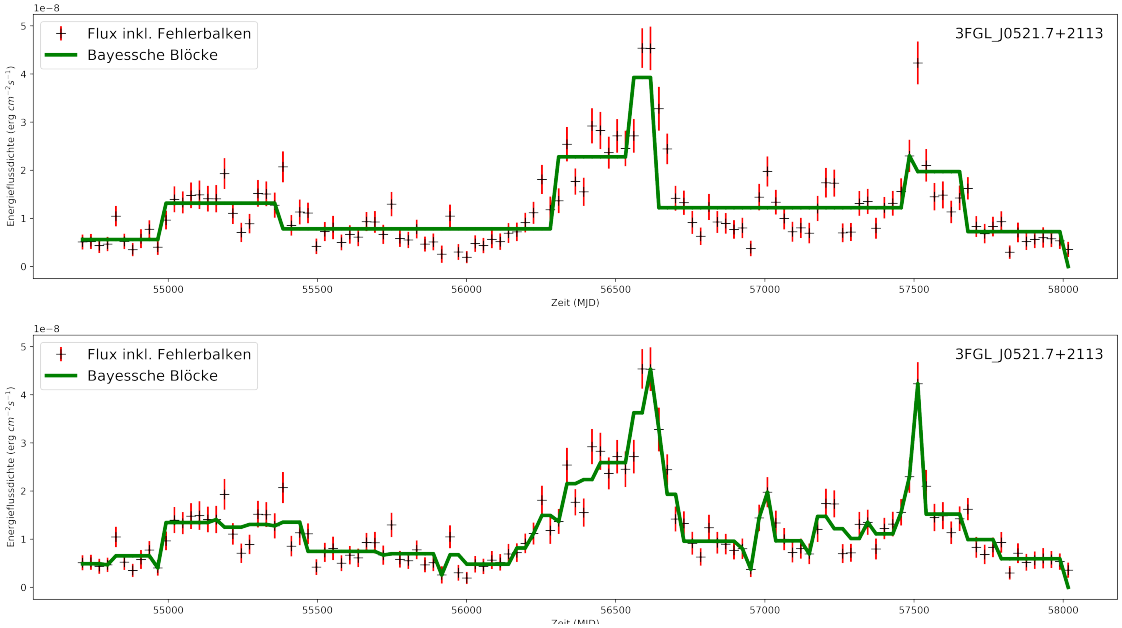

Fig. 4.2: Lichtkurven der Quelle TXS 0518+211: Anwendung der bayesschen Blöcke (nach Scargle) mit $\sigma = 5.0 \cdot 10^{-9}\, erg\, cm^{-2} s^{-1}$ (oben) und $\sigma = 1.0 \cdot 10^{-9}\, erg\, cm^{-2} s^{-1}$ (unten).

Unter Berücksichtigung der Verwendung randomisierter Werte lassen sich auch hierbei asymmetrische Fluxes deutlich besser und zufälliger erzeugen. (Scargle et al., 2013) hat gezeigt, dass sich astronomische Zeitreihendaten durch seine Bayesschen Blöcke repräsentieren lassen. Aufgrund dessen soll die Normalverteilung der Generierung künstlicher Energieflüsse zugrunde gelegt werden.

4.1.2 Implementierung

Die Idee ist, zunächst erst einmal die gesamte Lichtkurve ohne ihre Ausbrüche mit einem gleichmäßigen Grundrauschen, im Weiteren auch als Baseline bezeichnet, zu versehen. Im nächsten Schritt würden dann anhand der Parameter des Benutzers die Peaks mit ihren Höhen und Breiten hinzugefügt werden, um die Lichtkurve zu komplettieren.

In der Software wird davon ausgegangen, dass in einer Lichtkurve die Energieflüsse nach einem oder mehreren aufeinanderfolgenden Ausbrüchen immer wieder auf ein geringeres Level, ihre sogenannte Baseline, abfallen und um diesen Wert fluktuieren. Dieser kann willkürlich gewählt werden, da die in dieser Arbeit erzeugten Energieflüsse normalverteilten Zufallswerten entstammen und somit dimensionslos sind (arbitrary unit). Eine Anpassung des Wertes hat keine Auswirkung auf die Datenpunkte, da die Maxima der anschließend integrierten Ausbrüche nicht ab-

solut, sondern relativ zum Grundrauschen gesetzt werden (siehe Parameter Flux Factor). Um dennoch eine plausible Größenordnung widerzuspiegeln, wurde der Wert hier auf $baseline = 1.0 \cdot 10^{-10}$ festgelegt. Im Sinne der Lichtkurven handelt es sich bei einem Grundrauschen jedoch nicht um einen konstanten Wert, sondern um eine gedachte Baseline, um die die Datenpunkte fluktuieren. Dies wird erreicht, indem für jeden der 119 Datenpunkte eine Normalverteilung erzeugt wird und aus den zu Grunde liegenden Werten der Maximalwert den Energiefluss in der Lichtkurve repräsentiert. Hierdurch erreicht man zwar keine Oszillation um die Baseline, jedoch eine um einen leicht erhöhten Wert. Mit dem Ziel, möglichst unterschiedliche Werte zu generieren, verwenden alle Datenpunkte den gleichen Erwartungswert μ, jedoch unterschiedliche Standardabweichungen σ. Diese stammen aus einem einfachen Zufallsgenerator mit diskreter Gleichverteilung aus dem Bereich 0 bis zum Baseline-Wert.

```
for index in range(119):                                          1
        baseline = 0.0000000001                                   2
        sigma = numpy.random.uniform(low=0, high=baseline         3
           /5, size=1)
        values = numpy.random.normal(loc=baseline, scale=         4
           sigma, size=250)
        Y[index] = max(values)                                    5
```

Listing 4.1: Verkürzter Codeauszug aus der Generierung des Grundrauschens: Die Methode **uniform** generiert einen Zufallswert (size=1) aus einer gleichmäßigen Verteilung, die durch „low“ und „high“ begrenzt ist, wohingegen **normal** 250 gaußverteilte Werte erzeugt. Abschließend wird der Maximale Wert der Liste **values** als „Strahlungswert“ verwendet.

Bei den Indizes, die vom Benutzer als Flux deklariert worden sind, wird noch vor dem Aufruf von **normal**-Funktion der Baseline-Wert mit dem entsprechenden Flux-Factor multipliziert. Hieraus entsteht der absolute Wert des Fluxes für die Lichtkurve, was das weitere Vorgehen bei der Flux-Berechnung vereinfacht.

Wie bereits unter 3.2.1 (Parametrisierung) erwähnt, hat der Anwender die Möglichkeit, die Flux-Positionen, -Faktoren und -Breiten zu beeinflussen. Die Höhen der Peaks wurden, wie im letzten Absatz beschrieben, bereits an den richtigen Stellen in die Lichtkurve eingearbeitet, nicht jedoch ihre Breiten. Da hierbei etwas mehr Logik anfällt, ist ihre Implementierung getrennt vom vorigen Schritt.

In 4.1.1 (Basismodell) wurde diskutiert, die Strahlungsausbrüche im Grundaufbau den Normalverteilungen nachzuempfinden. Die Eingabe des Benutzers im Feld **Flux Width** beschreibt, wie viele Datenpunkte die Halbwertsbreite (FWHM) eines Fluxes einschließen soll. Sie repräsentiert die Breite einer Gaußverteilung auf

halber Höhe seines Maximums (Full Width at Half Maximum) (Figur 4.3). Bei der Normalverteilung lassen sich FWHM und σ einfach ineinander umrechnen (Weisstein, 2002):

$$FWHM = 2\sqrt{2\ln 2}\,\sigma \approx 2,3548\,\sigma \tag{4.1}$$

$$\sigma = \frac{FWHM}{2\sqrt{2\ln 2}} \tag{4.2}$$

Setzt man jeweils die **Flux Width** in (4.2) ein, ergibt sich daraus das zu verwendende σ. Da für die Berechnung der Breiten lediglich die Form des später generierten Histogramms entscheidend ist, kann der Erwartungswert hier frei gewählt werden.

Die 250 errechneten Zufallswerte werden im darauffolgenden Schritt in die bekannte histografische Darstellung umgewandelt, um die gewünschte Gaußverteilung zu erhalten. Dafür muss zunächst noch festgelegt werden, aus wie vielen Kanten (Bins) das Histogramm bestehen soll. Diese Menge repräsentiert gleichzeitig auch später die Anzahl an Datenpunkten, die für den Peak verwendet werden. Es gilt also einen sinnvollen Wert für die Bins zu finden, der den Bereich von 4σ (95,45% der Daten) abdeckt. Messwerte, die außerhalb dieses Intervalls liegen, sind so klein, dass sie im Grundrauschen verschwinden und deshalb nicht neu gesetzt werden müssen. Da die vom Benutzer angegebene **Flux Width** den FWHM darstellt, erlangt man über folgende Formel eine sinnvolle Schätzung:

$$bins = 4\,\sigma \underset{(4.2)}{=} 4\,\frac{FWHM}{2\sqrt{2\ln 2}} = 2\,\frac{FWHM}{\sqrt{2\ln 2}} \tag{4.3}$$

Mit der Erstellung des Histogramms erhält man die relative Häufigkeitsdichte der einzelnen Kanten (Figur 4.3). Da hier ausschließlich die Verteilung von primärem Interesse ist, werden anschließend die absoluten Häufigkeiten skaliert, um die Energieflüsse der Fluxes auszudrücken. Die Werte werden in die Lichtkurve so eingepflegt, sodass das Maximum des Histogramms den Index des deklarierten Ausbruchs erhält (Figur 4.4).

```
for flux_index in range(len(flux_positions)):
        sigma = flux_widths[flux_index]/2*np.sqrt(2*np.log
            (2))
        values = numpy.random.normal(loc=0, scale=sigma,
            size=250)
        n_bins = numpy.ceil(4*sigma)
        dY,dX = numpy.histogram(values, bins=n_bins)
```

```
6
        factor = Y[flux_positions[flux_index]]/max(dY)                    7
        for i in range(len(dY)):                                          8
                dY[i] = factor * dY[i]                                    9
```

Listing 4.2: Verkürzter Codeauszug aus der Generierung der Fluxes: Hier wird über alle Indizes iteriert, die per Anwender Fluxes sind. In Zeile 3 werden die normalverteilten Daten erzeugt. Zeile 4 berechnet nach Formel (4.3) die Anzahl der bins und rundet diese auf eine natürliche Zahl auf (**ceil**). Die relativen Häufigkeiten aus dem Histogram werden nach **dY** geschrieben (Zeile 5) und anschließend skaliert. Das Maximum aus **dY** erhält den zu Beginn errechneten Flux-Wert (Zeile 7) und setzt somit den Faktor für die übrigen Werte.

Um jedoch zu verhindern, dass sich mehrere nahegelegene Ausbrüche überschreiben, werden die simulierten Energieflüsse der einzelnen Histogramme nur dann gesetzt, wenn der betreffende Index in der Lichtkurve nicht bereits zuvor angepasst wurde oder größer als der derzeitig verwendete Wert ist. Letzteres Kriterium sichert z.B. den Fall, dass ein kurzer, hoher Peak, der sich direkt vor einem breiten, flacheren befindet, dennoch in der Lichtkurve erscheint, anstatt von diesem aufgrund seiner FWHM verschluckt zu werden.

```
if not((i in updatedFluxIndexes) & (Y[i] >= dY[histIndex])) :             1
        Y[i] = dY[histIndex]                                              2
        updatedFluxIndexes.append(i)                                      3
```

Listing 4.3: Der mithilfe des Histogramms generierte Strahlungswert wird nur dann übernommen, wenn der Index nicht in der Liste der bereits überschriebenen Indizes zu finden ist, oder größer als der bisherige Wert ist.

4.2 Leistungsdichtespektrum

Nachdem man nun in der Lage ist, künstliche Lichtkurven zu generieren, die den realen optisch sehr ähneln, müssen sie auf ihren Realitätsbezug hin überprüft werden. Die Analysen können auf die unterschiedlichste Eigenschaften abzielen, wobei sich in dieser Arbeit auf die der spektralen Leistungsdichte konzentriert wird.

Man erhält sie durch Fourier-Transformation der Autokorrelationsfunktion des Signals. Das Leistungsdichtespektrum (engl. PSD) gibt Aufschluss darüber, welche Leistung die einzelnen Frequenzen besitzen. Da es sich bei einer Lichtkurve um ein nicht periodisches Signal handelt, zeigt sich hier ein breites Spektrum an aktiven Frequenzen, in dem jedoch die Leistung der niederen Frequenzen überwiegt. Legt

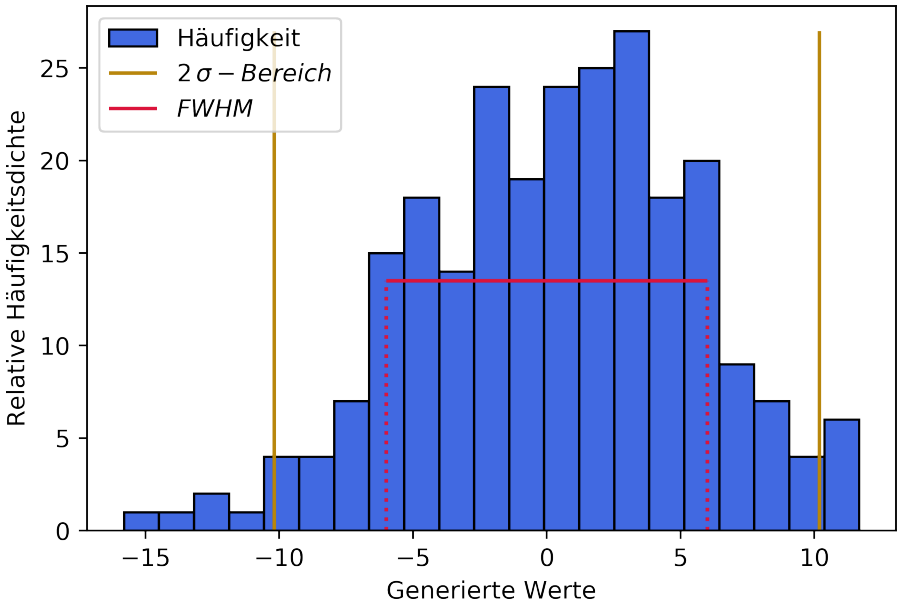

Fig. 4.3: Histogramm mit $\mu = 0$ und Flux-Width = FWHM = 12. Nach Formel (4.3) erhält man $bins = 20.4$, was auf $bins = 21$ aufgerundet wird. Die beiden beigen Linien markieren den 2σ-Bereich. Die horizontale, rote Linie repräsentiert die Halbwertsbreite auf halber Höhe des Maximums.

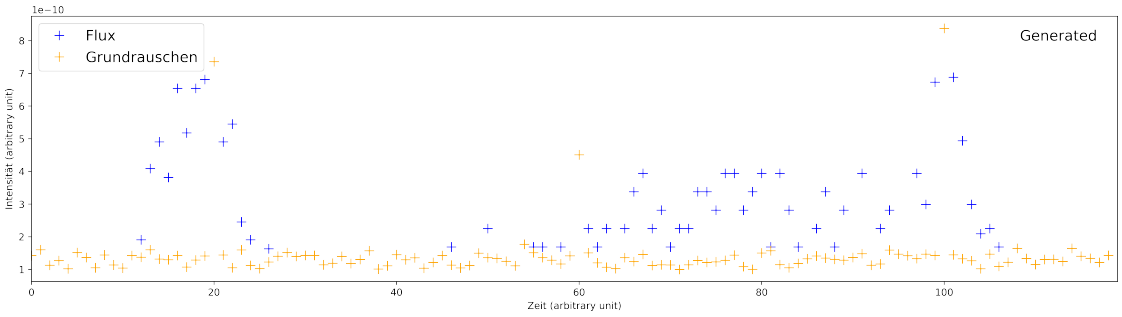

Fig. 4.4: Die orangen Punkte in der generierten Lichtkurve stellen das ursprüngliche Grundrauschen dar. Die anschließend integrierten Fluxes sind in Blau dargestellt. Die fertige Lichtkurve enthält ausschließlich alle blauen Punkte und die orangenen, die nicht von einem blauen überlagert wurden (Figur 5.6). Der Ausbruch um die Zeit 15 bis 25 entstammt dem Histogramm aus Figur 4.3.

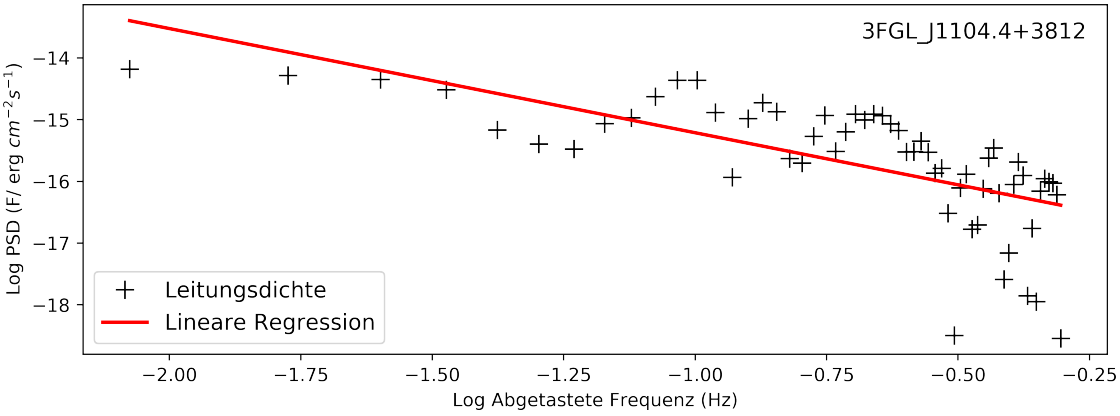

Fig. 4.5: Leistungsdichtespektrum des Blazars Markarjan 421 (Lichtkurve siehe Figur 2.3). Die Steigung der ausgleichenden Geraden (rot) beträgt $m = -1.688 \pm 0.253$.

man eine ausgleichende Gerade durch ein PSD, beträgt ihre Steigung (engl. Slope) nach (Abdo, Ackermann, Ajello et al., 2010) für die 9 hellsten FSRQs im Durchschnitt -1.4 ± 0.1 und für die 6 hellsten BL Lacs -1.7 ± 0.3 (siehe Figur 4.5). Ein solches Verhalten wird auch als Brownsches- oder rotes Rauschen (engl. „red noise“) bezeichnet. Man spricht hier von unkorrelierten Fluktuationen, bei denen die Leistungsdichte mit steigender Frequenz abfällt. Bezogen auf die Lichtkurven kann man auf längeren Zeitskalen größere Variabilitäten der Amplitude feststellen (Chatterjee et al., 2012). Ein Signal, das durchgängig ähnlich schwankt und um einen konstanten Wert zu fluktuieren scheint, nennt man weißes Rauschen (engl. „white noise“). Im PSD zeigt es eine nur geringe negative, bis gar positive Steigung (Figur 4.6, unten).

Der Astronom ist durch eine solche Auswertung in der Lage, zu erkennen, welche physikalischen Prozesse hinter der Entstehung der Energieflussdichten stecken könnten.

Vor diesem Hintergrund ist das Ziel, die Parameter der Software so zu optimieren, dass die zufällige Generierung von Lichtkurven vergleichbare Steigungen im PSD umfasst. In Folgenden soll allein auf die Erfüllung dieses Kriteriums hingearbeitet werden.

Um die Aussage von (Abdo, Ackermann, Ajello et al., 2010) nachvollziehen zu können, wurde zunächst die mittlere Steigung der vorliegenden Quellen berechnet. Figur 4.7 (links) zeigt, dass nach erster Betrachtung der Mittelwert (auch engl. „mean“) bei $mean = -0.536 \pm 0.335$ liegt, was eine deutliche Abweichung hierzu darstellt. Im Hinblick auf den Versuch der Beseitigung dieser Unstimmig-

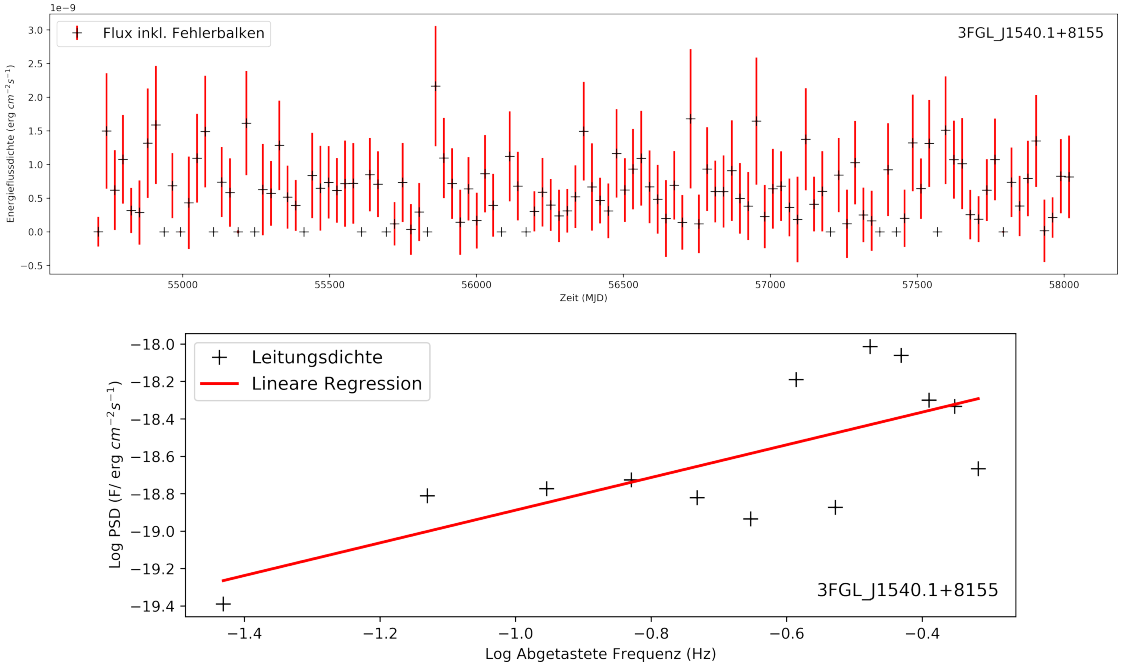

Fig. 4.6: Lichtkurve (oben) und Leistungsdichtespektrum (unten) des BL Lac-Blazars 1ES 1544+820 mit mittlerer Steigung $m = 0.873 \pm 0.249$.

keiten wurden die Korrelationskoeffizienten der linearen Regressionen näher inspiziert. Diese sagen aus, wie weit die Datenpunkte von der ausgleichenden Geraden abweichen. Sie stellt also ein Maß für die Präzision der linearen Regression dar (Lawrence & Lin, 1989). Es zeigt sich, dass fast alle Geraden mit einem Determinationskoeffizient (Quadrat des Korrelationskoeffizienten) $R^2 < 0.2$ Steigungen zwischen $-1 \leq m \leq 1$ aufwiesen (Figur 4.8). Filtert man diese heraus, verschiebt sich der Mittelwert auf $mean_{0.2} = -0.982 \pm 0.293$ und die Steigungen um 0 herum verschwinden fast vollständig (Figur 4.7 - Mitte). Eine noch striktere Filterung ($R^2 < 0.5$) zeigt dann eine mittlere Steigung von $mean_{0.5} = -1.405 \pm 0.247$, welche der obigen Aussage nahe kommt. Ein solcher Quality-Cut soll deshalb die gesamte Arbeit über mit berücksichtigt werden.

Da Lichtkurven optisch sehr unterschiedliche Erscheinungsbilder aufzeigen, ist die Entscheidung, ob eine Generierte insofern einer Realen ähnelt, subjektiv. Im Weiteren soll sich deshalb auf die Eigenschaften des Leistungsdichtespektrums fokussiert werden. Wie sehen also die Power spectral densities in unserem Falle aus?

Für die folgenden Betrachtungen wurden, identisch zu der Anzahl an vorliegenden Quellen, ein Datensatz mit 2278 zufällig generierte Lichtkurven verwendet. Auf diesen Daten beruht Figur 4.10, die starke Parallelen zu Figur 4.8 aufweist. Dies lässt vermuten, dass auch in der histografischen Darstellung große Ähnlichkeiten

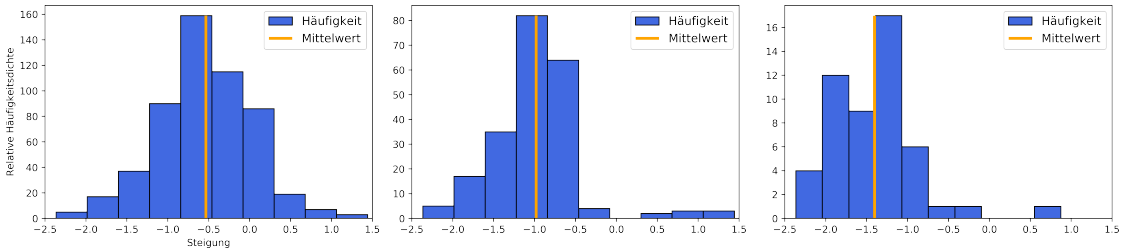

Fig. 4.7: Während im linken Histogramm alle Steigungen enthalten sind, zeigt das Mittlere nur die mit $R^2 \geq 0.2$ und das Rechte $R^2 \geq 0.5$. Die orange Linie markiert jeweils den Mittelwert aller Slopes.

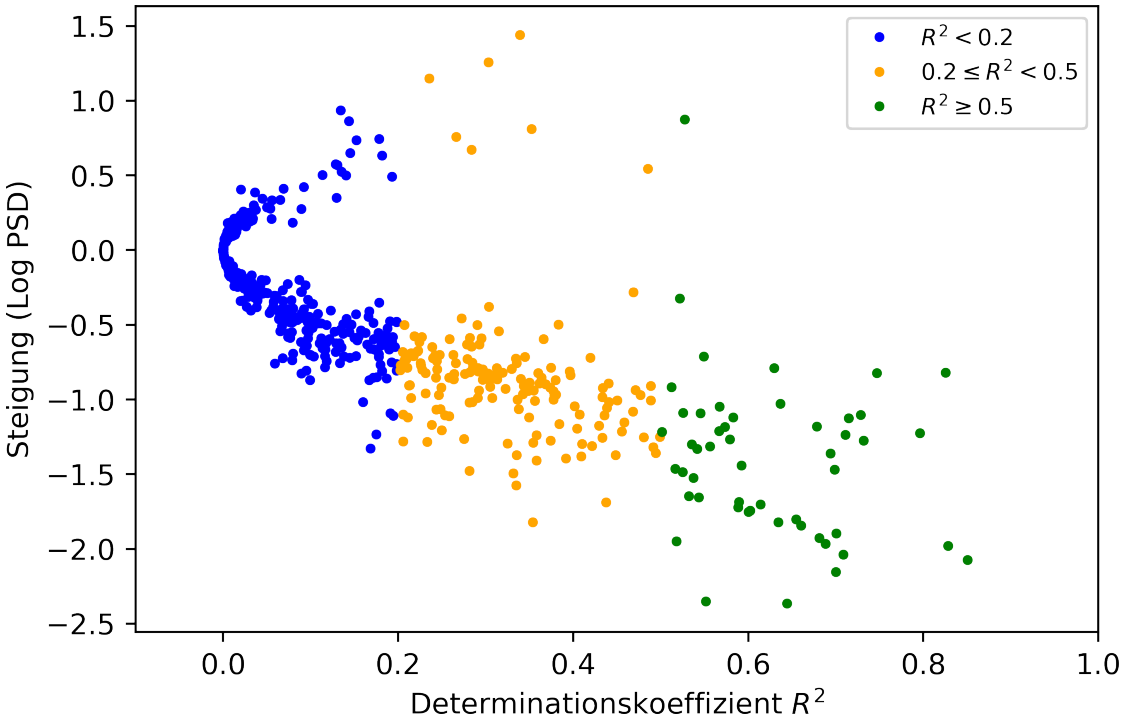

Fig. 4.8: Die nur schwach korrelierten Steigungen weisen größtenteils Werte zwischen -1 und +1 auf. Je stärker die Korrelation wird, desto weiter fällt die mittlere Steigung ab.

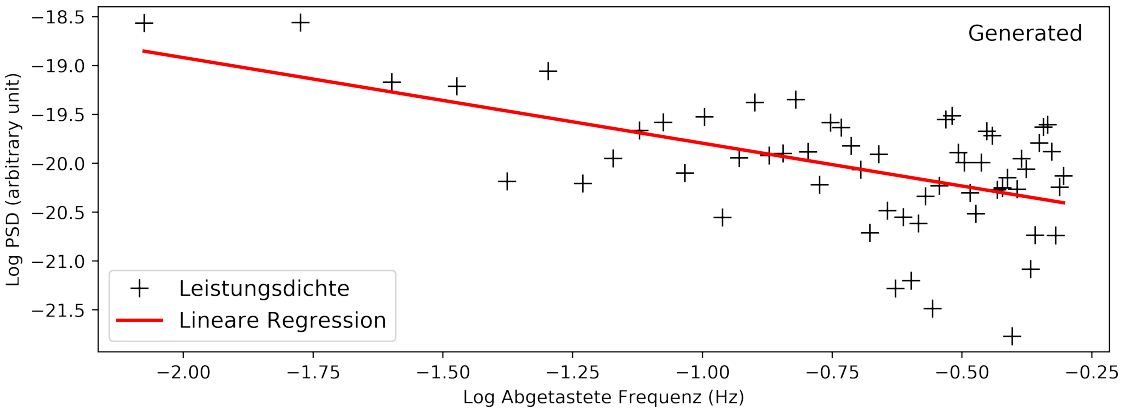

Fig. 4.9: Analoges PSD der generierten Lichtkurve aus Figur 4.4. Die Steigung beträgt hier $m = -0.876 \pm 0.175$.

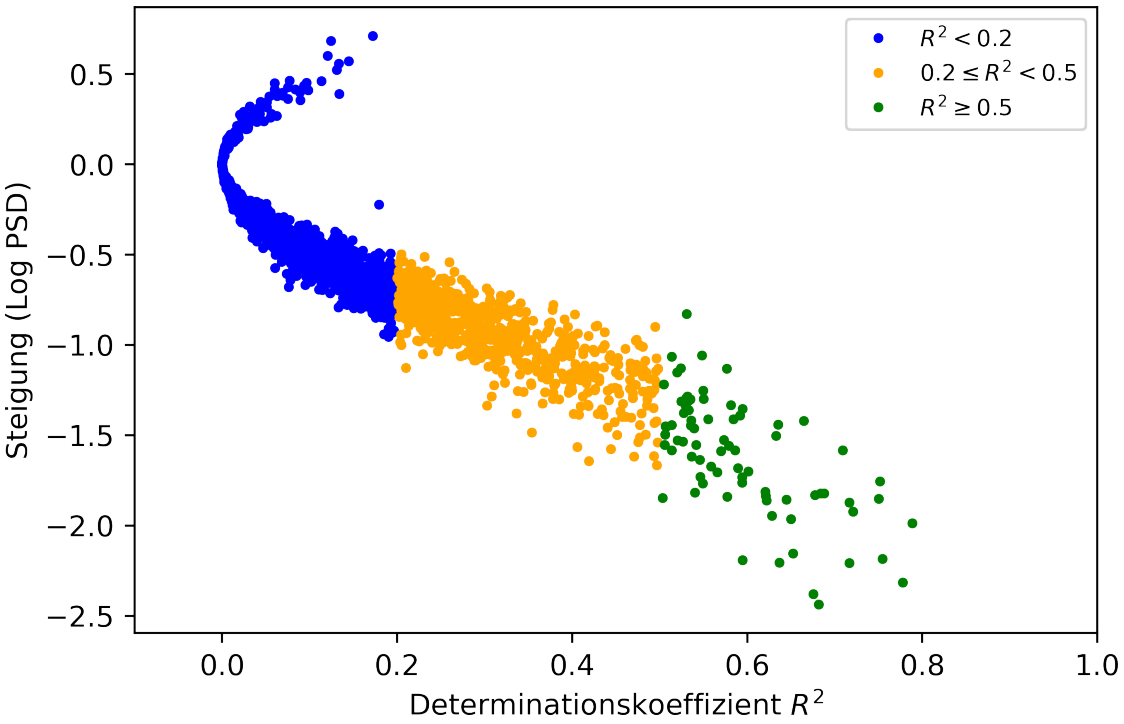

Fig. 4.10: Steigung zu Determinationskoeffizient für die zufällig generierten Quellen: Die Verteilung ist nahe zu identisch zu der in Figur 4.8 (reale Quellen), jedoch sind die Steigungen für jedes R^2 geringer gestreut.

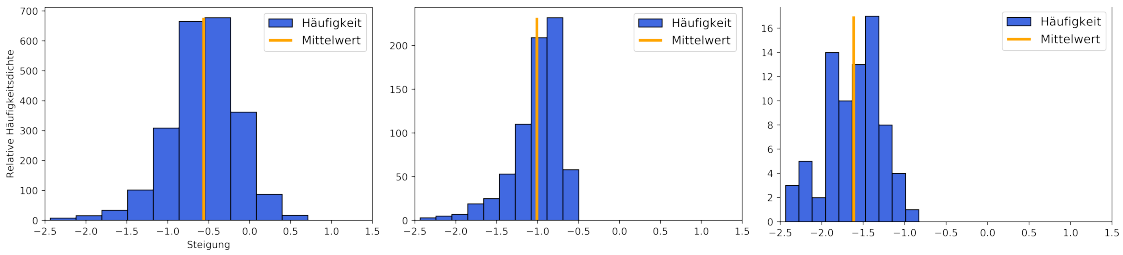

Fig. 4.11: Histogramm der generierten Lichtkurven: Analog zu Figur 4.7 sind links alle Steigungen enthalten, in der Mitte wurde nach $R^2 \geq 0.2$ und rechts nach $R^2 \geq 0.5$ gefiltert.

zu erwarten sind. In der Tat zeigen alle drei Diagramme (Figur 4.11) fast identische Strukturen und Mittelwerte, wobei die schmalere Verteilung auf die zuvor beobachtete, geringere Streuung im PSD zurückzuführen ist. Der Mittelwert beträgt in der linken Darstellung $mean_{gen} = -0.562 \pm 0.192$ ($mean_{real} = -0.536 \pm 0.335$ bei realen Lichtkurven), in der mittleren $mean_{gen,0.2} = -1.011 \pm 0.184$ ($mean_{real,0.2} = -0.982 \pm 0.293$) und rechts $mean_{gen,0.5} = -1.616 \pm 0.176$ (im Vergleich: $mean_{real,0.5} = -1.405 \pm 0.247$).

Da die generierten Lichtkurven sowohl vor, als auch nach der Filterung äußerst ähnliche Mittelwerte und Verteilungen aufweisen, kann man zu dem Schluss kommen, dass die Software mit den zufällig generierte Lichtkurven tatsächlich im Bezug auf die Eigenschaften im PSD eine gute Nachstellung realer Lichtkurven ermöglicht. Um den Zusammenhang zwischen den Parametern und der resultierenden Steigung im Leistungsdichtespektrum besser verstehen zu können, sollen diese im Folgenden analysiert und ihre Auswirkung getestet werden.

5. PARAMETEROPTIMIERUNG

Mit der Verwendung der zufälligen Generierung erhält man ein breites Spektrum an Lichtkurven, die unterschiedlichste Formen annehmen und daraus resultierend verschiedene Leistungsdichtespektrum aufweisen. Um den Einfluss der anfangs gewählten Parameter auf die PSDs erkennen zu können, sollen diese folgend untersucht und, wenn möglich, optimiert werden. Hierfür wurden zunächst alle Variablen ausfindig gemacht, die eine direkte Wirkung auf die Generierung der Datenwerte ausüben. Es handelt sich hierbei sowohl um vom Benutzer regulierbare, als auch programminterne Parameter. Wie zuvor wurde hier jeweils ein Datensatz, bestehend aus 2278 zufällig generierten Lichtkurven, zum Vergleich verwendet.

5.1 Verwendete Datenpunkte pro Lichtkurve

Die Aufzeichnungen der Energieflüsse, die mir im FITS-Format zur Verfügung stehen, reichen vom 20. August 2008 bis 04. Oktober 2017. Da jeder Datenpunkt einen Monat Beobachtung repräsentiert, umfasst jede Quelle 119 Datenpunkte. Die Anpassung der Anzahl verwendeter Datenpunkte hätte zur Folge, dass die Frequenzen des Signals nach der Fourier-Transformation anders verteilt wären, was in der Tat eine Änderung im PSD bewirken würde. Dies wäre aber nicht zweckmäßig, da sich durch eine analoge Änderung die Leistungsdichtespektren der originalen Lichtkurven ebenfalls verändern würden (siehe Figur 5.1). Demnach wird die Anzahl bei 119 Datenpunkten belassen.

5.2 Zufallswerte pro Normalverteilung

Jeder Datenpunkt einer generierten Lichtkurve entstammt einer Normalverteilung. Sie alle entsprechen jeweils dem Maximalwert der Verteilung, die je nach Status (Ausbruch oder Grundrauschen) aus unterschiedlichen Parametern (μ, σ) berechnet wird. Derzeit werden pro Datenpunkt 250 Werte verwendet, um für diesen eine Gaußverteilung zu erzeugen. Eine Anpassung dieser Anzahl in positive Richtung hätte zur Folge, dass die Wahrscheinlichkeit steigt, einen größeren Extremwert zu erhalten. Da unterschiedliche σ verwendet werden, würde dies die Variabilität in den Amplituden erhöhen, was nach (Chatterjee et al., 2012) im PSD in eine

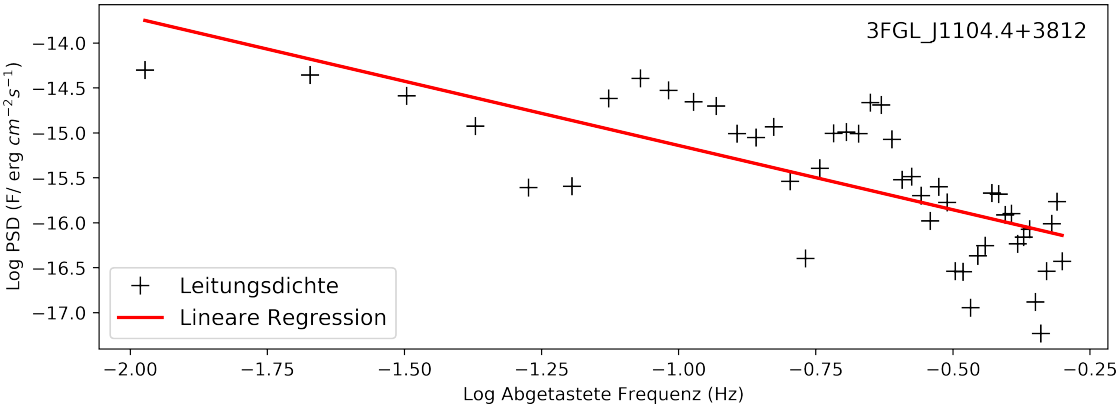

Fig. 5.1: Leistungsdichtespektrum des Blazars Markarjan 421. Im Gegensatz zu Figur 4.5 werden hier nur die 94 signifikantesten Datenpunkte einbezogen. Die Steigung beträgt hier $m = -1.429 \pm 0.205$.

stärkere negative Steigung zeigen würde. Im Gegensatz dazu würde ein Herabsetzen des Wertes ein gegenteiliges Verhalten bewirken. Des Weiteren hat die Anzahl einen großen Einfluss auf die benötigte Rechenzeit zur Erzeugung der Lichtkurven. Die Reduktion des Wertes von 250 auf 50 zeigte Folgendes: Wie in Figur 5.2 zu sehen ist, konzentrieren sich die Slopes bei gleicher Anzahl an Lichtkurven eher im Bereich geringerer Steigung. Die Histogramme (Figur 5.3) haben bei gleicher Filterung ($R^2 \geq 0.2$ und $R^2 \geq 0.5$) entsprechend geringere Mittelwerte und eine schmalere Verteilung.
Wird hingegen die Anzahl auf 500 gesetzt, bestätigt sich obige Aussage: Figur 5.4 verdeutlicht, dass die Variabilität der Amplituden gestiegen ist und somit im Durchschnitt größere negative Steigungen auftreten. Dies zeigt sich auch noch einmal in den Histogrammen in Figur 5.5. Nicht zu vernachlässigen ist auch die um das 4-fach angestiegene Rechenzeit für die Generierung im Vergleich zu 250 verwendeten Zufallswerten.
Durch diese Betrachtung wird deutlich, dass sich bereits allein durch die Anpassung der Anzahl an verwendeten Zufallswerte die Steigungen in den Leistungsdichtespektren ändern und so sich auch die Verteilungen in den entstehenden Histogrammen verschieben. Da es, was das Maß der Steigungen betrifft, kein richtig und kein falsch gibt und die mir vorliegenden realen Lichtkurven lediglich als Orientierung dienen, wird der Wert bei 250 belassen. Dieser kann im Programmcode dennoch ohne Weiteres bei Bedarf angepasst werden.

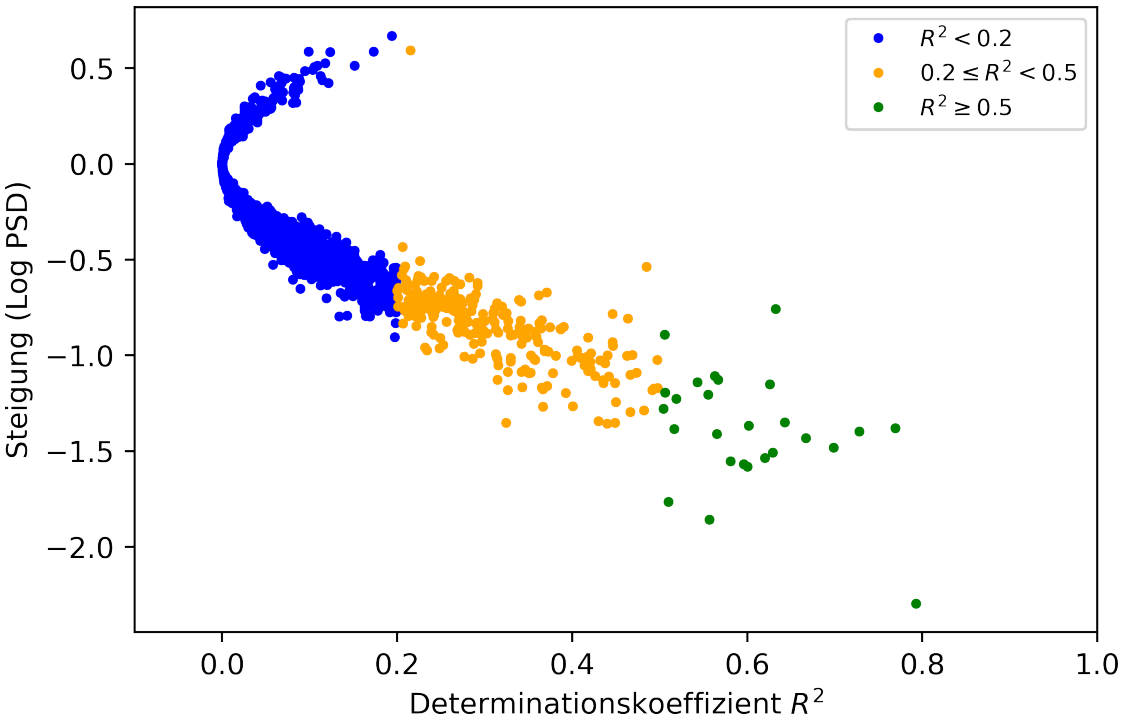

Fig. 5.2: Verwendung von 50 Zufallswerten pro Normalverteilung (im Unterschied zu Figur 4.10). Die Steigungen der Slopes sind hier eher gering und demnach im blauen Bereich konzentriert.

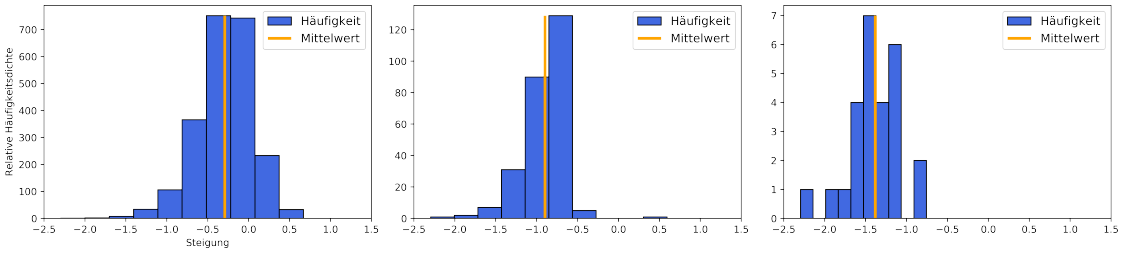

Fig. 5.3: Histogramm der generierten Lichtkurven mit 50 Zufallswerten pro Normalverteilung. Eine Filterung analog zu Figur 4.11 ergibt hier Mittelwerte von $mean = -0.293 \pm 0.183$ (links), $mean_{0.2} = -0.898 \pm 0.168$ (Mitte) und $mean_{0.5} = -1.382 \pm 0.149$ (rechts).

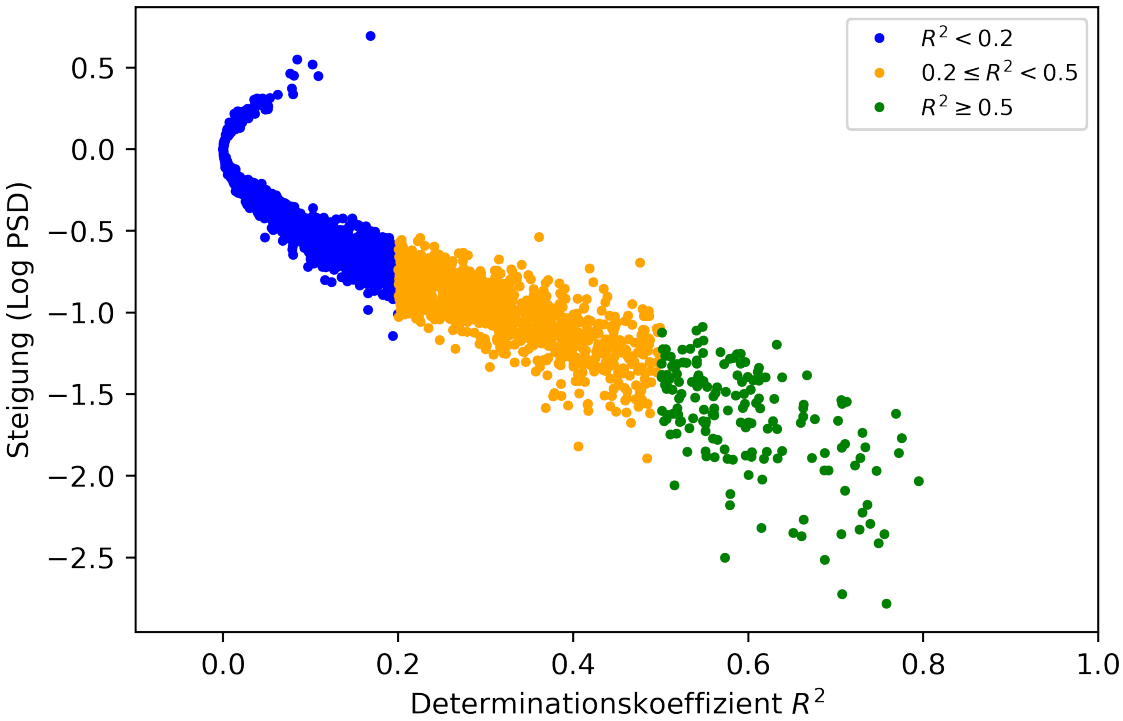

Fig. 5.4: Verwendung von 500 Zufallswerten pro Normalverteilung. Im Vergleich zu Figur 4.10 sind die Steigungen der Slopes hier eher steiler orientiert und demnach im orangen/grünen Bereich.

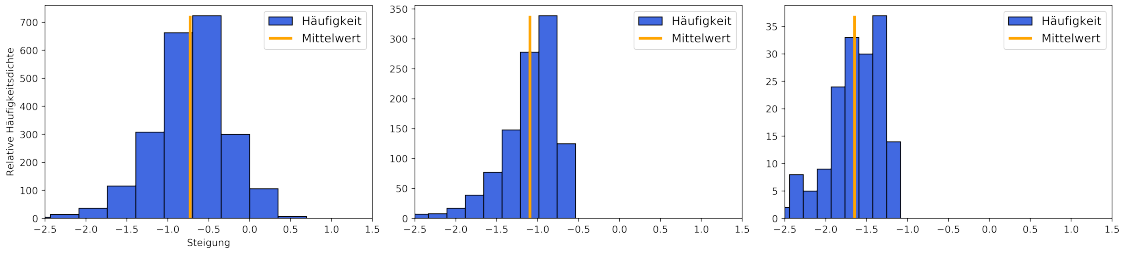

Fig. 5.5: Histogramm der generierten Lichtkurven mit 500 Zufallswerten pro Normalverteilung. Nach der Filterung ergeben sich die Mittelwerte zu $mean = -0.726 \pm 0.196$ (links), $mean_{0.2} = -1.095 \pm 0.189$ (Mitte) und $mean_{0.5} = -1.648 \pm 0.178$ (rechts).

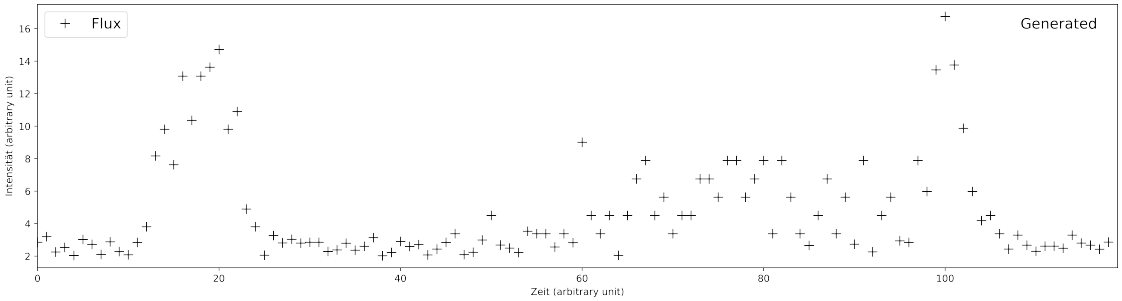

Fig. 5.6: Die aus Figur 4.4 generierte Lichtkurve, hier mit $baseline = 2$.

5.3 Wert des Grundrauschens

Das Grundrauschen ist in jeder Quelle vorhanden und stellt den quasi ihren ruhenden Zustand dar. Auf diese Baseline bauen die Energieflüsse auf. Da diese in den generierten Lichtkurven an sich erst einmal dimensionslos sind und die Ausbrüche im Verhältnis zum Grundrauschen errechnet werden, bewirkt eine Anpassung des Wertes lediglich ein Anheben bzw. Absenken des Wertebereichs. Die Verteilung bleibt davon unbeeinflusst, sodass keine veränderten Ergebnisse zu erwarten sind. Figur 5.6 wurde mit $baseline = 2$ erzeugt. Eine Lichtkurve mit identischen Parametern (gleicher Seed-Key!) aber $baseline = 1.0 \cdot 10^{-10}$ stimmt komplett mit Figur 5.6 überein, zeigt jedoch Werte um den Faktor $2 \cdot 10^{10}$ erhöht.

5.4 Anzahl der Ausbrüche

Unterscheidet man bei den generierten Lichtkurven nach der Anzahl der vorhandenen Peaks, lassen sich leichte Unterschiede erkennen: Lichtkurven, die den Parametern nach keine Ausbrüche vorweisen, fluktuieren mit ähnlichen Amplituden überhalb des Baseline-Wertes und weisen demnach im PSD eine Steigung $m \sim 0$ auf („white noise“). Es zeichnet sich jedoch ab, dass Lichtkurven mit einem bis drei Ausbrüchen mit steigender Korrelation eine größere negative Steigung aufweisen als Quellen, die vier bis zehn Fluxes enthalten (vgl. Figur 5.7 und Figur 5.8). Ein in diesem Zuge neu generierter Datensatz, bestehend aus 2278 Quellen, die ausschließlich nur einen bis drei Ausbrüche umfassen, zeigt ähnlich zu Figur 5.4 eine leicht erhöhte Konzentration im steileren, grünen Bereich. In histografischer Betrachtung entspricht das einem Unterschied im Mittelwert zu einem Datensatz, der bis zu zehn Fluxes enthält, sowohl mit Filterung nach $R^2 \geq 0.2$ und $R^2 \geq 0.5$, als auch ohne, von bis zu $\Delta m = -0.1$.

Durch die Reduzierung der Anzahl an Ausbrüchen in der zufälligen Generierung auf Maximal drei kann man somit durchaus eine mittlere Steigung der Lichtkurven im PSD von $mean_{1-3} = -1.725 \pm 0.180$ erreichen.

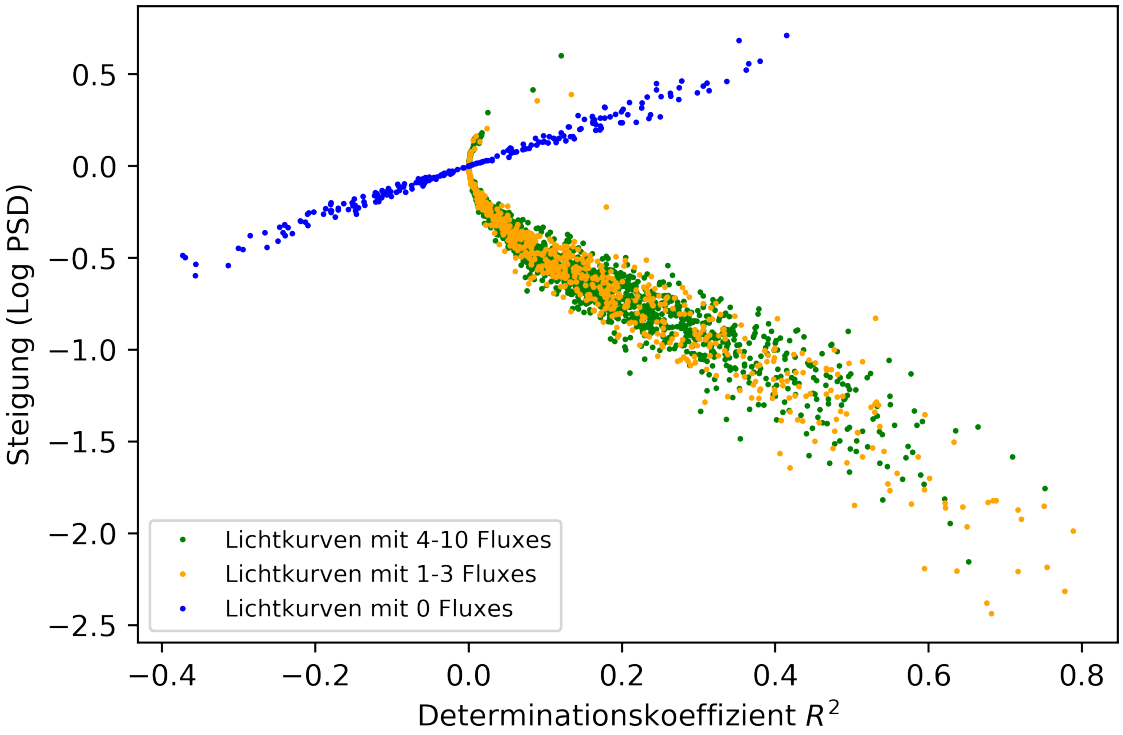

Fig. 5.7: Die blauen Punkte markieren alle Lichtkurven mit 0 Ausbrüchen, in Orange die mit 1-3 und in Grün die mit 4-10. Es ist zu beachten, dass die Grafik 1453 grüne und 614 orangene Datenpunkte zeigt.

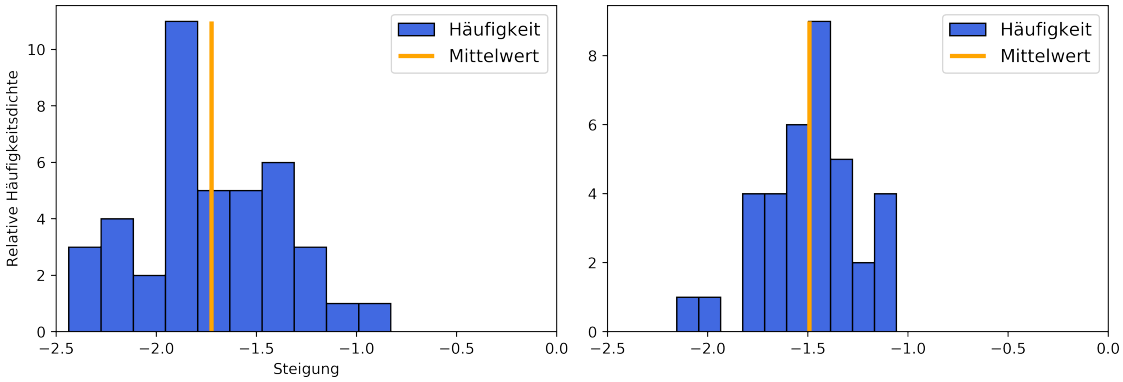

Fig. 5.8: In diesen Histogrammen ist klar zu erkennen, dass nach einer Filterung von $R^2 \geq 0.5$ die Steigungen bei 1-3 Fluxes (links, $mean_{1-3} = -1.725 \pm 0.180$) deutlich steiler sind als bei 4-10 Fluxes (rechts, $mean_{4-10} = -1.492 \pm 0.171$).

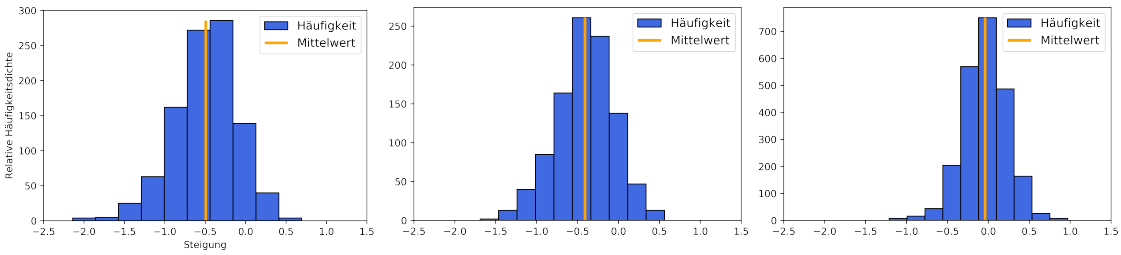

Fig. 5.9: Links: $\mu = 5$, $\sigma = 2, mean_{5,2} = -0.494 \pm 0.189$, Mitte: $\mu = 3$, $\sigma = 1, mean_{3,1} = -0.409 \pm 0.187$, Rechts: $\mu = 1$, $\sigma = 0.5, mean_{1,0.5} = -0.040 \pm 0.183$

5.5 Faktor des Ausbruchs zum Grundrauschen

Die Faktoren, die gesetzt werden, um den Ausschlag eines Ausbruchs zu definieren, werden nicht wie andere Parameter aus einer gleichmäßigen Verteilung gewonnen, sondern aus einer Normalverteilung gezogen. Ein Flux erhält diesen zufällig gaußverteilten Wert x ($\mu = 10$, $\sigma = 5$), wenn $x > 1$ erfüllt ist. Dies wurde so umgesetzt, da sich in der Regel Ausbrüche erkennbar vom Grundrauschen abheben und eine gleichmäßig verteilte Zufallszahl dies nur zum Teil erfüllt. Schwache Peaks werden bereits durch die Fluktuationen der Baseline abgedeckt. Aber was für Auswirkungen bringt eine Modifikation mit sich?

Das Herabsetzen des Erwartungswerts μ hat zur Folge, dass die Amplituden der Ausbrüche geringer werden und somit ihre Variabilität sinkt. Nach (Chatterjee et al., 2012) ist dies begleitet von abflachenden, also gegen 0 gehende Steigungen im Leistungsdichtespektrum („white noise“). Dieses Verhalten kann auch in Figur 5.9 beobachtet werden.

Im Gegenzug kann ebenfalls mit steigendem μ eine, wenn auch nur leicht, stärker werdende negative Steigung festgestellt werden.

Die Frage ist nun, ob man hierbei von einer Optimierung sprechen kann, da bei einem Herabsetzen des Erwartungswertes die Amplituden der Ausbrüche geringer werden und so die Lichtkurven aus optischer Betrachtung nicht mehr die Diversität widerspiegeln, wie es die realen tun. Eine analoge Aussage trifft auch auf eine Erhöhung zu. Die Konfiguration wird somit bis auf Weiteres im Ursprungszustand belassen.

5.6 Halbwertsbreiten

Die FWHMs sind sehr breit gefächert. Sie müssen mindestens eine Breite von einem Datenpunkt umfassen, damit dieser Flux in der Lichtkurve dargestellt werden kann, und kann maximal 100 Datenpunkte einnehmen, da es durchaus vorkommen kann, dass sich aufgrund eines speziellen Ereignisses der Ausbruch über längere Zeit

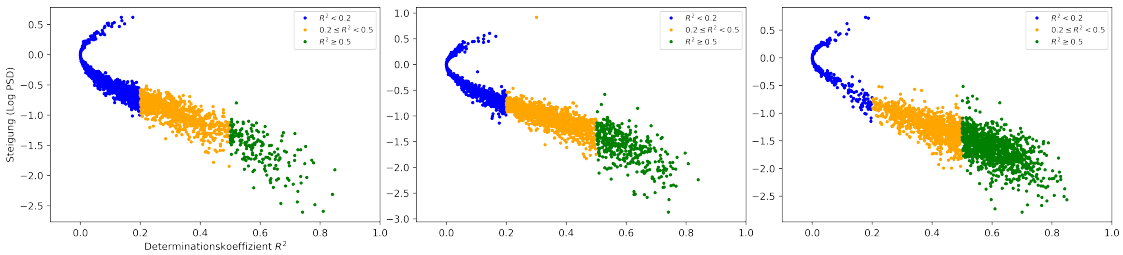

Fig. 5.10: Alle drei Grafiken zeigen die Verteilung der Steigungen im PSD von 2278 zufällig generierten Lichtkurven. Links wurde die maximale Breite auf 75 Datenpunkte begrenzt, in der Mitte auf 50 und rechts auf 25.

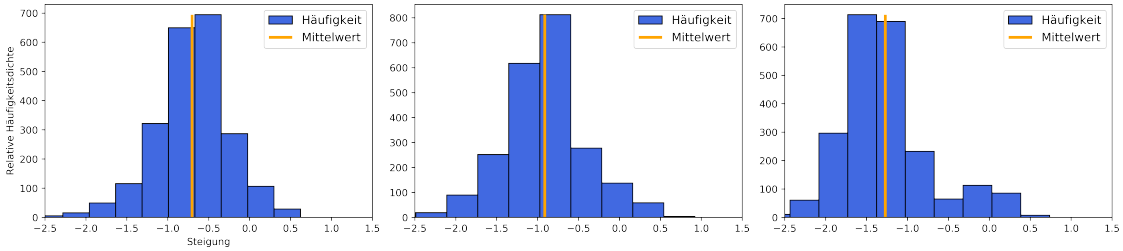

Fig. 5.11: Bei FWHM $\leq$ 75 erzielt man $mean_{gen,75} = -0.703 \pm 0.193$ (links), bei FWHM $\leq$ 50 beträgt er $mean_{gen,50} = -0.913 \pm 0.192$ und FWHM $\leq$ 25 zeigt $mean_{gen,25} = -1.272 \pm 0.189$.

hält. Da die Zufallsgenerierung auf einer gleichmäßigen Verteilung beruht, können solch große Halbwertsbreiten des öfteren auftreten.

Um das Verhalten bei den Anpassungen nachvollziehen zu können, wurde eine schrittweise Reduzierung durchgeführt. Dabei wurden Breiten aus den Wertebereichen $\text{FWHM}_{75} = randint(1, 76)$, $\text{FWHM}_{50} = randint(1, 51)$ und $\text{FWHM}_{25} = randint(1, 26)$ verwendet (Der erste Parameter ist inklusive in der Zufallsgenerierung, der zweite exklusive). In Figur 5.10 erkennt man deutlich, dass mit der geringer werdenden maximalen Breite die Lichtkurven steilere negative Steigungen erhalten.

Durch Anpassung dieses Parameters scheint also ein grundlegende Verbesserung der mittleren Steigung im PSD möglich zu sein (Figur 5.11). Filtert man nach $R^2 \geq 0.2$, erreicht man mittlere Slopes von $mean_{75,0.2} = -1.069 \pm 0.182$ bis $mean_{25,0.2} = -1.438 \pm 0.188$ und kommt bei $R^2 \geq 0.5$ von $mean_{75,0.5} = -1.588 \pm 0.170$ bis $mean_{25,0.5} = -1.649 \pm 0.174$. Ähnlich dem vorigen Parameter (Faktor) stellt sich auch hier die Frage, ob eine Begrenzung der maximalen Breite auf 25 Datenpunkte zweckmäßig ist, um optisch reale Lichtkurven nachzuahmen.

6. FAZIT

Die Untersuchungen zeigen, dass das im Rahmen dieser Arbeit entwickelte Programm durchaus in der Lage ist, im Hinblick auf die charakteristischen Eigenschaften im Leistungsdichtespektrum, realitätsnahe Lichtkurven zu erzeugen. PSDs beschreiben die Leistung, die einzelne Frequenzen in einem Signals besitzen, wobei ihre ausgleichenden Geraden nach (Abdo, Ackermann, Ajello et al., 2010) für Blazare charakteristisch sind und im Durchschnitt für die 9 hellsten FSRQs -1.4 ± 0.1 und die 6 hellsten BL Lacs -1.7 ± 0.3 betragen. Die manuelle Konfiguration von Flux-Positionen, -Faktoren und -Breiten erlaubt es, Energieflüsse zu generieren, die in Lichtkurven unterschiedlichste Variationen in Form und Struktur annehmen können. Auch führen die unterschiedlichen Parametrisierungen hierbei zu abwechslungsreichen Verläufen und Steigungen in den PSDs, was für eine effektvolle Diversität an generierbaren Datenreihen spricht.

Die in (Abdo, Ackermann, Ajello et al., 2010) beschriebenen Steigungen werden nach einen Quality-Cut auf die ausgleichenden Geraden (Determinationskoeffizient $R^2 \geq 0.5$) für die vorliegenden 2278 realen Quellen bestätigt. Die im Anschluss unter die Lupe genommenen generierten Lichtkurven zeigen starke Parallelen zu den eben erzielten Ergebnissen: Während im PSD der Mittelwert für alle Steigungen mit $R^2 \geq 0.5$ $mean_{real,0.5} = -1.405 \pm 0.247$ beträgt, erreicht man bei gleicher Filterung mit den imitierten Quellen sogar einen Durchschnitt von $mean_{gen,0.5} = -1.616 \pm 0.176$. Die Steigungen der PSDs in eine histografische Form gebracht, zeigen für beide Datensätze sowohl ohne Filterung, als auch mit Quality-Cuts bei $R^2 \geq 0.2$ und $R^2 \geq 0.5$ vergleichbare Verteilungen und Wertebereiche. Dies belegt, dass es mit der Software in der derzeitigen Konfiguration durchaus möglich ist, reale Lichtkurven im Bezug auf ihre Eigenschaften in Leistungsdichtespektrum nachzustellen.

In der Folge werden sowohl programminterne, als auch vom Benutzer konfigurierbare Parameter darauf untersucht, welchen Einfluss diese auf die resultierende Steigung in den PSDs haben. Es wird schnell ersichtlich, dass bei vielen dieser Variablen eine Justierung Veränderungen im Leistungsdichtespektrum mit sich bringt.

So haben Tests an den generierten Lichtkurven ergeben, dass eine Anpassung der derzeit 119 verwendeten Datenpunkte pro Lichtkurve durchaus eine grundsätzliche Änderung im PSD herbeiführen kann. Nach (Chatterjee et al., 2012)

wird die Steigung steiler, wenn bei längeren Zeitskalen größere Variabilitäten in der Amplitude auftreten. Dieser Ansatz wird jedoch nicht weiter verfolgt, da sich eine Erhöhung des Wertes für die realen Quellen nicht überprüft lässt und so keine Vergleiche gezogen werden können.

Auch eine Änderung an der Anzahl der zu verwendenden Zufallszahlen für die Erzeugung einer Normalverteilung bewirkt sichtbare Unterschiede bei Betrachtung der Slopes in Leistungsdichtespektren. Eine Reduzierung sorgt für geringere Extremwerte in der Verteilung, wodurch die Variabilität in den Lichtkurven sinkt und sich die mittlere Slope weiter Richtung 0 orientiert. Im Gegensatz zu den mit 250 Zufallswerten generierten Lichtkurven ($mean_{250} = -1.616 \pm 0.176$) zeigt sich hier $mean_{50,0.5} = -1.382 \pm 0.149$. Eine konträre Anpassung bewirkt ein entsprechend gegensätzliches Verhalten ($mean_{500,0.5} = -1.648 \pm 0.178$).

Weiter besteht die Möglichkeit, den Parameter des Grundrauschens anzupassen. Der Wert hat jedoch ausschließlichen Einfluss auf den Wertebereich der generierten Energieflüsse. Zwei mit identischen Parametern erzeugte Lichtkurven (gleiche Seed-Keys!), aber unterschiedlichen Baselines, besitzen auch identische Strahlungswerte, die aber um den Faktor, der zwischen beiden Baseline-Werten liegt, verschoben sind. Eine Änderung in den PSDs ist hier also nicht vorzufinden.

Eine recht interessanteste Änderung lässt sich hingegen durch die Eingrenzung der Anzahl an zu integrierenden Strahlungsausbrüchen pro Lichtkurve bei der zufälligen Generierung erzielen. Bei separater Betrachtung der Histogramme, aufgeteilt nach der Anzahl vorhandener Fluxes, stellt man fest, dass Lichtkurven, die einen bis maximal drei Ausbrüche besitzen, mit steigender Korrelation steilere negative Steigungen aufweisen ($mean_{1-3} = -1.725 \pm 0.180$) als die, die vier bis zehn umfassen ($mean_{4-10} = -1.492 \pm 0.171$) . Gestützt wird diese Beobachtung auch durch die Erkenntnis, dass sich bei den realen Lichtkurven nur selten mehrere starke Ausbrüche in einer Zeitreihe vorfinden lassen.

Ein weiterer Parameter, der einen Unterschied im Leistungsdichtespektrum bezweckt, ist der Faktor, mit dem die Ausschläge bei der zufälligen Generierung gebildet werden. Ein Richtung 1 gehender Wert lässt die Steigung im PSD abflachen und erzeugt Lichtkurven, die zunehmend weißem Rauschen ähneln. Eine Anpassung in positive Richtung ($\mu > 10$) verursacht folglich auch eine stärker ins negativ gerichtete Steigung. Dennoch sollte sich hierbei die Frage gestellt werden, inwiefern eine Anpassung dieses Wertes das natürliche Erscheinungsbild einer realen Lichtkurve widerspiegelt.

Zu guter Letzt bezweckt auch die Eingrenzung der Halbwertsbreiten bei der zufälligen Generierung einer veränderte mittlere Steigung im PSD. Während der Mittelwert bei $FWHM \leq 100$ $mean_{0.5} = -1.616 \pm 0.176$ beträgt, lässt sich durch eine Herabsetzung von $FWHM \leq 25$ der Mittelwert $mean_{25,0.5} = -1.649 \pm 0.174$ erzielen. Auch an dieser Stelle muss überlegt werden, ob eine maximale Halbwerts-

breite von 25 Datenpunkten der Realität nahe kommt.

Es existieren demnach mehrere Parameter, deren Justierung Änderungen in der mittleren Steigung im Leistungsdichtespektrum hervorrufen. Diesbezüglich stellt die Software für die Generierung künstlicher Lichtkurven ein umfassendes Werkzeug dar, das auch künftig im Falle des Einsatzes einer künstlichen Intelligenz dazu verwendet werden kann, das neuronale Netz mit ausreichend Trainingsdaten zu füttern.

ABKÜRZUNGSVERZEICHNIS

AGN	Active galactic nucleus
BL Lac	BL Lacerta
FGST	Fermi Gamma-ray Space Telescope
FITS	Flexible Image Transport System
FSRQ	flat-spectrum radio quasar
FWHM	Full Width at Half Maximum
GBM	Gamma-ray Burst Monitor
GUI	Graphical user interface
LAT	Large Area Telescope
MJD	Modifiziertes Julianisches Datum
PSD	Power spectral density
SMSL	Supermassives Schwarzes Loch
TS	Teststatistik

Literatur

Abdo, A., Ackermann, M., Agudo, I., Ajello, M., Aller, H., Aller, M., ... others (2010). The spectral energy distribution of fermi bright blazars. *The Astrophysical Journal*, *716* (1), 30.

Abdo, A., Ackermann, M., Ajello, M., Antolini, E., Baldini, L., Ballet, J., ... others (2010). Gamma-ray light curves and variability of bright fermi-detected blazars. *The Astrophysical Journal*, *722* (1), 520.

Albert, J., Aliu, E., Anderhub, H., Antoranz, P., Armada, A., Baixeras, C., ... others (2007). Discovery of very high energy γ-ray emission from the low-frequency-peaked bl lacertae object bl lacertae. *The Astrophysical Journal Letters*, *666* (1), L17.

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., ... Streicher, O. (2013, Oktober). Astropy: A community Python package for astronomy. *Astronomy and Astrophysics*, *558*, A33. (Last accessed on 2019-09-13) doi: 10.1051/0004-6361/201322068

Atwood, W., Abdo, A. A., Ackermann, M., Althouse, W., Anderson, B., Axelsson, M., ... others (2009). The large area telescope on the fermi gamma-ray space telescope mission. *The Astrophysical Journal*, *697* (2), 1071.

Blandford, R. & Payne, D. (1982). Hydromagnetic flows from accretion discs and the production of radio jets. *Monthly Notices of the Royal Astronomical Society*, *199* (4), 883–903.

Burke, B. F., Graham-Smith, F. & Wilkinson, P. N. (2019). *An introduction to radio astronomy.* Cambridge University Press.

Chatterjee, R., Bailyn, C., Bonning, E., Buxton, M., Coppi, P., Fossati, G., ... Urry, C. (2012). Similarity of the optical-infrared and γ-ray time variability of fermi blazars. *The Astrophysical Journal*, *749* (2), 191.

Dagnello, S. (o. J.). *Agns seen at different angles.* Zugriff auf `https://public.nrao.edu/gallery/agns-seen-at-different-angles/` (Last accessed on 2019-08-27)

Giommi, P., Colafrancesco, S., Cutini, S., Marchegiani, P., Perri, M., Pittori, C., ... others (2008). Agile and swift simultaneous observations of the blazar s50716+ 714 during the bright flare of october 2007. *Astronomy & Astrophysics*, *487* (3), L49–L52.

Hunter, J. D. (2007). *Matplotlib: A 2d graphics environment* (Bd. 9) (Nr. 3). IEEE Computer Society. (Last accessed on 2019-09-13)

Jones E, P. P., Oliphant E et al. (2001–). *SciPy: Open source scientific tools for Python.* Zugriff auf `http://www.scipy.org/` (Last accessed on 2019-09-13)

Jovanović, P. & Popović, L. Č. (2009). X-ray emission from accretion disks of agn: signatures of supermassive black holes. *arXiv preprint arXiv:0903.0978*.

Kantel-Chaos-Team. (2010). *Warum python?* Zugriff auf http://www.pythonmania.de/article/warum.html (Last accessed on 2019-08-20)

Kazmierczak, J. (2018). *Nasa's fermi satellite celebrates 10 years of discoveries.* Zugriff auf https://www.nasa.gov/feature/goddard/2018/nasa-s-fermi-satellite-celebrates-10-ye (Last accessed on 2019-08-08)

Krolik, J. H. (1999). *Active galactic nuclei: from the central black hole to the galactic environment.* Princeton University Press.

Lawrence, I. & Lin, K. (1989). A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, 255–268.

Meegan, C., Lichti, G., Bhat, P., Bissaldi, E., Briggs, M. S., Connaughton, V., ... others (2009). The fermi gamma-ray burst monitor. *The Astrophysical Journal*, *702* (1), 791.

Oliphant, T. E. (2006). *A guide to numpy.* Trelgol Publishing. Zugriff auf https://numpy.org/ (Last accessed on 2019-09-13)

Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., ... Contributors, A. (2018, September). The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *Astronomical Journal*, *156*, 123. (Last accessed on 2019-09-13) doi: 10.3847/1538-3881/aabc4f

Python. (o. J.). Zugriff auf http://www.inztitut.de/blog/glossar/python/ (Last accessed on 2019-08-20)

Python Software Foundation. (2018). *Python programming language.* Zugriff auf https://www.python.org/downloads/release/python-367/ (Last accessed on 2019-09-13)

Sambruna, R. M. (1997). Soft x-ray properties of flat-spectrum radio quasars. *The Astrophysical Journal*, *487* (2), 536.

Scargle, J. D., Norris, J. P., Jackson, B. & Chiang, J. (2013). Studies in astronomical time series analysis. vi. bayesian block representations. *The Astrophysical Journal*, *764* (2), 167.

The SciPy community. (2019). *Random sampling.* Zugriff auf https://docs.scipy.org/doc/numpy-1.16.0/reference/generated/numpy.random.RandomSt (Last accessed on 2019-09-14)

Vestergaard, M. & Peterson, B. M. (2006). Determining central black hole masses in distant active galaxies and quasars. ii. improved optical and uv scaling relationships. *The Astrophysical Journal*, *641* (2), 689.

Weisstein, E. W. (2002). Gaussian function.

DANKSAGUNG

Ich möchte mich recht herzlich bei allen bedanken, die mich während der Anfertigung meiner Bachelorarbeit begleitet und mir zur Seite gestanden haben!

Zunächst gilt mein Dank Prof. Dr. Karl Mannheim, der es mir zu meiner großen Freude ermöglicht hat, Astronomie und Informatik in dieser Arbeit zu verbinden. Anhand Ihrer gezielten Fragestellungen im Bezug auf mein Thema und Ihrer Hinweise konnte ich einen interessanten roten Leitfaden durch die Arbeit legen.

Des Weiteren möchte ich mich bei Prof. Dr. Sara Buson für die Bereitstellung des Datensatzes, der für die Arbeit unabdingbar ist, ausdrücklich bedanken. Mit Ihrem Engagement motivierten Sie mich stets zum Fortschritt und hatten als Spezialistin für Fermi-Daten auf jede Frage eine Antwort.

Außerdem möchte ich meine große Dankbarkeit an Paul Ray Burd ausdrücken. Durch unsere engere Zusammenarbeit blieb mir jederzeit das konkrete Ziel vor Augen und ich wusste, woran ich gerade war. Deine Expertise verlieh mir zusätzlichen Auftrieb und dein stets offenes Ohr verhalf mir zu einem erfolgreichen Entwicklungsprozess bei der Arbeit.

Abschließend möchte ich mich herzlich bei meinen Eltern bedanken, die mich während meines Bachelorstudiums durchgehend unterstützt und mir Kraft gegeben haben.

SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel **Simulation von Fermi-LAT Blazar Lichtkurven mit Zufallsprozessen durch Optimierung der Leistungsdichtespektren** selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Würzburg, den 23.09.2019

Niklas Richter

ANHANG

.1 Programmcode

```
# Import necessary libraries
import numpy as np
import numpy.random as nr

from tkinter import *
from tkinter import messagebox
from tkinter import filedialog
from tkinter import ttk

# Definition of the lightcurve generating function
def generateDistributedWaveform(flux_positions: list,
    flux_factors: list, flux_widths: list, seed):
    # preparatiom
    nr.seed(seed)

    flux_positions = [ int(x) for x in flux_positions ]
    flux_factors = [ float(x) for x in flux_factors ]
    flux_widths = [ float(x) for x in flux_widths ]

    flux_positions = np.array(flux_positions)
    flux_factors = np.array(flux_factors)
    flux_widths = np.array(flux_widths)

    # set internal parameters
    n_randomDistribution = 250 # number of random values
        for a single normal distribution
    n_baseline = 0.0000000001 # baseline value
    n_datapointsPerLightcurve = 119 # number of datapoints
        per lightcurve

    # prepare result array
    results = np.empty(shape = [3,], dtype=object)
    results[1] = n_randomDistribution
    results[2] = n_baseline

    # factor for the fwhm (full width at half maximum)
    conversionFactor = 2*np.sqrt(2*np.log(2))
```

```
    X = [i for i in range(n_datapointsPerLightcurve)]
    Y = np.empty(shape = [n_datapointsPerLightcurve], dtype
       =object)
    updatedFluxIndexes = []

    # iterate through every datapoint and calculate the
       flux value depending on the state of the datapoint (
       baseline or peak)
    for index in range(n_datapointsPerLightcurve):
        baseline = n_baseline
        sigma = nr.uniform(0,baseline/5)
        array_index = np.where(flux_positions == index)
        # current index is a declareted as peak
        if (len(array_index[0]) == 1):
            baseline = flux_factors[array_index[0]] *
               baseline # multiply factor with baseline
               value
        values = []
        # get random gaussian distributed values for flux
           calculation
        values = nr.normal(baseline, sigma, size =
           n_randomDistribution)
        maxY = max(values) # use the max value as flux to
           get different (fluctuating) values for baseline
        Y[index] = maxY
    meanY = np.mean(Y)

    # index is again listed as peak but now adding the
       individual widths to the peaks
    for flux_index in range(len(flux_positions)):
        values = []
        sigma = flux_widths[flux_index] / conversionFactor
           # formula to get sigma of fwhm
        values = nr.normal(0, sigma, size =
           n_randomDistribution)
        # calculate number of needed bins and then generate
            histogram
        n_bins = np.ceil(4*sigma)
        dY,dX = np.histogram(values, bins = int(n_bins))
        dY = np.array(dY, dtype=float)
```

```
        # calculate factor to scale the bin values of the histogram to flux values and apply it to all bin-values
        factor = Y[flux_positions[flux_index]]/max(dY)

        for i in range(len(dY)):
            dY[i] = factor * dY[i]
        # calculate starting index (the maximum peak value will be assigned to the index of the flux position)
        maxIndex = np.argmax(dY)
        startingIndex = int(flux_positions[flux_index]-maxIndex)
        internalIndex = 0
        # iterating through all datapoints involved in the width of a peak
        for i in range(startingIndex, startingIndex + len(dY)):
            # preventing out of bounds exception
            if((i >= 0) & (i < len(Y))):
                # just update flux value if higher than mean (otherwise it is not a peak)
                if(dY[internalIndex] >= meanY):
                    # when trying to set an index that has already been updated previously use the higher value
                    if not((i in updatedFluxIndexes) & (Y[i] >= dY[internalIndex])):
                        Y[i] = dY[internalIndex]
                        updatedFluxIndexes.append(i)
            internalIndex = internalIndex + 1

    results[0] = Y
    return results

# Selection of "specific data" or "randomized data"
def dataSourceRadioSelection():
    selection = dataSourceRadioVal.get()
    if selection == 1:
        showSpecificInputFields()
```

```
    elif selection == 2:
        showRandomInputFields()
    else:
        messagebox.showerror(title = "Error", message = "An error has occurred, please restart the generator.")
        mainwindow.quit()

# Gray out "Seeds" if "Number of lightcurves" is selected and the other way
optionRadioUseSeed = None
def optionRadioSelection():
    global optionRadioUseSeed
    selection = optionRadioVal.get()
    if selection == 1:
        optionRadioUseSeed = True
        seedsEntry.configure(state='normal')
        nLightcurvesEntry.configure(state='disabled')
    elif selection == 2:
        optionRadioUseSeed = False
        seedsEntry.configure(state='disabled')
        nLightcurvesEntry.configure(state='normal')

    else:
        messagebox.showerror(title = "Error", message = "An error has occurred, please restart the generator.")
        mainwindow.quit()

# Display buttons and inut fields for "specific data"
def showSpecificInputFields():
    infoLabel = Label(mainwindow, anchor = "n", wraplengt
        =800, text="Here you can either produce a single
        lightcurve with a given seed (select 'seed') or you
        can generate multiple lightcurves with the same
        parameters but random seeds (select 'number of
        lightcurves').\nFor more detailed information take a
        look at the Howto.")
    infoLabel.place(x = 50, y = 100, width=800, height=60)
    infoLabel.config(font=("Arial", 12))
```

```
    fluxPositionsLabel.place(x = 100, y = 200, width=100, height=20)
    fluxPositionsEntry.place(x = 220, y = 200, width=220, height=30)
    fluxFactorsLabel.place(x = 100, y = 240, width=100, height=20)
    fluxFactorsEntry.place(x = 220, y = 240, width=220, height=30)
    fluxWidthsLabel.place(x = 100, y = 280, width=100, height=20)
    fluxWidthsEntry.place(x = 220, y = 280, width=220, height=30)
    seedsLabel.place(x = 100, y = 320, width=100, height=20)
    seedsEntry.place(x = 220, y = 320, width=220, height=30)
    nLightcurvesEntry.place(x = 650, y = 320, width=150, height=30)
    global nLightcurvesLabel
    nLightcurvesLabel.place_forget()
    nLightcurvesLabel = Radiobutton(mainwindow, text="Number of lightcurves", variable = optionRadioVal, value=2, command = optionRadioSelection)
    nLightcurvesLabel.config(font=("Arial", 12))
    nLightcurvesLabel.place(x = 460, y = 320, width=170, height=20)

    global optionRadioUseSeed
    optionRadio = optionRadioVal.get()
    if optionRadio == 1:
        seedsEntry.configure(state='normal')
        nLightcurvesEntry.configure(state='disabled')
    elif optionRadio == 2:
        seedsEntry.configure(state='disabled')
        nLightcurvesEntry.configure(state='normal')
    else:
        seedsEntry.configure(state='disabled')
        nLightcurvesEntry.configure(state='disabled')

# Display buttons and inut fields for "randomized data"
```

```
def showRandomInputFields():
    infoLabel = Label(mainwindow, anchor = "n", wraplengt
        =800, text="Here you can generate multiple
        lightcurves with different (randomized) parameters
        and seeds.\nFor more detailed information take a
        look at the Howto.")
    infoLabel.place(x = 50, y = 100, width=800, height=60)
    infoLabel.config(font=("Arial", 12))
    global fluxPositionsLabel
    global fluxPositionsEntry
    global fluxFactorsLabel
    global fluxFactorsEntry
    global fluxWidthsLabel
    global fluxWidthsEntry
    global seedsLabel
    global seedsEntry
    global nLightcurvesLabel
    fluxPositionsLabel.place_forget()
    fluxPositionsEntry.place_forget()
    fluxFactorsLabel.place_forget()
    fluxFactorsEntry.place_forget()
    fluxWidthsLabel.place_forget()
    fluxWidthsEntry.place_forget()
    seedsLabel.place_forget()
    seedsEntry.place_forget()
    nLightcurvesLabel.place_forget()
    nLightcurvesLabel = Label(mainwindow, text="Number of
        lightcurves", anchor = "e")
    nLightcurvesLabel.config(font=("Arial", 12))
    nLightcurvesLabel.place(x = 460, y = 320, width=170,
        height=20)
    nLightcurvesEntry.place(x = 650, y = 320, width=150,
        height=30)
    nLightcurvesEntry.configure(state='normal')

# Button "Generate" pressed. Check for valid inputs and
    start generation process
def generateLC():
    # get user input
    fluxPositionsInput = fluxPositionsEntry.get()
```

```
    fluxFactorsInput = fluxFactorsEntry.get()
    fluxWidthsInput = fluxWidthsEntry.get()
    seedInput = seedsEntry.get()
    nLightcurvesInput = nLightcurvesEntry.get()
    # create array from comma separated user input
    fluxPositionsInputArray = fluxPositionsInput.split(",")
    fluxFactorsInputArray = fluxFactorsInput.split(",")
    fluxWidthsInputArray = fluxWidthsInput.split(",")
    # remove spaces and check if input contain digits
    fluxPositionsInputArray = [x.strip(' ') for x in
       fluxPositionsInputArray]
    fluxPositionsInputArray = [ int(x) for x in
       fluxPositionsInputArray if x.isdigit() ]
    fluxFactorsInputArray = [x.strip(' ') for x in
       fluxFactorsInputArray]
    fluxFactorsInputArray = [ float(x) for x in
       fluxFactorsInputArray if float(x) ]
    fluxWidthsInputArray = [x.strip(' ') for x in
       fluxWidthsInputArray]
    fluxWidthsInputArray = [ float(x) for x in
       fluxWidthsInputArray if float(x) ]

    selection = dataSourceRadioVal.get()

    # check if seed is valid user input
    if not ((seedInput != "") & (seedInput.isdigit())):
        seedInput = nr.randint(4294967296, dtype= 'u8')
    seedInput = int(seedInput)
    if((seedInput < 0) | (seedInput >= 4294967296)):
        seedInput = nr.randint(4294967296, dtype= 'u8')
    seedInput = int(seedInput)

    # check if number of lightcurves is valid user input
    if ((selection == 2) | ((selection == 1) & (
       optionRadioUseSeed == False))):
        if ((nLightcurvesInput != "") & (nLightcurvesInput.
           isdigit())):
            nLightcurvesInput = int(nLightcurvesInput)

            if(nLightcurvesInput <= 0):
```

```
                messagebox.showerror(title = "Error",
                    message = "Invalid 'number of
                    lightcurves' input. Please enter valid
                    value and try again!")
        else:
            messagebox.showerror(title = "Error", message =
                "Invalid 'number of lightcurves' input.
                Please enter valid value and try again!")

    # ckeck if all flux-parameters have the same length
    if(len(fluxPositionsInputArray) != len(
        fluxFactorsInputArray)) | (len(
        fluxPositionsInputArray) != len(fluxWidthsInputArray
        )):
        messagebox.showerror(title = "Error", message = "
            The number of positions, factors and widths you
            provided are unequal. Please enter valid values
            and try again!")

    # check for valid flux positions input
    if not(all((i >= 0) & (i <= 118) for i in
        fluxPositionsInputArray)):
        messagebox.showerror(title = "Error", message = "
            Values in flux positions musst be 0<=index<=
            118! Please enter valid values and try again!")

    # check for valid flux factors input
    if not(all(i > 1 for i in fluxFactorsInputArray)):
        messagebox.showerror(title = "Error", message = "
            Values <=1 detected in flux factors! Please
            enter valid values and try again!")

    # check for valid flux widths input
    if not(all(i >= 1 for i in fluxWidthsInputArray)):
        messagebox.showerror(title = "Error", message = "
            Values <1 detected in flux widths! Please enter
            valid values and try again!")
```

```
    mainwindow.filename = filedialog.asksaveasfilename(
        initialdir = "/", title = "Select file", filetypes =
         (("Numpy Array","*.npy"),("all files","*.*")))
    if mainwindow.filename == "":
        return

    if selection == 1:
        if optionRadioUseSeed == True:
            generateSingleSourcesWithSeed(
                fluxPositionsInputArray,
                fluxFactorsInputArray, fluxWidthsInputArray,
                 seedInput)
            showSuccessSpecSeed(fluxPositionsInputArray,
                fluxFactorsInputArray, fluxWidthsInputArray,
                 seedInput, mainwindow.filename)
        elif optionRadioUseSeed == False:
            generateSameSourcesWithRandSeed(
                fluxPositionsInputArray,
                fluxFactorsInputArray, fluxWidthsInputArray,
                 nLightcurvesInput)
            showSuccessSpecSources(fluxPositionsInputArray,
                 fluxFactorsInputArray, fluxWidthsInputArray
                , nLightcurvesInput, mainwindow.filename)
            removeProgressbar()
        else:
            messagebox.showerror(title = "Error", message =
                 "Invalid selection between use seed or
                number of lightcurves, please select and try
                again.")
    elif selection == 2:
        generateSourcesWithRandSeed(nLightcurvesInput)
        showSuccessRandom(nLightcurvesInput, mainwindow.
            filename)
        removeProgressbar()
    else:
        messagebox.showerror(title = "Error", message = "An
            error has occurred, please restart the
            generator.")
        mainwindow.quit()
```

```
# Initialize generation of a single lightcurve with given flux-parameters and given seed
def generateSingleSourcesWithSeed(fluxPositions, fluxFactors, fluxWidths, seed):
    final = np.zeros([1, 5], dtype=object)

    # rewrite final array
    final[0,0] = seed
    final[0, 1] = fluxPositions
    final[0, 2] = fluxFactors
    final[0, 3] = fluxWidths

    # generate the lightcurves
    final[0, 4] = generateDistributedWaveform(final[0, 1], final[0, 2], final[0, 3], final[0, 0])

    np.save(mainwindow.filename, final)

# Initialize generation of a given number of lightcurves with randomized flux-parameters seeds
def generateSourcesWithRandSeed(nSources):
    # initialize progressbar and counter
    progressbar.place(x = 120, y = 380, width=500, height=25)
    progressbar["maximum"] = nSources
    progressbar["value"] = 0
    counterLabel.place(x = 650, y = 380, width=200, height=30)
    counterLabel.config(font=("Arial", 12))

    # disable any user interaction possibility
    nLightcurvesEntry.configure(state='disabled')
    nLightcurvesLabel.configure(state='disabled')
    fluxPositionsEntry.configure(state='disabled')
    fluxFactorsEntry.configure(state='disabled')
    fluxWidthsEntry.configure(state='disabled')
    seedsEntry.configure(state='disabled')
    seedsLabel.configure(state='disabled')
    radio1.configure(state='disabled')
    radio2.configure(state='disabled')
```

```
    generateButton.configure(state='disabled')
    exitButton.configure(state='disabled')

    # init result array
    final = np.zeros([nSources, 5], dtype=object)

    seeds = nr.randint(4294967296, size = nSources, dtype =
        'u8') # one seed per lightcurve
    n_fluxes = nr.randint(0, 11, size = nSources) # number
       of fluxes in single lightcurve
    sum_fluxes = np.sum(n_fluxes) # number of fluxes in all
        lighcurves together
    flux_positions = nr.randint(0, 119, size = sum_fluxes)
       # indexes of every flux position
    flux_widths = nr.randint(1, 101, size = sum_fluxes) #
       fwhm (full width at half maximum) of all fluxes
    flux_factors = np.empty(shape=(sum_fluxes,))

    index = 0
    while (index < sum_fluxes):
        factor = nr.normal(10, 5) # factor of flux outburst
        if(factor > 1):
            flux_factors[index] = factor
            index = index + 1

    # rewrite final array
    final[0:nSources+1,0] = seeds
    index_start = 0
    for source in range(len(n_fluxes)):
        n_fluxes_source = n_fluxes[source]
        index_end = index_start + n_fluxes_source
        final[source, 1] = flux_positions[index_start:
           index_end]
        final[source, 2] = flux_factors[index_start:
           index_end]
        final[source, 3] = flux_widths[index_start:
           index_end]
        index_start = index_end

    # generate the lightcurves
```

```
    for source in range(nSources):
        final[source, 4] = generateDistributedWaveform(
            final[source, 1], final[source, 2], final[source
            , 3], final[source, 0])
        progressbar["value"] = source + 1
        counterLabel.config(text = str(source + 1) + " / "
            + str(nSources))
        mainwindow.update()
    np.save(mainwindow.filename, final)

# Initialize generation of a given number of lightcurves
    flux-parameters and randomized seeds
def generateSameSourcesWithRandSeed(fluxPositions,
    fluxFactors, fluxWidths, nSources):
    # Init progressbar and counter
    progressbar.place(x = 120, y = 380, width=500, height
        =25)
    progressbar["maximum"] = nSources
    progressbar["value"] = 0
    counterLabel.place(x = 650, y = 380, width=200, height
        =30)
    counterLabel.config(font=("Arial", 12))

    # disable any user interaction possibility
    nLightcurvesEntry.configure(state='disabled')
    nLightcurvesLabel.configure(state='disabled')
    fluxPositionsEntry.configure(state='disabled')
    fluxFactorsEntry.configure(state='disabled')
    fluxWidthsEntry.configure(state='disabled')
    seedsEntry.configure(state='disabled')
    seedsLabel.configure(state='disabled')
    radio1.configure(state='disabled')
    radio2.configure(state='disabled')
    generateButton.configure(state='disabled')
    exitButton.configure(state='disabled')

    final = np.zeros([nSources, 5], dtype=object)

    seeds = nr.randint(4294967296, size = nSources, dtype =
        'u8') # one seed per lightcurve
```

```
355
356     # rewrite final array
357     final[0:nSources+1,0] = seeds
358     for source in range(nSources):
359         final[source, 1] = fluxPositions
360         final[source, 2] = fluxFactors
361         final[source, 3] = fluxWidths
362
363     # generate the lightcurves
364     for source in range(nSources):
365         final[source, 4] = generateDistributedWaveform(
              final[source, 1], final[source, 2], final[source
              , 3], final[source, 0])
366         progressbar["value"] = source + 1
367         counterLabel.config(text = str(source + 1) + "␣/␣"
              + str(nSources))
368         mainwindow.update()
369
370     np.save(mainwindow.filename, final)
371
372 # Messagebox contents of "About"
373 def showAbout():
374     m_text = "\
375 **********************\n\
376 Author:␣Niklas␣Richter\n\
377 Date:␣23.09.2019\n\
378 Version:␣1.0\n\
379 **********************"
380     messagebox.showinfo(title = "About", message = m_text)
381
382 # Messagebox contents in case of successful generation of a
        lightcurve with given flux-parameters and seed
383 def showSuccessSpecSeed(positions, factors, widths, seed,
      filelocation):
384     m_text = "\
385 You␣successfully␣generated␣a␣lightcurve␣with␣the␣following␣
      parameters:\n\
386 **********************\n\
387 Flux␣positions:␣" + str(positions) + "\n\
388 Flux␣factors:␣" + str(factors) + "\n\
```

```
Flux widths: " + str(widths) + "\n\                                    389
Seed: " + str(seed) + "\n\                                             390
File location: " + str(filelocation) + "\n\                            391
**********"                                                           392
    messagebox.showinfo(title = "Success", message = m_text            393
        )
                                                                       394
# Messagebox contents in case of successful generation of              395
    multiple lightcurves with given flux-parameters
def showSuccessSpecSources(positions, factors, widths,                 396
    nSources, filelocation):
    m_text = "\                                                        397
You successfully generated multiple similar lightcurves                398
    with just random seed:\n\
**********\n\                                                          399
Number of lightcurves: " + str(nSources) + "\n\                        400
Flux positions: " + str(positions) + "\n\                              401
Flux factors: " + str(factors) + "\n\                                  402
Flux widths: " + str(widths) + "\n\                                    403
File location: " + str(filelocation) + "\n\                            404
**********"                                                           405
    messagebox.showinfo(title = "Success", message = m_text            406
        )
                                                                       407
# Messagebox contents in case of successful generation of              408
    multiple lightcurves with randomized flux-parameters
def showSuccessRandom(nSources, filelocation):                         409
    m_text = "\                                                        410
You successfully generated multiple randomized lightcurves             411
    :\n\
**********\n\                                                          412
Number of lightcurves: " + str(nSources) + "\n\                        413
File location: " + str(filelocation) + "\n\                            414
**********"                                                           415
    messagebox.showinfo(title = "Success", message = m_text            416
        )
                                                                       417
# Messagebox contents of "Evaluation"                                  418
def showEval():                                                        419
        m_text = "\                                                    420
```

```
**********\n\
Evaluation of the result array\n\
**********\n\
The returned result array is a NumPy-Array. A dataset
    containing e.g. 2300 lightcurves will have a shape of
    (2300,5). Each lightcurve contains 5 sub-arrays.\n\n\
(x, 0) contains the used seed for data generation.\n\
(x, 1) contains the array with the used flux positions.\n\
(x, 2) contains the array with the used flux factors.\n\
(x, 3) contains the array with the used flux widths.\n\
(x, 4, 0) contains the array with the calculated flux
    values.\n\
(x, 4, 1) contains the number of random values used for a
    single normal distribution.\n\
(x, 4, 2) contains the used baseline value."
        messagebox.showinfo(title = "Howto", message=m_text
            )

# Messagebox contents of "HowTo"
def showHowTo():
        m_text = "\
**********\n\
Specific data\n\
**********\n\
Flux Positions: Enter the positions of the peaks. The
    positions can range from 0 to 118 (both including).\n\
Flux Factors: The factors multiplied by the baseline value
    results in the amplitude of the peaks. The input should
    be > 1. The number of given factors must equal the
    number of positions.\n\
Flux Widths: The corresponding FWHMs to the fluxes. The
    input should be > 1. The number of given widths must
    equal the number of positions.\n\
Selecting 'Seed' will generate one single lightcurve with
    the given flux-parameters and the given seed (integer
    between 0 and 2**32 - 1 inclusive). If you leave this
    field empty a random seed will be assigned.\n\
Selecting 'Number of lightcurves' will generate the given
    amount of lightcurves with the given flux-parameters. A
    random seed will be assigned to every single lightcurve
```

```
    .\n\n\
Multiple inputs should always be comma separated!\n\n\n\
**********\n\
Randomized data\n\
**********\n\
The given number of lightcurves will be generated with
    random flux-parameters for every single lightcurve.\n\
Every lightcuve has none or a maximum of 10 peaks. The
    number is randomized as well as the seed, flux positions
    and the flux widths.\n\
The flux factor is a value from a normal distribution with
    mean = 10 and sigma = 5, which is always > 1."
          messagebox.showinfo(title = "Howto", message=m_text
              )

# Remove progressbar after a successful generation
def removeProgressbar():
    progressbar.place_forget()
    counterLabel.place_forget()
    nLightcurvesEntry.configure(state='normal')
    nLightcurvesLabel.configure(state='normal')
    fluxPositionsEntry.configure(state='normal')
    fluxFactorsEntry.configure(state='normal')
    fluxWidthsEntry.configure(state='normal')
    seedsEntry.configure(state='normal')
    seedsLabel.configure(state='normal')
    radio1.configure(state='normal')
    radio2.configure(state='normal')
    generateButton.configure(state='normal')
    exitButton.configure(state='normal')

# Mainwindow configuration
mainwindow = Tk()
mainwindow.resizable(False, False)
mainwindow.geometry("900x500")
mainwindow.title("Lightcurve Generator")

# Setup menu bar
menuBar = Menu(mainwindow)
menu_file = Menu(menuBar, tearoff=0)
```

```
menu_file.add_command(label="Howto", command=showHowTo)
menu_file.add_command(label="Evaluation", command=showEval)
menu_file.add_command(label="About", command=showAbout)
menu_file.add_command(label="Exit", command=mainwindow.quit
    )
menuBar.add_cascade(label="File", menu=menu_file)
mainwindow.config(menu=menuBar)

# Setup radiobuttons
dataSourceRadioVal = IntVar()
radioLabel = Label(mainwindow, text="Choose whether you
    want to generate lightcurves with your own or randomized
    data:")
radioLabel.place(x = 150, y = 20, width=600, height=20)
radioLabel.config(font=("Arial", 12))
radio1 = Radiobutton(mainwindow, text="Specific data",
    variable = dataSourceRadioVal, value=1, command =
    dataSourceRadioSelection)
radio1.place(x = 250, y = 45, width=120, height=20)
radio1.config(font=("Arial", 12))
radio2 = Radiobutton(mainwindow, text="Randomized data",
    variable = dataSourceRadioVal, value=2, command =
    dataSourceRadioSelection)
radio2.place(x = 500, y = 45, width=150, height=20)
radio2.config(font=("Arial", 12))

# Setup menu buttons
generateButton = Button(mainwindow, text="Generate",
    command=generateLC)
generateButton.place(x = 730, y = 430, width=120, height
    =40)
generateButton.config(font=("Arial", 10))
exitButton = Button(mainwindow, text="Exit", command=
    mainwindow.quit)
exitButton.place(x = 50, y = 430, width=120, height=40)
exitButton.config(font=("Arial", 10))

# Setup user input fields
infoLabel = Label(mainwindow, anchor = "n", wraplengt=800,
    text="")
```

```
fluxPositionsLabel = Label(mainwindow, text="Flux Positions ", anchor = "w")
fluxPositionsLabel.config(font=("Arial", 12))
fluxPositionsEntry = Entry(mainwindow, bd=5)
fluxFactorsLabel = Label(mainwindow, text="Flux Factors", anchor = "w")
fluxFactorsLabel.config(font=("Arial", 12))
fluxFactorsEntry = Entry(mainwindow, bd=5)
fluxWidthsLabel = Label(mainwindow, text="Flux Widths", anchor = "w")
fluxWidthsLabel.config(font=("Arial", 12))
fluxWidthsEntry = Entry(mainwindow, bd=5)
optionRadioVal = IntVar()
seedsLabel = Radiobutton(mainwindow, text="Seed", variable = optionRadioVal, value=1, command = optionRadioSelection)
seedsLabel.config(font=("Arial", 12))
seedsEntry = Entry(mainwindow, bd=5)
nLightcurvesLabel = Radiobutton(mainwindow, text="Number of lightcurves", variable = optionRadioVal, value=2, command = optionRadioSelection)
nLightcurvesLabel.config(font=("Arial", 12))
nLightcurvesEntry = Entry(mainwindow, bd=5)
progressbar = ttk.Progressbar(mainwindow, orient ="horizontal", mode ="determinate")
counterLabel = Label(mainwindow, anchor = "w")

# Run UI
mainwindow.mainloop()
```

.2 Zur Auswertung verwendeter Code

```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as nr
import scipy.signal as ss
from scipy.stats import linregress as lr
from astropy.io import fits

# load files
df = fits.open('../FITS/
   MonthlyLC1GeV_cleaned_fixSpec_sourceListAll2283.fits')
array = np.load("random2000.npy", allow_pickle=True)

# print single lighcurve
def printSingleLightcurve():
    source = array[0]
    fluxes=source[4][0]
    print("Seed: " + str(source[0]))
    print("Positions: " + str(source[1]))
    print("Factors: " + str(source[2]))
    print("Widths: " + str(source[3]))
    Y = np.array(fluxes)
    X = [ i for i in range(len(fluxes))]

    # use fits lightcurve instead
    index = 961
    Y = np.array(df[1].data[index])[51]
    X = (df[1].data[index][3]-df[1].data[index][2])/2+df
       [1].data[index][2]

    plt.figure(figsize=(20,5))
    plt.xlabel('Zeit (MJD)')
    plt.ylabel('Energieflussdichte (erg $cm^{-2}s^{-1}$)')
    #plt.errorbar(X, Y, yerr = df[1].data[index][44],
       ecolor = 'red', elinewidth = 2, linewidth = 0,
       marker = '+', markersize = 10, c = 'black', label =
       'Flux inkl. Fehlerbalken')
    plt.errorbar(X, Y, linewidth = 0, marker = '+',
```

```
        markersize = 10, c = 'black', label = 'Flux inkl.
        Fehlerbalken')
    #plt.legend(loc=1, prop={'size': 15})
    #plt.text(0.02, 0.95, df[1].data[index][56],
        horizontalalignment='left', verticalalignment='top',
         transform=plt.gca().transAxes, fontsize=15)
    plt.legend(loc=2, prop={'size': 15})
    plt.text(0.85, 0.95, df[1].data[index][56],
        horizontalalignment='left', verticalalignment='top',
         transform=plt.gca().transAxes, fontsize=15)
    #plt.savefig('./' + str(df[1].data[index][56]) + '.png
        ', dpi = 500)

# Show Power Spectral Density for fits data
def showPSDforFITS(index: int, ts: int):
    y = []
    for i in range(len(df[1].data[index][51])):
        if df[1].data[index][23][i] > ts:
            y.append(df[1].data[index][51][i])
    print('n datapoints: ' + str(len(y)))

    x_welch, y_welch = ss.welch(y)

    if(len(x_welch) > 0):
        x_welch = np.delete(x_welch, 0)
        y_welch = np.delete(y_welch, 0)

        x_welch = np.log10(x_welch)
        y_welch = np.log10(y_welch)

        slope, intercept, rValue, pValue, stderr = lr(
            x_welch, y_welch)

    plt.figure(figsize=(10, 3.5))
    plt.xlabel('Log Abgetastete Frequenz (Hz)')
    plt.ylabel('Log PSD (F/ erg $cm^{-2}s^{-1}$)')
    plt.plot(x_welch, y_welch, linewidth = 0, marker = '+',
        markersize = 10, c = 'black', label = '
        Leitungsdichte')
    plt.plot(x_welch, slope*x_welch+intercept, linewidth =
```

```
        2, c = 'red', label = 'Lineare Regression')

    plt.legend(loc=3, prop={'size': 12})
    plt.text(0.76, 0.95, df[1].data[index][56],
        horizontalalignment='left', verticalalignment='top',
         transform=plt.gca().transAxes, fontsize=12)
#     plt.legend(loc=2, prop={'size': 12})
#     plt.text(0.76, 0.12, df[1].data[index][56],
    horizontalalignment='left', verticalalignment='top',
    transform=plt.gca().transAxes, fontsize=12)

    print('Slope: ' + str(slope) + ' +- ' + str(stderr))
#     plt.savefig('./psd_' + str(df[1].data[index][56]) + '
    _ts-' + str(ts) + '.png', dpi = 500)

# Show Power Spectrum Density for generated lightcurves
def showPSDforGen(array, index: int):
    y = array[index][4][0].tolist()
    print('n_datapoints: ' + str(len(y)) + ', Fluxes: ' +
        str(len(array[index][1])))

    x_welch, y_welch = ss.welch(y)

    x_welch = np.delete(x_welch, 0)
    y_welch = np.delete(y_welch, 0)

    x_welch = np.log10(x_welch)
    y_welch = np.log10(y_welch)

    slope, intercept, rValue, pValue, stderr = lr(x_welch,
        y_welch)

    plt.figure(figsize=(10, 3.5))
    plt.xlabel('Log Abgetastete Frequenz (Hz)')
    plt.ylabel('Log PSD (arbitrary unit)')
    plt.plot(x_welch, y_welch, linewidth = 0, marker = '+',
         markersize = 10, c = 'black', label = '
        Leistungsdichte')
    plt.plot(x_welch, slope*x_welch+intercept, linewidth =
        2, c = 'red', label = 'Lineare Regression')
```

```
    plt.legend(loc=3, prop={'size': 12})
    plt.text(0.86, 0.95, 'Generated', horizontalalignment='
        left', verticalalignment='top', transform=plt.gca().
        transAxes, fontsize=12)
#     plt.legend(loc=2, prop={'size': 12})
#     plt.text(0.76, 0.12, df[1].data[index][56],
    horizontalalignment='left', verticalalignment='top',
    transform=plt.gca().transAxes, fontsize=12)

    print('Slope: ' + str(slope) + ' +- ' + str(stderr))
#     plt.savefig('./psd_generatedLC.png', dpi = 500)

# Show correlation - slopes graphic
def correlationSlopes(correlations, slopes):
    s = []
    sS = []
    m = []
    mS = []
    l = []
    lS = []

    correlations = np.power(correlations,2)

    for i in range(len(slopes)):
        if (correlations[i] < 0.2):
            s.append(correlations[i])
            sS.append(slopes[i])
        if ((correlations[i] >= 0.2) and (correlations[i] <
            0.5)):
            m.append(correlations[i])
            mS.append(slopes[i])
        if (correlations[i] >= 0.5):
            l.append(correlations[i])
            lS.append(slopes[i])

    plt.plot(s, sS, marker = '.', linewidth = 0, color = '
        blue', markersize = 5, label = '$R^{2} < 0.2$')
    plt.plot(m, mS, marker = '.', linewidth = 0, color = '
        orange', markersize = 5, label = '$0.2 \leq R^{2} <
```

```
        0.5$')
    plt.plot(l, lS, marker = '.', linewidth = 0, color = '
        green', markersize = 5, label = '$R^{2}\geq 0.5$')
    plt.xlabel('Determinationskoeffizient $R^2$')
    plt.ylabel('Steigung (Log PSD)')
    plt.xlim([-0.1, 1])
    plt.legend(loc=1, prop={'size': 8})

#     plt.savefig('./slope_correlation2_gen_00.png', dpi =
    500)
#     plt.plot(np.power(correlationsReal,2), slopesReal,
    marker = '.', linewidth = 0)

# Show histogram for fits lightcurves
def showAvgPSDforFITS(ts: int):
    fluxSources = []
    for source in range(len(df[1].data)):
        fluxes = []
        for flux in range(len(df[1].data[source][51])):
            if df[1].data[source][23][flux] > ts:
                fluxes.append(df[1].data[source][51][flux])
        fluxSources.append(fluxes)

    slopes = []
    correlations = []
    errors = []
    for flux in range(len(fluxSources)):
        x_welch, y_welch = ss.welch(fluxSources[flux])

        if(len(x_welch) > 10):
            x_welch = np.delete(x_welch, 0)
            y_welch = np.delete(y_welch, 0)

            slope, intercept, rValue, pValue, stderr = lr(
                np.log10(x_welch), np.log10(y_welch))

            if(rValue*rValue > 0.5):
                slopes.append(slope)
                correlations.append(rValue)
                errors.append(stderr)
```

```
161
    print('Mean: ' + str(np.mean(slopes)) + ' +- ' + str(np 162
        .mean(errors)) + ', n_slopes: ' + str(len(slopes)))
163
    n, bins, patches = plt.hist(slopes, edgecolor = 'black' 164
        , color = 'royalblue', label = 'Haeufigkeit')
    plt.xlabel('Steigung') 165
    plt.ylabel('Relative Haeufigkeitsdichte') 166
    plt.vlines(np.mean(slopes), 0, max(n), colors='orange', 167
        linewidth = 3, label = 'Mittelwert')
    plt.xlim([-2.5, 1.5]) 168
    plt.legend(loc=1, prop={'size': 12}) 169
#     plt.savefig('./histogram_slopes_real_00.png', dpi = 170
    500)
    return slopes, correlations, errors 171
172
# Show histogram for generated lightcurves 173
def showAvgPSDforGen(array): 174
    slopes = [] 175
    correlations = [] 176
    errors = [] 177
    for flux in range(len(array)): 178
        x_welch, y_welch = ss.welch(array[flux][4][0]. 179
            tolist())
180
        if(len(x_welch) > 10): 181
            x_welch = np.delete(x_welch, 0) 182
            y_welch = np.delete(y_welch, 0) 183
184
            slope, intercept, rValue, pValue, stderr = lr( 185
                np.log10(x_welch.tolist()), np.log10(y_welch
                .tolist()))
186
            if(rValue*rValue > 0.5): 187
                slopes.append(slope) 188
                correlations.append(rValue) 189
                errors.append(stderr) 190
191
    print('Mean: ' + str(np.mean(slopes)) + ' +- ' + str(np 192
        .mean(errors)) + ', n_slopes: ' + str(len(slopes)))
```

```
193
194     n, bins, patches = plt.hist(slopes, edgecolor = 'black'
            , color = 'royalblue', label = 'Haeufigkeit')
195     plt.xlabel('Steigung')
196     plt.ylabel('Relative Haeufigkeitsdichte')
197     plt.vlines(np.mean(slopes), 0, max(n), colors='orange',
            linewidth = 3, label = 'Mittelwert')
198     plt.xlim([-2.5, 1.5])
199     plt.legend(loc=1, prop={'size': 12})
200 #     plt.savefig('./histogram_slope_gen_05.png', dpi = 500)
201     return slopes, correlations, errors
```