|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Подъёмно-транспортные системы (РК4)

***ОТЧЕТ***

***ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

*Оптимальное проектирование механизма передвижения козлового крана с помощью метода динамического программирования.*

Студент \_\_\_\_\_РК4-81\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Клёнов А.А.\_\_\_\_\_

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Пармузин Д.Б.\_\_\_

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Пармузин Д.Б.\_\_\_

(Подпись, дата) (И.О.Фамилия)

*2020 г,*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой РК4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Л. Тропин

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме Оптимальное проектирование механизма передвижения козлового крана с помощью метода динамического программирования\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_РК4-81\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_Клёнов\_\_\_\_Александр\_\_\_\_Андреевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_ нед., 50% к \_\_\_ нед., 75% к \_\_ нед., 100% к \_\_\_ нед.

***Техническое задание*** доказать эффективность метода многокритериальной оптимизации на примере проектирования механизма передвижения козлового крана\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_\_22\_\_\_листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Пармузин Д.Б.\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Клёнов А.А.\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Содержание**

Проблема …………………………………………………………………………4

Актуальность …………………………………………………………………….4

Цель исследования ………………………………………………………………4

Материал и методы исследования ……………………………………………...4

Исследование …………………………………………………………………….7

Программа: Часть 1 …………………………………………………………….10

Программа: Часть 2 …………………………………………………………….12

Программа: Часть 3 …………………………………………………………….16

Программа: Часть 4 …………………………………………………………….17

Программа: Часть 5 …………………………………………………………….19

Программа: Часть 6 …………………………………………………………….20

Результат выполнения программы …………………………………………....21

Выводы ………………………………………………………………………….22

Список используемой литературы …………………………………………....22

*Проблема*: сложность выбора оптимальной конструкции и компоновки при проектировании механизма передвижения козлового крана

*Актуальность*: в данной работе я рассматриваю метод многокритериального оптимального проектирования, который позволяет быстро и качественно разработать механизм передвижения крана с учётом всех требований. Перспективность метода заключается в том, что он универсален и может быть использован при проектировании любых машин и механизмов для снижения затрат на изготовление и эксплуатацию.

*Цель исследования*: показать эффективность метода на примере проектирования механизма передвижения козлового крана

*Материал и методы исследования:*

К крановым механизмам зачастую предъявляется широкий спектр противоречивых требований. Поэтому конструктору довольно сложно выделить среди множества вариантов решений тот, который бы являлся наилучшим. Здесь на помощь приходит один из методов многокритериальной оптимизации, основанный на применении принципа Парето [4,4] и позволяющий оценивать различные решения по нескольким параметрам. Метод называется «динамическое программирование».

Суть принципа Парето заключается в том, что среди всех возможных решений механизма Х существует некоторое множество решений , между которыми лицом, принимающим решения (ЛПР), не может быть выражено чёткое предпочтение. Данное множество называется множеством парето-оптимальных решений или множеством Парето. Любое парето-оптимальное решение превосходит другие парето-оптимальные решения хотя бы по одному из критериев оптимальности, но при этом уступает любому из них также хотя бы по одному другому критерию. В то же время каждое парето-оптимальное решение превосходит любое прочее решение, не вошедшее во множество Парето, по всем критериям. Формирование множества Парето является первой задачей, которую необходимо решить при оптимальном проектировании механизма передвижения крана.

Для сравнения отдельных решений задачи оптимального проектирования механизма передвижения используется векторный критерий оценки качества решения, включающий в себя параметры, оказывающие непосредственное влияние на технико-экономические свойства машины. В состав векторного критерия для механизма передвижения должны быть включены следующие величины: себестоимость механизма С, как параметр определяющий капитальные затраты на изготовление; масса механизма М, как параметр, определяющий эксплуатационные затраты и стоимость монтажа механизма; КПД механизма η, также оказывающий влияние на эксплуатационные затраты. Требования, предъявляемые к конструкции механизма передвижения и имеющие обязательный характер, задаются в виде ограничений, например, условие обеспечения необходимой прочности узлов механизма.

На предварительной стадии ЛПР должно определить наиболее перспективные компоновки механизма передвижения и кинематические схемы привода, отличающиеся типом применяемого редуктора или мотор-редуктора, наличием или отсутствием открытой зубчатой передачи, а также способом соединения вала колеса с тихоходным валом редуктора или мотор-редуктора. Механизм передвижения при этом представляется как сложная техническая система, состоящая из отдельных модулей: 1–модуль электродвигателей и системы управления; 2–модуль приводных и неприводных колёсных установок и кранового рельсового пути; 3–модуль трансмиссии (включает в себя редуктор, соединительные муфты и трансмиссионные валы); 4–модуль открытой зубчатой передачи (при её наличии); 5–модуль тормоза (или тормозов); 6–объединённый модуль всех металлических конструкций балансиров (при их наличии). Состав модулей может меняться в зависимости от рассматриваемой кинематической схемы привода.

Формирование множества парето-оптимальных решений для механизма передвижения следует проводить отдельно для каждого сочетания выбранных ЛПР компоновок механизма передвижения и кинематических схем привода, при этом задача сводится к выбору оптимальных решений для каждого из модулей. В качестве решений для отдельных модулей могут быть рассмотрены как серийно выпускаемые покупные изделия (электродвигатели, колёсные установки, редукторы, мотор-редукторы, соединительные муфты и тормоза), так и оригинальные конструкции. При этом должны быть составлены базы данных по всем основным элементам механизма, в состав которой входят все серийно выпускаемые и оригинальные конструкции с указанием их массы, себестоимости, КПД и других параметров.

На каждый из отдельных модулей накладывается своя система ограничений: по прочности, надёжности, технологичности, условиям сборки и др. На всю систему механизма передвижения целиком также накладывается ряд ограничений, прежде всего ограничение по заданной техническим заданием скорости передвижения крана. На скорость передвижения крана оказывают влияние: частота вращения вала выбранного электродвигателя , диаметр применяемого ходового колеса , передаточное отношение редуктора uр и открытой зубчатой передачи uоп. При этом изменение одного из перечисленных параметров неизбежно должно сопровождаться изменением какого-либо другого с тем, чтобы в итоге была обеспечена необходимая скорость передвижения крана v. Таким образом, в процессе поиска оптимального решения помимо прочего необходимо определить оптимальные значения вышеперечисленных взаимосвязанных параметров.

Для того чтобы можно было легко менять распределение указанных выше параметров, определяющих скорость передвижения крана, но имеющих различную размерность, необходимо ввести новый безразмерный параметр υ, называемый степенью редукции.

Требуемая степень редукции для всей системы равна:

 (1)

где nдвmax – максимальная частота вращения вала электродвигателя для всех имеющихся в базе данных электродвигателей; Dkmax – максимальный диаметр ходового колеса из всех имеющихся в базе; upmin , uопmin – минимальные имеющиеся в базе данных значения передаточного отношения редукторов и открытой зубчатой передачи.

Для отдельных решений каждого из модулей степень редукции определится следующим образом:

Для 1-го модуля:  (2)

Для 2-го модуля:  (3)

Для 3-го модуля:  (4)

Для 4-го модуля:  (5)

где nдвf , Dkf , upf , uопf – соответственно обороты двигателя, диаметр ходового колеса, передаточные числа редуктора и открытой зубчатой передачи для некоторого f-решения, f – номер решения в базе данных для соответствующего модуля.

Таким образом, условие обеспечения необходимой скорости движения может быть записано в следующем виде:

 (6)

где υx – степень редукции для некоторого рассматриваемого х-го решения.

 (7)

Предварительно для каждого из модулей должны быть сформированы частные множества парето-оптимальных решений. В дальнейшем на основе метода динамического программирования, рассматривая сочетания различных частных парето-оптимальных решений для первого и второго модуля, будет составлено общее множество парето-оптимальных решений для первых двух модулей. Далее, рассматривая сочетания полученных общих парето-оптимальных решений для двух первых модулей с частными парето-оптимальными решениями для третьего модуля, формируется множество общих парето-оптимальных решений для первых трёх модулей. Повторив аналогичную процедуру для оставшихся модулей, на последнем шаге будет получено множество парето-оптимальных решений для всей системы механизма для рассматриваемого варианта сочетания компоновки механизма с кинематической схемой привода.

*Исследование:*

Возьмём за основу для проектирования кинематическую схему механизма передвижения крана с раздельным приводом (рис. 1).

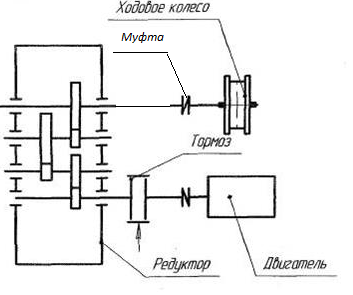


Рис. 1 – Кинематическая схема

Данный механизм не включает открытую зубчатую передачу и для него не будут предусмотрены балансиры, как это делается в случае многоколёсных систем. Это значительно упрощает задачу.

Для автоматизации проектирования я написал программу на языке Python. Программа демонстрирует алгоритм формирования множества парето-оптимальных решений на каждом шагу. Рассмотрим задачи, которые должна решать программа:

* Загрузить базы данных в формате Excel для каждого из рассматриваемых модулей.
* Запросить у пользователя ввод ограничений, накладываемых на каждый модуль в отдельности, а также требуемой скорости перемещения крана.
* Сформировать на основании введённых ограничений частные парето-оптимальные решения.
* Объединить все частные решения в одно множество, которое удовлетворяет заданной для крана скорости перемещения.
* Сократить множество до 5 значений, чтобы человек был в состоянии проанализировать каждый из вариантов в отдельности и выбрать один из них.

В качестве исходных данных программа имеет список, в котором содержатся относительные пути к файлам Excel. Все файлы Excel должны находится в той же директории (папке), что и исполняемый файл (с расширением .exe). Если это условие не выполняется, то программа не сможет найти и загрузить файлы и аварийно завершится.

Теперь рассмотрим содержимое баз данных:

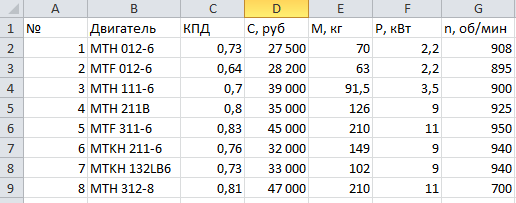


Рис. 2 – «Двигатели»

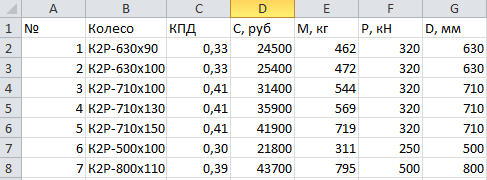


Рис. 3 – «Колёсные установки»

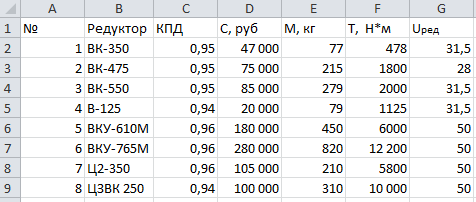


Рис. 4 – «Редукторы»

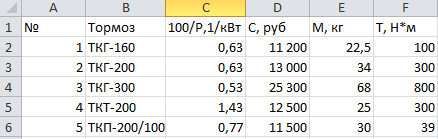


Рис. 5 – «Тормоза»

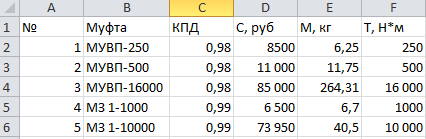


Рис. 6 – «Муфты»

Каждому изделию в базе данных сопоставлены 3 параметра, по которым ведётся оптимизация: КПД, себестоимость (С) и масса (М). В столбце «F» содержится параметр, по которому пользователь вводит ограничения (задаёт минимум и максимум параметра). Это позволяет отсеять те строки таблицы, которые не соответствуют результатам проектировочных расчётов. И наконец в столбце «G» располагается параметр, определяющий кинематические характеристики данного модуля. Этот столбец есть только у таблиц «Двигатели», «Колёсные установки» и «Редукторы», так как именно их сочетание определяет скорость передвижения крана. Остальные изделия не оказывают влияние на скорость.

Вводимые пользователем ограничения, основываются на предварительных проектировочных расчётах, получаемых для крана с заданными скоростью передвижения, грузоподъёмностью, пролётом и т.д.

Окно пользовательского ввода (рис. 7):

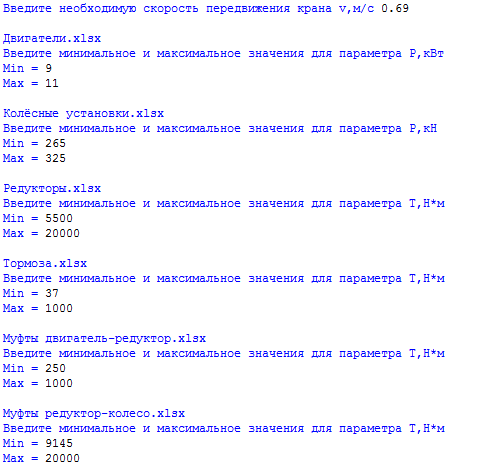
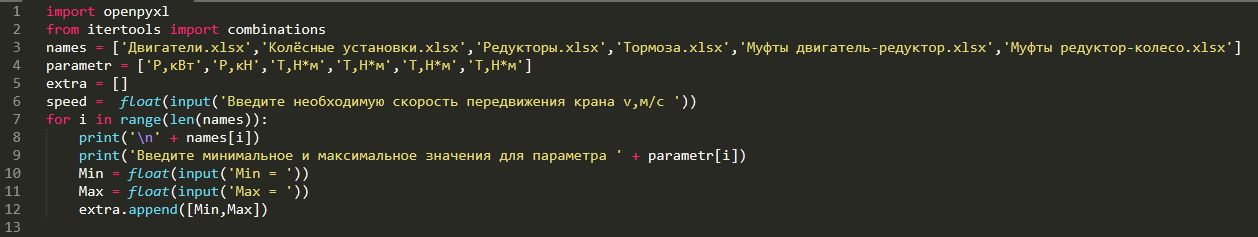


Рис. 7

Теперь рассмотрим код программы с необходимыми комментариями, чтобы продемонстрировать работу алгоритма оптимизации.

*Часть 1 – ввод данных пользователем и их сохранение в памяти программы*



В строках (1) и (2) я импортирую библиотеки, которые содержат нужные мне функции. Так *openpyxl* используется для работы с файлами Excel (загрузка, манипулирование строками и столбцами и т.д.). В строке (2) я импортирую не всю библиотеку, а только функцию *combinations* библиотеки *itertools.*

Строки (3) , (4) и (5) содержат особые структуры данных, которые называют *списками.* Содержимое списка записывают в квадратных скобках [] через запятую. Каждый элемент списка имеет свой индекс. Нумерация начинается с нуля. Чтобы обратиться к элементу списка, нужно написать имя переменной, в которой хранится список, а потом в квадратных скобках указать индекс элемента в списке. Пример: a = [1,2] . a[0] = 1, a[1] = 2.

В списке *names* находятся имена загружаемых файлов Excel (представлены строками, содержимое строки записывается в кавычках), в списке *parametr* – обозначения и размерности параметров, по которым вводятся ограничения, а список *extra*, в котором будут храниться значения минимумов и максимумов элементов списка *parametr* – пока пуст.

В строке (6) определяется переменная *speed*. В ней хранится значение, введённое пользователем с помощью функции *input.* Так как эта функция возвращает строку, а нам нужно число, то мы преобразуем результат функцией *float.* Пример: до float было ‘5.5’ – после стало 5.5. С 5.5 можно проводить математические операции (сложение, вычитание, умножение), а с ‘5.5’ нельзя.

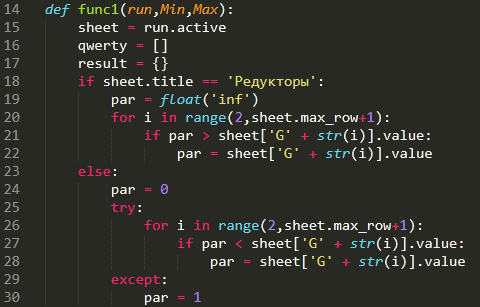
В строке (7) мы видим конструкцию цикла, различаемую по ключевому слову *for.* Переменная *i* будет определять количество итераций цикла. Функция *range* возвращает упорядоченный список чисел. Например, range(1,6) вернёт 1,2,3,4,5. Последнее число, указанное в скобках, не включается в список. Если указывается в скобках одно число, то перечисление идёт с нуля. Функция *len* возвращает длину списка. Так список names включает 6 элементов. Значит, его длина равняется 6. Подытожим сказанное: цикл изменяет значение переменной i от 0 до 5, что позволяет её использовать в качестве индекса для элементов списка names и parametr.

Всё, что следует после двоеточия в цикле, располагается с отступом и называется *телом цикла.* Как только все команды (строки) в теле закончатся, к переменной i прибавляется 1 и всё начинается сначала. Так будет продолжаться до тех пор, пока i не станет равной 6, то есть выйдет за пределы списка range.

Функция *print* строк (8) и (9) возвращает всё то, что будет высвечиваться на экране. Так на рисунке 7 синим цветом показан результат действия функции print, а чёрным цветом – функции input.

В строке (12) мы используем метод *append* для списка extra, который добавляет в его конец указанный элемент. В данном случае мы добавляем список [Min,Max]. Min и Max были определены в строках (10) и (11). Таким образом, в памяти программы сохраняются данные, введённые пользователем.

*Часть 2 – функция формирования частных парето-оптимальных решений*



Если вы встречаетесь с задачей, для выполнения которой требуется многократно совершить одни и те же действия, то логично использовать для этого *функцию.* Например, действия  *дойти до магазина, набрать корзину с продуктами, расплатиться с кассиром,* можно заменить словосочетанием *сходить в магазин.* И если вам надо совершить покупки в различных магазинах, вы можете просто сказать «сходить в магазин А», «сходить в магазин Б» и т.д.

В программировании, чтобы не прописывать один и тот же код для повторяющихся задач, например, вывода текст на экран, так же используют функции. Мы уже знакомы с функциями print, input, range, float и др. Они являются стандартными и выделяются в редакторе кода голубым цветом. Можно написать и свою функцию. Она должна начинаться с ключевого слова *def* (строка 14), затем следует имя функции (в данном случае *func1*), а в скобках указывают принимаемые функцией параметры. У нас 3 параметра: *run* – переменная, в которую мы поместим загруженные данные с файла Excel; Min и Max – значения, хранящиеся в списке extra.

Всё, что следует после двоеточия в функции, располагается с отступом и называется *телом функции.* Важно понимать, что все переменные, которые были объявлены в функции, существуют только внутри неё. После завершения функции, они стираются из памяти.

В строке (15) мы создаём переменную *sheet*, в которую помещаем с помощью метода *active* содержимое первого листа файла Excel (именно на первом листе, который является при загрузке файла активным, и располагается база данных).

В строке (16) мы создаём пустой список *qwerty*. В нём будут находиться номера строк базы данных, которые не войдут в множество парето-оптимальных решений.

В строке (17) мы создаём особую структуру данных, которая называется *словарь*. Словарь состоит из пар *ключ-значение,* записываемых в круглых скобках {} через запятую. Ключ отделяется от значения двоеточием. В отличие от списков, элементы словаря не индексируются. Чтобы получить доступ к какому-либо значению словаря, нужно указать имя переменной, в которой хранится словарь, а в квадратных скобках указать ключ. Пример: a = {‘Имя’: ‘Вася’, ‘Возраст’:25}. Тогда a[‘Имя’] = ‘Вася’.

На строке (18) мы видим *условную конструкцию*. Её отличает ключевое слово *if.* Если условие, которое следует после if, выполняется, то программа переходит к блоку, расположенному с отступом после двоеточия. Если же условие не выполняется, то программа переходит к блоку, находящемуся после ключевого слова *else* (если оно есть).

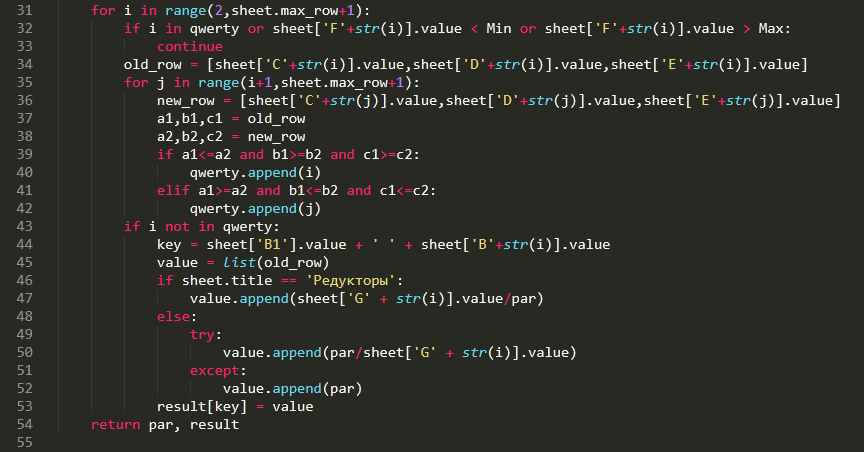
Также на строке (18) мы видим, что к переменной sheet применяют метод *title*. Этот метод возвращает имя листа, который хранится в переменной. Если это имя совпадёт со строкой ‘Редукторы’, то выполнится код ниже.

Далее в строке (19) мы вводим переменную *par*, присваивая ей первоначальное значение равное бесконечности (float(‘inf’) – так в Python можно задать бесконечно большое число). Переменная par будет содержать минимальное (в случае редукторов) или максимальное (в случае двигателей и колёсных установок) значение параметра, определяющего кинематические характеристики механизма передвижения. Для всех других модулей мы примем par равным 1, так как на скорость крана они не влияют.

В строке (20) встречается уже знакомая нам конструкция цикла. Этот цикл должен перебрать все значения столбца ‘G’ таблицы с редукторами (доступ к значениям ячеек таблицы осуществляется с помощью метода *value;* в квадратных скобках указывают адрес ячейки, значение которой мы хотим узнать), начиная со второй строки и заканчивая последней (чтобы получить номер последней строки, надо воспользоваться методом *max\_*row). В результате минимальное из всех значений он присваивает переменной par. Стоит упомянуть, что адрес ячейки, например ‘G2’, мы получаем слиянием строки ‘G’ и строки ‘2’. Чтобы из числа 2 ( а номер строки – это как раз число), получить строку мы должны применить функцию *str.*

Если имя листа будет не ‘Редукторы’, то мы должны найти максимальное значение параметра par. Для этого мы присваиваем этой переменной первоначальное значение 0 (строка 24).

Далее на строке (25) мы видим конструкцию, которая называется *исключение.* Исключение нужно для того, чтобы при возникновении ошибки в программе, она не завершилась, а выполнила действия, указанные программистом. Сначала выполняется блок кода, отмеченный ключевым словом *try.* Если возникла ошибка, то программа выполняет код, отмеченный ключевым словом *except.* Я предусмотрел исключение в программе, потому что не все базы данных имеют столбец ‘G’ (см. рис.5 и рис.6). Как раз для таких баз данных значение параметра par должно быть равным 1 (строка 30), что и выполняется программой за счёт блока except.



На строке (31) мы видим цикл, который проходит по всем строкам таблицы, начиная со второй. Далее (строка 32) мы видим условную конструкцию, которая включает в себя 3 условия, разделённые логическим «или» (по-английски *or*): если хотя бы одно из условий выполняется, то переходим к шагу *continue.* Continue прекращает дальнейшее выполнение тела цикла и начинает новую итерацию (повышает значение переменной i на 1 и возвращает выполнение программы к строке 32).

На строке (34) мы видим список *old\_row*, значениями которого являются содержимое ячеек ‘C’, ‘D’, ‘E’ текущей строки таблицы. Эти ячейки соответствуют КПД, себестоимости и массе соответственно, то есть тем параметрам, по которым мы ведём оптимизацию. На строке (36) мы видим аналогичный список *new\_row*, но его значениям соответствует уже следующая строка таблицы, так как в строке (35) мы переменной j присвоили в цикле значение на 1 больше, чем переменной i (с помощью функции range). Чтобы упростить понимание кода, я ввёл переменные a1,b1,c1, которым присвоил содержимое списка old\_row. a1 соответствует первому элементу списка, b1 – второму, c1 – третьему. То же самое можно сказать и про переменные a2,b2,c2, только они соответствуют списку new\_row.

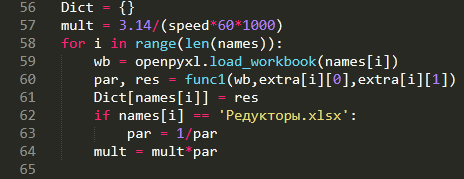
Далее идут условия, с помощью которых и формируются частные парето-оптимальные решения для данного модуля. Если i-я строка по всем параметрам (КПД, себестоимость и масса) уступает j-й строке (речь идёт о строке 39; в ней присутствует логическое умножение *and*: все условия должны быть верны, чтобы выполнился блок кода, расположенный ниже), то добавляем номер этой строки в список qwerty, так как это решение нам не подходит. Если же условие ложное, то проверяем j-ю строку по отношению к i-й (строка 41; слово  *elif*  означает else if). В случае ложности строки (41), добавляем номер j в список qwerty (строка 42).

Если i-я строка после завершения цикла по строкам j не попала в список qwerty (строка 43) , то значит она должна войти в множество частных парето-оптимальных решений. Для этого я создаю ключ *key* для словаря (строка 44), в котором будет информация о виде изделия (двигатель, редуктор и т.д.) и наименование изделия (например, ТГК-160). Значением же по ключу *value* будет выступать содержимое списка old\_row. Но так как в Python список передаётся по ссылке, а не по значению (это значит, что если значение переменной изменится, то поменяется и список, на который переменная ссылается), то мы к списку old\_row применяем функцию *list*, которая преобразует старый список в новый и тем самым обезопасит нас от нежелательного изменения списка old\_row (строка 45).

Затем в список value надо добавить ещё один элемент, который будет являться степенью редукции для данного решения. В случае работы с базой данных ‘Редукторы’ (строка 46), степень редукции будет определяться строкой (47), в случае базы данных ‘Двигатели’ или ‘Колёсные установки’ – строкой (50) , в остальных случаях, для которых нет столбца ‘G’ (именно поэтому я использовал исключение, чтобы предотвратить аварийное завершение программы) – строкой (52). После добавления степени редукции в список value можно добавить пару ключ-значение в словарь (строка 53).

Как я уже упоминал, переменные после выхода из функции стираются из памяти. Поэтому, если нужно использовать какую-то переменную в дальнейшем, необходимо вернуть её функцией с помощью метода *return.* Нам понадобятся переменная par, так как она используется в определении степени редукции всей системы, и переменная result, которая несёт информацию о множестве частных парето-оптимальных решениях для данного модуля. Вернём их методом return (строка 54).

*Часть 3 – сохранение частных решений в словарь и определение степени редукции всей системы*



Для сохранения всех частных парето-оптимальных решений создадим словарь *Dict* (строка 56). Далее в строке (57) определяем переменную *mult,*присваивая ей первоначальное значение. Эта переменная будет содержать в себе значение степени редукции для всей системы, определяемой по формуле (1).

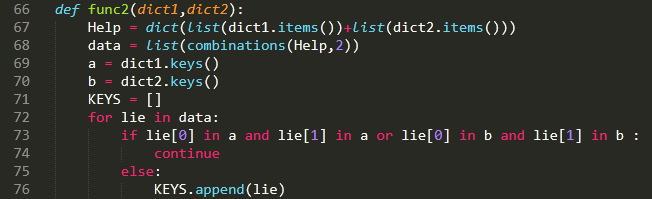
Затем нам нужно загрузить все файлы Excel, имена которых содержатся в списке names. Для этого мы используем цикл (строка 58). Загрузка файлов осуществляется с помощью метода *load\_worbook* библиотеки openpyxl. В скобках указывается путь к файлу,а в данном случае элемент списка names, так как он и представляет собой путь (строка 59). Загруженный файл сохраняем в переменной *wb.*

В строке (60) мы присваиваем переменным *par* и *res* результаты выполнения функции func1. Функция принимает на вход загружаемый файл в виде переменной wb, а также элементы списка extra, в котором напоминаю, хранятся значения минимумов и максимумов параметра, ограничения которого вводил пользователь.

В строке (61) мы добавляем в словарь Dict пару ключ-значение. Ключом является имя загружаемого файла, значением – словарь с частным парето-оптимальным решением, который вернула функция func1.

Далее мы формируем переменную mult согласно формуле (1) Для этого мы её предыдущее значение умножаем на переменную par (строка 64). Но так как по формуле (1) передаточное отношение для редуктора находится в знаменателе, то мы переменную par, которая возвращает для базы данных с редукторами это отношение, изменяем, как указано в строке (63). Предварительно мы, конечно, проверяем, что файл относится к редукторам (строка 62).

*Часть 4 – функция объединения частных решений в одно множество*

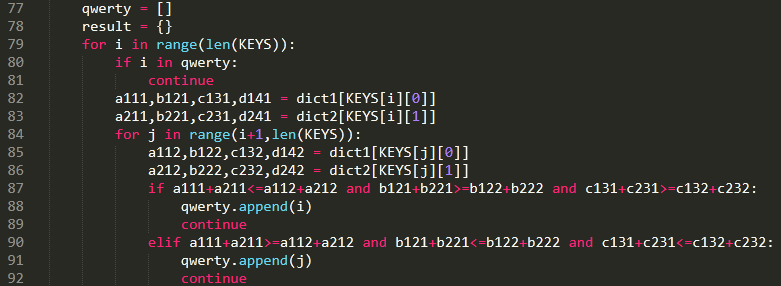


На строке (66) мы определяем новую функцию *func2,* которая в качестве входных данных принимает два параметра: словарь *dict1* и словарь *dict2*. Dict1 будет содержать общее решение для рассмотренных ранее модулей, полученное на предыдущем шаге, а dict2 содержит частные парето-оптимальные решения текущего модуля. Далее мы создаём новый словарь *Help*, с помощью функции *dict* (строка 67), который вмещает в себя пары ключ-значение как dict1, так и dict2. Для этого мы пользуемся методом словаря, который возвращает эти пары в виде списка и который называется *items.* Так же мы применяем к полученным парам функцию list, чтобы избежать изменения словарей dict1 и dict2.

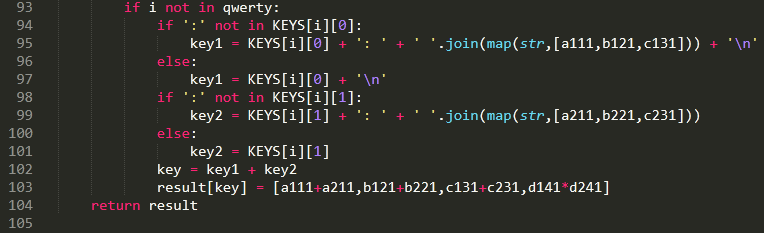
В строке (68) мы комбинируем все ключи словаря Help по парам с помощью функции *combinations,* которую мы импортировали в самом начале программы, и присваиваем результат переменной *data*.

Далее в строках (69) и (70) мы создаём переменные *a* и *b,* которым присвоили список ключей словарей dict1и dict2 соответственно с помощью метода *keys.* Затем мы создаём пустой список *KEYS* (строка 71).

Наконец мы с помощью цикла (строка 72) проходим по всем парам ключей, находящихся в списке data. Если каждый ключ из пары принадлежит одному и тому же списку (или a, или b), то начинаем выполнять следующую итерацию (строки 73 и 74). Если же ключи относятся к разным спискам, то мы пару эти ключей добавляем в список KEYS (строка 76).

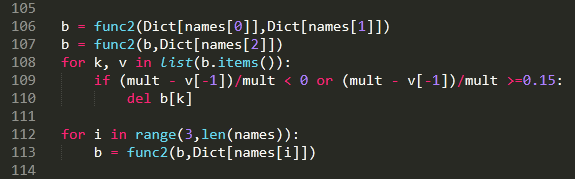


Далее (строки 77-92) идёт алгоритм отсеивания неугодных решений, аналогичный алгоритму функции func1, показанный в строках 31-42. Основное отличие в том, что для проверки используются не сами переменные, а их суммы, так как функция работает не с одной базой данных, а с двумя (dict1 и dict2).



И наконец мы должны как-то сохранить наши результаты, чтобы вернуть их из функции. Для копилки результатов всё так же будем использовать словарь result (как мы это делали с функцией func1). Ключом этого словаря будут все выбранные программой изделия, для которых через двоеточие перечисляются их параметры (КПД, себестоимость, масса). Различные изделия отделяются друг от друга через символ новой строки ‘\n’ (строка 97). Если для изделия ещё не определены его параметры (нет двоеточия после наименования изделия; строка 94), то это выполняется в строке (95). Здесь мы элементы списка параметров (a111,b121,c131) преобразуем в строки с помощью функции *map*, которая применяет функцию *str* к каждому элементу списка, а затем соединяем параметры через пробел ‘ ‘ с помощью метода *join.* Аналогичные действия проделываем и для ключа *key2*. Затем складываем строки, содержащиеся в ключах *key1*и *key2* (строка 102) и результат используем в качестве ключа для словаря result. Значением же будет список, показанный в строке (103). Последним элементом этого списка является степень редукции для всех рассмотренных ранее модулей, которая формируется согласно формуле (7). В заключительной строке функции (строка 104) мы возвращаем словарь result из функции c помощью метода return.

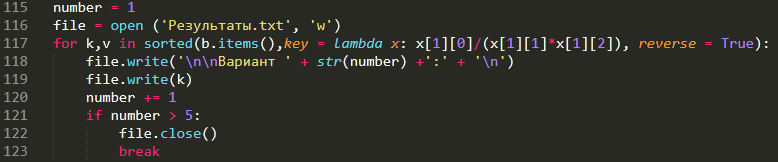
*Часть 5 – отсеивание вариантов, неудовлетворяющих скорости передвижения крана и объединение всех частных решений в одно множество.*

**

В строках (106) и (107) мы получаем общее решение для первых 3-х модулей, которые при отсутствии открытой зубчатой передачи должны полностью удовлетворять скорости передвижения козлового крана. Естественно, что в этом множестве решений будут и те, которые не удовлетворяют условиям. Поэтому мы запускаем цикл по парам ключ-значение (строка 108), где переменная k – это текущий ключ, а v – текущее значение по ключу. Далее мы проверяем выполнение условия (строка 109), которое является интерпретацией формулы (6). Если условие не выполняется, то данное решение мы удаляем из словаря b (строка 110). В условии v[-1] означает последний элемент списка v, который является степенью редукции для данного решения, a mult – это степень редукции для всей системы в целом, которую мы находили ранее.

Отсеяв неугодные решения, продолжаем объединение частных решений в одно множество и наконец получаем на последнем шаге множество общих решений для всей системы в целом (строки 112 и 113).

*Часть 6 – вывод результатов в отдельный файл*

**

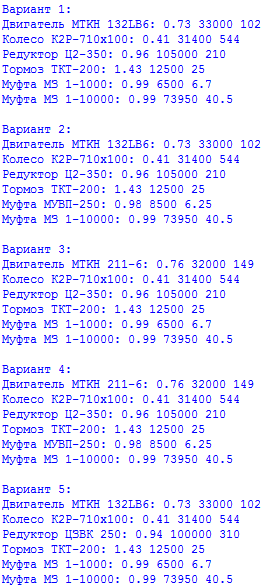
На предыдущем шаге мы получили, наконец, словарь b, ключами которого являются искомые решения. Но этих решений может быть очень много, и человек будет не в состоянии проанализировать каждое из них, чтобы потом выбрать только одно. Поэтому я счёл необходимым ввести счётчик вариантов, предельным значением которого я выбрал число 5 (строка 121). Первоначальное же значение счётчика *number* равняется 1 (строка 115). Как только значение переменной number перевалит за 5 (строка 121), срабатывает инструкция *break* (строка 123)*,* которая завершает цикл (строка 117) преждевременно.

Отдельно стоит поговорить про цикл на строке (117). В нем итерации идут по паре ключ-значение (k-v). Мы это уже встречали ранее, например, в строке (108). Но здесь цикл пробегается не просто по ключам и значениям словаря, а по отсортированному словарю благодаря функции *sorted.* Сортируется словарь по определённому мной правилу: по значению параметра  ( чем этот показатель больше, тем лучше). Этот параметр я присваиваю функции *lambda* – особой функции языка Python. Но сортировка по умолчанию происходит от меньшего числа к большему, то есть все нужные мне значения будут находиться в конце списка. Поэтому я меняю порядок сортировки: от большего числа к меньшему за счёт конструкции *reverse = True.*

Я решил вывести результаты выполнения программы в отдельный файл. В строке (116) я создаю файл с именем *Результаты.txt* и открываю его в режиме записи (обозначается ‘w’). Затем записываем содержимое ключей в файл (строка 119) и когда число вариантов становится равным 5, закрываем файл (строка 122).

Программа закончена, но есть одна проблема: как её использовать тем, у кого нет интерпретатора Python? Очевидно, что нужно файл, в котором написан код, преобразовать в формат, читаемый на любом компьютере. Этим форматом является *.exe.* Назовём исполняемый файл *Парето* и запустим его. Введя необходимые данные, которые запрашивают у пользователя (можно ввести те, которые указаны на рис. 7), мы получаем наконец файл с результатами.

*Результат выполнения программы:*



Так выглядит результат выполнения программы. Мы видим 5 вариантов, каждый из которых представлен полным набором используемых изделий. Через двоеточие для изделия указаны КПД, себестоимость в рублях и масса в килограммах соответственно (для тормозов я не нашёл значений КПД при заполнении баз данных, поэтому использовал аналогичный параметр: 100/ потребляемая мощность).

На последней стадии человек сам должен выбрать один из вариантов на основании личных соображений. Например, я бы выбрал вариант 5, потому что там используется редуктор ЦЗВК, а он вертикального типа, в отличие от редуктора Ц2. В механизмах передвижения крана чаще всего используются именно вертикальные редукторы.

**Выводы:** предложенный метод оптимального проектирования имеет следующие преимущества: позволяет автоматизировано рассмотреть применение всех перспективных по мнению ЛПР компоновок механизма, кинематических схем привода, а также всех доступных на конкретном заводе-изготовителе комплектующих механизма, что в режиме «ручного» проектирования практически невозможно из-за ограниченных сроков подготовки технического проекта.

Хоть я в данной работе не реализовал всех возможностей метода оптимального проектирования (не стал рассматривать различные компоновки и кинематические схемы в виду сложности разработки программы), я доказал его эффективность на примере конкретной схемы.

**Список используемой литературы:**

1. Чернова Н.М. Развитие теории оптимального проектирования механизмов грузоподъёмных кранов пролётного типа: дис. … докт. техн. наук. – Саратов. 2009. – 372 с.

2. Адитья Бхаргава. Грокаем алгоритмы. 2017 – 290 с.

3. Свейгарт Эл. Автоматизация рутинных задач с помощью Python. 2017. – 592 с.

4. Ногин. В.Д. Принятие решений в многокритериальной среде: количественный подход. – М.: ФИЗМАТЛИТ, 2005 – 176 с.