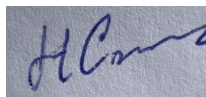


Институт: Новых материалов и технологий

КУРСОВАЯ РАБОТА

по теме: Разработка программного модуля на примере расчёта
и построения подшипника качения

Руководитель: Котел Наталья Сергеевна



Студент: Русинов Егор Константинович



Группа: НМТВ-123902

г. Екатеринбург
2023

Задание на курсовую работу

Студент Русинов Егор Константинович

группа НМТВ-123902

специальность/направление подготовки 09.03.02 Информационные системы и технологии

1. Тема курсовой работы

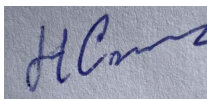
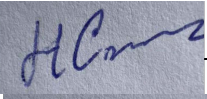
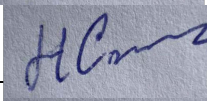
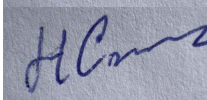
Разработка программного модуля на примере расчёта и построения подшипника качения

2. Содержание работы, в том числе состав графических работ и расчётов

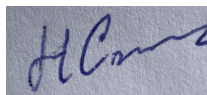
1. Разработать программный модуль на примере расчёта и построения подшипника качения
2. По исходным данным задачи №1 построить 3D-модель подшипника качения в системе КОМПАС 3D
3. Оформить пояснительную записку к курсовой работе и выложить её на LMS-платформе «Курсовое проектирование» для студентов УрФУ

3. Дополнительные сведения

4. План выполнения курсовой работы

Наименование элементов курсовой работы	Сроки	Примечания	Отметка о выполнении
Разработка программного модуля на примере расчёта и построения подшипника качения	04.03.23- 25.03.23		
Построение 3D-модели подшипника качения в CAD-системе КОМПАС 3D	01.04.23- 22.04.23		
Оформление пояснительной записки и размещение её в сервисе «Курсовое проектирование»	29.04.23- 20.05.23		
Защита курсовой работы	22.05.23- 11.06.23		

Руководитель Н. С. Котел / И. О. Фамилия /



СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАСЧЕТА ПОДШИПНИКА НА ДОЛГОВЕЧНОСТЬ.....	6
АЛГОРИТМ ПРОГРАМНОГО МОДУЛЯ	9
КОД ПРОГРАМНОГО МОДУЛЯ	12
ПОСТРОЕНИЕ 3D-МОДЕЛИ ПОДШИПНИКА В КОМПАС 3D.....	28
ТЕСТ ПРОГРАМНОГО МОДУЛЯ	32
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ ...	35
ПРИЛОЖЕНИЯ	36

ВВЕДЕНИЕ

Разработка программных модулей для автоматических расчетов и построения 3D моделей различных инженерных объектов является актуальной темой в современной инженерной практике. Такие программные модули имеют широкий диапазон применения, включая автомобильную, авиационную, судостроительную, нефтегазовую, металлургическую промышленность и многие другие отрасли, где используются подшипники качения.

Подшипник качения – это важный элемент механизмов, обеспечивающий высокую надежность и долговечность их работы. В свою очередь, правильный расчет параметров подшипника качения является ключевым в проектировании сложных механизмов.

Основными целями данной работы являются:

1. Изучение ГОСТа 831-75 Подшипники шариковые радиально-упорные однорядные.
2. Закрепление навыков программирования на языке Python, создание приложения с графическим интерфейсом для расчета и черчения подшипника качения.
3. Изучение техник и возможностей системы моделирования КОМПАС 3D, закрепление навыков на практике создания 3D-модели подшипника качения.
4. Разработка программного модуля с графическим интерфейсом для расчета основных технических параметров подшипника качения, таких как долговечность, а также динамической отрисовки чертежа подшипника/

Python (версия 3.10) — это язык программирования, который широко используется в интернет-приложениях, разработке программного обеспечения с графическим интерфейсом, науке о данных и машинном обучении (ML). Разработчики используют Python, потому что он эффективен, прост в изучении и работает на разных платформах. Программы на языке Python можно скачать бесплатно, они совместимы со всеми типами систем и повышают скорость разработки. Для создания приложений с графическим интерфейсом используется библиотека tkinter, предоставляющая широкие возможности для его развития.

КОМПАС-3D – это один из самых популярных и многофункциональных инструментов трехмерного моделирования. Это программное обеспечение является продуктом русской компании АСКОН и позволяет полностью автоматизировать процесс создания проектной документации согласно всем требованиям Единой Системы Конструкторской Документации или ЕСКД.

Использование компьютерных технологий в проектировании механизмов позволяет сократить время и затраты на разработку продукции и повышает качество и надежность механизма. Разработка программного модуля на языке Python позволит закрепить и отточить навыки программирования, работа с библиотекой tkinter поможет наработать опыт создания и

наполнения визуально красивых и удобных программ, моделирование.

Основная задача модуля – определение параметров подшипника, соответствующих условиям задачи. Модуль базируется на методах математического расчета, используя данные о геометрии детали, силах, трениях, режимах работы и других факторах, оказывающих влияние на работу подшипника качения. Данные из задачи используются для построения трехмерной модели подшипника, а также для расчета его рабочих характеристик.

В данной работе будут рассмотрены методы разработки программного продукта на языке Python в среде разработки PyCharm, с использованием стандартной библиотеки tkinter, а также использование программы Компас-3D для создания трехмерной модели подшипника. Будут описаны алгоритмы решения расчетных задач, отображающие ход выполнения программных операций и результаты работы.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАСЧЕТА ПОДШИПНИКА НА ДОЛГОВЕЧНОСТЬ

Расчёт подшипников на долговечность производится исходя из их динамической грузоподъёмности.

Динамической грузоподъёмностью радиальных и радиально-упорных подшипников называется постоянная радиальная нагрузка, которую группа идентичных подшипников с неподвижным наружным кольцом сможет выдержать в течение расчетного срока службы, исчисляемого в часах.

Долговечность подшипника определяется как срок службы до появления признаков контактной усталости металла на любом из колец или тел качения.

Под расчётным сроком службы понимают срок службы партии подшипников, в которых не менее 90% одинаковых подшипников, при одной и той же нагрузке и частоте вращения должны обработать без появления на рабочих поверхностях раковин и отслаивания.

Зависимость между номинальной долговечностью (расчётным сроком службы), динамической грузоподъёмностью и действующей на подшипник нагрузкой определяется формулой

$$L = \left(\frac{C}{P} \right)^p$$

где: L- номинальная долговечность, млн. оборотов;

C- динамическая грузоподъёмность;

P – эквивалентная динамическая нагрузка;

p – показатель степени в формуле долговечности (для шариковых подшипников p=3, для роликовых p= 10/3 или 3,33).

Эквивалентной динамической нагрузкой для радиальных шариковых и радиально – упорных подшипников называется постоянная радиальная нагрузка, которая при приложении её к подшипнику с вращающимся внутренним кольцом и неподвижным наружным обеспечивает такой же расчётный срок службы, как при действительных условиях нагружения и вращения. Для этих типов подшипников эквивалентная динамическая нагрузка определяется по формуле:

$$P = (X * V * F_r + Y * F_a) * K_o * K_T$$

где: Fr – постоянная по величине и направлению радиальная нагрузка, Н;

Fa- постоянная по величине и направлению осевая нагрузка, Н;

X - коэффициент радиальной нагрузки;

Y – коэффициент осевой нагрузки;

V – коэффициент вращения (V=1);

K_б – коэффициент безопасности;

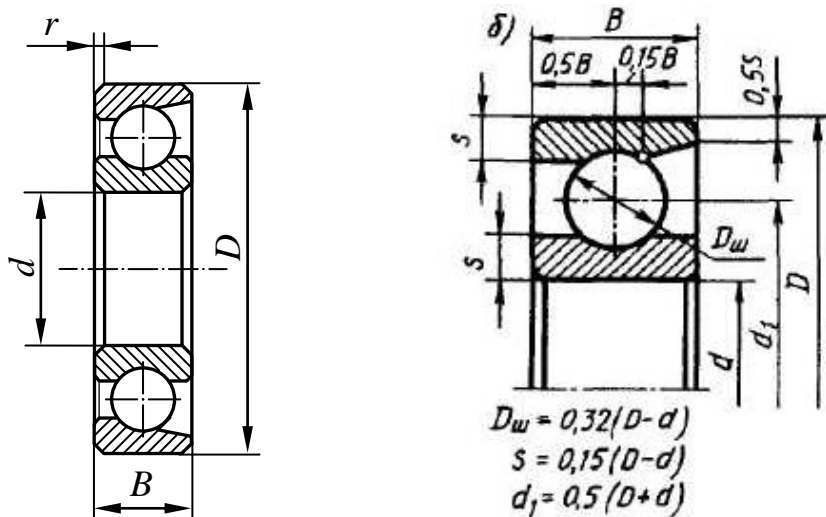
K_T – температурный коэффициент.

Эквивалентная динамическая нагрузка для подшипников, не обладающих осевой или радиальной нагрузкой может быть определена по формулам ниже:

$$P = X * V * F_r * K_b * K_T$$

$$P = Y * F_a * K_b * K_T$$

Согласно моему варианту (№2), в модуле будет рассчитываться и чертиться подшипник шариковый радиально-упорный легкой серии № 36204 (ГОСТ 831-75) по следующей схеме:



d – диаметр отверстия внутреннего кольца радиального и радиально-упорного подшипника и «тугого» кольца одинарного упорного подшипника;

d_1 – диаметр отверстия свободного кольца упорного подшипника;

D – внешний диаметр наружного кольца радиального и радиально-упорного подшипника и свободного кольца упорного подшипника;

B – ширина колец радиальных и радиально-упорных подшипников или ширина внутреннего кольца, если ширины колец не одинаковы;

Размеры тел качения для основных типов подшипников можно определить по следующим формулам:

Для шарикоподшипников:

S – толщина кольца $S = 0,15 * (D - d)$,

d_1 – ось шарика $d_1 = 0,5 \cdot (D + d)$,

D_w – диаметр шарика $D_w = 0,32 \cdot (D - d)$.

Данные для расчета и построения шарикового радиально-упорного подшипника легкой серии № 36204 (ГОСТ 831-75) приведены в Приложении.

АЛГОРИТМ ПРОГРАММНОГО МОДУЛЯ

Структура программного модуля

Для разделения логики работы программы и дальнейшего повторного использования кода, было принято решение разбить ее на структуру следующего вида:

main.py

bearing (подшипник)

1. *__init__.py*

2. *calculate.py*

3. *core.py*

где: **main.py** – главный исполняемый файл, который можно запускать из командной строки с помощью команды `python main.py`. В нем определены главные переменные с данными из Приложения (постоянная по величине и направлению радиальная нагрузка, постоянная по величине и направлению осевая нагрузка, коэффициент радиальной нагрузки, коэффициент осевой нагрузки, коэффициент вращения, коэффициент безопасности, температурный коэффициент – для расчета долговечности подшипника; диаметр отверстия внутреннего кольца радиального и радиально-упорного подшипника и «тугого» кольца одинарного упорного подшипника, диаметр отверстия свободного кольца упорного подшипника, внешний диаметр наружного кольца радиального и радиально-упорного подшипника и свободного кольца упорного подшипника, ширина колец радиальных и радиально-упорных подшипников или ширина внутреннего кольца, если ширины колец не одинаковы – для построения чертежа подшипника).

bearing – программный пакет для расчета и отрисовки рабочего окна приложения, включающий в себя следующие модули:

__init__.py – инициализатор пакета. В него импортируются и в нем определяются все необходимые классы и функции для расчета и построения подшипника. Также в данном модуле реализуется функция для наполнения главного окна программы необходимыми полями.

calculate.py – модуль, в котором определены все основные функции, формулы и классы для необходимых вычислений и построения расчетов. Основные классы: **CalcData** – для расчета параметров эквивалентной динамической нагрузки и номинальной долговечности, и **DrawCoords** – для расчета координат точек, необходимых для построения 2D-чертежа подшипника.

core.py – модуль, в котором определены все необходимые классы для кастомизации,

наполнения и осуществления черчения на основном окне. Использование определенных в этом модуле классов планируется в соответствии с паттерном проектирования “Mixins”. Основные функции и классы наследуются или переопределяют функции библиотеки tkinter.

Мета-алгоритм программного модуля:

1. Создание главного окна.
 - 1.1. Создать главное окно размером 1200 на 600 пикселей.
 - 1.2. Изменить название главного окна на «Расчет подшипника на долговечность».
 - 1.3. Изменить иконку главного окна на подшипник.
2. Наполнить главное окно полями для ввода данных.
 - 2.1. Добавить на главное окно поле «Начертить подшипник».
 - 2.2. Добавить на главное окно поля для ввода данных подшипника.
Осуществить привязку добавленных полей к внутренним переменным для отслеживания изменения значений.
 - 2.3. Добавить на главное окно кнопки «Значения по умолчанию» и «Начертить подшипник» с привязкой к внутренним функциям для наполнения и считывания данных из вышестоящих полей.
 - 2.4. Добавить на главное окно поле «Рассчитать подшипник».
 - 2.5. Добавить на главное окно поля для ввода данных подшипника.
Осуществить привязку добавленных полей к внутренним переменным для отслеживания изменения значений.
 - 2.6. Добавить на главное окно кнопки «Значения по умолчанию» и «Рассчитать подшипник» с привязкой к внутренним функциям для наполнения и считывания данных из вышестоящих полей.
 - 2.7. Добавить на главное окно поля для вывода параметров эквивалентной динамической нагрузки и номинальной долговечности подшипника.
3. Наполнить главное окно полями для черчения.
 - 3.1. Добавить на главное окно поле для создания чертежа
 - 3.2. Добавить на созданное в п. 3.1. поле два прямоугольника с помощью методов классов и модуля `сое.ру`.
 - 3.3. Привязать первый прямоугольник к кнопке «Начертить подшипник»
 - 3.4. Привязать второй прямоугольник к изображению 3D-модели подшипника

Работа программного модуля:

1. При нажатии на кнопку «Значения по умолчанию» в полях соответствующего столбца, выше нажатой кнопки появляются значения из переменной, определенной в главном исполняемом файле main.py. Данные по умолчанию взяты из таблиц в Приложении.
2. При нажатии на кнопку «Рассчитать подшипник» в полях соответствующего столбца, ниже нажатой кнопки появляются вычисленные значения эквивалентной динамической нагрузки и номинальной долговечности соответственно по формулам, приведенным в разделе Теоретические основы расчета подшипника на долговечность. При изменении исходных данных для внесения корректировок в ответ необходимо повторно нажать данную кнопку.
3. При нажатии на кнопку «Начертить подшипник» в центральном прямоугольнике появляется чертеж подшипника с проведенными осями, заштрихованными поверхностями и соответствующими пропорциями. В правом прямоугольнике появляется изображение 3D-модели подшипника по ГОСТ 831-75. При изменении исходных данных для внесения корректировок в построенный чертеж необходимо повторно нажать данную кнопку. Изображение 3D-модели подшипника при этом не меняется.

КОД ПРОГРАМНОГО МОДУЛЯ

Код *main.py*

```
from bearing import *                                # Импортируем все необходимое из пакетов

defaults_calc = {'C': 15.7,                          # Данные из таблицы 3.1
                 'p': 3,                             # "Исходные данные для расчёта
                 'Fr': 2.08,                          # подшипников на долговечность"
                 'Fa': 3.142,
                 'X': 0.46,
                 'Y': 1.421,
                 'V': 1,
                 'k_b': 1.3,
                 'k_t': 1}

defaults_draw = {'d': 20,                            # Данные из таблицы 3.3
                 'D': 47,                             # "Подшипники шариковые радиально-упорные
                 'B': 14,                             # однорядные (ГОСТ 831-75)"
                 'r': 1.5}

def main(draw_data, calc_data):
    data = DrawData(draw_data)
    calc = CalcData(calc_data)

    win = MyWin(data, calc)
    draw_places(win)

    image = get_image('images/goal.png')
    get_window(win, data, image)
    win()

if __name__ == '__main__':
    main(defaults_draw, defaults_calc)
```

Код *__init__.py*

```
from .core import *
from .calculate import *
```

```

class MyWin(ButtonWin, LabelWin, CustomWin, DrawWin):
    pass

def draw_places(win) -> None:
    win.draw_rect(5, 5, 395, 595)
    win.draw_rect(405, 5, 795, 595)

def create_draw(win: MyWin, data, im):
    def inner():
        left_angle = (5, 5)
        size = (390, 590)
        for k in win.entries_draw:
            data[k] = float(win.entries_draw.get(k).get())

        coords = DrawCoords(data, left_angle, size)
        coords.resize_data()
        coords.add_params()
        center = coords.get_center()
        # Очищаем место
        win.clear_place(5, 5, 395, 595)
        win.clear_place(405, 5, 795, 595)
        # Рисуем внешний контур
        out_fig = coords.get_outer_polygon()
        win.draw_line(*(out_fig + out_fig[:2]), size=3)
        # Рисуем внутренний цилиндр
        in_cil = coords.get_inner_cylinder()
        win.draw_line(*(in_cil + in_cil[:2]), size=3)
        # Рисуем косые
        win.draw_line(in_cil[0] - coords['r'], in_cil[1] - coords['r'], in_cil[0],
in_cil[1], size=3)
        win.draw_line(in_cil[2] + coords['r'], in_cil[3] - coords['r'], in_cil[2]
, in_cil[3], size=3)
        win.draw_line(in_cil[4] + coords['r'], in_cil[5] + coords['r'], in_cil[4],
in_cil[5], size=3)
        win.draw_line(in_cil[6] - coords['r'], in_cil[7] + coords['r'], in_cil[6],
in_cil[7], size=3)
        # Рисуем цилиндр до шариков, статор
        big_cil = coords.get_big_cylinder()
        win.draw_line(*(big_cil + big_cil[:2]), size=3)
        # Штрихуем статор

```

```

up_in_fill = coords.fill_up_stator()      # Верхняя часть
for i in up_in_fill:
    win.draw_line(*i, size=1)
down_in_fill = coords.fill_down_stator()  # Нижняя часть
for i in down_in_fill:
    win.draw_line(*i, size=1)

# Рисуем внутреннюю для подшипника сторону
up_lines = coords.get_upinner_lines()
win.draw_line(*up_lines[:4], size=3)
win.draw_line(*up_lines[4:], size=3)
down_lines = coords.get_downinner_lines()
win.draw_line(*down_lines[:4], size=3)
win.draw_line(*down_lines[4:], size=3)

# Штрихуем ротор
up_out_fill = coords.fill_up_rotor()      # Верхняя часть
for i in up_out_fill:
    win.draw_line(*i, size=1)
down_out_fill = coords.fill_down_rotor()  # Нижняя часть
for i in down_out_fill:
    win.draw_line(*i, size=1)
win.draw_polygon(*coords.get_block(), size=5)

# Рисуем подшипники
up_cir = coords.get_upcircle_params()
win.draw_circle(*up_cir, size=3)
down_cir = coords.get_downcircle_params()
win.draw_circle(*down_cir, size=3)

# Рисуем осевые линии
win.draw_punctir(center[0] - 3, center[1], center[0] + 4, center[1],
size=1)
win.draw_punctir(center[0], center[1] - 4, center[0], center[1] + 4,
size=1)
win.draw_punctir(center[0] - 20 - coords['B'] // 2, center[1], center[0] -
5, center[1], size=1)
win.draw_punctir(center[0] + 6, center[1], center[0] + 20 + coords['B'] //
2, center[1], size=1)
win.draw_punctir(center[0], center[1] - coords['D'] // 2 - coords['r'],
center[0], center[1] + coords['D'] // 2 + coords['r'])
win.draw_punctir(center[0] - 0.5 * coords['Dw'] - 10, center[1] - 0.5 *
coords['d1'],
center[0] + 0.5 * coords['Dw'] + 10, center[1] - 0.5 *
coords['d1'])
win.draw_punctir(center[0] - 0.5 * coords['Dw'] - 10, center[1] + 0.5 *
coords['d1'],

```

```

        center[0] + 0.5 * coords['Dw'] + 10, center[1] + 0.5 *
coords['dl'])

    # Добавляем 3d чертеж
    win.draw_3d(405, 5, im)
    return inner

def get_window(win, data: DrawData, im):
    win.customize('Расчет подшипника на долговечность', (1200, 600))
    # Поля для черчения
    win.add_label_field('Начертить подшипник', (15, 10), 10)
    win.add_data_field('d', 'd', 'мм', (30, 50), 30)
    win.add_data_field('D', 'D', 'мм', (30, 80), 30)
    win.add_data_field('B', 'B', 'мм', (30, 110), 30)
    win.add_data_field('r', 'r', 'мм', (30, 140), 30)

    win.add_button('Значения по умолчанию', win.set_data, (10, 350), 20)
    win.add_button('Начертить подшипник', create_draw(win, data, im), (10, 380),
20)

    # Поля для расчета
    win.add_label_field('Рассчитать подшипник', (200, 10), 10)
    win.add_calc_field('C', 'C', ' ', 'кН', (220, 50), 30)
    win.add_calc_field('p', 'p', ' ', (220, 80), 30)
    win.add_calc_field('Fr', 'Fr', ' ', 'кН', (220, 110), 30)
    win.add_calc_field('Fa', 'Fa', ' ', 'кН', (220, 140), 30)
    win.add_calc_field('X', 'X', ' ', (220, 170), 30)
    win.add_calc_field('Y', 'Y', ' ', (220, 200), 30)
    win.add_calc_field('V', 'V', ' ', (220, 230), 30)
    win.add_calc_field('k_b', 'Kb', ' ', (220, 260), 30)
    win.add_calc_field('k_t', 'KT', ' ', (220, 290), 30)

    win.add_button('Значения по умолчанию', win.set_calc, (200, 350), 20)
    win.add_button('Рассчитать подшипник', win.get_calc, (200, 380), 20)
    win.add_answer_field('P', 'P', ' ', 'кН', (220, 410), 30)
    win.add_answer_field('L', 'L', ' ', 'млн об.', (220, 440), 30)

```

Код *calculate.py*

```

__all__ = ['DrawData', 'DrawCoords', 'CalcData']

class DrawData:
    """Класс для расчета недостающих параметров подшипника"""

```

```

def __init__(self, data: dict) -> None:
    """Инициализатор данных класса
    :param data: dict - Словарь с имеющимися данными о подписнике"""
    self.base_data = data
    self.data = dict.fromkeys(self.base_data, 0)

def __mul__(self, other):
    """Метод для пропорционального увеличения параметров подписника"""
    for i in self.data:
        self.data[i] *= other

def __getitem__(self, item):
    return self.data[item]

def __setitem__(self, key, value):
    self.data[key] = float(value)

class CalcData:
    """Класс для расчета долговечности подписника"""
    def __init__(self, data: dict) -> None:
        self.base_data = data
        self.data = dict.fromkeys(self.base_data, 0)

    def get_P(self):
        """Метод для расчета динамической нагрузки"""
        X = float(self.data.get('X'))
        V = float(self.data.get('V'))
        Fr = float(self.data.get('Fr'))
        Y = float(self.data.get('Y'))
        Fa = float(self.data.get('Fa'))
        Kb = float(self.data.get('k_t'))
        KT = float(self.data.get('k_t'))
        return round((X * V * Fr + Y * Fa) * Kb * KT, 4)

    def get_L(self):
        """Метод для расчета долговечности"""
        C = float(self.data.get('C'))
        p = float(self.data.get('p'))
        P = float(self.get_P())
        return round((C / P) ** p, 6)

    def __getitem__(self, item):

```



```

        return self.data[item]

    def __setitem__(self, key, value):
        self.data[key] = value

class DrawCoords:
    """Класс для расчета координат ключевых точек чертежа"""
    def __init__(self, data: DrawData, field_coords: tuple = (5, 5), field_size:
tuple = (390, 590)) -> None:
        """Инициализируем размеры подшипника и поля для чертежа"""
        self.data = data
        self.field_coords = field_coords
        self.field_size = field_size

    def add_params(self):
        """Метод для расчета дополнительных параметров подшипника"""
        self['S'] = 0.15 * (self.data['D'] - self.data['d'])
        self['Dw'] = 0.32 * (self.data['D'] - self.data['d'])
        self['d1'] = 0.5 * (self.data['D'] + self.data['d'])

    def resize_data(self):
        """Метод для пропорционального увеличения параметров data"""
        k = min((self.field_size[0] - 20) // float(self.data['B']),
(self.field_size[1] - 50) // float(self.data['D']))
        self.data * k

    def get_center(self):
        """Метод для получения координат x и y центра чертежа"""
        x = self.field_coords[0] + self.field_size[0] // 2
        y = self.field_coords[1] + self.field_size[1] // 2
        return x, y

    def get_inner_cilinder(self):
        """Метод для получения координат внутренней полости подшипника"""
        cent = self.get_center()
        x1 = cent[0] - 0.5 * self['B'] + self['r']           # Левый верхний
        y1 = cent[1] - self['d'] // 2
        x2 = cent[0] + 0.5 * self['B'] - self['r']           # Правый верхний
        y2 = y1
        x3 = cent[0] + 0.5 * self['B'] - self['r']           # Правый нижний
        y3 = cent[1] + (self['d'] - self['d'] // 2)
        x4 = x1                                               # Левый нижний

```

```

        y4 = y3
        return x1, y1, x2, y2, x3, y3, x4, y4

    def get_big_cilinder(self):
        """Метод для получения координат внутреннего цилиндра, условного
        статора"""
        cent = self.get_center()
        x1 = cent[0] - 0.5 * self['B']           # Левый верхний
        y1 = cent[1] - self['d'] // 2 - self['S']
        x2 = cent[0] + 0.5 * self['B']           # Правый верхний
        y2 = y1
        x3 = cent[0] + 0.5 * self['B']           # Правый нижний
        y3 = cent[1] + (self['d'] - self['d'] // 2) + self['S']
        x4 = x1                                  # Левый нижний
        y4 = y3
        return x1, y1, x2, y2, x3, y3, x4, y4

    def get_outer_polygon(self):
        """Метод для получения координат внешнего контура чертежа"""
        cent = self.get_center()
        x1 = cent[0] - 0.5 * self['B'] + self['r']
        y1 = cent[1] - self['D'] // 2
        x2 = cent[0] + 0.5 * self['B'] - self['r']
        y2 = y1
        x3 = cent[0] + 0.5 * self['B']
        y3 = y1 + self['r']
        x4 = x3
        y4 = cent[1] + self['D'] // 2 - self['r']
        x5 = x2
        y5 = cent[1] + self['D'] // 2
        x6 = x1
        y6 = y5
        x7 = cent[0] - 0.5 * self['B']
        y7 = y4
        x8 = x7
        y8 = y3
        return x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7, x8, y8

    def get_upinner_lines(self):
        """Метод для получения координат внутренних линий верхней части условного
        ротора"""
        cent = self.get_center()
        if self['S'] >= self['r']:

```

```

        x1 = cent[0] - 0.5 * self['B']
    else:
        x1 = cent[0] - 0.5 * self['B'] + (self['r'] - self['S'])
    y1 = cent[1] - self['D'] // 2 + self['S']
    x2 = cent[0]
    y2 = y1
    if 0.5 * self['S'] >= self['r']:
        x3 = cent[0] + 0.5 * self['B']
    else:
        x3 = cent[0] + 0.5 * self['B'] - (self['r'] - 0.5 * self['S'])
    y3 = cent[1] - self['D'] // 2 + 0.5 * self['S']
    x4 = cent[0] + 0.15 * self['B']
    y4 = cent[1] - 0.5 * self['d1'] - (self['Dw'] ** 2 / 4 - (0.15 *
self['B']) ** 2) ** 0.5
    return x1, y1, x2, y2, x3, y3, x4, y4

def get_downinner_lines(self):
    """Метод для получения координат внутренних линий нижней части условного
ротора"""
    cent = self.get_center()
    if self['S'] >= self['r']:
        x1 = cent[0] - 0.5 * self['B']
    else:
        x1 = cent[0] - 0.5 * self['B'] + (self['r'] - self['S'])
    y1 = cent[1] + self['D'] // 2 - self['S']
    x2 = cent[0]
    y2 = y1
    if 0.5 * self['S'] >= self['r']:
        x3 = cent[0] + 0.5 * self['B']
    else:
        x3 = cent[0] + 0.5 * self['B'] - (self['r'] - 0.5 * self['S'])
    y3 = cent[1] + self['D'] // 2 - 0.5 * self['S']
    x4 = cent[0] + 0.15 * self['B']
    y4 = cent[1] + 0.5 * self['d1'] + (self['Dw'] ** 2 / 4 - (0.15 *
self['B']) ** 2) ** 0.5
    return x1, y1, x2, y2, x3, y3, x4, y4

def get_upcircle_params(self):
    """Метод для получения координат верхнего подшипника"""
    cent = self.get_center()
    x = cent[0]
    y = cent[1] - 0.5 * self['d1']
    r = 0.5 * self['Dw']

```

```

        return x - r, y - r, x + r, y + r

def get_downcircle_params(self):
    """Метод для получения координат нижнего подшипника"""
    cent = self.get_center()
    x = cent[0]
    y = cent[1] + 0.5 * self['d1']
    r = 0.5 * self['Dw']
    return x - r, y - r, x + r, y + r

def fill_up_stator(self, space=10):
    """Метод для получения координат штрихующих линий верхней части статора"""
    cent = self.get_center()
    left_up_x = cent[0] - 0.5 * self['B']
    left_up_y = cent[1] - self['d'] // 2 - self['S']
    right_up_x = cent[0] + 0.5 * self['B']
    right_up_y = left_up_y
    ans = []
    for i in range(0, int(self['B'] + self['S'] - self['r']), space):
        if i > self['B']:
            x1 = right_up_x
            y1 = right_up_y + i - self['B']
        else:
            x1 = left_up_x + i
            y1 = left_up_y
        if self['S'] - self['r'] < i < self['S'] + self['r']:
            x2 = left_up_x + (i - (self['S'] - self['r'])) / 2
            y2 = left_up_y + self['S'] - self['r'] + (i - (self['S'] -
self['r'])) / 2
        elif i <= self['S'] - self['r']:
            x2 = left_up_x
            y2 = left_up_y + i
        else:
            x2 = left_up_x + i - self['S']
            y2 = left_up_y + self['S']
        ans.append((x1, y1, x2, y2))
    return ans

def fill_down_stator(self, space=10):
    """Метод для получения координат штрихующих линий нижней части статора"""
    cent = self.get_center()
    left_up_x = cent[0] - 0.5 * self['B']
    left_up_y = cent[1] + self['d'] // 2

```

```

right_up_x = cent[0] + 0.5 * self['B']
right_up_y = left_up_y
start = int(self['r']) if self['r'] - int(self['r']) < 0.5 else
int(self['r']) + 1
ans = []
for i in range(start, int(self['B'] + self['S']), space):
    if self['B'] - self['r'] < i < self['B'] + self['r']:
        x1 = right_up_x - self['r'] + (i - (self['B'] - self['r'])) / 2
        y1 = left_up_y + (i - (self['B'] - self['r'])) / 2
    elif i <= self['B'] - self['r']:
        x1 = left_up_x + i
        y1 = left_up_y
    else:
        x1 = right_up_x
        y1 = right_up_y + i - self['B']
    if i < self['S']:
        x2 = left_up_x
        y2 = left_up_y + i
    else:
        x2 = left_up_x + i - self['S']
        y2 = left_up_y + self['S']
    ans.append((x1, y1, x2, y2))
return ans

def fill_up_rotor(self, space=10):
    """Метод для получения координат штрихующих линий верхней части ротора"""
    cent = self.get_center()
    down_lines = self.get_upinner_lines()[4:]
    left_down_x = cent[0] - 0.5 * self['B']
    left_down_y = cent[1] - self['D'] // 2 + self['S']
    right_down_x = cent[0] + 0.5 * self['B']
    right_down_y = cent[1] - self['D'] // 2 + self['S']
    k = (down_lines[1] - down_lines[3]) / (down_lines[0] - down_lines[2])
    b = down_lines[3] - down_lines[2] * k
    ans = []
    for i in range(0, int(self['S'] + self['B'] - self['r']), space):
        if i <= self['S'] - self['r']:
            x1 = left_down_x
            y1 = left_down_y - i
        elif self['S'] - self['r'] < i < self['S'] + self['r']:
            x1 = left_down_x + (i - (self['S'] - self['r'])) / 2
            y1 = left_down_y - (self['S'] - self['r']) - (i - (self['S'] -
self['r'])) / 2

```

```

        else:
            x1 = left_down_x + i - self['S']
            y1 = left_down_y - self['S']
            if i < 0.65 * self['B']:
                x2 = left_down_x + i
                y2 = left_down_y
            elif 0.65 * self['B'] <= i <= self['B'] + 0.5 * self['S']:
                my_k = abs(k)
                my_b = left_down_y - down_lines[3]
                s = i - 0.65 * self['B']
                dx = (s - my_b) / (my_k + 1)
                add = int(dx) if dx - int(dx) < 0.5 else int(dx) + 1
                x2 = left_down_x + 0.65 * self['B'] + add
                y2 = k * x2 + b
            else:
                x2 = right_down_x
                y2 = right_down_y - 0.5 * self['S'] - (i - self['B'] - 0.5 *
self['S'])

            ans.append((x1, y1, x2, y2))

    return ans

def fill_down_rotor(self, space=10):
    """Метод для получения координат штрихующих линий нижней части ротора"""
    cent = self.get_center()
    up_lines = self.get_downinner_lines()[4:]
    left_down_x = cent[0] - 0.5 * self['B']
    left_down_y = cent[1] + self['D'] // 2
    right_down_x = cent[0] + 0.5 * self['B']
    right_down_y = cent[1] + self['D'] // 2
    k = (up_lines[1] - up_lines[3]) / (up_lines[0] - up_lines[2])
    b = up_lines[3] - (right_down_y - self['S'])
    start = int(self['r']) if self['r'] - int(self['r']) < 0.5 else
int(self['r']) + 1
    ans = []
    for i in range(start, int(0.5 * self['S'] + self['B']), space):
        dot = up_lines[2] - b / k
        if i <= self['S']: # Верхняя граница
            x1 = left_down_x
            y1 = left_down_y - i
        elif self['S'] < i <= dot - (left_down_x - self['S']) + self['S']:
            x1 = left_down_x - self['S'] + i
            y1 = left_down_y - self['S']
        else:

```

```

        # x = left_down_x + (i - self['S'])
        # x1 = (left_down_x + (i - self['S'])) + b / (1 - k)
        # y1 = left_down_y - 2 * self['S'] + k * x + 1.6 * b
        x1 = left_down_x - self['S'] + i
        y1 = left_down_y - self['S']

        if i < self['B'] - self['r']:
            # Нижняя граница
            x2 = left_down_x + i
            y2 = left_down_y
        elif self['B'] - self['r'] <= i <= self['B'] + self['r']:
            x2 = left_down_x + (self['B'] - self['r']) + (i - (self['B'] -
self['r'])) / 2
            y2 = left_down_y - (i - (self['B'] - self['r'])) / 2
        else:
            x2 = right_down_x
            y2 = right_down_y - 0.5 * self['S'] - (i - self['B'] - 0.5 *
self['S'])

        ans.append((x1, y1, x2, y2))

    return ans

def get_block(self):
    cent = self.get_center()
    up_lines = self.get_downinner_lines()[4:]
    side1 = (up_lines[0] - 2, up_lines[1] - 2, up_lines[2], up_lines[3] - 2)
    y = cent[1] + self['d'] // 2 + self['S']
    side2 = (cent[0], y + 2, cent[0] + 0.5 * self['B'] - 2, y + 2)
    side3 = (cent[0], y + 2, cent[0] + 0.5 * self['B'] - 2, y + 2,
            up_lines[0] - 2, cent[1] + self['D'] // 2 - self['r'])
    return side1 + side2 if self['r'] <= 0.5 * self['S'] else side1 + side3

def __getitem__(self, item):
    return float(self.data[item])

def __setitem__(self, key, value):
    self.data[key] = float(value)

```

Код *core.py*

```

__all__ = ['CustomWin', 'LabelWin', 'ButtonWin', 'DrawWin', 'get_image']

import tkinter as tk

```

```

class CoreWin:
    """Ядро главного окна. Подтягивает необходимые объекты из tkinter"""
    def __init__(self) -> None:
        self.main_win = tk.Tk()

    def __call__(self) -> None:
        self.main_win.mainloop()

class CustomWin(CoreWin):
    """Класс-кастомизатор главного окна. Добавляет название, размеры и иконку"""
    def customize(self, title: str, size) -> None:
        self.main_win.title(title)
        self.main_win.geometry(f'{size[0]}x{size[1]}+200+100')
        self.main_win.resizable(False, False)
        icon = tk.PhotoImage(file='images/icon.png')
        self.main_win.iconphoto(False, icon)

class LabelWin(CoreWin):
    """Класс для добавления в окно полей ввода и информационных подписей"""
    def __init__(self) -> None:
        super().__init__()
        self.entries_draw = {}
        self.entries_calc = {}
        self.answers = {}

    def add_data_field(self, entry_name: str, label_start: str, label_end: str,
position: tuple, dx: int) -> None:
        """Метод для добавления в окно поля вида  $F = |0.034|$ , мм"""
        tk.Label(self.main_win, text=f'{label_start} =').place(x=position[0],
y=position[1])
        entry = tk.Entry(self.main_win, width=6)
        entry.place(x=position[0] + dx, y=position[1])
        self.entries_draw[entry_name] = entry
        tk.Label(self.main_win, text=f', {label_end}').place(x=position[0] + dx *
2.5, y=position[1])

    def add_calc_field(self, entry_name: str, label_start: str, label_end: str,
position: tuple, dx: int) -> None:
        """Метод для добавления в окно поля вида  $F = |0.034|$ , мм"""
        tk.Label(self.main_win, text=f'{label_start} =').place(x=position[0],
y=position[1])

```



```

        entry = tk.Entry(self.main_win, width=6)
        entry.place(x=position[0] + dx, y=position[1])
        self.entries_calc[entry_name] = entry
        tk.Label(self.main_win, text=f'{label_end}').place(x=position[0] + dx *
2.5, y=position[1])

    def add_answer_field(self, entry_name: str, label_start: str, label_end: str,
position: tuple, dx: int) -> None:
        """Метод для добавления в окно поля вида F = |0.034 |, мм"""
        tk.Label(self.main_win, text=f'{label_start} =').place(x=position[0],
y=position[1])
        entry = tk.Entry(self.main_win, width=9)
        entry.place(x=position[0] + dx, y=position[1])
        self.answers[entry_name] = entry
        tk.Label(self.main_win, text=f'{label_end}').place(x=position[0] + dx * 3,
y=position[1])

    def add_label_field(self, label_text: str, position: tuple, size: int) ->
None:
        """Метод для добавления в окно поля вида 'Начертить подшипник'"""
        tk.Label(self.main_win, text=f'{label_text}', font=('Arial',
size)).place(x=position[0], y=position[1])

class ButtonWin(CoreWin):
    """Класс для добавления в окно именованных кнопок"""
    def __init__(self, data, calc):
        super().__init__()
        self.data = data
        self.calc = calc

    def add_button(self, but_text: str, but_func, position: tuple, size: int) ->
None:
        """Метод для добавления в окно кнопки вида |Начертить подшипник|"""
        tk.Button(self.main_win, text=f'{but_text}', command=but_func,
width=size).place(x=position[0], y=position[1])

    def set_data(self) -> None:
        """Метод для подстановки в окна значений по умолчанию из переменной data.
Перед постановкой данные удаляются"""
        for i in self.entries_draw:
            self.entries_draw.get(i).delete(0, tk.END)
            value = self.data.base_data.get(i)

```

```

        self.entries_draw[i].insert(-1, value)
        self.data[i] = value

    def set_calc(self) -> None:
        """Метод для подстановки в окна значений по умолчанию из переменной calc.
        Перед постановкой данные удаляются"""
        for i in self.entries_calc:
            self.entries_calc.get(i).delete(0, tk.END)
            value = self.calc.base_data.get(i)
            self.entries_calc[i].insert(-1, value)
            self.calc[i] = value

    def get_calc(self) -> None:
        """Метод для подстановки в окна значений L и P по расчетам класса
        CalcData"""
        for i in self.entries_calc:
            val = self.entries_calc.get(i).get()
            self.calc[i] = val
        P = self.calc.get_P()
        self.answers.get('P').delete(0, tk.END)
        self.answers.get('P').insert(-1, P)
        L = self.calc.get_L()
        self.answers.get('L').delete(0, tk.END)
        self.answers.get('L').insert(-1, L)

class DrawWin(CoreWin):
    """Класс для рисования чертежа"""
    def __init__(self):
        super().__init__()
        self.canvas = tk.Canvas(bg="white", width=800, height=600)
        self.canvas.place(x=395, y=0)

    def draw_line(self, *args, size=1) -> None:
        """Метод для рисования ломаной линии толщины size по координатам args"""
        self.canvas.create_line(*args, width=size)

    def draw_rect(self, *args, size=1) -> None:
        """Метод для рисования прямоугольника толщины size по координатам args"""
        self.canvas.create_rectangle(*args, width=size)

    def draw_circle(self, *args, size=1) -> None:
        """Метод для рисования окружности толщины size по координатам args"""

```

```

        self.canvas.create_oval(*args, width=size, fill='white')

def clear_place(self, *args, size=1):
    """Метод для удаления чертежа"""
    self.canvas.create_rectangle(*args, width=size, fill='white')

def draw_polygon(self, *args, size=3, fill=True):
    """Метод для рисования окружности толщины size по координатам args"""
    self.canvas.create_polygon(*args, width=size, fill='white')

def draw_3d(self, x, y, image):
    self.canvas.create_image(x, y, anchor='nw', image=image)

def draw_punctir(self, *args, size=1) -> None:
    """Метод для рисования пунктирной линии толщины size по координатам
args"""
    self.canvas.create_line(*args, width=size, dash=5)

def get_image(path):
    return tk.PhotoImage(file=path, height=590, width=390)

```

ПОСТРОЕНИЕ 3D-МОДЕЛИ ПОДШИПНИКА В КОМПАС 3D

Построение фрагмента подшипника

На основе данных из Приложения рассчитаем и построим вспомогательные отрезки и вспомогательные оси:

$$B = 14$$

$$r = 1,5$$

$$D = 47$$

$$d = 20$$

$$S = 0,15 \cdot (D - d) = 0,15 \cdot (47 - 20) = 4,05$$

$$d1 = 0,5 \cdot (D + d) = 0,5 \cdot (47 + 20) = 33,5$$

$$Dw = 0,32 \cdot (D - d) = 0,32 \cdot (47 - 20) = 8,64$$

Вертикальные вспомогательные:

$$1. \quad d / 2 = 10$$

$$2. \quad d / 2 + r = 11,5$$

$$3. \quad d / 2 + S = 14,05$$

$$4. \quad d1 / 2 = 16,75$$

$$5. \quad D / 2 - S = 19,45$$

$$6. \quad D / 2 - S / 2 = 21,475$$

$$7. \quad D / 2 - r = 22$$

$$8. \quad D / 2 = 23,5$$

Горизонтальные вспомогательные:

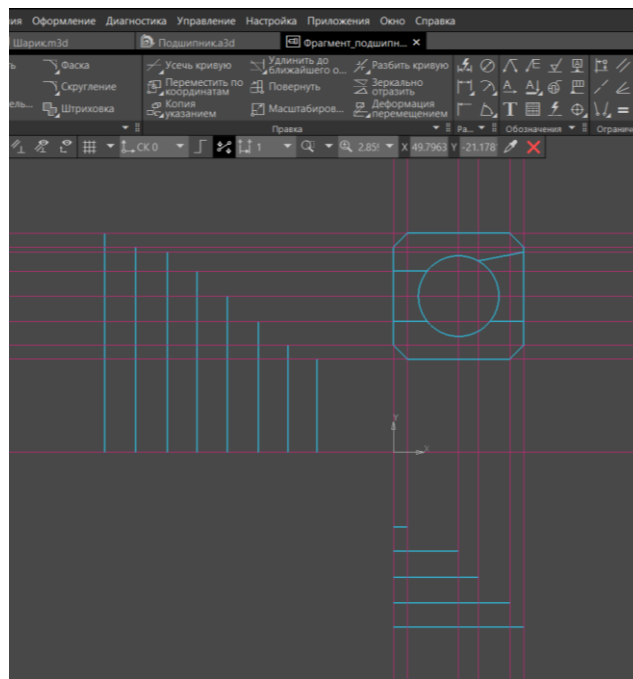
$$1. \quad r = 1,5$$

$$2. \quad B / 2 = 7$$

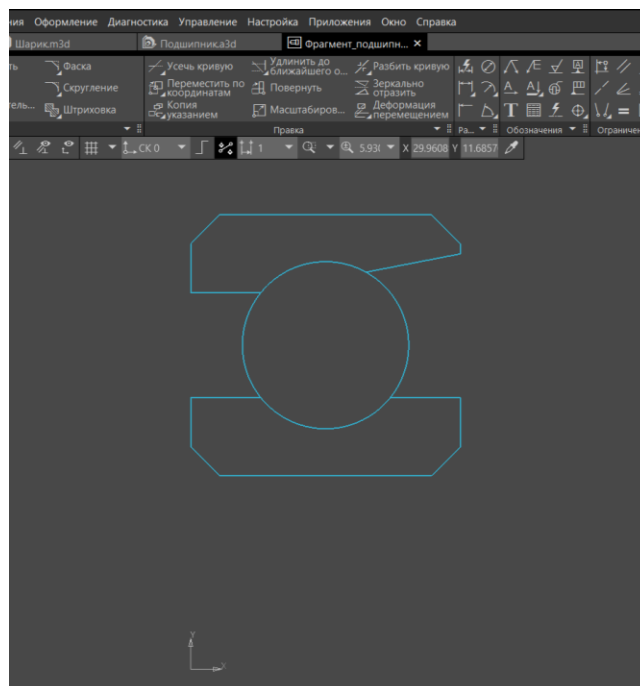
$$3. \quad 0,65 \cdot B = 9,1$$

$$4. \quad B - r = 12,5$$

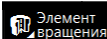
$$5. \quad B = 14$$



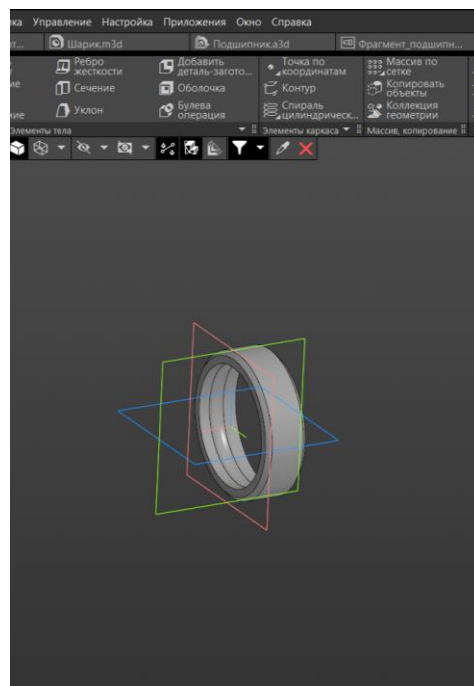
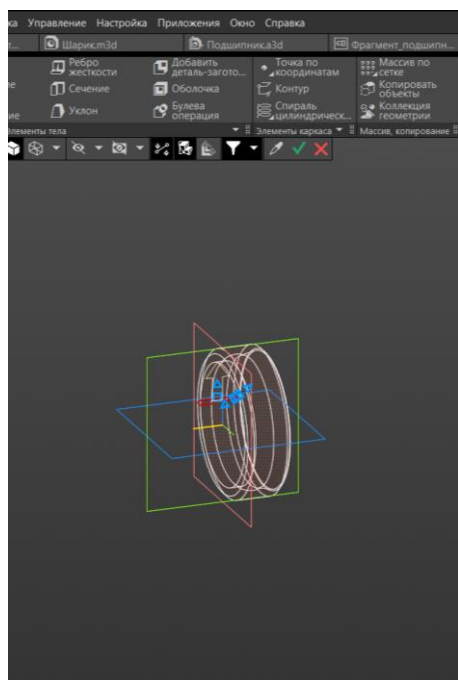
Уберем вспомогательные линии и отрезки:



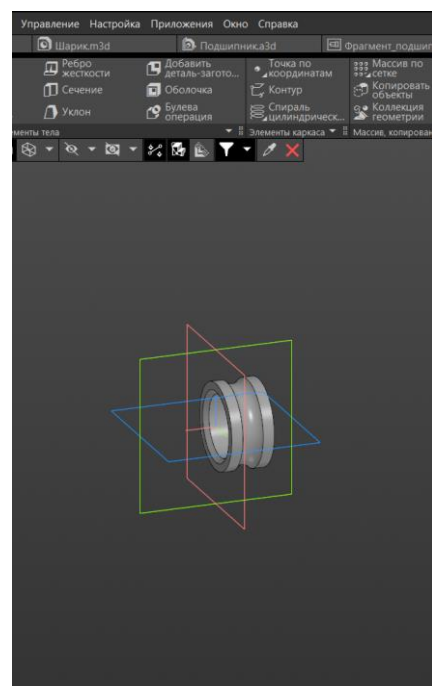
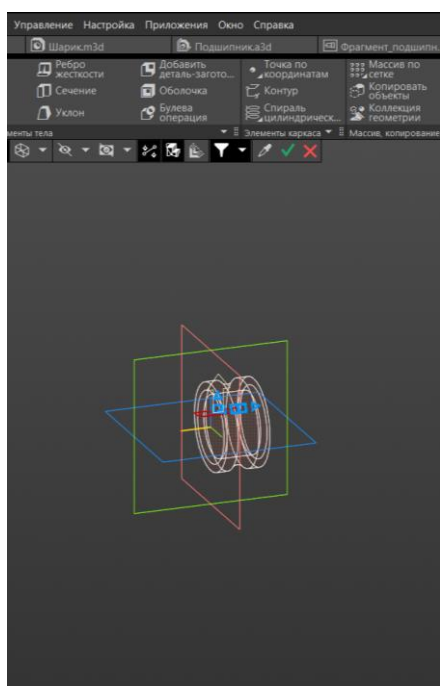
Построение деталей подшипника


Разобьем полученный фрагмент на 3 части (внешняя внутренняя, шарик) и построим на их основе детали с помощью элемента вращения :

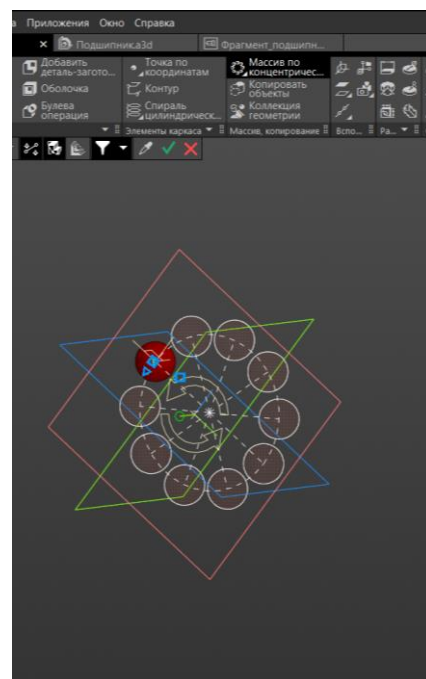
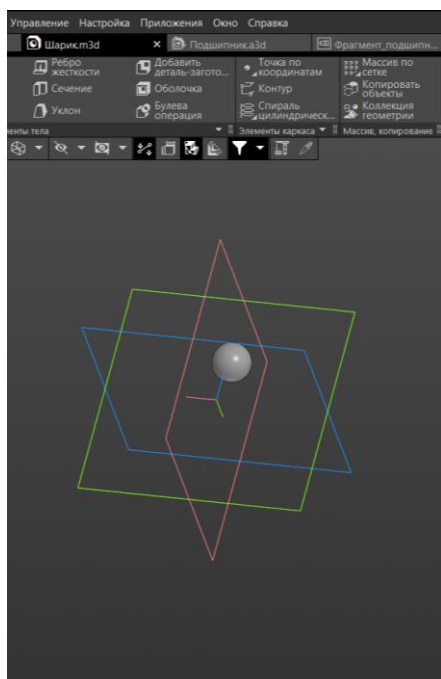
Внешняя часть



Внутренняя часть

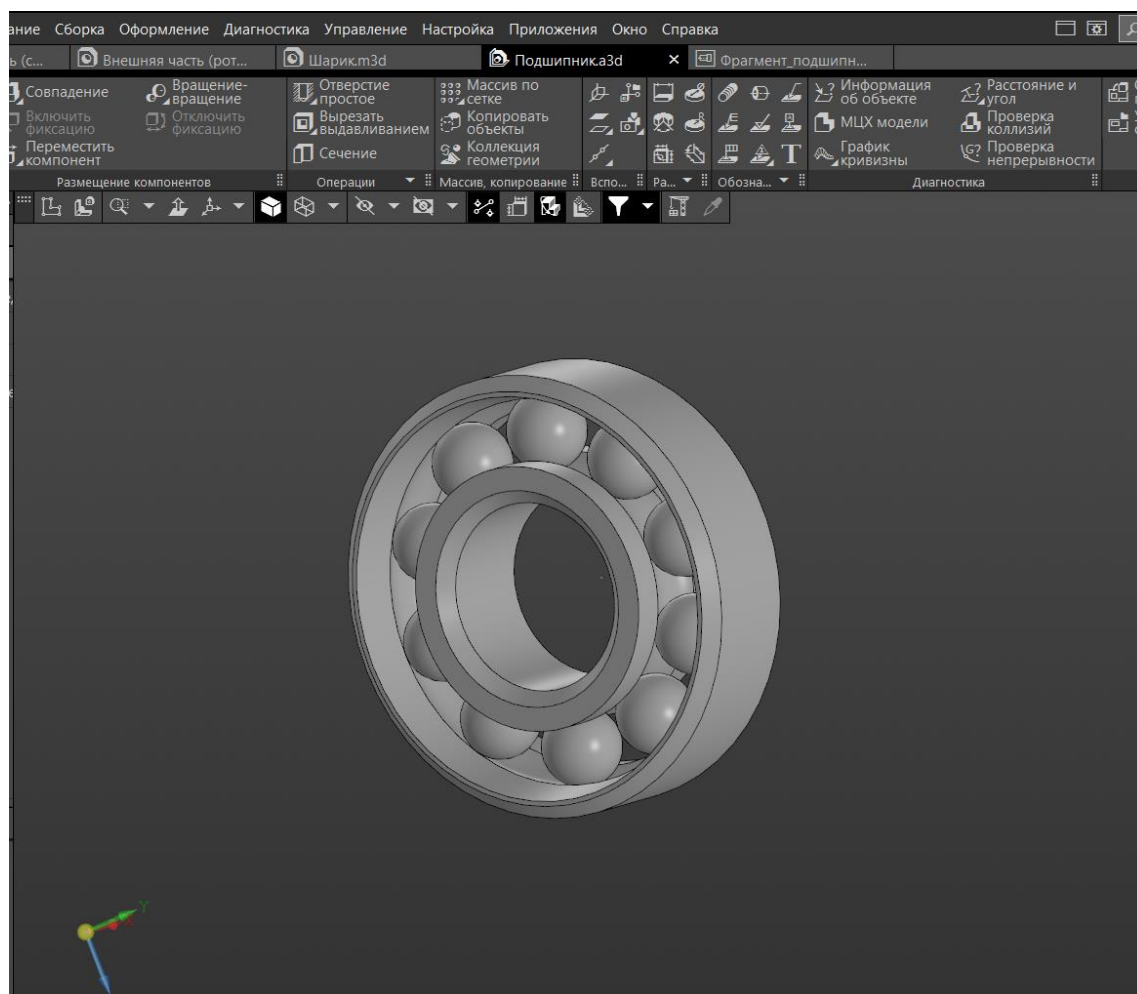


Шарики подшипника построим с помощью элемента вращения и массива по концентрической сетке  Массив по концентричес...:



Объединение деталей в сборку

Соберем из всех деталей итоговую модель (сборку) подшипника, зафиксируем изображение и добавим в программный модуль:



ТЕСТ ПРОГРАМНОГО МОДУЛЯ

Главное меню

Расчет подшипника на долговечность

Начертить подшипник	Рассчитать подшипник		
$d =$ <input type="text"/> , мм	$C =$ <input type="text"/> , кН		
$D =$ <input type="text"/> , мм	$p =$ <input type="text"/>		
$B =$ <input type="text"/> , мм	$F_r =$ <input type="text"/> , кН		
$r =$ <input type="text"/> , мм	$F_a =$ <input type="text"/> , кН		
	$X =$ <input type="text"/>		
	$Y =$ <input type="text"/>		
	$V =$ <input type="text"/>		
	$K_b =$ <input type="text"/>		
	$K_T =$ <input type="text"/>		
<input type="button" value="Значения по умолчанию"/>	<input type="button" value="Значения по умолчанию"/>		
<input type="button" value="Начертить подшипник"/>	<input type="button" value="Рассчитать подшипник"/>		
	$P =$ <input type="text"/> , кН		
	$L =$ <input type="text"/> , млн об.		

Подстановка значений по умолчанию

Расчет подшипника на долговечность

Начертить подшипник	Рассчитать подшипник		
$d =$ <input type="text" value="20"/> , мм	$C =$ <input type="text" value="15.7"/> , кН		
$D =$ <input type="text" value="47"/> , мм	$p =$ <input type="text" value="3"/>		
$B =$ <input type="text" value="14"/> , мм	$F_r =$ <input type="text" value="2.08"/> , кН		
$r =$ <input type="text" value="1.5"/> , мм	$F_a =$ <input type="text" value="3.142"/> , кН		
	$X =$ <input type="text" value="0.46"/>		
	$Y =$ <input type="text" value="1.421"/>		
	$V =$ <input type="text" value="1"/>		
	$K_b =$ <input type="text" value="1.3"/>		
	$K_T =$ <input type="text" value="1"/>		
<input type="button" value="Значения по умолчанию"/>	<input type="button" value="Значения по умолчанию"/>		
<input type="button" value="Начертить подшипник"/>	<input type="button" value="Рассчитать подшипник"/>		
	$P =$ <input type="text"/> , кН		
	$L =$ <input type="text"/> , млн об.		

Расчет долговечности подшипника

Расчет подшипника на долговечность

Начертить подшипник

d = 20, мм

D = 47, мм

B = 14, мм

r = 1.5, мм

Значения по умолчанию

Начертить подшипник

Рассчитать подшипник

C = 15.7, кН

p = 3

Fr = 2.08, кН

Fa = 3.142, кН

X = 0.46

Y = 1.421

V = 1

Kb = 1.3

KT = 1

Значения по умолчанию

Рассчитать подшипник

P = 5.4216, кН

L = 24.283794, млн об.

Чертеж подшипника

Расчет подшипника на долговечность

Начертить подшипник

d = 20, мм

D = 47, мм

B = 14, мм

r = 1.5, мм

Значения по умолчанию

Начертить подшипник

Рассчитать подшипник

C = 15.7, кН

p = 3

Fr = 2.08, кН

Fa = 3.142, кН

X = 0.46

Y = 1.421

V = 1

Kb = 1.3

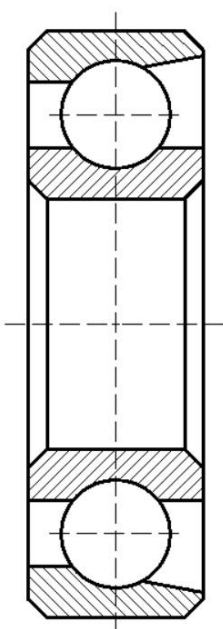
KT = 1

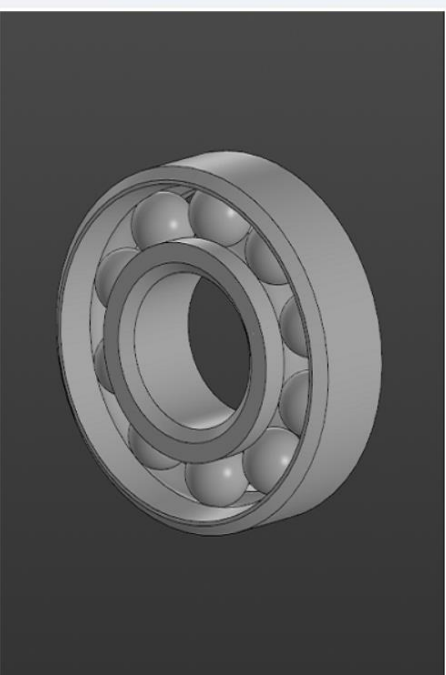
Значения по умолчанию

Рассчитать подшипник

P = 5.4216, кН

L = 24.283794, млн об.





33

ЗАКЛЮЧЕНИЕ

Данная курсовая работа была разработана на языке Python в интерактивной среде разработки PyCharm с использованием стандартной библиотеки tkinter, а также с помощью системы трехмерного проектирования КОМПАС 3D. Благодаря легко читаемому синтаксису языка и простому, но мощному функционалу всех инструментов курсовая работа разработана на высоком уровне и в короткие сроки.

В ходе курсовой работы были достигнуты следующие результаты:

Были освоены навыки построения 3D-моделей в системе КОМПАС 3D, отточены умения программирования на языке Python, а также был разработан программный модуль, который позволяет производить автоматизированный расчет на долговечность и построение подшипника качения.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

1. ГОСТ 831-75 Подшипники шариковые радиально-упорные однорядные. Типы и основные размеры (с Изменением N 1) // Электронный фонд правовых и нормативно-технических документов URL: <https://docs.cntd.ru/document/1200012724> (дата обращения: 16.05.2023).
2. Лутц М. Изучаем Python. - 5-е изд. - Диалектика, 2019. - 832 с.
3. Д. Бейдер Чистый Python. Тонкости программирования для профи. - СПб.: Питер, 2019. - 288 с.
4. tkinter — Python interface to Tcl/Tk // Python Documetation URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 16.05.2023).
5. Герасимов А.А. Самоучитель КОМПАС-3D v19. - СПб.: БХВ-Петербург, 2021. - 624 с.
6. Чагина А. В., Большаков В. П. 3D-моделирование в КОМПАС-3D версий v17 и выше. Учебное пособие для вузов. - СПб.: Питер, 2021. - 256 с.
7. Ознакомление и построение подшипника качения в программе Компас 3D v18 // YouTube URL: <https://www.youtube.com/watch?v=o5ha4DnIYO4> (дата обращения: 16.05.2023).

ПРИЛОЖЕНИЯ

Таблица 1

Исходные данные для расчёта подшипников на долговечность

Тип, серия подшипника	Радиально-упорный шариковый однорядный легкой серии № 36204
Частота вращения вала, об/мин, n	1444,5
Коэффициент безопасности, K_b	1,3
Температура подшипникового узла, град.	100
Температурный коэффициент, K_T	1
Режим работы	Средний равновероятный
Коэффициент эквивалентности,	0,25
Радиальная нагрузка кН, F_r	2,08
Осевая нагрузка, кН, F_a	3,142
Динамическая грузоподъемность, кН, C	15,7
Статическая грузоподъемность, кН, C_0	8,31
Параметр осевого нагружения, e	0,21
Коэффициент радиальной нагрузки, X	0,46
Коэффициент осевой нагрузки, Y	1,421

Таблица 2

Подшипники шариковые радиально-упорные однорядные легкой серии (ГОСТ 831-75)

Обозначение	36204
d	20
D	47
B	14
r	1,5
C	15,7
C0	8,31