МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательноеучреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П.Королёва»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Моделирование и исследование распространения пучков Эйри и Пирси

по направлению подготовки
01.03.02 Прикладная математика и информатика (уровень бакалавриата)

Студент	И.А. Родин
Руководитель ВКР, к.ф.-м.н., доцент	С.А. Дегтярев
Нормоконтролёр	С.В. Суханов

Самара 2019

ЗАДАНИЕ

фывфывМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательноеучреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П.Королёва» (Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

УТВЕРЖДАЮ
Заведующий кафедрой

_____А.В. Куприянов
’’ ___ ’’ _____ 20 ___ г.

РЕФЕРАТ

Выпускная квалификационная работа бакалавра: 65 страниц, 32 рисунков, 14 источника, 1 приложение.

Презентация: n слайдов.

ПУЧКИ ЭЙРИ, ПУЧКИ ПИРСИ, БЕЗДИФРАКЦИОННЫЕ ПУЧКИ, ПРЕОБРАЗОВАНИЕ ФУРЬЕ, ПРЕОБРАЗОВАНИЕ ФРЕНЕЛЯ, УСКОРЯЮЩИЕСЯ ПУЧКИ, ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Объектом исследования данной научно-исследовательской работы являются бездифракционные пучки Эйри и Пирси, представленные в данном исследовании в виде комплексной функции действительных переменных.

Целью данной научно-исследовательской работы является построение и рассмотрение пучков, их свойств, а также результата применения к данным пучкам различных операторов распространения.

Разработана программа, осуществляющая построение рассмотренных пучков, вычисление необходимых характеристик, численное преобразование Френеля, двумерное и одномерное преобразования Фурье на произвольных участках рассмотрения. Был произведен анализ влияния параметров моделирования на выходной результат.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках данной научно-исследовательской работы рассматриваются и моделируются несколько разновидностей бездифракционных пучков, а именно пучки Эйри и Пирси. Эти пучки обладают рядом уникальных свойств, которые являются причиной большого интереса со стороны современного научного сообщества. Среди важных свойств пучков Эйри можно озвучить следующие [1, 2]:

- ускорение;
- самофокусировка;
- бездифракционность;
- самовосстановление.

Остановимся на каждом из свойств подробнее.

Данные пучки являются ускоряющимися, то есть распространяются по некоторой криволинейной траектории, зависящей от определенных параметров. Благодаря данному свойству могут быть достигнуты области пространства, скрытые от прямого наблюдения.

Второе свойство пучков вызывает, пожалуй, наибольший интерес, так как открывает возможность использования данных пучков в мире микромеханики и, в частности, в сфере различных микроманипуляций. Суть свойства заключается в том, что если расположить в пространстве несколько источников излучения, то благодаря свойству ускорения в определенной точке такая оптическая установка создаст фокус – некоторую область пространства, при попадании частиц в которую эти частицы будут там удерживаться и перемещаться вслед за этой областью. Экспериментальная реализация данного подхода уже была продемонстрирована на примере перемещения частиц через препятствие.

Третье свойство представляет существенный интерес для современной оптики, потому что пучки Эйри не изменяют своего вида при распростране-

нии. Однако здесь стоит заметить, что данное утверждение справедливо для математической модели, бесконечной абстракции, когда входное распределение берется определенным на протяжении всей вещественной оси. На практике, к сожалению, такого результата добиться невозможно, поскольку для возможности моделирования распространения различных пучков их начальное распределение урезается до какого-то конечного отрезка рассмотрения. В этом случае свойства неподверженности дифракции все же сохраняется, однако только при распространении на конечные расстояния. В теории данное свойство может быть использовано для передачи информации, поскольку если гарантируется, что внешний вид сигнала не изменился, то после его приема можно точно установить параметры сигнала в точке отправления и таким образом передать информацию. Однако для реализации данной технологии еще предстоит решить перечень определенных проблем.

Последнее свойство пучков заключается в их способности восстаналивать свой общий вид после определенного времени расространения при встрече некоторого препятствия на своем пути.

Вид этих пучков берет свое начало в теории катастроф [3]. Свое название пучки Эйри получили от интеграла Эйри, который был получен Джорджем Эйри для объяснения оптических каустик, таких как те, что проявляются в виде радуги [4]. Воспроизвести и пронаблюдать пучки Эйри впервые удалось ученым из Флориды в 2007 году [1, 4].

Пучки Пирси также получили свое название от одноименного интеграла, который часто применяется для решения проблем распространения волн и дифракции [5] Несмотря на то, что некоторыми из перечисленных выше свойств пучки Пирси не обладают, например, свойством ускорения, существует возможность с помощью нехитрых манипуляций привести их к виду, сильно напоминающему по внешнему виду и свойствам пучки Эйри. Аналогичные рассуждения справедливы и для пучков Эйри.

В рамках данной работы реализована программа, осуществляющая построение пространственных пучков Эйри и Пирси в качестве исследовательской составляющей, а так же программный модуль, позволяющий промоделировать формирование данных пучков при прохождении через линзу с различными физическими характеристиками и распространение на свободное расстояние как с использованием, так и без использования оптических элементов. Среди параметров, рассматриваемых в данной работе, будут изучены амплитуда, фаза и вещественная и мнимая части выходного поля.

1 Обзор теоретических основ для ускоряющихся пучков

Уравнение, описывающее пучки Эйри, является аналитическим решением следующего дифференциального уравнения:

$$\frac{\partial^2 \omega}{\partial z^2} = z\omega,$$

и выглядит следующим образом:

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} exp[\frac{i\alpha t^3}{3} + ixt]dt, \tag{1}$$

что является известным результатом [6, 7]. Принимая во внимание формулу одномерного обратного преобразования Фурье, имеющую следующий вид [8]:

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} f(x)exp[ixt]dt, \tag{2}$$

нетрудно заметить, что равенство (2) также является преобразованием Фурье от функции:

$$F(t) = exp[\frac{i\alpha t^3}{3}], \tag{3}$$

которая в дальнейшем будет рассматриваться в качестве входного поля в виде пространственного спектра рамках исследования.

Уравнение, описывающее пучки Пирси, задается интегралом Пирси:

$$Pe(x, y) = \int_{-\infty}^{\infty} exp[i(t^4 + xt^2 + yt)]dt,$$

что тоже в свою очередь является обратным преобразованием Фурье от функции следующего вида:

$$F(x, t) = exp[i(t^4 + xt^2)],$$

которая будет рассматриваться как входное поле при моделировании пучков Пирси [9].

Для численного интегрирования было принято решение использовать стандартную формулу прямоугольников, так как она представляется наиболее простой в реализации и имеет наибольшую точность среди всех способов численного интегрирования нулевого порядка. В классическом варианте формула выглядит следующим образом [10]:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} f(\frac{x_{i+1}+x_i}{2})(x_{i+1}-x_i), \tag{4}$$

однако в рамках данной работы исходная функция является функцией двух аргументов, поэтому результатом такого численного интегрирования станет значение выходной функции в конкретной точке, то есть:

$$f(u_j) \approx \sum_{i=0}^{n-1} f(\frac{x_{i+1}+x_i}{2})(x_{i+1}-x_i)exp[i\frac{k}{f}(\frac{x_{i+1}+x_i}{2})u_j].$$

Однако не всегда научный интерес может ограничиться информацией, получаемой из одномерного преобразования Фурье. Во многих случаях значительно нагляднее продемонстрировать результат моделирования в виде поверхности. Для достижения данного результата используется двумерное преобразование Фурье, которое имеет следующий вид [8]:

$$G(u,v) = \frac{k}{f}\int_{-\infty}^{+\infty}\int f(x,y)exp[\frac{-ik}{f}(xu+yv)]dxdy, \tag{5}$$

где x, y – пространственные переменные входного поля;

u, v – пространственные переменные выходного поля;

k – волновое число;

f – фокусное расстояние.

В данной работе двойное интегрирование в формуле тоже было заменено численным аналогом.

Кроме того, определенный интерес в рамках данного исследования может представлять реализация преобразования Френеля, физический смысл которого заключается в распространении пучка на некоторое расстояние z. Общий вид одномерного преобразования Френеля [11]:

$$F(u) = -\sqrt{\frac{ik}{2\pi z}} exp[ikz] \int\limits_{-\infty}^{+\infty} f(x) exp[\frac{ik}{2z}(x-u)^2], \tag{6}$$

где z– расстояние распространения.

Кроме того, множество случаев, для которых допустимо использовать данную формулу, ограничивается условием $x - u \ll z$.

Стоит отметить, что преобразование Фурье моделирует распространение в ситуации, когда начальная и конечная плоскости рассмотрения являются фокальными плоскостями линзы. Данное ограничение не всегда удовлетворяет интерес исследований, поэтому в рамках этой научной работы было принято решение совместить преобразование Френеля и Фурье, чтобы иметь возможность рассмотреть выходное распределение на некотором произвольном расстоянии от выходной фокальной плоскости линзы. Достичь подобного результата можно за счет включения в оператор распространения на свободное расстояние оптического элемента. В математическом смысле это значит, что преобразование Френеля умножается на экспоненту определенного вида. В данном случае воспользуемся следующей реализацией:

$$exp[-ik\frac{x^2}{2f}].$$

Тогда раскроем квадрат в показателе экспоненты, вынесем за знак интеграла слагаемое, содержащее u^2, так как оно не зависит от переменной интегрирования, и внесем добавленный оптический элемент под знак интеграла. После группировки некоторых слагаемых получим:

$$F(u) = -\sqrt{\frac{ik}{2\pi z}} exp[ikz] exp[\frac{iku^2}{2z}] \int\limits_{-\infty}^{+\infty} f(x) exp[\frac{ikx^2}{2}(\frac{1}{z} - \frac{1}{f})] exp[ik\frac{ux}{z}] dx,$$

где f – фокусное расстояние линзы;

z – расстояние от начальной плоскости до линзы.

2 Описание программной реализации пучков и операторов распространения

Для написания программы использовался язык программирования Python, так как он позволяет легко реализовать весь необходимый функционал, предлагает большой выбор готовых средств для использования, сравнения и проверки результатов. Работа велась в среде разработки JetBrains PyCharm, поскольку она удобна и предоставляет множество возможностей, существенно упрощающих и ускоряющих написание исходного кода.

Одним из вариантов решения поставленной задачи является использование готовой библиотеки для быстрого преобразования Фурье. Подобный функционал предоставляют различные библиотеки, в рамках данной работы были использованы библиотеки Numpy и Scipy [12]. Однако область их применения в рамках дискретного преобразования Фурье ограничена теми случаями, когда массивы аргументов для входной и выходной функции одинаковы. Это ограничение противоречит целям, преследуемым в данной работе, поэтому после получения первых результатов было принято решение отказаться от использования данного программного обеспечения и реализовать собственный модуль численного интегрирования методом прямоугольников, реализующий преобразование Фурье на произвольных массивах аргументов.

Спецификой рассматриваемых пучков является факт того, что функции, описывающие их – комплексные экспоненты, поэтому при работе и детальном рассмотрении полученных результатов не обойтись без работы с функциями, которые позволяют получить некоторые характеристики из массива комплексных чисел. В данной работе подобными функциями являются функция взятия модуля комплексного числа, функция взятия фазы комплексного числа, а также функции взятия действительной и мнимой части. Кроме того, требовалась программная реализация экспоненциальной функции, которая

корректно работала с комплекснозначными аргументами. Такой функционал предоставляет библиотека cmath [13] и Numpy. В ходе написания программы было выяснено, что библиотека cmath работает в разы медленнее, чем аналогичный функционал из библиотеки Numpy, поэтому выбор был сделан в пользу последней.

В качестве численного метода интегрирования был выбран метод центральных прямоугольников, потому что он наиболее прост в реализации и предоставляет наибольшую точность среди всех методов нулевого порядка. Кроме того, особенности рассматриваемых пучков, а именно наличие быстро осциллирующих составляющих, позволяет использовать метод прямоугольников практически без потери точности относительно, например, метода Симпсона. Отказ от использования готовых реализаций численного интегрирования связан с трудностями с параметризацией и общим сложным видом интегрируемых функций, рассматриваемых в работе.

Для визуализации результатов использовался модуль Matplotlib [14], так как он наиболее прост в использовании и предоставляет весь необходимый функционал для задач, поставленных в рамках данной научноисследовательской работы.

На рисунках 1 и 2 представлен внешний вид среды разработки Pycharm и пользовательский интерфейс взаимодействия с графиками библиотеки Matplotlib.

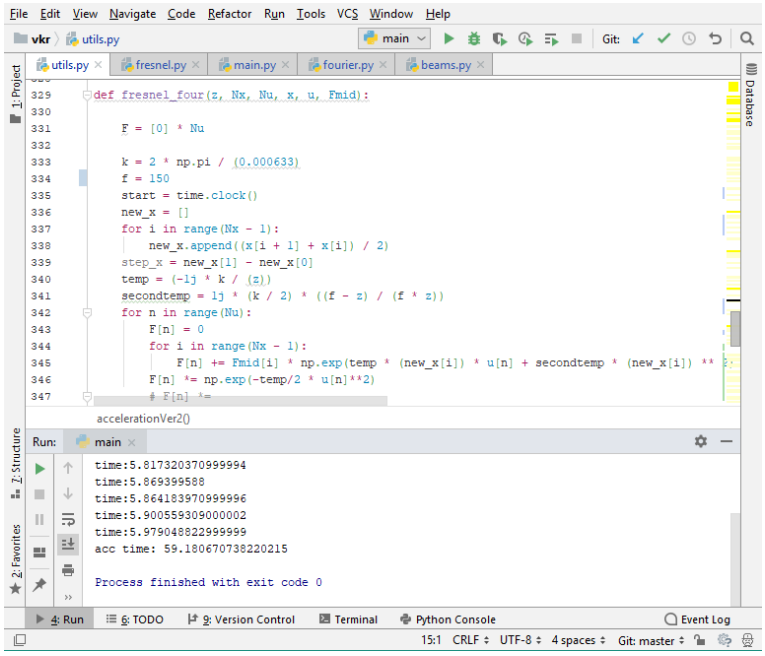

Рисунок 1 – Внешний вид среды разработки PyCharm

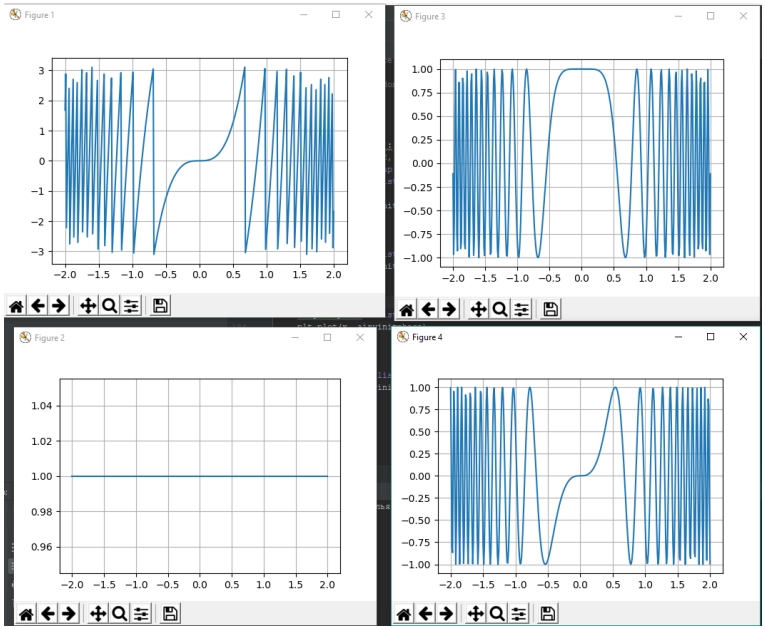

Рисунок 2 – Интерфейс взаимодействия с графиками, предоставляемый билиотекой Matplotlib

3 Построение входных распределений пучков Эйри, Пирси и их модификаций в виде пространственных спектров

3.1 Численное моделирование входного распределения пучков Эйри

Входное поле для пучков Эйри имеет вид (3).

Исходя из вида комплексной функции, нетрудно заметить, что амплитуда входного поля для любого аргумента будет равна 1. В свою очередь фазой входного поля будет является кубическая парабола, однако поскольку фаза комплексного числа изменяется в пределах от $-\pi$ до π или от 0 до 2π , то концы этой параболы будут представлять из себя набор сгущающихся линий по краям картинки.

Кроме того, рассматриваемая функция имеет параметр-множитель у кубического члена показателя экспоненты. Варьируя этот параметр, можно получить различную фазу входного поля, что в дальнейшем может существенно сказаться на амплитуде и фазе выходного поля [6]. В остальном представление данной функции в виде стандартной функции языка программирования Python полностью удовлетворяет требованиям, которые накладывает общий подход к реализации программного комплекса. Поскольку объем данных, которые подаются на вход преобразованию Фурье сильно влияет на качество выходных данных, то будет рационально увеличивать коэффициент перед кубическим слагаемым, так как таким образом увеличивается объем полезных данных на той же длине рассматриваемого отрезка, то есть можно получить более качественный результат за то же время моделирования. Результаты моделирования для различных значений параметра представлены на рисунках 3 и 4.

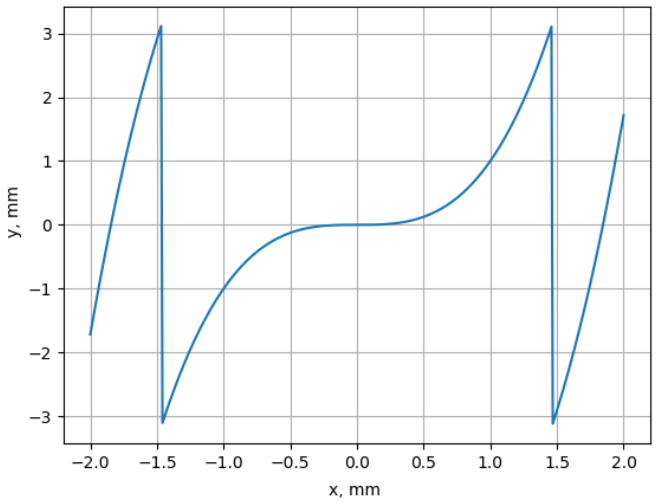

Рисунок 3 – Фаза входного поля Эйри при $\alpha = 1$

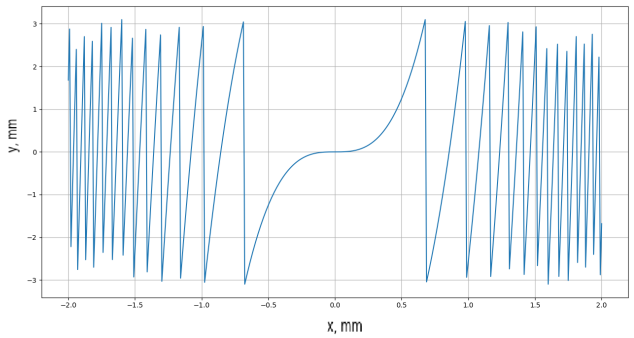

Рисунок 4 – Фаза входного поля Эйри при $\alpha = 30$

Сравнивая данные изображения, нетрудно заметить, что с увеличением параметра значительно увеличивается объем информации на одном и том же рассматриваемом отрезке. Этим приемом можно воспользоваться в дальнейшем, чтобы получить более информативное изображение выходного поля без увеличения времени вычислений, возникающего вследствие увеличения исследуемого отрезка.

Аналогичным образом можно получить двумерную картину фазы начального распределения пучков, которая в дальнейшем будет использоваться в качестве входных данных для двумерного преобразования Фурье. Вид двумерного пучка Эйри:

$$f(x,y) = exp(\frac{i\alpha x^3 + i\alpha y^3}{3}).$$

Двумерная фаза входного распределния представлена на рисунке 5.

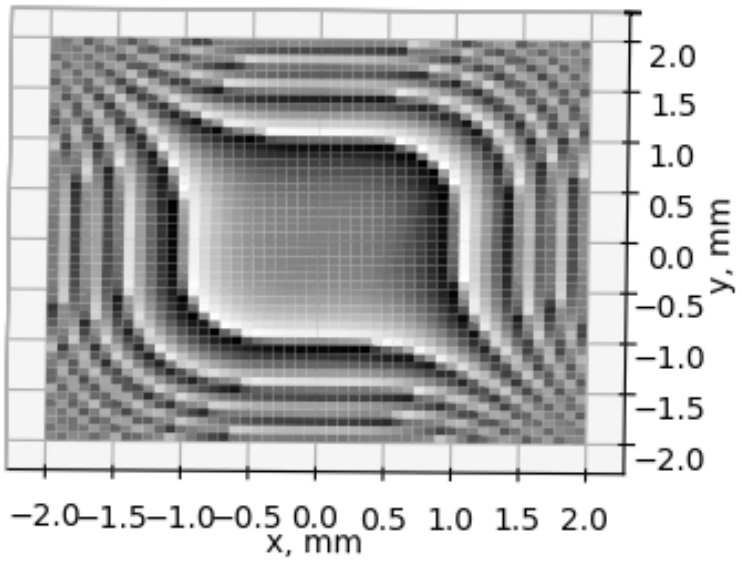

Рисунок 5 – Двумерная фаза входного поля Эйри при $\alpha = 3$

Программная реализация входного поля пучков Эйри и графической интерпретации его амплитуды, фазы, действительной и мнимой части приведена в разделе 1 приложения А.

3.2 Численное моделирование входного распределения пучков Пирси

Общий вид моделируемого входного поля для пучков Пирси может быть представлен следующим образом:

$$f(x,t) = exp[i(x^4 + tx^2)].$$

По аналогии с входным полем для пучков Эйри, помимо основного аргумента x в функции присутствует параметр, варьируя который можно изменять фазу входного поля и изменять характеристики выходного поля. Амплитуда входного поля также равна 1. Существенным различием между этими двумя функциями является то, что фаза входного поля пучков Эйри представляет собой нечетную функцию, в то время как фаза входного поля пучков Пирси, представленная на рисунке 6 – функция четная, что окажет определенное влияние на внешний вид выходных полей рассматриваемых пучков.

Для пространственного спектра пучков Пирси просто получить двумерную картину, так как она получается декартовым произведением массивов значений выходной функции по осям OX и OY, которые идентичны. На рисунке 6 представлена фаза входного поля Пирси.

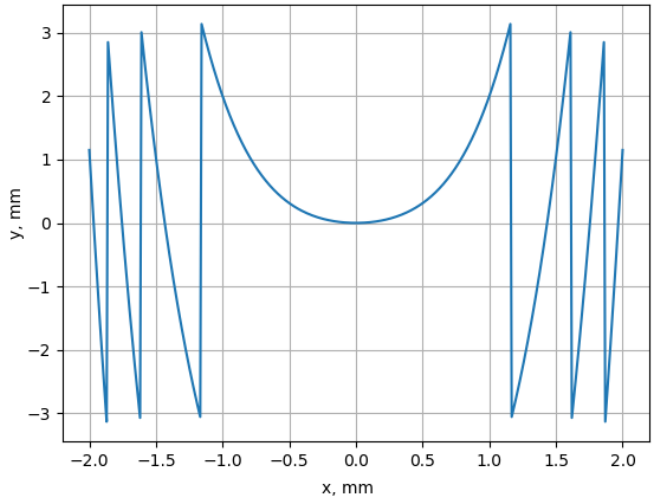

Рисунок 6 – Фаза входного поля Пирси $\alpha = 1$

Схожим образом получается двумерный вид пучка:

$$f(x, y) = \exp(i(x^4 + y^4 + tx^2 + ty^2))$$

и двумерная фаза входного распределения, представленная на рисунке 7.

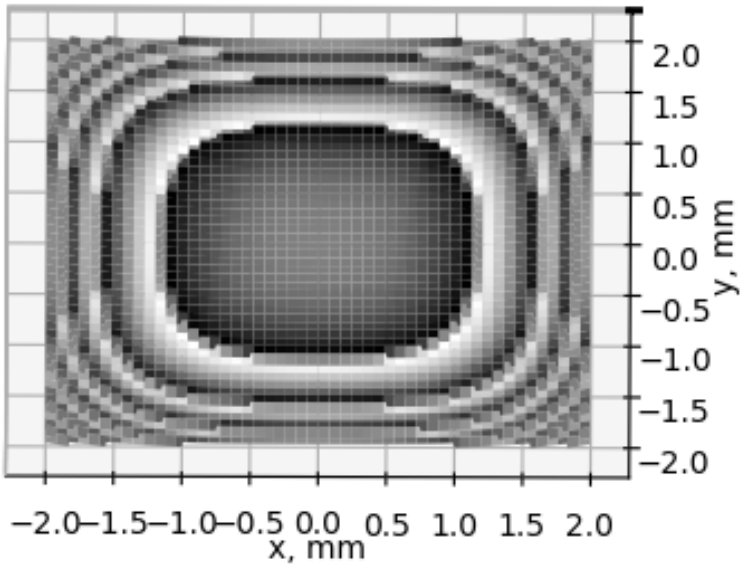

Рисунок 7 – Двумерная фаза входного поля Пирси при $\alpha = 1$

Программная реализация входного поля пучков Пирси и графической интерпретации его амплитуды, фазы, действительной и мнимой части приведена в разделе 1 приложения А.

3.3 Изменение характеристик пространственных спектров

Нетрудно заметить, что основное отличие между описанными выше пучками заключается в виде функции в показателе экспоненты. Это базовое отличие определяет некоторые свойства пучков, например, нечетность функции делает пучок ускоряющимся, а четность влечет за собой обратный эффект. Однако в некоторых ситуациях проще модифицировать существующих пучок, нежели создавать новый, поэтому произведение над пучками подобных преобразований представляет определенный научный интерес.

Для пучков Эйри задача заключается в модификации функции в показателе экспоненты с целью добиться ее четности. Наиболее простая реализация данной задачи заключается в добавлении модуля к аргументу функции. В таком случае входная функция примет следующий вид:

$$AiEven(x) = exp[-it\frac{|x|^3}{3}].$$

Поскольку амплитуда входного поля по-прежнему остается единицей, обратим внимание на график фазы входной функции, вид которой представлен на рисунке 8.

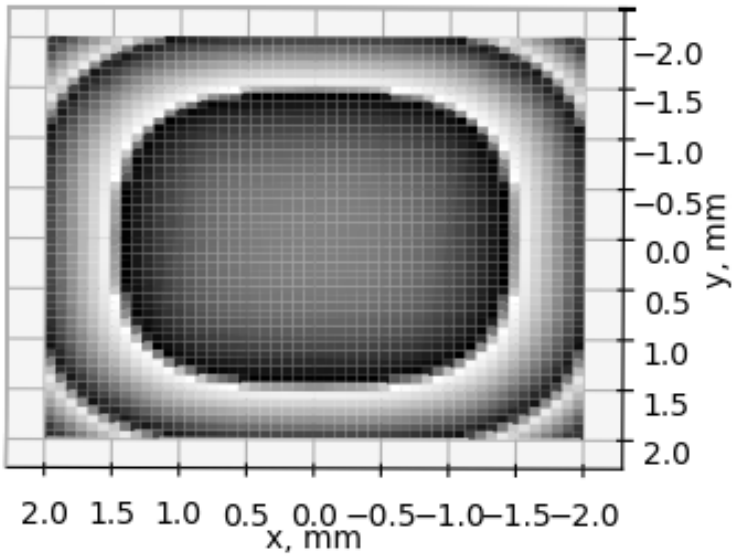

Рисунок 8 – Двумерная фаза модифицированного входного поля Эйри при $\alpha = 3$

Нетрудно заметить общую схожесть получившейся картины с ранее продемонстрированной фазой входного поля пучков Пирси. Таким образом, на данном этапе уже можно высказать предположение, что общий вид выходной картины модифицированного пучка структурно будет повторять внешний вид пучка Пирси.

Аналогичные рассуждения справделивы и для пучков Пирси. В данном случае ставится обратная задача – приведение четной функции в показателе экспоненты к нечетному аналогу, однако такого же элегантного решения проблемы нет, а наиболее удобной реализацией станет кусочно-заданная функция следующего вида:

$$PeOdd(x) = \begin{cases} \exp(i(x^4 + tx^2)) & : x \geq 0 \\ \exp(i(-x^4 - tx^2)) & : x < 0 \end{cases}$$

Тогда внешний вид входного распределения для двумерного модифицированного пучка Пирси примет вид, приведенный на рисунке 9.

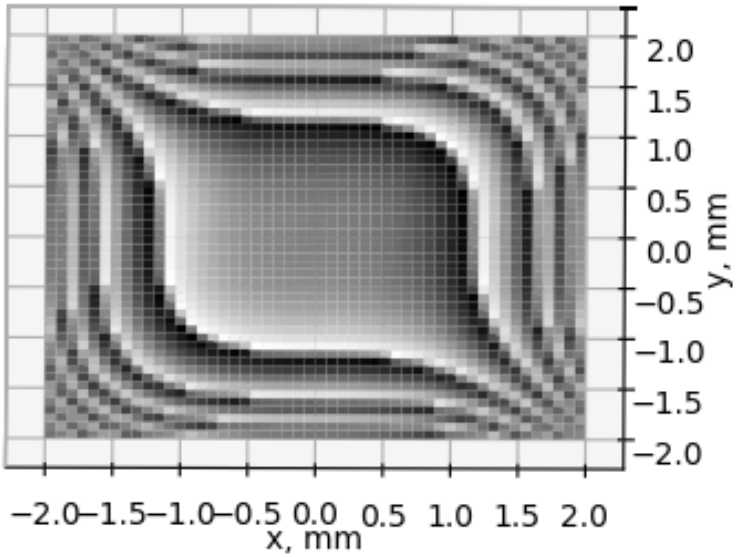

Рисунок 9 – Двумерная фаза модифицированного входного поля Эйри при $\alpha = 1$

В данном случае также достигнут желаемый результат: картина входного поля модифицированного пучка схожа с входным распределением стандартного пучка Эйри. В дальнейшем будет произведено сравнение результатов моделирования пар схожих пучков для оценки степени их подобия.

Подробная реализация программного модуля, осуществляющего построение входного распределения модифицированных пучков и визуализацию отдельных их параметров, представлена в разделе 1 приложения А.

4 Моделирование формирования пучков с помощью преобразования Фурье

Основной прием, который можно использовать при моделировании формирования пучков с помощью линзы – это перебор различных параметров: некоторые из них будут обусловлены физическими характеристиками линзы, а некоторые – особенностями методов создания пучков.

Кроме того, поскольку в рамках данной работы был реализован метод численного интегрирования, имеется возможность варьировать размер и отрезок покрытия входного и выходного массивов аргументов. Массив аргументов выходной функции будет оказывать не такое сильное влияние на общую картину, тогда как увеличение размерности и уменьшение интервала между отсчетами массива аргументов входной функции будет в значительно большей степени оказывать влияние на эксперимент. Это объясняется тем, что для подсчета конечного количества значений выходного поля требуется усечь область, на которой рассматривается входное поле со всей вещественной оси до какого-то отрезка. При таком подходе преобразование Фурье от исходной функции будет представлять из себя свертку исходного вида преобразования Фурье с функцией sinc:

$$\int_{c_1}^{c_2} Ai(x)exp[-ixt]dx = \frac{c_2 - c_1}{2\pi} \int_{-\infty}^{\infty} exp[\frac{ix^3}{3} + \frac{i(c_1 + c_2)}{2}(x - t)] \times \\ \times sinc[\frac{c_2 - c_1}{2}(x - t)]dx,$$

которая при увеличении рассматриваемого отрезка до всей вещественной оси обратится в дельта-импульс, вернув преобразованию Фурье исходный вид. Из этого можно сделать вывод, что влияние функции $sinc$ на точность преобразования будет ослабевать по мере увеличения отрезка рассмотрения входного поля [6].

Программная реализация функции численного интегрирования, выпол-

няющей преобразование Фурье представлена в разделе 2 приложения А. Логика работы программы такова, что функция интегрирования одинаково работает со всеми видами входных полей, так что его использование для моделирования распространения пучков конкретного вида требует только указания искомой функции в качестве параметра метода.

Как уже было сказано ранее, физический смысл преобразования Фурье – распространение пучка через линзу. Приведем условную схему оптической установки, соответствующей данному физическому процессу. Ее внешний вид представлен на рисунке 10.

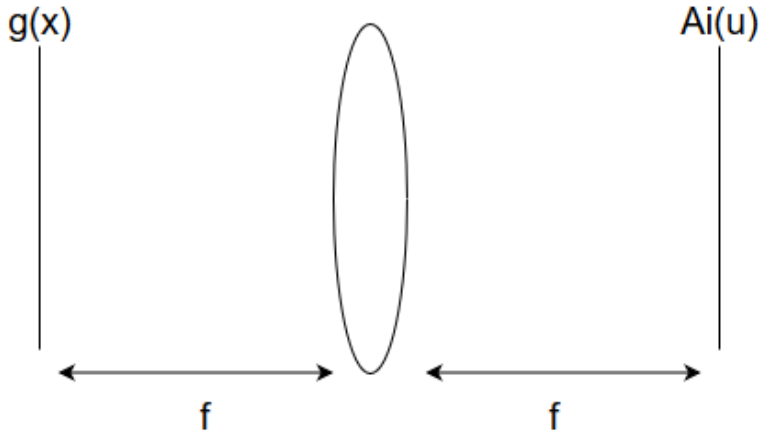

Рисунок 10 – Схема оптической установки для распространения пучка через линзу

На данной схеме видны две плоскости: входная, находящаяся слева от линзы, и выходная, находящаяся с противоположной стороны. Обе плоскости находятся на фокусном расстоянии f от линзы, то есть являются фокальными. На входной плоскости начинает свое распространение входная функция пучка, а на выходной плоскости формируется результат моделирования – сформированный пучок.

На следующих рисунках представлены результаты моделирования формирования пучков Эйри и Пирси с различными параметрами. Для краткости воспользуемся следующими обозначениями: x – массив аргументов входной функции, представляющий собой набор равноудаленных друг от друга эле-

ментов на вещественной оси абсцисс на участке от левой до правой границы, u – массив аргументов выходной функции, представляющий собой набор равноудаленных друг от друга элементов на вещественной оси абсцисс на участке от левой до правой границы, α – значение параметра функции входного поля.

Результат моделирования пучка Эйри представлен на рисунке 11.

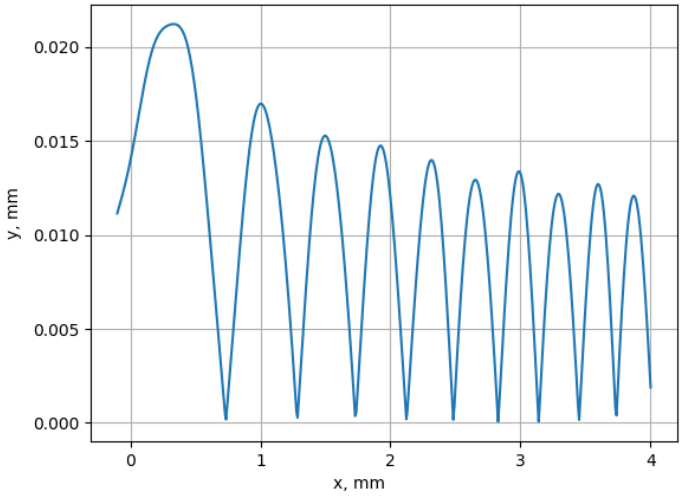

Рисунок 11 – Амплитуда выходного поля пучков Эйри, $x \in [-2; 2], \alpha = 10$

Как уже было сказано выше, одной из целей, преследуемых в данной работе, является выявление влияния различных параметров на выходную картину. Рассмотрим поочередно сравнение выходных картин при изменении фокусного расстояния линзы, параметра входной функции и входного отрезка рассмотрения. Результаты моделирования приведены на рисунках 12 - 14.

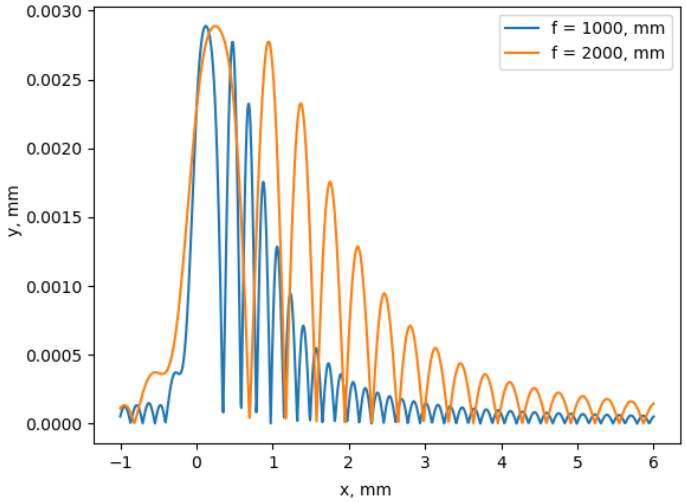

Рисунок 12 – Сравнение пучков Эйри для различных фокусных расстояний

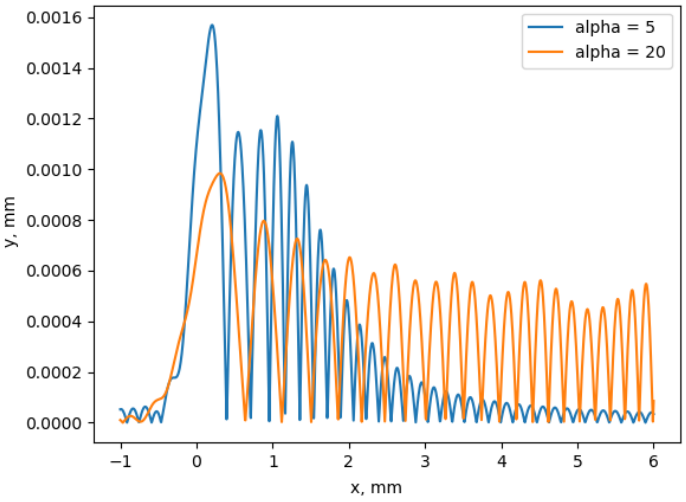

Рисунок 13 – Сравнение пучков Эйри, полученных для различных параметров входной функции

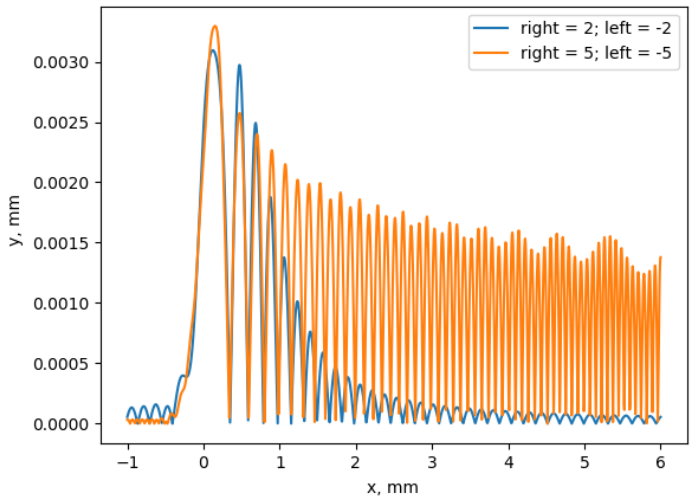

Рисунок 14 – Сравнение пучков Эйри для различных входных отрезков

Ранее в данной работе было предположено, что увеличение параметра входной функции приведет к похожему результату, который можно получить с помощью увеличения отрезка рассмотрения. Теперь можно утверждать, что данное предположение было верно, поскольку можно наблюдать, насколько существенно увеличилось потенциальное расстояние распространения, как при увеличении входного отрезка, так и при увеличении параметра входной функции. Увеличение фокусного расстояния линзы дает привычный и понятный эффект, который можно назвать масштбаированием, то есть визуально он воспринимается как приближение картинки, чему и соответствует увеличение фокусного расстояния линзы, например, в фотоаппарате.

Стоит отметить, что в теории амплитуда выходного поля равномерно затухает, так что волнообразность максимумов графиков на рисунках 13 - 14 является дефектом, который неизбежно возникнет при моделировании или физической реализации. Как уже было сказано выше, это объясняется необходимостью сужения до конечного отрезка области рассмотрения входной функции.

Проведем аналогичные сравнения для пучков Пирси. Результат моделирования формирования пучка Пирси представлен на рисунке 15, а сравнение

различных реализаций при изменении параметров на рисунках 16 - 18.

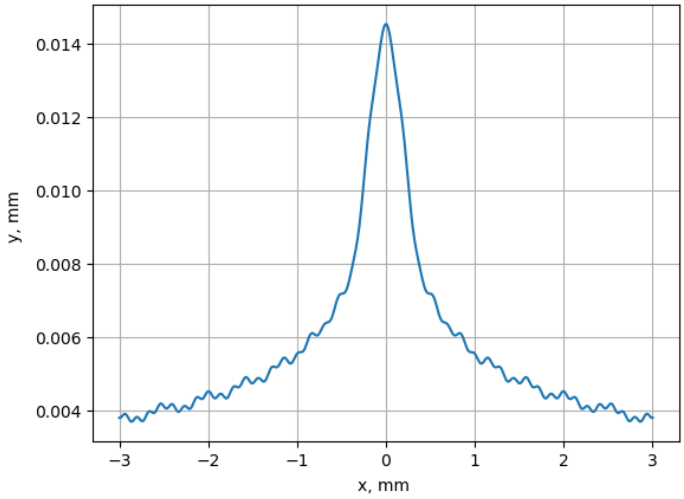

Рисунок 15 – Амплитуда выходного поля пучков Пирси при $\alpha = 1, f = 1000$

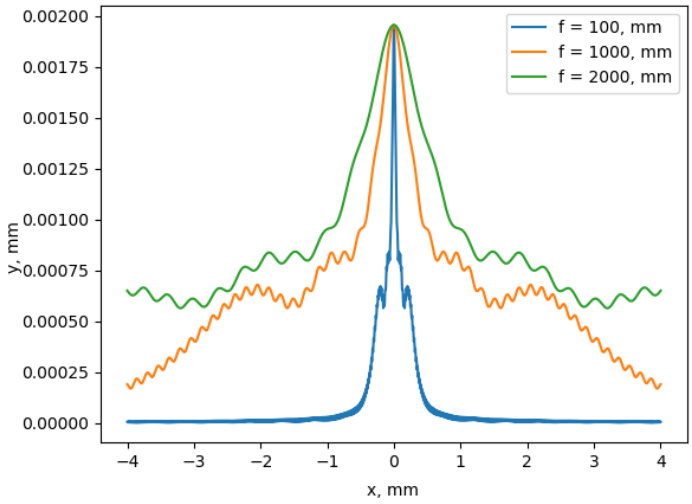

Рисунок 16 – Сравнение пучков Пирси для различных фокусных расстояний

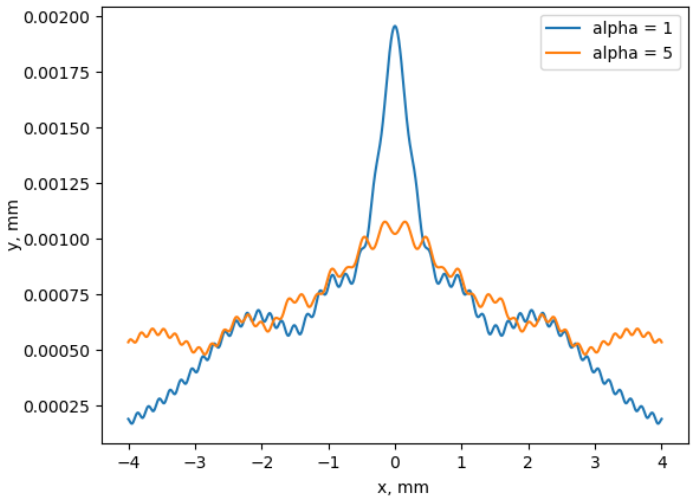

Рисунок 17 – Сравнение пучков Пирси, полученных для различных параметров входной функции

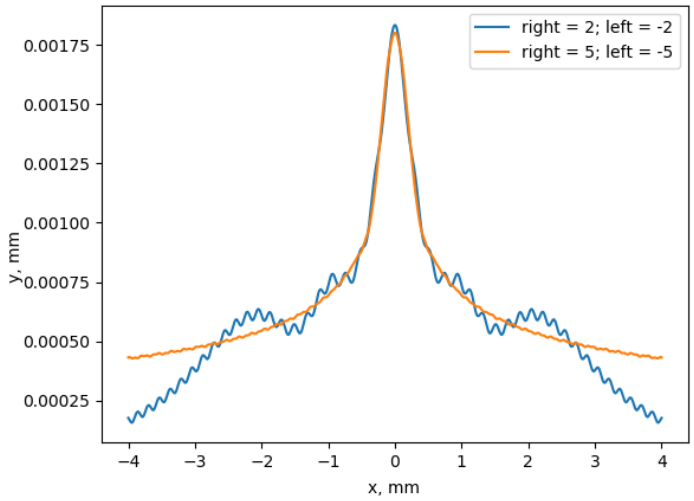

Рисунок 18 – Сравнение пучков Пирси для различных входных отрезков

Общий ход рассуждений при описании сравнений пучков Пирси с различными параметрами идентичен рассуждениям относительно пучков Эйри. Стоит только отметить, что колебания на ветвях графиков объясняются эффектом Гиббса из-за ограниченной апертуры оптической установки [15].

Далее рассмотрим изменения в картине амплитуды выходного поля при изменении параметра входного поля и фокусного расстоянии линзы f на примере пучков Пирси с зафиксированными параметрами $x \in [-1; 2], u \in [-3; 3]$.

Сравнивая графики на рисунках 19 и 20, можно увидеть, какое влияние оказывает несимметричность массива аргументов входной функции относительно отрезка рассмотрения выходной функции.

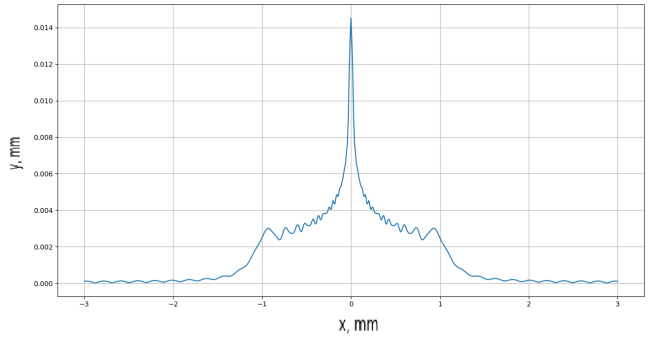

Рисунок 19 – Амплитуда выходного поля пучков Пирси, $x \in [-3; 3], \alpha = 1$

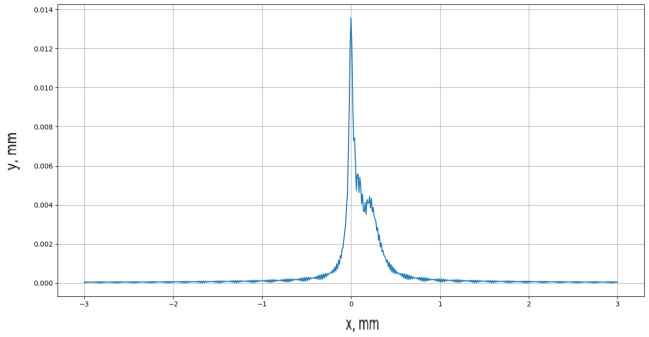

Рисунок 20 – Амплитуда выходного поля пучков Пирси, $x \in [-1; 2], \alpha = 1$

Можно легко убедиться в корректности результатов, сравнив полученные изображения с готовыми результатами, опубликованными в научных статьях по этой теме [16].

Попробуем убедиться в том, что модификации, произведенные над пучками в предыдущем разделе, действительно привели к желаемому результату и изменили их свойства ожидаемым образом. Для начала подвергнем рассмотрению модифицированный пучок Эйри, функция в показателе экспоненты у которого теперь стала четной функцией. Результат моделирования представлен на рисунке 21.

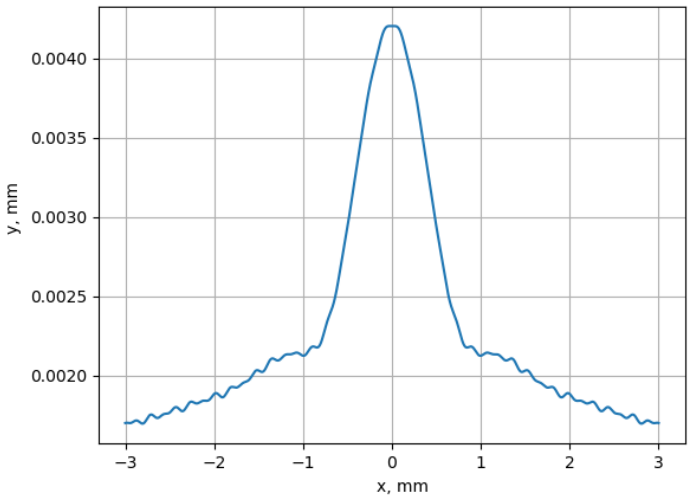

Рисунок 21 – Амплитуда выходного поля модифицированного пучка Эйри, $\alpha = 5, f = 2000$

Нетрудно заметить схожесть графика на последнем рисунке с, например, рисунком 15. Данная схожесть наблюдается для разных фокусных расстояний и параметров, однако в этом нет ничего удивительного, так как фактически входные функции пучков остаются разными, хотя и имеют ряд одинаковых свойств.

Несколько интереснее ситуация обстоит во втором случае, когда модифицировался пучок Пирси, так как из нашего предположения следует, что новая реализация должна была получить некоторые новые свойства, в частности, свойство ускорения. Рассмотрим результат моделирования, представленный на рисунке 22.

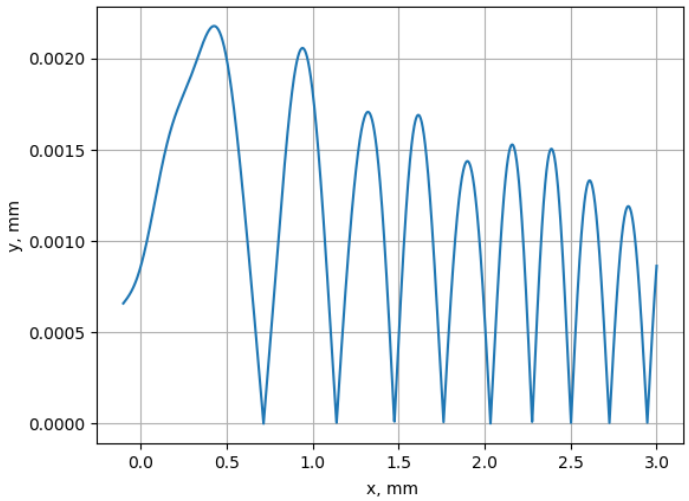

Рисунок 22 – Амплитуда выходного поля модифицированного пучка Пирси, $\alpha = 5, f = 2000$

Очевидно сходство рисунка 22 с рисунком 11, однако хотелось бы убедиться в том, что, кроме внешнего сходства, подобные преобразования передают некоторые свойства пучков. Проверка этой гипотезы осуществляется в разделе 6.

5 Моделирование распространения пучков с помощью преобразования Френеля

Рассмотрим результаты, полученные в ходе применения к рассматриваемым пространственным спектрам оператора распространения на произвольное расстояние. В целом, существенных отличий преобразования Френеля от Фурье не так много, а кроме того, преобразование Френеля связано с преобразованием Фурье и может быть представлено через через него, однако данный факт выходит за рамки исследования данной научной работы. Как уже объяснялось ранее, в теории бесконечные пучки Пирси и Эйри являются бездифракционными, однако на практике нам приходится обрезать отрезки рассмотрения или моделирования. В таком случае пучки остаются бездифракционными только на определенном расстоянии. Этим объясняется изменение внешнего вида пучков при увеличении расстояния распространения. Стоит отметить, что прохождение луча через линзу по своей сути ускоряет его свободное распространение, поэтому сформировать пучки Эйри или Пирси можно не только с помощью описанного ранее преобразования Фурье, но и с помощью распространения входного луча на достаточно большое расстояние.

Алгоритм вычисления значения интеграла в заданной точке представлен на рисунке 23.

```
for i in range(Nx - 1):
    F[n] += func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i]) \
            * np.exp((1j * k / (2 * z)) * ((x[i] + x[i + 1]) / 2 - u_n) ** 2)
```

Рисунок 23 – Фрагмент кода, отвечающий за вычисление значения интеграла в заданной точке

Приведем общий вид оптической схемы, выполняющей преобразование Френеля, на рисунке 24.

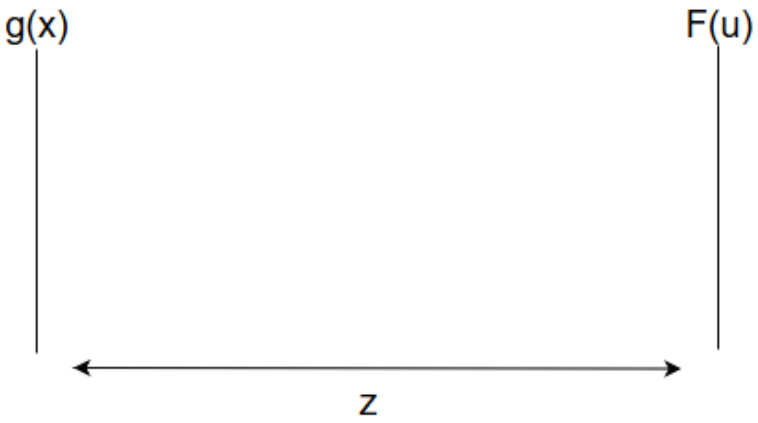

Рисунок 24 – Схема оптической установки для свободного распространения луча

Общий вид преобразования Френеля (6), как и основные принципы перехода к формулам численного интегрирования (4), уже были описаны в работе, так что перейдем к рассмотрению полученных результатов, представленных на рисунках 25 - 26.

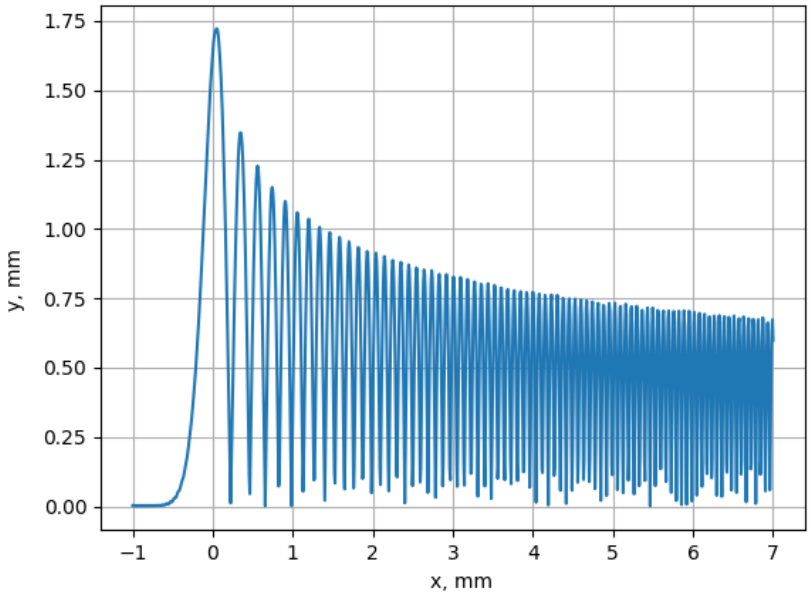

Рисунок 25 – График амплитуды пучка Эйри при $z = 300$мм

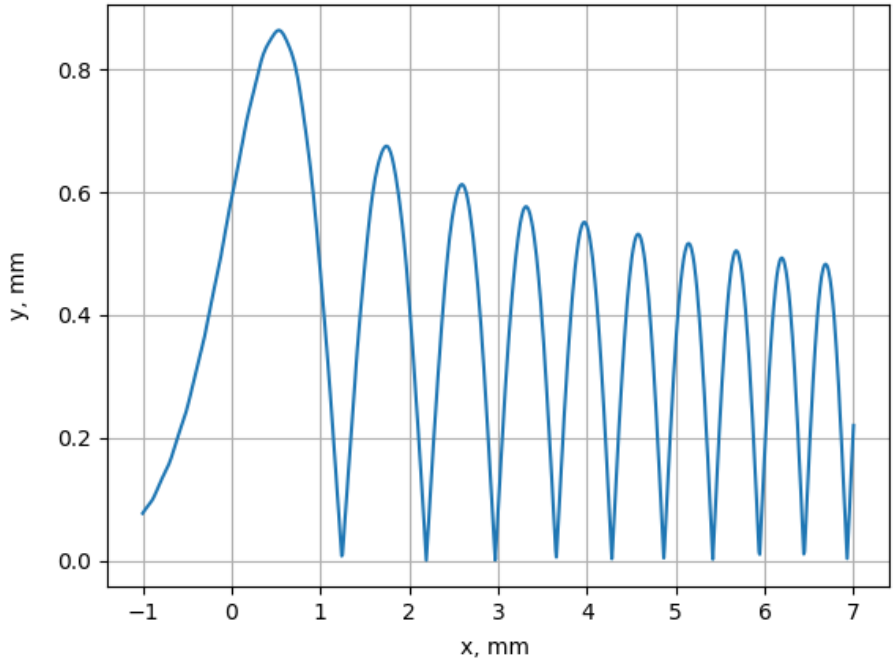

Рисунок 26 – График амплитуды пучка Эйри $z = 1200$мм

Подробное описание реализации программного модуля, осуществляющего преобразование Френеля, приведено в разделе 3 приложения А.

6 Моделирование распространения пучка вблизи фокальной области

В процессе моделирования может возникнуть интерес к случаю, который не описывается приведенными выше операторами распространения. Например, может возникнуть необходимость распространять уже сформированный пучок на некоторое расстояние вблизи фокальной области. Нетрудно догадаться, что в сути своей реализация данного случая требует совмещения физического смысла преобразований Фурье и Френеля. Рассмотрим этот случай подробнее.

Для реализзации данного случая существуют две опции:

1) включение оптического элемента в оператор распространения на свободное расстояние;

2) распространение пучка, полученного при прохождении входной функции через линзу.

Как уже было описано в разделе теоретических сведений, преобразование Френеля справедливо только для расстояний распространения много больше размеров входного окна. Второй способ не требует проведения никаких дополнительных математических преобразований, однако он не снимает ограничения на расстояние распространения, то есть с его помощью будет проблематично получить картину в ближней зоне. Первый способ в свою очередь позволяет в некоторой степени этим ограничением пренебречь, однако требует проведения дополнительных математических преобразований, которые делают это возможным. Дополним для этого преобразование Френеля (6) следующим оптическим элементом:

$$exp[-ik\frac{x^2}{2f}].$$

Тогда раскроем квадрат в показателе экспоненты, вынесем за знак интеграла слагаемое, содержащее u^2, так как оно не зависит от переменной инте-

грирования, и внесем добавленный оптический элемент под знак интеграла. После группировки некоторых слагаемых получим:

$$F(u) = -\sqrt{\frac{ik}{2\pi z}} exp[ikz] exp[\frac{iku^2}{2z}] \int\limits_{-\infty}^{+\infty} f(x) exp[\frac{ikx^2}{2}(\frac{1}{z} - \frac{1}{f})] exp[ik\frac{ux}{z}] dx,$$

где f – фокусное расстояние линзы;

z – расстояние от начальной плоскости до линзы.

По физическому смыслу оба случая идентичны, так что оптические схемы будут для них одинаковыми, хотя стоит отметить, что первый случай позволяет получить картину и до фокусного расстояния линзы, в то время как использование второго способа дает возможность увидеть картину только после его прохождение через линзу.

Однако не стоит забывать, что при использовании второго варианта в процессе вычислений входные данные дважды подвергаются усеканию, что не может не отразиться на качестве выходной картины и допустимых областях рассмотрения. Иными словами после формирования пучка дальнейшему распространению подвергается только та его часть, которая попала в выходное окно для преобразования Фурье, так что, вообще говоря, рассматривать существенно большее выходное окно для дальнейшего преобразования Френеля возможно, но практически не имеет смысла. Учтем данный факт при дальнейшем моделировании.

Вид оптической схемы, осуществляющей оба описанных преобразования, представлен на рисунке 27.

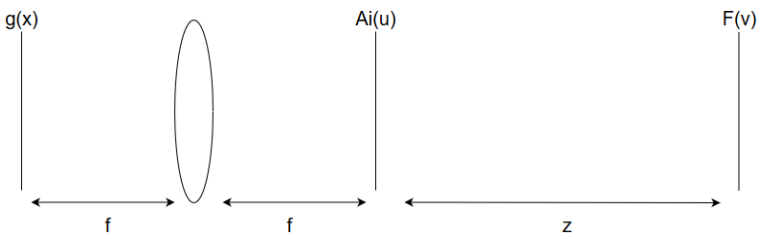

Рисунок 27 – Схема оптической установки для свободного распространения пучка, сформированного с помощью линзы

Данная схема не нуждается в подробном описании, поскольку является последовательным объединением схем, изображенных на рисунках 10 и 24, описанных ранее.

Реализация интегрирования в данной функции аналогична предыдущим реализациям. Также стоит заметить, что для данного случая справедливо замечание о получения двумерного преобразования из одномерного в частных случаях.

Поскольку две опции, обозначенные в начале раздела, несколько разделили сферы собственного применения, то приведем оба результата. Результаты моделирования, полученные с помощью второго способа будут в ыглядеть аналогично иллюстрациям из предыдущего раздела. Первый способ тоже подходит для этой цели, однако это отличный инструмент, чтобы продемонстрировать свойство ускорения, которое присуще некоторым видам пучков. В данной работе данным свойством обладают пучки Эйри и, по выдвинутому предположению, модификация пучка Пирси, при котором его входная функция становится нечетной функцией. Свойство ускорения будет наблюдаться в ближней зоне, то есть при формировании пучка в точке близкой к точке начала распространения.

Продемонстрируем результат моделирования с помощью второго способа на рисунке 28. Нетрудно увидеть существенное сходство с полученными ранее результатами, например, 25.

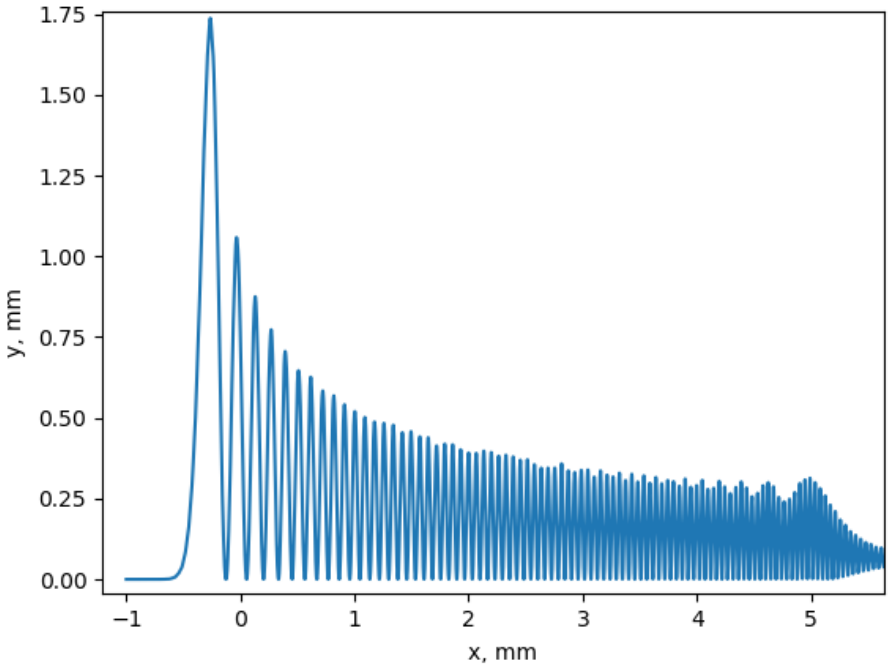

Рисунок 28 – График амплитуды пучка Эйри при $f = 300$мм, $z = 400$мм

Перейдем теперь к рассмотрению свойства ускорения, которое легко можно получить с помощью первого способа. Для этого зафиксируем начальную и конечную точки рассмотрения и пройдемся по получившемуся отрезку с зафиксированным шагом, вычисляя на каждой итерации полученное преобразование для заданного расстояния. Результатом данного вычислительного эксперимента станут изображения рисунках 29 - 30.

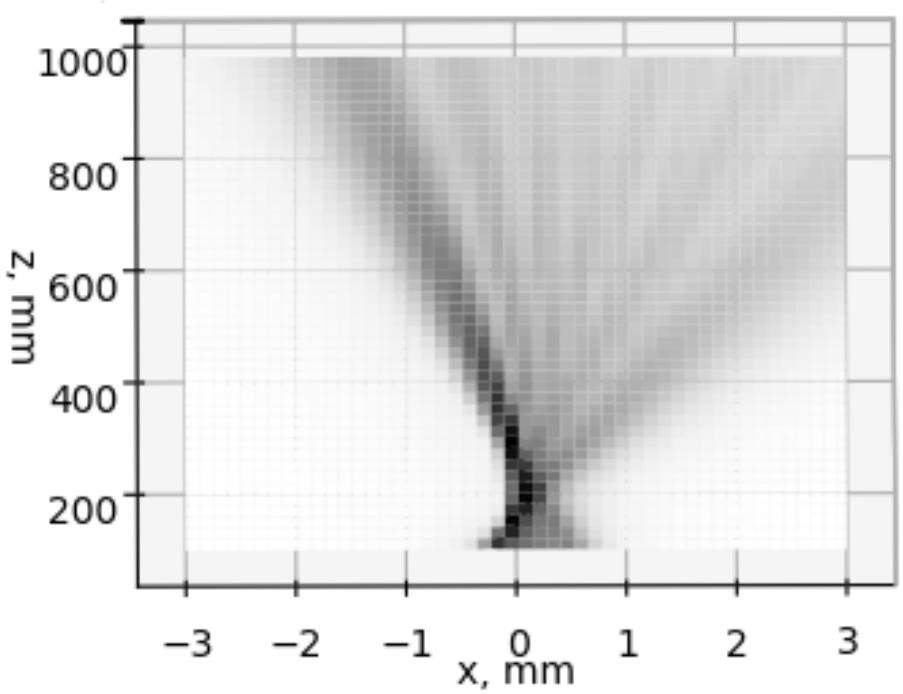

Рисунок 29 – Ускорение пучка Эйри, $f = 100$, мм

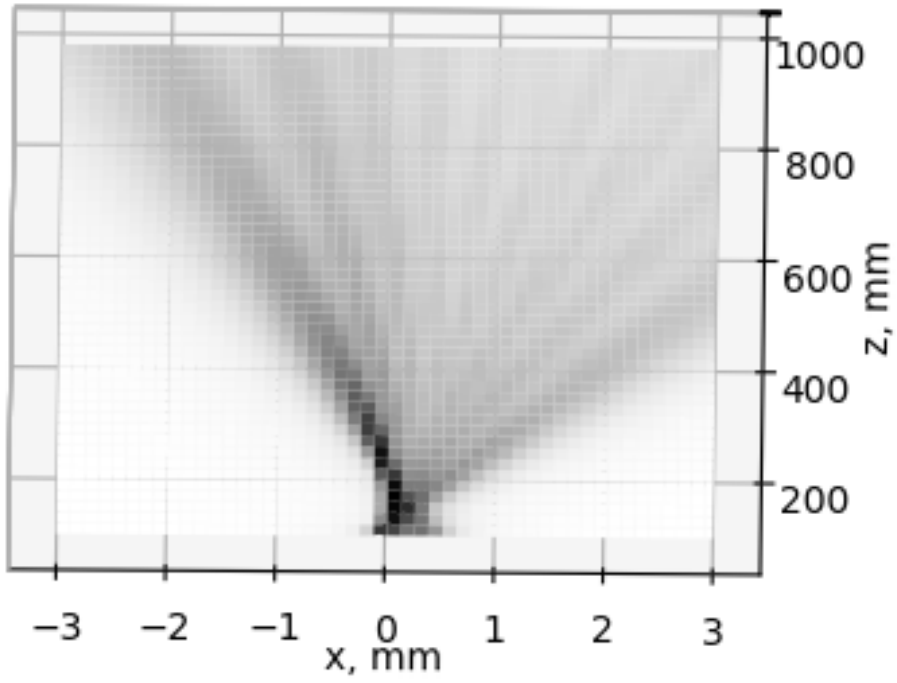

Рисунок 30 – Ускорение нечетного пучка Пирси, $f = 100$, мм

В данной цветовой карте темные темные области находятся выше светлых областей. В таком случае на рисунке 30 зависимость отклонения наиболее ярко выраженного пика от расстояния распространения отлична от линей-

ной. Именно это явление и называется ускорением, и в данном эксперименте было подтверждено наличие свойства ускорения у модифицированного пучка Пирси, что также находит подтверждение в [1]. Кроме того, можно заметить, что наиболее узкая и четкая область изображения соответствует приблизительному фокусному расстоянию линзы, что отвечает общеизвестным принципам работы собирающей линзы.

Таким образом, за исключением минорных отличий, была продемонстрирована возможность модификации пучков Эйри и Пирси в аналогичные друг другу с приобретением свойств желаемого пучка.

Подробное описание модуля, реализующего совмещение преобразований Френеля и Фурье, приведено в разделе 4 приложения А.

7 Формирование двумерных пучков

В общем виде двумерное преобразование Фурье уже было описано равенством (5).

Аналогично прежним рассуждениям, для вычисления двумерного преобразования Фурье использовалось численное интегрирование методом прямоугольников. Полная реализация преобразования приведена в приложении.

Нетрудно заметить, что вычислительная сложность используемого алгоритма составляет $O(n^4)$, если мы рассматриваем одинаковые квадратные входные и выходные области. Для получения удовлетворительный точности результатов время выполнения программы может превышать несколько минут. В связи с этим стоит заметить, что в частном случае, когда входная и выходная области симметричны, двумерное преобразование Фурье может быть получено из одномерного с помощью факторизации. Это осуществимо за счет разделимости подынтегрального выражения на два идентичных, каждое из которых зависит только от одной переменной. Данный метод позволяет существенно сократить время вычислений даже для моделирования с большей точностью, однако применим только в частном случае, например, он не будет подходить в случае намеренного зануления некоторых участков входного распределения.

На рисунках 31 - 32 приведены результаты моделирования двумерного преобразования Фурье.

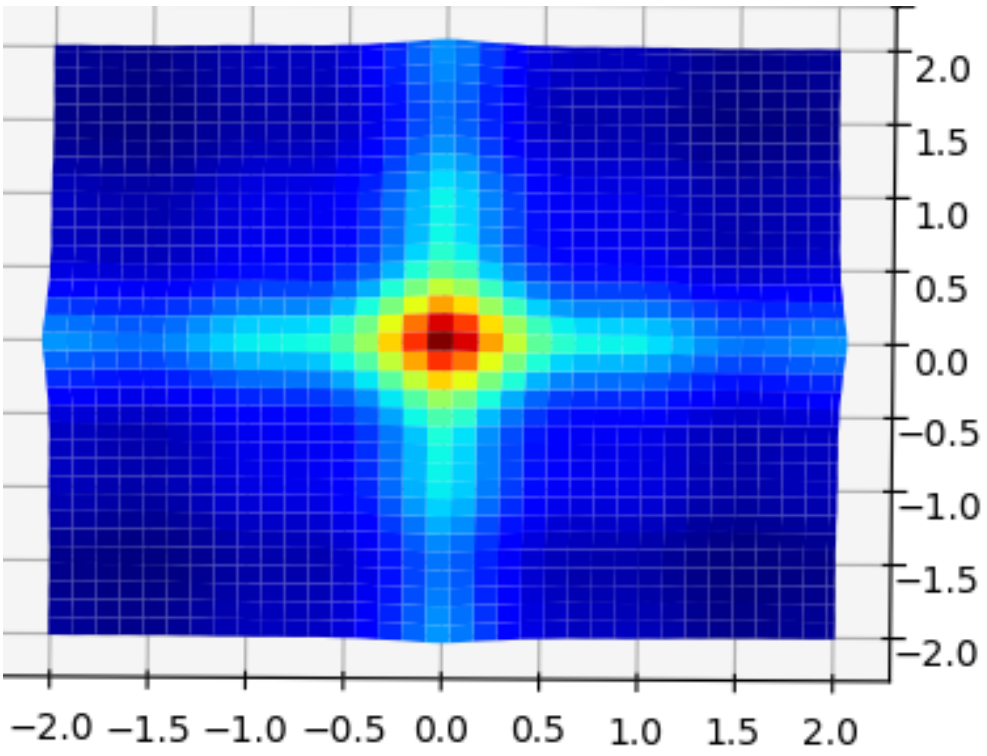

Рисунок 31 – График амплитуды двумерного пучка Пирси при $f = 650$мм

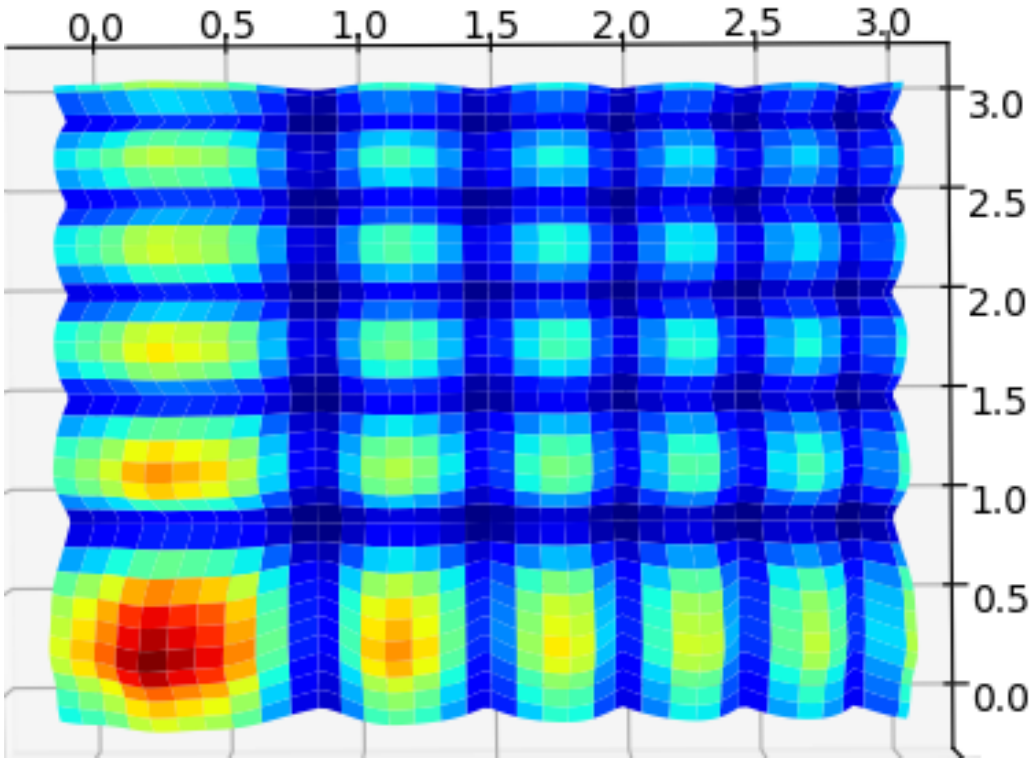

Рисунок 32 – График амплитуды двумерного пучка Эйри при $f = 650$мм

ЗАКЛЮЧЕНИЕ

В рамках данной работы были реализованы программные модули построения входных распределений в виде пространственных спектров пучков Эйри и Пирси, моделирования формирования пучков при прохождении через линзы и распространении на свободное расстояние, а также через оптическую систему, позволяющую моделировать распространение пучков вблизи фокальной области. Можно отметить, что результаты работы программы соответствуют действительности, так как совпадают с результатами, представленными в статьях, посвященных рассмотрению проблем аналогичной тематики, которые использовались в качестве библиографических источников. Также в пользу корректности результатов говорит дублирование некоторых результатов, полученных различными способами. В ходе выполнения работы были установлены некоторые закономерности между значениями параметров входных полей и операторов распространения и общим видом выходного поля. Было установлено, что увеличение объема входных данных положительно сказывается на общем виде выходной картины и расстоянии потенциального распространения, что достигается за счет увеличения отрезка рассмотрения входной функции, а также увеличения параметра входной функции. Было отмечено, что увеличение фокусного расстояния линзы при моделировании распространения пучков масштабирует график, что соответствует привычному принципу работы собирающей линзы.

Над пучками были проведены модификации с целью выявить возможность приобретения рассматриваемыми пучками некоторых свойств, характерных для пучков, чей внешний вид был взят в качестве желаемого результата для модификаций. Данное предположение нашло подтверждение: в частности, модифицированный пучок Пирси стал обладать свойством ускорения, чего нельзя сказать о его классическом виде.

Без внесения конструктивных изменений использование разработанной программы можно распространить на большое семейство пучков, что может являться объектом исследования в дальнейших работах. Кроме того, объектом исследования может стать свойство недифрагированности пучков, как представляющее существенный инетерес современной науки. Реализованный модуль действительно можно использовать для моделирования и анализа распространения пучков перед переходом к физической реализации оптической установки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Siviloglou, G.A. Observation of Accelerating Airy Beams [Текст] / G. A. Siviloglou [and other] // Physical Review Letters. – 2007. – Vol. 99(21), 213901.

2 Airy beam [Электронный ресурс] // Википедия : свободная энцикл. – Электрон. дан. – [Б. м.], 2018. – URL: https://en.wikipedia.org/wiki/Airy_beam (дата обращения: 11.11.2018).

3 Catastrophe theory [Электронный ресурс] // Википедия : свободная энцикл. – Электрон. дан. – [Б. м.], 2018. – URL: https://en.wikipedia.org/wiki/Catastrophe_theory (дата обращения: 10.12.2018).

4 Хонина, С. Н. Ограниченные одномерные пучки Эйри: лазерный веер [Текст] / С.Н. Хонина, С.Г. Волотовский // Компьютерная оптика. – 2008 – Т. 32, № 2. – С.168-174.

5 Auto-focusing and self-healing of Pearcey beams [Текст] / James D. Ring [and others] // Optics express – 2012 – Vol. 20, № 17. – P. 18955-18966.

6 Khonina, S. N. Specular and vortical Airy beams [Текст] / Svetlana N. Khonina // Optics Communications – 2011 – № 284(19) – P. 4263–4271

7 M. Abramowitz Handbook of Mathematical Functions [Текст] / M. Abramowitz, I.A. Stegun. – Dover, 1972. – 456 p.

8 Зорич В. А. Математический анализ [Текст]/ Зорич В. А. — М.: Физматлит, 1984. — 544 с.

9 Virtual source of a Pearcey beam [Текст] / Dongmei Deng [and others] // Optics Letters. – 2014. – Т. 39, № 9. – P. 2703-2706

10 Самарский А.А. Численные методы [Текст] / Самарский А.А., Гулин А.В. — М.: Наука. Гл. ред. физ-мат. лит., 1989.— 432 с.

11 Залманзон Л.А. Преобразования Фурье, Уолша, Хаара и их применение в управлении, связи и других областях [Текст]/ Залманзон Л.А. — М.: Наука. Гл. ред. физ.-мат. лит., 1989. - 496 с.

12 Numpy and Scipy Documentation [Электронный ресурс] // Официальный сайт документации библиотек Numpy и Scipy – Электрон. дан. – 2018 URL: https://docs.scipy.org/doc/ (дата обращения: 11.11.2018)

13 Mathematical functions for complex numbers [Электронный ресурс] // Документация: стандартная библиотека Python – Электрон. дан. – 2018. – URL: https://docs.python.org/2/library/cmath.html (дата обращения: 14.11.2018)

14 Matplotlib Documentation [Электронный ресурс] // Официальный сайт библиотеки Matplotlib – Электрон. дан. – 2018. – URL: https://matplotlib.org/3.0.2/index.html (дата обращения: 15.11.2018)

15 Gibbs phenomenon [Электронный ресурс] // Википедия : свободная энцикл. – Электрон. дан. – [Б. м.], 2019. – URL: https://en.wikipedia.org/wiki/Gibbs_phenomenon (дата обращения: 25.5.2019)

16 Acelerating optical beams [Текст] / Miguel A. Bandres [and others] // Optics and photonics news – 2013 – Vol. 24(6) – P. 30-37

ПРИЛОЖЕНИЕ А

Код программы

А.1 Модуль, отвечающий за входное распределение пучков

```
class Params:
    pe_param = 10
    Aiparam = 20
    sigma = 1000

def setPeParam( param):
    Params.pe_param = param

def setAiParam( param):
    Params.Aiparam = param

def getAiParam():
    return Params.Aiparam

def getPeParam():
    return Params.pe_param

def Pe(x):
    return np.exp(1j * (x ** 4) + getPeParam() * 1j * (x ** 2))

def PeGauss(x):
    return np.exp(x**2/Params.sigma)*np.exp(1j * (x ** 4)
                                    + getPeParam() * 1j * (x ** 2))

def Pe_with_param(x, t):
    return np.exp(1j * (x ** 4) + t * 1j * (x ** 2))

def PeOdd(x):
    if (x >= 0):
        return np.exp(1j * (x ** 4) + getPeParam() * 1j * (x ** 2))
    else:
        return np.exp(1j * -(x ** 4) + getPeParam() * 1j * -(x ** 2))
```

```
def PeOdd2d(x, y):
    if ((x >= 0) & (y >= 0)):
        return np.exp(1j * (x ** 4) + getPeParam() * 1j * (x ** 2) + 1j *
                    (y ** 4) + getPeParam() * 1j * (y ** 2))
    elif ((x >= 0) & (y < 0)):
        return np.exp(1j * (x ** 4) + getPeParam() * 1j * (x ** 2) + 1j
                    * -(y ** 4) + getPeParam() * 1j * -(y ** 2))
    elif ((x < 0) & (y > 0)):
        return np.exp(1j * -(x ** 4) + getPeParam() * 1j * -(x ** 2) + 1j
                    * (y ** 4) + getPeParam() * 1j * (y ** 2))
    else:
        return np.exp(1j * -(x ** 4) + getPeParam() * 1j * -(x ** 2) + 1j
                    * -(y ** 4) + getPeParam() * 1j * -(y ** 2))

def Gauss(x):
    return np.exp(-x ** 2 / Params.sigma ** 2)

def Ai(x):
    return np.exp(1j * getAiParam() * (x ** 3) / 3)


def AiEven(x):
    return np.exp(-1j * getAiParam() * (abs(x) ** 3) / 3)


def AiEven2d(x, y):
    return np.exp(1j * getAiParam() * (abs(x) ** 3) / 3 + 1j
                * getAiParam() * (abs(y) ** 3) / 3)


def AiI(x):
    return cmath.exp(I * getAiParam() * (x ** 3) + I * x)

def Pe2d(x, y):
    return np.exp(1j * (x ** 4 + y ** 4) + 1j * (x ** 2 + y ** 2))


def Ai2d(x, y):
    return np.exp(1j * getAiParam() * x ** 3 + 1j
                * getAiParam() * y ** 3)
```

```
def getInitPe2d():
    xright = 2
    xleft = -2
    yright = 2
    yleft = -2
    nx = 151
    ny = 151
    x = np.linspace(xleft, xright, nx)
    y = np.linspace(yleft, yright, ny)
    xstep = abs(x[1] - x[0])
    ystep = abs(y[1] - y[0])
    xmid = []
    ymid = []
    for i in range(nx - 1):
        xmid.append((x[i + 1] + x[i]) / 2)
        ymid.append((y[i + 1] + y[i]) / 2)
    xmid = np.array(xmid)
    ymid = np.array(ymid)
    Fphase = []
    for i in range(len(xmid)):
        xarr = []
        for j in range(len(ymid)):
            xarr.append(Pe2d(xmid[i], ymid[j]) * xstep * ystep)

        Fphase.append(xarr)
    return Fphase, xmid, ymid

def plotAiPhase():
    a = np.linspace(-1, 1, 100)
    Fabs = list(map(lambda x: np.angle(Ai(x)), a))
    plt.plot(a, Fabs)
    plt.grid()
    plt.show()

def plotPePhase():
    a = np.linspace(-2, 2, 100)
    Fabs = list(map(lambda x: np.angle(PeOdd(x)).real, a))
    plt.plot(a, Fabs)
    plt.grid()
```

```
    plt.show()

def getInitAi2d():
    xright = 2
    xleft = -2
    yright = 2
    yleft = -2
    nx = 151
    ny = 151
    x = np.linspace(xleft, xright, nx)
    y = np.linspace(yleft, yright, ny)
    xstep = abs(x[1] - x[0])
    ystep = abs(y[1] - y[0])
    xmid = []
    ymid = []
    for i in range(nx - 1):
        xmid.append((x[i + 1] + x[i]) / 2)
        ymid.append((y[i + 1] + y[i]) / 2)
    xmid = np.array(xmid)
    ymid = np.array(ymid)
    Fphase = []
    for i in range(len(xmid)):
        xarr = []
        for j in range(len(ymid)):
            xarr = np.append(xarr, Ai2d(xmid[i], ymid[j]) * xstep * ystep)
        Fphase.append(xarr)
    Fphase = np.array(Fphase)
    return Fphase, xmid, ymid

def getInitPeOdd2d():
    xright = 2
    xleft = -2
    yright = 2
    yleft = -2
    nx = 501
    ny = 501
    x = np.linspace(xleft, xright, nx)
    y = np.linspace(yleft, yright, ny)
    xstep = abs(x[1] - x[0])
    ystep = abs(y[1] - y[0])
```

```
    xmid = []
    ymid = []
    for i in range(nx - 1):
        xmid.append((x[i + 1] + x[i]) / 2)
        ymid.append((y[i + 1] + y[i]) / 2)
    xmid = np.array(xmid)
    ymid = np.array(ymid)
    Fphase = []
    for i in range(len(xmid)):
        xarr = []
        for j in range(len(ymid)):
            xarr.append(PeOdd2d(x[i], y[j]) * xstep * ystep)
        Fphase.append(xarr)

    for i in range(nx - 1):
        for j in range(ny - 1):
            a = Fphase[i][j]
            a = np.angle(a).real
            Fphase[i][j] = a
    Fphase = np.array(Fphase)

    x, y = np.meshgrid(x[0:nx - 1:1], y[0:ny - 1:1])
    a = 1
    Fphase = np.array(Fphase)
    fig = plt.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
    ax.plot_surface(x, y, Fphase, cmap=plt.cm.binary, antialiased=False)
    pylab.show()

def getInitAiEven2d():
    xright = 2
    xleft = -2
    yright = 2
    yleft = -2
    nx = 501
    ny = 501
    x = np.linspace(xleft, xright, nx)
    y = np.linspace(yleft, yright, ny)
    xstep = abs(x[1] - x[0])
    ystep = abs(y[1] - y[0])
```

```
    xmid = []
    ymid = []
    for i in range(nx - 1):
        xmid.append((x[i + 1] + x[i]) / 2)
        ymid.append((y[i + 1] + y[i]) / 2)
    xmid = np.array(xmid)
    ymid = np.array(ymid)
    Fphase = []
    for i in range(len(xmid)):
        xarr = []
        for j in range(len(ymid)):
            xarr.append(AiEven2d(x[i], y[j]) * xstep * ystep)
        Fphase.append(xarr)

    for i in range(nx - 1):
        for j in range(ny - 1):
            a = Fphase[i][j]
            a = np.angle(a).real
            Fphase[i][j] = a
    Fphase = np.array(Fphase)
    x, y = np.meshgrid(x[0:nx - 1:1], y[0:ny - 1:1])
    a = 1
    Fphase = np.array(Fphase)
    fig = plt.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
    ax.plot_surface(x, y, Fphase, cmap=plt.cm.binary, antialiased=False)
    pylab.show()


def plotPhasetAi2d():
    xright = 2
    xleft = -2
    yright = 2
    yleft = -2
    nx = 501
    ny = 501
    x = np.linspace(xleft, xright, nx)
    y = np.linspace(yleft, yright, ny)
    xstep = abs(x[1] - x[0])
    ystep = abs(y[1] - y[0])
```

```
    xmid = []
    ymid = []
    for i in range(nx - 1):
        xmid.append((x[i + 1] + x[i]) / 2)
        ymid.append((y[i + 1] + y[i]) / 2)
    xmid = np.array(xmid)
    ymid = np.array(ymid)
    Fphase = []
    for i in range(len(xmid)):
        xarr = []
        for j in range(len(ymid)):
            xarr.append(PeOdd2d(x[i], y[j]) * xstep * ystep)
        Fphase.append(xarr)

    for i in range(nx - 1):
        for j in range(ny - 1):
            a = Fphase[i][j]
            a = np.angle(a).real
            Fphase[i][j] = a
    Fphase = np.array(Fphase)

    x, y = np.meshgrid(x[0:nx - 1:1], y[0:ny - 1:1])
    a = 1
    Fphase = np.array(Fphase)
    fig = pylab.figure()
    axes = fig.gca(projection='3d', proj_type='ortho')
    axes.set_xlabel('x, mm')
    axes.set_ylabel('y, mm')
    axes.plot_surface(x, y, Fphase, cmap=plt.cm.binary)
    pylab.show()
```

A.2 Модуль, отвечающий за применение преобразования Фурье к данным

```
def fourierArr(Nx, x_right, x_left, f, u, func):
    x = np.linspace(x_left, x_right, Nx)
    k = 2 * np.pi / (0.000633 * f)
    start = time.clock()
    Fmid = [0] * (Nx - 1)
    for i in range(Nx - 1):
```

```
        Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
    output = []
    temp = -1j*k/2
    multiplier = (x_right - x_left) / (Nx)
    for j in range(len(u)):
        val = 0
        u_j = u[j]
        for i in range(Nx - 1):
            val += Fmid[i] * np.exp((temp * (x[i] + x[i + 1]) * u_j))
        output.append(val * multiplier)
    Fabs = list(map(abs, output))
    plt.plot(u, Fabs)
    return np.array(output)


def fourierDot(Nx, x_right, x_left, f, u, func):
    x = np.linspace(x_left, x_right, Nx)
    k = 2 * np.pi / (0.000633 * f)
    start = time.clock()
    Fmid = [0] * (Nx - 1)
    for i in range(Nx - 1):
        Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
    output = 0
    temp = -1j * k / 2
    for i in range(Nx - 1):
        output += Fmid[i] * np.exp( temp* (x[i] + x[i + 1]) * u)
    output = output * (x_right - x_left) / (Nx)
    return output


def Fourier(Nx, Nu, u_left, u_right, x_left, x_right, f, is2d, func):
    x = np.linspace(x_left, x_right, Nx)
    F = [0] * Nu
    u = np.linspace(u_left, u_right, Nu)
    k = 2 * np.pi / (0.000633 * f)
    start = time.clock()
    Fmid = [0] * (Nx - 1)
    for i in range(Nx - 1):
        Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
```

```
    temp = -1j * k / 2
    for j in range(Nu):
        F[j] = 0
        for i in range(Nx - 1):
            F[j] += Fmid[i] * np.exp((temp * (x[i] + x[i + 1]) * u[j]))
    F = list(map(lambda x: x * (x_right - x_left) / (Nx), F))
    Fabs = list(map(abs, F))
    end = time.clock()
    print("time:" + str(end - start))
    if (is2d):
        F2dAbs = []
        for i in range(Nu):
            row = [j * Fabs[i] for j in Fabs]
            F2dAbs.append(row)

        arrayF2dAbs = np.array(F2dAbs)
        z, y = np.meshgrid(u, u)

        fig = pylab.figure()
        axes = Axes3D(fig)
        axes.plot_surface(z, y, arrayF2dAbs, cmap=plt.cm.jet)
        print("another done")
    else:
        plt.plot(u, Fabs)
        plt.xlabel('x, mm')
        plt.ylabel('y, mm')
        plt.grid()
        plt.show()
    return Fabs


def Fourier2d():
    init, x, y = beams.getInitPe2d()
    start = time.clock()
    uleft = -2
    uright = 2
    vleft = -2
    vright = 2

    nv = 101
```

```
    nu = 101
    koef = 2 * np.pi / (0.000633 * 1000)
    u = np.linspace(uleft, uright, nu)
    v = np.linspace(vleft, vright, nv)
    output = []
    len_x = len(init[0])
    len_y = len(init)
    koef_mul_i = -1j * koef
    for i in range(nu):
        outputline = []
        u_i = u[i]
        for j in range(nv):
            integrateelem = 0
            v_j = v[j]
            for k in range(len_x): # x
                x_k = x[k]
                x_u = x_k * u_i
                for p in range(len_y): # y
                    integrateelem += init[k][p] * np.exp(koef_mul_i * (x_u + y[p] * v_j))
            outputline.append(integrateelem)
    for i in range(nu):
        for j in range(nv):
            a = output[i][j]
            output[i][j] = abs(a)

    end = time.clock()
    print("TIME: ", end - start)
    output = np.array(output)
    u, v = np.meshgrid(u, v)
    fig = pylab.figure()
    axes = Axes3D(fig)
    axes.plot_surface(u, v, output, cmap=plt.cm.jet)
    pylab.show()
```

A.3 Модуль, отвечающий за применение преобразования Френеля к данным

```
def fresnel(z, Nx, Nu, x_right, x_left, u_right, u_left, func):
    maxvals = []
```

```
x = np.linspace(x_left, x_right, Nx)
F = [0] * Nu
u = np.linspace(u_left, u_right, Nu)
wavelength = 0.000633
k = 2 * np.pi / (wavelength)
temp = (1j * k / (2 * z))
start = time.clock()
Fmid = [0] * (Nx - 1)
for i in range(Nx - 1):
    Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
for n in range(Nu):
    F[n] = 0
    u_n = u[n]
    for i in range(Nx - 1):
        F[n] += Fmid[i] * np.exp(temp * ((x[i] + x[i + 1]) / 2 - u_n) ** 2)
F = list(map(lambda x: x * np.exp(1j*k*z) * np.sqrt(-1j*k/(2*np.pi*z)), F))
print(np.sqrt(-1j * k / (2 * np.pi * z)))
print('a')

Fabs = list(map(lambda x: abs(x) ** 2, F))
end = time.clock()
maxVal = np.amax(Fabs)
maxvals.append(maxVal)
mean = reduce(lambda x1, x2: x1 + x2, Fabs) / len(Fabs)
print('maxval:', str(maxVal))
print('mean:', str(mean))
print('diff:', str(maxVal / mean))
print('time:' + str(end - start))

if (False):
    F2dAbs = []
    for i in range(Nu):
        row = [j * Fabs[i] for j in Fabs]
        F2dAbs.append(row)

    arrayF2dAbs = np.array(F2dAbs)
    z, y = np.meshgrid(u, u)

    fig = pylab.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
```

```
        ax.plot_surface(z, y, arrayF2dAbs, cmap=plt.cm.jet)
        print("another done")
    else:
        plt.plot(u, Fabs)
        plt.xlabel('x, mm')
        plt.ylabel('y, mm')

    plt.xlim((u_left, u_right))
    plt.ylim((0, np.max(maxvals)*1.1))
    plt.legend(('param=1', 'param=5', 'param=20', 'param=40'))
    plt.grid()
    plt.show()
    return Fabs

def fresnelDot(z, Nx, x_right, x_left, u, func):
    x = np.linspace(x_left, x_right, Nx)
    k = 2 * np.pi / (0.000633)
    temp = (1j * k / (2 * z))
    start = time.clock()
    Fmid = [0] * (Nx - 1)
    for i in range(Nx - 1):
        Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
    out = 0
    for i in range(Nx - 1):
        out += Fmid[i] * np.exp(temp * ((x[i] + x[i + 1]) / 2 - u) ** 2)
    out *= np.exp(1j * k * z) * np.sqrt(-1j * k / (2 * np.pi * z))
    return out

def fresnelArr(z, new_u, out_u, Ffour):
    F = [0] * len(out_u)
    wavelength = 0.000633
    k = 2 * np.pi / (wavelength)
    temp = (1j * k / (2 * z))
    step_u = new_u[1] - new_u[0]
    for n in range(len(out_u)):
        F[n] = 0
        out_u_n = out_u[n]
        for i in range( len(new_u)):
            F[n] += Ffour[i] * np.exp(temp * (new_u[i] - out_u_n) ** 2)
    F = list(map(lambda x: x * np.exp(1j*k*z) * np.sqrt(-1j*k/(2*np.pi * z)), F))
```

```
    Fabs = list(map(lambda x: abs(x) ** 2, F))
    plt.plot(out_u, Fabs)
    plt.show()
    end = time.clock()
    return Fabs
```

A.4 Модуль, содержащий в себе вспомогательные функции для моделирования и построения

```
def acceleration(Nx, x_right, x_left, Nu, u_left, u_right):
    u = np.linspace(u_left, u_right, Nu)

    z_quan = 4
    repeat_time = 40
    z_0 = 100
    z_step = 10
    v = np.linspace(z_0, z_0 + z_step * (z_quan - 1), z_quan)
    output = []
    start = time.time()
    for i in range(z_quan):
        outputLine = fresnel.fresnel(i * z_step + z_0, Nx, Nu, x_right, x_left, u_right, u_left, be
        output.append(outputLine)
    output = np.array(output)
    end = time.time()
    print('acc time: ' + str(end - start))
    u, v = np.meshgrid(u, v)
    fig = plt.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
    ax.view_init(100, 100)
    ax.set_xlabel('x, mm')
    ax.set_ylabel('z, mm')
    ax.plot_surface(u, v, output, cmap=plt.cm.binary)
    pylab.show()



def accelerationVer3(Nx, x_right, x_left, Nu, u_left, u_right):
    u = np.linspace(u_left, u_right, Nu)
    z_quan = 1
```

```
    z_0 = 400
    z_step = 40
    v = np.linspace(z_0, z_0 + z_step * (z_quan - 1), z_quan)


    out_u = np.linspace(u_left, u_right, Nu)
    output = []
    start = time.time()
    func = beams.Ai
    new_u = []
    for i in range(Nu - 1):
        new_u.append((u[i + 1] + u[i]) / 2)
    Ffour = fourier.fourierArr(Nx, x_right, x_left, 300, new_u, func)
    Fabs = list(map(lambda x: abs(x), Ffour))
    plt.plot(new_u, Fabs)
    plt.xlabel('x, mm')
    plt.ylabel('y, mm')
    plt.grid()
    plt.show()


    for i in range(z_quan):
        outputLine = fresnel.fresnelArr(i * z_step + z_0,new_u, out_u, Ffour)
        output.append(outputLine)

    output = np.array(output)
    end = time.time()
    print('acc time: ' + str(end - start))
    out_u, v = np.meshgrid(out_u, v)
    fig = plt.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
    ax.view_init(100, 100)
    ax.set_xlabel('x, mm')
    ax.set_ylabel('z, mm')
    ax.plot_surface(u, v, output, cmap=plt.cm.binary)
    pylab.show()

def fresfour(z, Nx, Nu, x_right, x_left, u_right, u_left, func, f, is2d):
    F = []
    u = np.linspace(u_left, u_right, Nu)
```

```
    out_u = np.linspace(1.1 * u_left, 1.1 * u_right, int(0.5 * Nu))
    k = 2 * np.pi / (0.000633)
    temp = (1j * k / (2 * z))
    start = time.clock()
    new_u = []
    for i in range(Nu - 1):
        new_u.append((u[i + 1] + u[i]) / 2)
    Ffour = fourier.fourierArr(Nx, x_right, x_left, f, new_u, func)
    print("done args")
    end = time.clock()
    print('time:' + str(end - start))
    Fabs = list(map(lambda x: abs(x), Ffour))
    plt.plot(new_u, Fabs)
    plt.xlabel('x, mm')
    plt.ylabel('y, mm')
    plt.grid()
    plt.show()

    for n in range(len(out_u)):
        buff = 0
        for i in range(Nu - 1):
            buff += Ffour[i] * np.exp(temp * (new_u[i] - out_u[n]) ** 2)
        F.append(buff)
    F = list(map(lambda x: x * np.exp(1j * k * z) * np.sqrt(-1j * k / (2 * np.pi * z)), F))
    Fabs = list(map(lambda x: abs(x), F))
    end = time.clock()

    print('time:' + str(end - start))
    if (is2d):
        F2dAbs = []
        for i in range(Nu):
            row = [j * Fabs[i] for j in Fabs]
            F2dAbs.append(row)

        arrayF2dAbs = np.array(F2dAbs)
        z, y = np.meshgrid(u, u)

        fig = pylab.figure()
        ax = fig.gca(projection='3d', proj_type='ortho') ax.plot_surface(z, y, arrayF2dAbs, cmap=pl
↪ pylab.show()
```

```
        print("another done")
    else:
        plt.plot(out_u, Fabs)
        plt.xlabel('x, mm')
        plt.ylabel('y, mm')
        plt.grid()
        plt.show()



def accelerationVer2(Nx, x_right, x_left, Nu, u_left, u_right):
    u = np.linspace(u_left, u_right, Nu)
    x = np.linspace(x_left, x_right, Nx)
    z_quan = 45
    z_0 = 100
    z_step = 20
    v = np.linspace(z_0, z_0 + z_step * (z_quan - 1), z_quan)
    output = []
    start = time.time()
    Fmid = [0] * (Nx - 1)
    func = beams.PeOdd
    for i in range(Nx - 1):
        Fmid[i] = func((x[i] + x[i + 1]) / 2) * (x[i + 1] - x[i])
    for i in range(z_quan):
        outputLine = fresnel_four(i * z_step + z_0, Nx, Nu, x, u, Fmid)
        output.append(outputLine)

    output = np.array(output)
    end = time.time()
    print('acc time: ' + str(end - start))
    u, v = np.meshgrid(u, v)
    fig = plt.figure()
    ax = fig.gca(projection='3d', proj_type='ortho')
    ax.view_init(100, 100)
    ax.set_xlabel('x, mm')
    ax.set_ylabel('z, mm')
    ax.plot_surface(u, v, output, cmap=plt.cm.binary)
    pylab.show()

def fresnel_four(z, Nx, Nu, x, u, Fmid):
```

```
    F = [0] * Nu

    k = 2 * np.pi / (0.000633)
    f = 150
    start = time.clock()
    new_x = []
    for i in range(Nx - 1):
        new_x.append((x[i + 1] + x[i]) / 2)
    step_x = new_x[1] - new_x[0]
    temp = (-1j * k / (z))
    secondtemp = 1j * (k / 2) * ((f - z) / (f * z))
    for n in range(Nu):
        F[n] = 0
        for i in range(Nx - 1):
            F[n] += Fmid[i] * np.exp(temp * (new_x[i]) * u[n] + secondtemp * (new_x[i]) ** 2
↪ F[n] *= np.exp(-temp/2 * u[n]**2)
        F = list(map(lambda x: x * np.exp(1j * k * z) * np.sqrt(-1j * k / (2 * np.pi * z)), F))
    Fabs = list(map(abs, F))
    end = time.clock()
    print('time:' + str(end - start))

    plt.plot(u, Fabs)
    return Fabs

def normalizeArray(arr):
    max = np.max(arr)
    arr /= max
    return arr


def beamsForDifferentFocus(Nx, x_right, x_left, Nu, u_left, u_right, f, func):
    u = np.linspace(u_left, u_right, Nu)
    x = np.linspace(x_left, x_right, Nx)
    leg = []
    for i in range(len(f)):
        a = fourier.fourierArr(Nx, x_right, x_left, f[i], u, func)
        leg.append('f = ' + str(f[i]) + ', mm')
    plt.legend(leg)
    plt.show()
```

```
def beamsForDifferentParam(Nx, x_right, x_left, f, Nu, u_left, u_right, params, func):
    u = np.linspace(u_left, u_right, Nu)
    x = np.linspace(x_left, x_right, Nx)
    leg = []
    print(func == beams.Ai)
    isAi = True if(func == beams.Ai) else False
    print(isAi)
    for i in range(len(params)):
        if(isAi):
            beams.setAiParam(params[i])
            print(beams.getAiParam())
        else:
            beams.setPeParam(params[i])
            print(beams.getPeParam())
        a = fourier.fourierArr(Nx, x_right, x_left, f, u, func)
        leg.append('alpha = ' + str(params[i]))
    plt.legend(leg)
    plt.show()

def beamsForDifferentInputRange(f, Nu, u_left, u_right, rights, lefts, func):
    f = 1000
    u = np.linspace(u_left, u_right, Nu)
    leg = []
    for i in range(len(rights)):
        a = fourier.fourierArr((rights[i]-lefts[i])*800+1, rights[i], lefts[i], f, u, func)
        leg.append('right = ' + str(rights[i]) + "; left = " + str(lefts[i]))
    plt.legend(leg)
    plt.show()
```