Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика

РАБОТА ПРОВЕРЕНА
Рецензент,
_____(И.О. Ф.)
«____»_____2018 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А.А.Замышляева
«____»_____2018 г.

Исследование генетических алгоритмов на примере решения квадратичной задачи о назначениях

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

ЮУрГУ–01.03.04.2018.35.ПЗ ВКР

Руководитель работы,
к.ф.-м.н., доцент
_____/А.В. Гренштейн
«____»_____2018 г.
Автор работы
Студент группы ЕТ-413
_____/ И.Г. Курносов
«____»_____2018 г.
Нормоконтролер,
к.ф.-м.н., доцент
_____/Д.А. Дрозин
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Курносов И.Г. Исследование колебаний в обрабатываемой детали в процессе шлифования. Челябинск: ЮУрГУ, ЕТ-413, 67с., 4ил., 9табл,, библиогр. список – 14наим., 1 прил.

Данная работа посвящена исследованию колебаний и модели шлифования в обрабатываемой детали. В работе была разработана математическая модель и программа, которая реализует данный процесс, оценена случайность задаваемой величины – шероховатости. Полученный результат говорит о том, что модель является приближенной реальному процессу.

ОГЛАВЛЕНИЕ

АННОТАЦИЯ ... 2
ВВЕДЕНИЕ ... 4
1. МЕТОДЫ ПРОЕКТИРОВАНИЯ ЦИКЛОВ ШЛИФОВАНИЯ ... 5
1.1. Особенности и суть процесса шлифования ... 5
1.2. Проектирование циклов круглого врезного шлифования ... 6
1.3. Выводы по разделу ... 9
2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ЦИКЛОВ ШЛИФОВАНИЯ ... 10
2.1. Уравнение силового взаимодействия в процессе резания ... 10
2.2. Сила резания ... 15
2.3. Минимизация параметров математической модели методом обезразмеривания ... 16
2.4. Выводы по разделу ... 19
3. ИССЛЕДОВАНИЕ КОЛЕБАНИЙ В ПРОЦЕССЕ ШЛИФОВАНИЯ ... 20
3.1. Учет параметров колебаний ... 20
3.2. Вывод уравнений для колебаний ... 20
3.3. Выводы по разделу ... 27
4. АНАЛИЗ РЕЗУЛЬТАТОВ ... 28
4.1. Линейная кубическая вязкость ... 28
4.2. Оценка шероховатости после обработки ... 29
4.3. Выводы по разделу ... 34
ЗАКЛЮЧЕНИЕ ... 36
БИБЛИОГРАФИЧЕСКИЙ СПИСОК ... 37
ПРИЛОЖЕНИЕ 1. Текст программы ... 39

ВВЕДЕНИЕ

Шлифование — метод обработки заготовок, путем резания. Помогает обеспечивать точность физических и механических свойств слоев обрабатываемой детали. А так же точность геометрических параметров поверхностей и их расположение.

Задачей машиностроения является — увеличение качества изготовляемых деталей, при поддержки высокой производительности технологических процессов.

Эффективное использование возможностей станка зависит от разработки качественного программного обеспечения, которое позволит автоматизировать процесс шлифования.

Цель работы: Произвести исследование и разработать модель, которая позволяет автоматизировать процесс шлифования. Исследовать колебания, возникающие в процессе шлифования. Ввести уточнение, в виде кубической линейной вязкости.

Задачи:

1) Изучение теории и методики моделирования процесса шлифования;
2) Создание математической модели взаимодействия инструмента и обрабатываемой детали;
3) Исследование колебаний и коррекция алгоритма, реализующего данную модель.

1. МЕТОДЫ ПРОЕКТИРОВАНИЯ ЦИКЛОВ ШЛИФОВАНИЯ

1.1.Особенности и суть процесса шлифования

По описанию Курдюкова В.И. [1], шлифование это процесс массового скоростного микрорезания или царапания поверхности твердых тел большим числом шлифующих зерен, сцементированных в инструмент с помощью связки. Процесс происходит при высоких скоростях – до 50 м/с, в некоторых случаях – до 150 м/с. Процесс шлифование материалов обеспечивает:

1) Высокую размерную и геометрическую точность.
2) Высокую производительность.
3) Высокое качество поверхностного слоя и низкую шероховатость.

Нет существенной разницы между процессами резания металлов. Сам процесс стоит рассматривать как суммарное царапание поверхностных слоев абразивными зернами. Чем больше число царапающих зерен на поверхности инструмента, тем выше качество обрабатываемого изделия.

Процесс резания при шлифовании имеет особенности. Шлифовальный круг имеет прерывистую режущую кромку. Абразивные зерна быстро скользят по металлу, вызывая большое тепловыделение. Процесс стружкообразования протекает непрерывно, снятие стружки происходит за короткое время. Процесс шлифования заключается в тонком срезании отдельными резцами материала детали.

Появляется вопрос: как обеспечить качество детали при меньших затратах времени на обработку изделия в процессе шлифования.

Данный вопрос решали такие исследователи как: Гузеев В.И., Нуркенов А.В., Гереншстейн А.В., Дьяконов А.А., Шипулин Л.В., Евтухов В.Г., Исаков Д.В., Медведев А.С., Лысов В.Е. ими было признано, что минимизация времени зависит от проектирования циклов шлифования, при которых весь путь инструмента к детали разделяется на ступени с различающейся радиальной подачей. При этом есть возможность к конкретным технологическим условиям адаптировать цикл обработки [14].

Оптимизация возможна на основе имитационной модели процесса, которая учитывает его разноплановые физические особенности: высокую теплонапряженность и множественное вероятностное микрорезание в технологической системе. [14]

Исследователи при проектировании циклов шлифования, делали ряд допущений, чтобы упросить расчеты. Из-за этого снижается область применения разработанных моделей. Шипулин Л.В., в своей работе [14] считает, что наиболее точная модель принадлежит Дьяконову А.А [4], но проблема его модели в том, что он не учитывает геометрию множественного микрорезания.

1.2. Проектирование циклов круглого врезного шлифования

Развитие проектирования циклов круглого врезного шлифования прошло три этапа: на основе производственного опыта и нормативных данных, а так же метод моделирования на основе экспериментов[1].

Благодаря системам числового программного управления, появилась возможность управления процессом обработки на уровне интеллектуального проектирования. За счет этого повышается производительность операции.

Производительность процесса шлифования зависит от этапов обработки. Черновое шлифование – обработка без токарной операции со снятием увеличенного припуска с заготовки. Данное шлифование обеспечивает высокую точность по 8-9-му квалитету и низкий параметр шероховатости, а так же не требует последующего предварительного шлифования. При этом в ходе процесса вероятней всего может произойти прижог на детали. Который может совпасть с глубиной дефектного слоя. Отсюда становится понятным, что глубина дефектного слоя не должна превышать остаточный припуск на конечном этапе.

Отсутствие прижога характерно менее производительными режимами резания[6]. Расчет предельной скорости подачи, необходимо производить с учетом возможного дефектного слоя в оставшейся части припуска.

Поскольку расчет величины дефектного слоя необходимым параметром при проектировании цикла шлифования, величина припуска для черновой и чистовой ступеней цикла обеспечивается за счет управления глубиной прижога.

Таким образом, расчет величины дефектного слоя является необходимым параметром при проектировании цикла шлифования. [8].

Контактная температура при круглом врезном шлифовании определяется по формуле:

(1.1)

где: $\alpha_в$ – коэффициент, учитывающий отвод тепла в инструмент (для абразивных кругов 1, а для алмазных ≥0,85); M – размерный коэффициент (для жаропрочных сталей М=1,9, а для углеродистых и низколегированных М=3,1); P_z – составляющая касательной силы резания, Н; l_k – длина дуги контакта, мм; F_k – площадь пятна контакта, мм^2; V – скорость резания м/с; $K_{сож}$ – коэффициент учитывающий снижение температуры; $t_ф$ – глубина шлифования, мм/об; $D_{кр}$ –

диаметр шлифовального круга, мм; d – диаметр заготовки, мм; V_s – окружная скорость заготовки. м/мин.

Глубина дефектного слоя определяется по формуле:

(1.2)

где: $D_{кр}$ – диаметр шлифовального круга, мм; d – диаметр заготовки, мм; A, X – коэффициенты; $t_ф$ – глубина резания, мм/об; $\theta_к$ – контактная температура в зоне резания, градусы; V_s – окружная скорость заготовки. м/мин.

При проектирования максимально производительной подачи, необходимо учитывать необходимую и фактическую мощность привода шлифовальной бабки и осыпаемость шлифовального круга. [7]

Потребная фактическая мощность привода определяется по формуле

(1.3)

где $N_э$ – эффективная потребная мощность резания, кВт; P_z – составляющая касательной силы резания; V – скорость резания м/с; η – КПД привода вращения шлифовального круга.

Ограничение по осыпаемости шлифовального круга имеет следующий вид

(1.4)

При условии

(1.5)

где $\sigma_в$ – предел прочности, Мпа; V – скорость резания м/с; $D_{кр}$ – диаметр шлифовального круга, м/с; $V_{окр}$ – окружная скорость заготовки. м/мин; N – зернистость шлифовального круга; С – номер структуры круга; $t_ф$ – глубина резания, мм/об; $\theta_к$ – контактная температура в зоне резания, градусы; $[T_у]$ – условный номер круга, ограничивающий самозатачивание.

Второй этап обработки характеризуется удалением малой части остаточного припуска, где основным требованием является выполнение условий по точности и качеству изделия.

Величина остаточного чистового припуска определяется как

(1.6)

где – остаточный суммарный припуск;

– остаточный припуск на черновой ступени.

Остаточный припуск необходимо удалить с учетом образовавшегося прижога при максимальной подаче, исключающей данный дефект.

Погрешность обработки, как величину, кумулятивную определяемую множеством параметров технологической системы, возможно оценить, как разницу фактического припуска и расчетного

(1.7)

где – расчетное значение припуска;

– фактическое значение припуска.

Качество изделия оценивается по шероховатости поверхности. Данное требование обеспечивается этапом выхаживания. Расчет данного этапа возможен на основе остаточного съема припуска за счет натяга в системе шлифовальный круг–деталь при отключенной поперечной подаче.

$$Ra_i = Rз_i - Sн_i = Rз_i - Y_i \quad (1.8)$$

где: Ra_i – шероховатость заготовки на *i*-ом обороте, $Rз_i$ – радиус заготовки на *i*-ом обороте, $Sн_i$ – подача шлифовального круга при выхаживании *i*-ом обороте, Y_i – натяг системы шлифовальный круг-заготовка и *i*-ом обороте.

Поскольку в задачи работы входит проектирование одномассовой модели, предполагается перейти от формул (1.1)–(1.7) к иному способу исчисления параметров. Допустимо заменить формирование припуска одним задаваемым параметром, а шероховатость поверхности детали задавать случайной величиной. [2]

Таким образом, рассчитав данные параметры системы в разработанной модели, является возможным расчет требуемого времени (количества оборотов заготовки) выхаживания при проектировании цикла шлифования.

1.3.Выводы по разделу

В данном разделе рассмотрены современные способы проектирования циклов шлифования, и наиболее важные результаты исследования в этой области. Рассмотрена оптимизация классической теории проектирования циклов шлифования на основе фактических параметров технологической системы.

Исходя из вышесказанного формируется задача оптимизации методики проектирования цикла шлифования, которая заключается в повышении эффективности процесса обработки за счет формирования цикла шлифования на основе фактических параметров технологической системы.

2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ЦИКЛОВ ШЛИФОВАНИЯ

2.1. Уравнение силового взаимодействия в процессе резания

Представим шлифовальный станок в виде трехмассовой модели (более подробно рассмотрено В.Г. Евтуховым в [5]). Аналогичная модель использовалась в работах [3], [12].

В процессе обработки шлифовальный круг массой , присоединенный к шлифовальной бабке массой , снимает припуск с заготовки массой . Заготовка присоединена к неподвижному основанию станка через демпфер с жесткостью пружины . Шлифовальный круг с жесткостью присоединен к бабке, которая в свою очередь присоединена к неподвижному основанию станка через демпфер с жесткостью . Коэффициент жесткости характеризует режущую способность и имитирует сопротивление шлифовального круга, что представлено на рисунке 2.1.

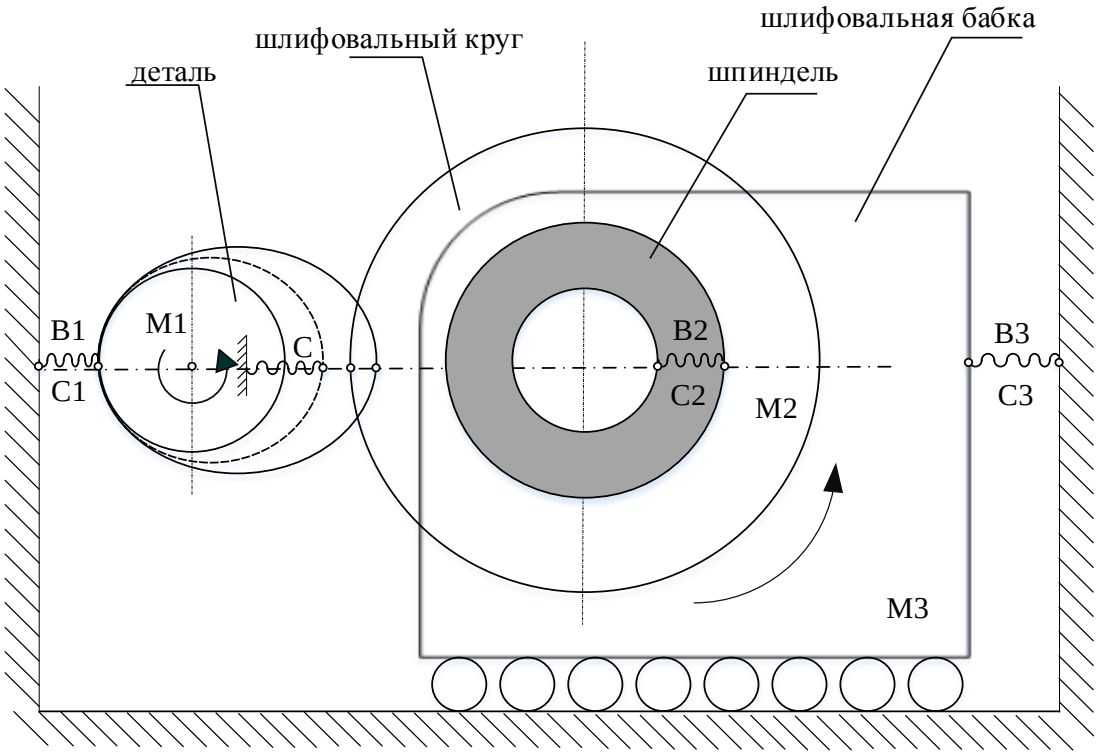


Рисунок 2.1 – Трехмассовая модель круглошлифовального станка

В данной модели осциллирует только деталь, что позволяет нам перейти к одномассовой модели (рисунок 2.2). Шлифовальный круг заменяем резцом, а процесс шлифования – процессом резания. При этом, будем считать, что подаваемый на деталь резец имеет идеально ровную форму.

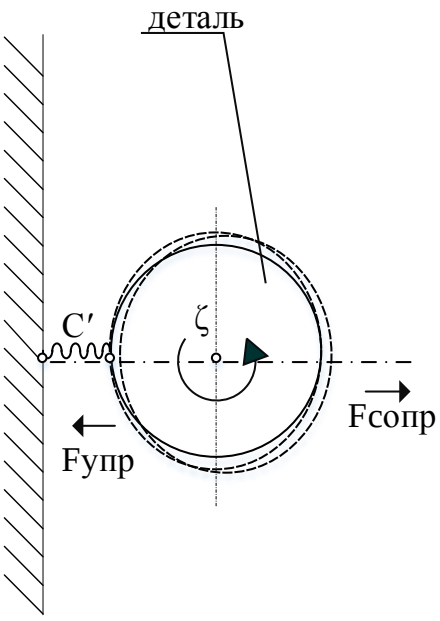


Рисунок 2.2 – Одномассовая модель круглошлифовального станка

Выведем уравнение движения детали. Согласно второму закону Ньтона (см.рисунок 2.2):

, (2.1)

где – сила сопротивления среды, Н;

– сила упругости, Н;

– масса детали, кг;

– ускорение координаты оси вращения детали, м/ .

Где сила сопротивления определяется как

, (2.2)

где – коэффициент сопротивления среды;

– скорость координаты оси вращения детали, м/с.

А сила упругости определяется как

(2.3)

где – жесткость пружины, Н/м,

– текущая координата оси вращения детали.

То уравнение движения детали примет вид:

(2.4)

При этом на деталь будет действовать сила резания, следовательно, уравнение (2.4) примет вид:

, (2.5)

где – сила резания, Н;

– глубина силы резания, м.

Найдем общее решение дифференциального уравнения (2.5). Решение будем искать в виде:

(2.6)

Тогда уравнение (2.4) преобразуется к виду:

(2.7)

Его решениями являются:

(2.8)

Введем следующие обозначения:

(2.9)

(2.10)

Следовательно, получим общее решение уравнения (2.5):

(2.11)

Теперь найдем частное решение уравнения. Для удобства перепишем уравнение (2.5) в виде:

(2.12)

Решение будем искать методом вариации произвольных постоянных (методом Лагранжа). Фундаментальную систему решений соответствующего однородного уравнения (2.4) образуют функции (2.13), (2.14)

(2.13)

(2.14)

Будем искать частное решение неоднородного уравнения в виде

(2.15)

(2.16)

(2.17)

(2.18)

Подставим выражения для производных в уравнение (2.15):

(2.19)

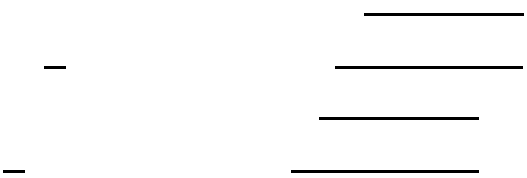
$=f(t)$

Преобразуем (2.19), учитывая (2.9) и (2.10):

(2.20)

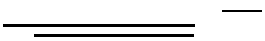
$=f(t)$

Для неизвестных функций получена система линейных дифференциальных уравнений второго порядка:

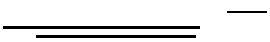


(2.21)

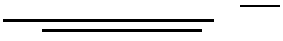
Откуда получаем



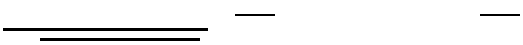
(2.22)



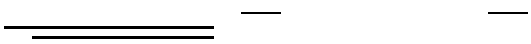
(2.23)



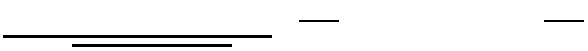
И тогда частным решением уравнения (2.12) является функция



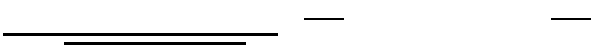
(2.24)



Соответственно, частным решением исходного уравнения (2.5) является функция



(2.25)



Полученные решения (2.11), (2.25) будут использованы в дальнейшем.

2.2.Сила резания

Резец, врезаясь в поверхность детали абразивными зернами, снимает стружку. Чем большую стружку необходимо срезать и чем прочнее материал детали, тем большую силу резания необходимо к нему приложить.

Определим глубину силы резания как (рисунок 2.3)

(2.26)

где – текущее расстояние от оси вращения детали до ее края, м;

– текущая координата края резца.

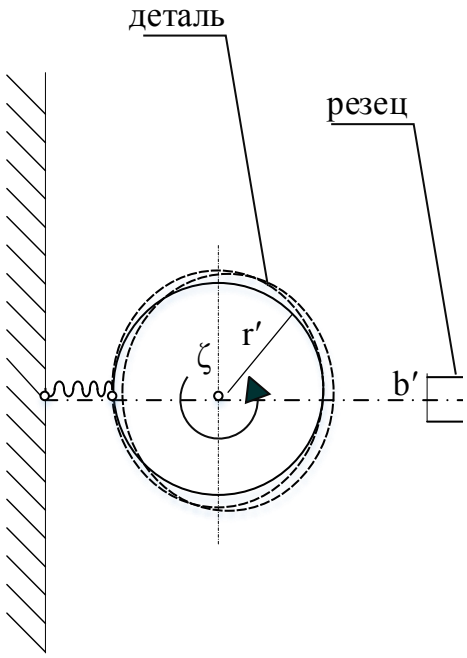


Рисунок 2.3 – Глубина силы резания

Глубина резания оказывает большое воздействие на ее силу. Силу резания определим как

(2.27)

где – предел прочности материала заготовки при высоких температурах,

;

– ширина шлифования, мм;

– первый коэффициент силы резания;
– угловая скорость детали, об/мин;
– требуемый радиус детали, мм;
– скорость резания, м/с;
– второй коэффициент силы резания;
– радиус шлифовального круга, мм;
– коэффициент затупления шлифовального круга.

При этом если (см.рисунок 2.3)

(2.28)

Резец не контактирует с поверхностью детали, следовательно сила резания (2.27)

(2.29)

Поскольку мы учитываем колебания оси вращения детали, то определим вертикальную составляющую силы резания

(2.30)

где – окружная скорость заготовки, м/мин;
– диаметр шлифовального круга, мм.

Данная составляющая оказывает большое влияние на точность обработки и на вибрации (отскоки), которые будут возникать в процессе резания.

2.3. Минимизация параметров математической модели методом обезразмеривания

Известно, что большое количество переменных математической модели характеризует трудоемкость расчета и идентификации ее параметров. Поэтому с целью минимизации параметров модели, а также повышения производительности расчета необходимо произвести обезразмеривание параметров (аналогично использовалась в работах [3], [12]). При этом главной задачей обезразмеривания является установление законов подобия между параметрами.

При считывании параметров миллиметры переводим в метры, минуты в секунды, обороты – в 2π рад. Затем установим три размерно-независимых

параметра: масса детали M [кг], требуемый радиус детали в конце процесса шлифования R [м], угловая скорость детали ω [1/с].

Исходные размерные параметры заменяем на безразмерные, что и представлено в таблице 2.1.

Таблица 2.1

Обезразмеривание параметров

Параметр	Безразмерный параметр
Сила резания	
Координата края резца в начальный момент времени	
Начальная координата оси детали	
Жесткость пружины	
Линейная вязкость детали	
Радиус детали в начальный момент времени	
Эксцентриситет детали в начальный момент времени	
Скорость подачи резца	
Текущая координата оси вращения детали	
Время	
Первый коэффициент силы резания	
Второй коэффициент силы резания	———
Масса детали	
Требуемый радиус детали в конце процесса шлифования	
Угловая скорость детали	

В результате обезразмеривания параметров модели уравнение движения детали (2.5) принимает вид

(2.31)

Известно, что значения параметров, получаемые с помощью методов численного решения, как правило отличаются от их истинных значений из-за наличия ошибки аппроксимации. Решения могут оказаться непригодным, если уравнение математической модели содержит переменные, значения которых отличаются по порядкам. Так, погрешности при определении параметров, порядки которых велики, могут быть не значимы для них самих, но в то же время они будут сильно искажать значения параметров меньших порядков. Поэтому

прежде чем перейти к созданию алгоритма для решения уравнений математической модели, необходимо привести эти уравнения к безразмерному виду, т.е. провести операцию обезразмеривания переменных, в результате которой все переменные математической модели будут иметь одинаковый порядок.

Перейдем от уравнения второго порядка к уравнению первого. Для этого введем массив размерностью два, в котором , , , . Тогда получим линейно однородную систему уравнений первого порядка (систему в нормальной форме Коши)

(2.32)

Обозначим через фундаментальную нормированную матрицу решений системы (2.32), то есть матрицу, столбцы которой являются решением этой системы, причем является единичной матрицей.

Дискретный шаг времени выберем так, чтобы выполнялись три условия:

1) является целым числом;

2) намного меньше периода собственных колебаний оси вращения детали вдоль оси ;

3) За промежуток времени сила резания меняется настолько мало, что на каждом таком промежутке эту силу можно считать постоянной (но на каждом промежутке постоянные разные).

Пусть

(2.33)

Тогда

(2.34)

Таким образом (2.34) является решением системы (2.32) для каждого дискретного значения времени. Дальнейшей задачей является нахождение фундаментальной нормированной матрицы решений, которая будет описывать решение (2.34).

2.4. Выводы по разделу

В данном разделе получена математическая модель, которая с учетом предложенных допущений, позволяет сформировать основу алгоритма проектирования цикла шлифования.

В разделе поставлена задача найти фундаментальную нормированную матрицу решений полученной нормальной системы уравнений (2.32). Произведено обезразмеривание параметров, которое устанавливает подобие между переменными. Произведенная минимизация количества параметров улучшит быстродействие расчетов, осуществляемых в программе.

3. ИССЛЕДОВАНИЕ КОЛЕБАНИЙ В ПРОЦЕССЕ ШЛИФОВАНИЯ

3.1. Учет параметров колебаний

Будем разбивать деталь на определенное количество секторов, представленных на рисунке 3.1.

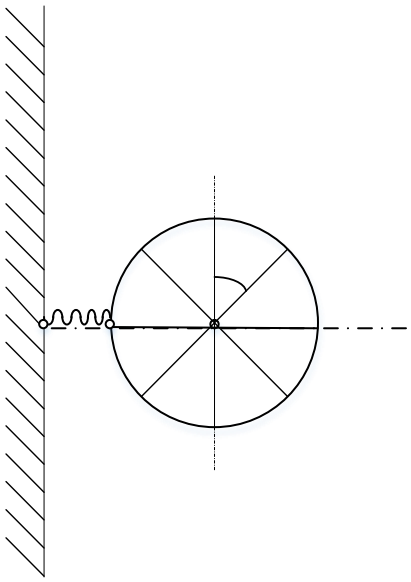


Рисунок 3.1 – Разбиение

Для этого изначально задаем число элементарных углов окружности, на которые делиться деталь. Затем находим значение элементарного угла, то есть такой минимальный угол, на который необходимо повернуть фигуру, чтобы она совпала сама с собой. При этом шаг по времени выбираем так, чтобы за один шаг было не более – колебаний. Таким образом, на каждом шаге разбиения будут учитываться колебания.

3.2. Вывод уравнений для колебаний

Рассмотрим необезрамеренное уравнение

(3.1)

Где – коэффициент сопротивления среды.
В силу формул (2.6)-(2.11) решением этого уравнения является

(3.2)

Где

(3.3)

(3.4)

Пусть

(3.5)

Тогда

(3.6)

Следовательно

(3.7)

(3.8)

Подставим решение (3.2) в (3.7)

(3.9)

При имеем:

(3.10)

(3.11)

(3.12)

Подставим (3.11)-(3.12) в (3.2) и (3.9)

(3.13)

(3.14)

Зная решение однородного уравнения, получим решение неоднородного

(3.15)

где – вектор, ;

– начальные условия.

Решение можно искать в виде (3.15), но отыщем его иным путем. Пусть имеется система из двух неизвестных

(3.16)

где – невырожденная матрица.

Найти функцию можно решением

(3.17)

С помощью фундаментальной матрицы решений ищем решение в виде

(3.18)

где – единичная матрица.

Начальные условия опишем как

(3.19)

(3.20)

Запишем характеристическое уравнение

(3.21)

Определитель этой матрицы (3.21)

(3.22)

Получаем из (3.22) характеристическое уравнение:

(3.23)

Подставляем (3.16) в уравнение (3.18). Получаем

(3.24)

(3.25)

То есть каждой матрице удовлетворяет свое характеристическое уравнение Получим

(3.26)

Подставляя (3.26) в уравнение (3.24). Получаем

(3.27)

Разделим полученное выражение

(3.28)

(3.29)

При начальных условиях (3.19), (3.20). Продифференцируем (3.29) и подставим (3.28)

(3.30)

Характеристическим уравнением для данного уравнения (3.20) является

(3.31)

Его решениями является

(3.32)

Следовательно, могут быть три случая. Если дискриминант меньше нуля

(3.33)

При

(3.34)

(3.35)

Если дискриминант равен нулю

(3.36)

При

(3.37)

(3.38)

(3.39)

И если дискриминант положителен (в остальных случаях)

(3.40)

То

(3.41)

(3.42)

(3.43)

Подставив начальные условия (3.19), (3.20) получаем

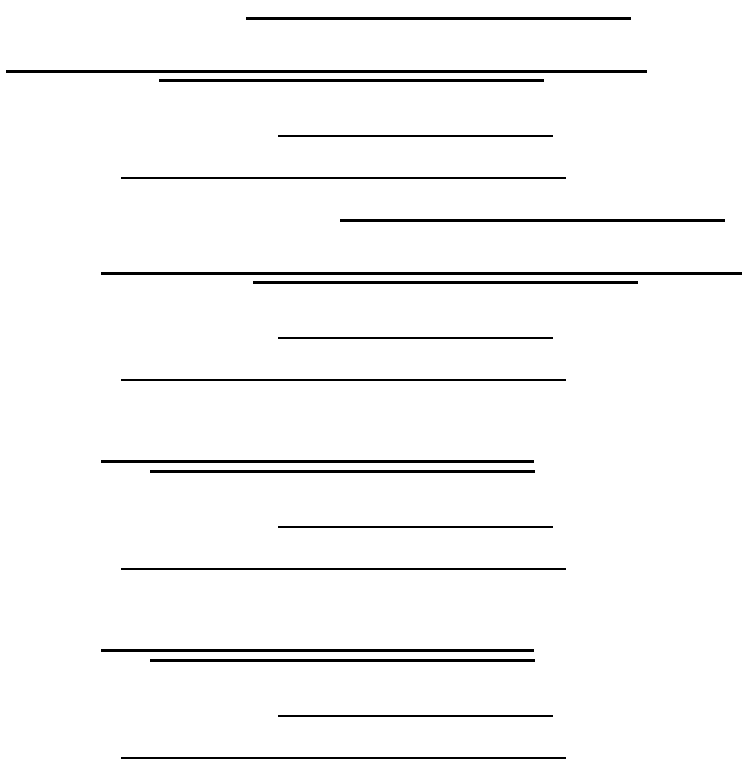
(3.44)

(3.45)

(3.46)

(3.47)

(3.48)



(3.49)

Таким образом, получена фундаментальная нормированная матрица решений. Теперь подставляя ее в (2.34) будем получать решение на каждом шаге. Коэффициенты, полученные в формулах (3.44)-(3.49) подбирает программа.

3.3. Выводы по разделу

В данном разделе показано, как производится разбиение, выбирается шаг по времени, учитывающее колебания, возникающие в процессе шлифования.

Полученные решения (3.44)-(3.49) являются фундаментальной нормированной матрицей решений и являются решением исходной задачи (2.32).

4. АНАЛИЗ РЕЗУЛЬТАТОВ

4.1.Линейная кубическая вязкость

Поскольку в результате возникающего резонанса возникают отскоки, введем поправку – линейную кубическую вязкость. Вязкость выбираем кубической, поскольку мы хотим, чтобы колебания затухали быстрее. Тогда уравнение (2.31) предстанет в виде:

, (4.1)

где – скорость резца;
– отладочный коэффициент.

Рассмотрим введенную поправку на устойчивость по Ляпунову. Уравнение (4.1) сведем к нормальной системе:

(4.2)

(4.3)

То есть

(4.4)

Рассмотрим функцию

(4.5)

Эта функцию положительная определена. Умножим первое уравнение системы (4.4) на —, а второе на . Тогда производная в силу системы функции имеет вид:

(4.6)

Ввиду того что , следует, что производная тоже отрицательна. Докажем, что тривиальное решение асимптотически устойчиво. В уравнении (4.1)

, , – не влияет:

Пусть , тогда

(4.7)

(4.8)

(4.9)

Правая часть в формуле (4.9) положительна, откуда следует, что решение дифференциального уравнения (4.1) является устойчивым по Ляпунову.

4.2.Оценка шероховатости после обработки

Шероховатость в представленной работе задается случайно. Это сделано с целью добавить в процесс обработки элемент случайности. В данном разделе поставлена задача рассмотреть, подчинена ли данная величина нормальному закону распределения.

Как известно, случайная величина подчиняется нормальному закону распределения, когда она подвержена влиянию большого числа случайных факторов, что является типичной ситуацией в анализе данных. Поэтому нормальное распределение является хорошей моделью для многих реальных процессов.

В качестве исходных параметров будем рассматривать полученный радиус в фиксированный момент времени. Упорядочим данный массив (представлено в таблице 4.1).

Таблица 4.1

Упорядоченный массив размерностью 121

59,8964	59,8965	59,8965	59,8966	59,8967	59,8968	59,8968	59,8969	59,897	59,8971
59,8972	59,8972	59,8973	59,8973	59,8974	59,8975	59,8975	59,8975	59,8976	59,8977
59,8978	59,8978	59,8978	59,8978	59,8979	59,898	59,898	59,898	59,898	59,8981
59,8981	59,8981	59,8981	59,8981	59,8982	59,8982	59,8982	59,8983	59,8983	59,8984
59,8984	59,8985	59,8985	59,8985	59,8986	59,8986	59,8986	59,8986	59,8986	59,8986
59,8987	59,8988	59,8988	59,8988	59,8988	59,8989	59,8989	59,899	59,8991	59,8991
59,8992	59,8992	59,8992	59,8993	59,8993	59,8993	59,8993	59,8994	59,8994	59,8994
59,8994	59,8994	59,8995	59,8995	59,8995	59,8995	59,8996	59,8996	59,8997	59,8997
59,8998	59,8998	59,8998	59,8998	59,8999	59,8999	59,8999	59,8999	59,9002	59,9002
59,9002	59,9003	59,9003	59,9004	59,9004	59,9005	59,9005	59,9005	59,9007	59,9008
59,9008	59,9009	59,9009	59,901	59,9011	59,9011	59,9012	59,9013	59,9014	59,9014
59,9015	59,9017	59,9017	59,9017	59,902	59,9023	59,9028	59,9029	59,9033	59,9033
59,9035									

Считается, что лучшим объемом выборки для качественной обработки данных является 120, поэтому выбрана близкая величина к данной.

Установим размер выборки:

(4.10)

Тогда длина частичного интервала:

(4.11)

Формула Стерджесса дает длины частичных интервалов, которые должны "накрыть" исходный интервал. Округляя до 4-го знака после запятой, получим

(4.12)

Сделаем равные отступы от исходных границ интервала. На отрезке [59,8964; 59,9035] разобьем массив на 8 интервалов длины

(4.13)

Для построения гистограммы относительных частот составим таблицу 4.2

Таблица 4.2

Вспомогательная таблица

№	Левая граница	Правая граница	Центр интервала	Частота	Относительная частота	Гистограмма
1	59,89638	59,89728	59,897	13	0,107438	119,3756
2	59,89728	59,89818	59,898	21	0,173554	192,8375
3	59,89818	59,89908	59,899	25	0,206612	229,5684
4	59,89908	59,89998	59,9	29	0,239669	266,2994
5	59,89998	59,90088	59,9	13	0,107438	119,3756
6	59,90088	59,90178	59,901	13	0,107438	119,3756
7	59,90178	59,90268	59,902	2	0,016529	18,36547
8	59,90268	59,90358	59,903	5	0,041322	45,91368

Несмещенную оценку математического ожидания (выборочную среднюю) определим как

(4.14)

Подставим значение в формулу

(4.15)

Для нахождения оценок дисперсии, асимметрии, эксцесса и коэффициента вариации составляем таблицу 4.3

Таблица 4.3

Вспомогательная таблица

Номер частичного интервала	(центр частичного интервала)		Сумма вариант частичного интервала		Слагаемые суммы для дисперсии	Слагаемые суммы для ассиметрии	Слагаемые суммы для эксцесса
1	59,9	0,000	13	-0,03	0,000	0,000	0,000
2	59,9	0,000	21	-0,03	0,000	0,000	0,000
3	59,9	0,000	25	-0,01	0,000	0,000	0,000
4	59,9	0,000	29	0,01	0,000	0,000	0,000
5	59,9	0,000	13	0,02	0,000	0,000	0,000
6	59,9	0,000	13	0,03	0,000	0,000	0,000
7	59,9	0,000	2	0,01	0,000	0,000	0,000
8	59,9	0,000	5	0,02	0,000	0,000	0,000

Выборочная дисперсия равна

(4.16)

Выборочное среднеквадратическое отклонение

(4.17)

Несмещенная оценка дисперсии

(4.18)

Несмещенная оценка СКО

(4.19)

Показатель асимметрии – третий центральный момент, отнесенный к кубу выборочного СКО

(4.20)

Показатель эксцесса – четвертый центральный момент, отнесенный к квадрату выборочной дисперсии, за вычетом 3-х

(4.21)

Коэффициент вариации

(4.22)

Для построения графика плотности нормального распределения составляется таблица 4.4

Таблица 4.4

Вспомогательная таблица

Номер частичного интервала	(центр частичного интервала)			*– плотность нормального распределения*	Гистограмма (для сравнения с плотностью)
1	59,897	0,000	-1,51	80,766	119,38
2	59,898	0,000	-0,94	161,25	192,84
3	59,899	0,000	-0,37	233,78	229,57
4	59,9	0,000	0,19	246,15	266,3
5	59,9	0,000	0,76	188,22	119,38
6	59,901	0,000	1,32	104,52	119,38
7	59,902	0,000	1,89	42,147	18,365
8	59,903	0,000	2,45	12,343	45,914

В точке максимума

(4.23)

(4.24)

В точке перегиба

(4.25)

(4.26)

(4.27)

Для построения графиков нормального и эмпирического распределения составляется таблица 4.5

Таблица 4.5

Вспомогательная таблица

№	граница интервала	нормальное распределение	эмпирическая функция
1	59,8964	0,0369	0,000
2	59,8973	0,1108	0,1074
3	59,8982	0,2557	0,281
4	59,8991	0,4637	0,4876
5	59,9	0,6824	0,7273
6	59,9009	0,8508	0,8347
7	59,9018	0,9458	0,9421
8	59,9027	0,985	0,9587
9	59,9036	0,9969	1

Имеется 1 интервал, содержащий менее 5-и точек (см. таблицу 4.3). Тогда объединим этот интервал с соседним. Получим 7 интервалов.

Для применения критерия Пирсона составляется таблица 4.6

Таблица 4.6

Вспомогательная таблица

Частота (число точек на частичном интервале)	– левая граница частичного интервала	– правая граница частичного интервала	$\frac{}{}$	$\frac{}{}$	значение функции Лапласа на левой границе	значение функции Лапласа на правой границе	разность значений функции Лапласа	– теоретическая частота по Лапласу	слагаемые статистики Пирсона
13	59,896	59,897	-1,79	-1,22	-0,5	-0,3892	0,1108	13,405	0,012
21	59,897	59,898	-1,22	-0,66	-0,3892	-0,2443	0,1449	17,531	0,686
25	59,898	59,899	-0,66	-0,09	-0,2443	-0,0363	0,208	25,17	0,001
29	59,899	59,89	-0,09	0,47	-0,0363	0,1824	0,2187	26,466	0,243
13	59,89	59,901	0,47	1,04	0,1824	0,3508	0,1684	20,38	2,673
13	59,901	59,902	1,04	1,61	0,3508	0,4458	0,095	11,493	0,198
7	59,902	59,904	1,61	2,74	0,4458	0,5	0,0542	6,555	0,03
3,843									

На уровне значимости *0,1* для числа степеней свободы *7 - 3 = 4* по таблицам получим Так как ,то гипотеза о нормальном распределении генеральной совокупности не отвергается.

Доверительный интервал для матожидания

((4.28)

(4.29)

(4.30)

Для доверительной вероятности получаем . Длина интервала . Для и доверительной вероятности получаем . Длина интервала .

Доверительный интервал для СКО

((4.30)

(4.31)

Полученная значимость по Колмогорову составляет 0,688794. Так как 0,688794>0,05, следовательно, шероховатость распределена по нормальному закону.

Ввиду очень малой значимости остаточная шероховатость явно неравномерная, зато и по Пирсону (χ^2) и, тем более, по Колмогорову шероховатость распределена по нормальному закону.

4.3.Выводы по разделу

В данном разделе показано, что в результате работы была введенна поправка – линейная кубическая вязкость, которая является устойчивой по Ляпунову. Также анализ случайности шероховатости детали показал, что она подчинена нормальному закону распределения по критерию Колмогорова, а, следовательно, является случайной.

Данные исследования получены с помощью программы, разработанной в Microsoft Visual Studio. Результаты работы программы приведены в таблицах 4.7, 4.8.

Таблица 4.7

Пример выходного результата программы

Шаг по времени 0,00277778 сек. Обрабатываемая деталь разделена на 120 секторов Предполагаемая продолжительность подачи резца -1,875 сек.						
Итерация	1	оборот	0,0083	Есть контакт	d =	-0,000312567
Итерация	4	оборот	0,0333	Нет контакта	d =	0,000523299
Итерация	110	оборот	0,9167	Есть контакт	d =	-0,00120523

Таблица 4.8

Пример выходного результата программы

Контур детали в конце процесса									
Итерация 1237, оборот 10.3083, время 3.43611 сек									
0.	0 °	радиус	59.899923 мм	глубина	-0.000009	мм	сила	0	Н
1.	3 °	радиус	59.899615 мм	глубина	0.0000013	мм	сила	0	Н
2.	6 °	радиус	59.898932 мм	глубина	0.0000041	мм	сила	0.001	Н

В приложении 1 приведен текст программы.

ЗАКЛЮЧЕНИЕ

Рассмотрены современные способы проектирования циклов шлифования, и наиболее важные результаты исследования в этой области. Рассмотрена оптимизация классической теории проектирования циклов шлифования на основе фактических параметров технологической системы.

Сформирована задача оптимизации методики проектирования цикла шлифования, которая заключается в повышении эффективности процесса обработки за счет формирования цикла шлифования на основе фактических параметров технологической системы.

Получена математическая модель, которая с учетом предложенных допущений, позволила сформировать основу алгоритма проектирования цикла шлифования.

Найдена фундаментальная нормированная матрица решений. Произведено обезразмеривание параметров, которое устанавливает подобие между переменными. Произведенная минимизация количества параметров улучшила быстродействие расчетов, осуществляемых в программе.

Сделано разбиение, выбирается шаг по времени, учитывающее колебания, возникающие в процессе шлифования.

В результате работы была введенна поправка – линейная кубическая вязкость, которая является устойчивой по Ляпунову. Также анализ случайности шероховатости детали показал, что она подчинена нормальному закону распределения по критерию Колмогорова, а, следовательно, является случайной.

В работе был изучен процесс проектирования циклов шлифования, рассмотрена и описана математическая постановка задачи. Исследованы колебания, возникающие в процессе, введена линейная кубическая вязкость. Полученный результат программы, говорит о том, что модель является приближенной реальному процессу.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Абразивная и алмазная обработка материалов: Справочник / Под. ред. А. Н. Резникова. М.: Машиностроение, 1977. 391 с.

2 Генис, Б.М. Шлифование на круглошлифовальных станках / Б.М. Генис, Л.Ш. Доктор, В.С. Терган // Учебное пособие. – М.: Высшая школа. – 1965 г.236 с., ил.

3 Гузеев, В.И. Наукоемкий процесс моделирования цикла врезного шлифования цилиндрических заготовок / В.И. Гузеев, А.Х. Нуркенов, А.В. Гереншгейн // Наукоемкие технологии в машиностроении. Том 10. – М.: Машиностроение, 2014. Стр. 9 – 13.

4 Дьяконов, А.А. Комплексное моделирование процесса плоского шлифования периферией круга / А.А. Дьяконов, Л.В. Шипулин // Наукоемкие технологии в машиностроении. – 2015. – №6(24). – С. 14–18.3.

5 Евтухов, В.Г. Моделирование процесса круглого врезного шлифования // Вестник СумДУ. – 2011. — № 1. – С. 124 – 133.

6 Исаков, Д.В. Проектирование производительных шлифовальных операций на основе расчетного определения эксплуатационных показателей шлифовальных кругов // Металлообработка. Том 2. – С.: Политехника, 2009. Стр. 9 – 15.

7 Калинин, Е.П. Теория и практика управления производительностью шлифования без прижогов с учетом затупления инструмента. – СПб.: Изд-во политехн. ун-та, 2009. – 358 с.

8 Кошин, А.А. Применимость параллельных вычислительных процессов в расчетных задачах технологии машиностроения / А.А. Кошин, А.А. Дьяконов // Технология машиностроения. 2010 . № 1 С. 45 – 47.

9 Лурье, Г.Б. Шлифование металлов. М.: Машиностроение, 1969. 172 с.

10 Медведев, А.С. Математическая модель и компьютерное моделирование процесса тонкого шлифования на координатно-шлифовальном станке / А.С. Медведев, В.Е. Лысов // Компьютерная интеграция производства и ИПИ технологии: Сб. ст. Всерос. науч.-практ. конф. Оренбург: ИПК ОГУ, 2007. С. 244 – 251.

11 Михелькевич, В.Н. Автоматическое управление шлифованием. М.: Машиностроение, 1975. 304 с.

12 Нуркенов, А.Х. Методика моделирования круглого врезного шлифования с учетом фактической жесткости технологической системы / А.Х. Нуркенов, А.В. Гереншгейн, Н. Машрабов // Материалы L1V международной научно-технической конференции “Достижения науки – агропромышленному производству”. Часть 111. /под редакцией докт. техн. Наук.

13 Переверзев, П.П. Моделирование технологических ограничений при оптимизации автоматических циклов шлифования // Вестник Южно-Уральского государственного университета: Сер. Машиностроение. 2012. № 12, Вып. 19. С. 165 – 168.

14 Шипулин, Л.В. Совершенствование методики проектирования операций плоского шлифования периферией круга на основе комплексного моделирования процесса // Современные проблемы науки и образования. 2013. № 2.

```
#include "stdafx.h"
#include <fstream>
#include <iostream>
#include <string>
#include <cmath>
#include <ctime>
#include <algorithm>
#include <conio.h>

double M_PI = std::acos(-1.0);
#include <stdio.h>
double ktf(double h);
double *formul(int n, double h, double(*func)(double), double R);
int endic(int n);
int sector(int n);
double publ(long N, double *ro, double *strang, double *h,
std::ofstream &ost);
char *krap(double *ss, int *kr, char *fr = 0);
char *krap(double ss, int *kr, char *fr = 0);
char *krap(double x, int p, int q, char ch = 0);
int matrkoef(double *A, double *P, double *P0, double dt, int n =
2);
double matrpubl(double *mtr, int n, std::ofstream &ost);
double prod(double *P, double *b, double *x, int n = 2);
double detal(double fee);
double rand(double a, double b);
double typtab(double x, int p);
double eq(double *x, double *p, int n = 2);
double podacha(double h);

int main()
{ try
     {
          double a, b, M, cd, vd, om, r, e, v, R, Rk, sgm, bh, Ka,
Kb, Kz, Vs,
          mu, press, krak, rnd, eps = 1e-5, per = 0.25, aa, bb;
          long oter, otermax = 100000, iter;
          double t, dt, l, fee, *q, h, h0, min, max, Qmax, Qmin, b0,
v0, s,
               *ro = 0, *syla = 0, *glub = 0, ks;
          double x[2], x0[2], Q[2], A[4], P[4], P0[4], F[2], *vrem;
          int i, j, k, kk, flag[9], fl = 1, vv = 1, n, nn = 2, jj =
0;
          long MN, ot = 0, tot = 0, it = 0, knt = 0;
          char *ss[] = { "сектор", "сектора", "секторов" };
          static long N, bob = 0, eob = 0, dob = 0;
          srand(time(NULL));
          std::ofstream ost("out");
          if (!ost)
          {
```

```
                throw std::string("ERROR! Can't open file for
writing.");
           }
           {
                std::ifstream ist("data");
                std::string tmp;
                if (!ist)
                {
                     throw std::string("ERROR! Can't open file for
reading.");
                }
                ist >> tmp >> a;
                ist >> tmp >> b;
                ist >> tmp >> M;
                ist >> tmp >> cd;
                ist >> tmp >> vd;
                ist >> tmp >> om;
                ist >> tmp >> R;
                ist >> tmp >> r;
                ist >> tmp >> e;
                ist >> tmp >> v;
                ist >> tmp >> dt;
                ist >> tmp >> Rk;
                ist >> tmp >> sgm;
                ist >> tmp >> bh;
                ist >> tmp >> Ka;
                ist >> tmp >> Kb;
                ist >> tmp >> Kz;
                ist >> tmp >> Vs;
                ist >> tmp >> mu;
                ist >> tmp >> press;
                ist >> tmp >> oter;
                ist >> tmp >> otermax;
                ist >> tmp >> iter;
                ist >> tmp >> krak;
                ist >> tmp >> dob;
                ist >> tmp >> rnd;
                ist >> tmp >> eps;
                ist >> tmp >> h;
                ist >> tmp >> n;
           }
           for (int i = 0; i < 9; i++)
           {
                flag[i] = 0;
           }
           if (om < 0.)
           {
                om = -om;
                flag[7] = 1;
           }
           if (n < 0)
```

```
{
     n = -n;          // Глубины и силы
     flag[5] = 1;
}
if (R < 0.)
{
     R = -R;           // Пишем исходные параметры
     flag[1] = 1;
}
if (bh < 0.)
{
     bh = -bh;
     flag[8] = 1;    // Пишем прогноз на оставшийся путь
}
if (dob < 0)
{
     dob = -dob;      // Записываем только оборот
     bob = 1;
}
if (r < 0.)
{
     r = -r;          // Пучать на начальной конфигурации
     flag[0] = 1;
}
if (M < 0.)
{
     M = -M;          // Регистрация отскоков и столкновений
     flag[2] = 1;
}
if (Rk < 0.)
{
     Rk = -Rk;        // Пишем координаты детали
     flag[3] = 1;
}
if (otermax < 0)
{
     otermax = -otermax;    // Пишем подачу
     flag[4] = 1;
}
if (krak < 0)
{
     krak = -krak;          // Пишем расстояние от реза до
детали
     flag[6] = 1;
}
aa = sgm*bh*om*2.*M_PI*R*Ka / Vs;      // Коэффициент при  t
t = sqrt(2.*R*Rk / (R + Rk));
bb = sgm*bh*Kz*t*Kb;                 // Коэффициент при √t
ktf(aa);
ktf(bb);
ktf(0.);
```

```
          if (flag[5])    // Глубины и силы
          {
               ost << "Техническая формула : F = " << aa << " * h + "
<< bb << " * sqrt(h) ";
               ost << "(F - -ньют., h - мм)." << std::endl;
          }
          q = formul(n, h, ktf, R);
          ktf(-5000.);
          v = v* om;           // был мм/об => мм/мин
          v = v / 60.;          // а теперь мм/сек
          om = om* M_PI / 30.;   // об/мин переводим в рад/сек
          t = M*R*om*om;
          aa = q[0] / t;
          bb = q[1] / t;
          mu *= R*R*om / M;
          ktf(aa);
          ktf(bb);
          ktf(0.);
          h /= R*(double)n;
          if (flag[5])   // Глубины и силы
          {
               ost << "Безразмерна формула : F' = " << aa << " * h +
" << bb << " * sqrt(h) ";
               ost << "(F'и h'- безразм.)." << std::endl;
               ost << "F = " << t << " * F', h = " << R << " * h'" <<
std::endl;
               ost << "     безразмер          безразмер       мм";
               ost << "                   Ньют" << std::endl;
               for (i = 0, aa = h; i < n; i++, aa += h)
               {
                    bb = ktf(aa);
                    ost << " h = " << krap(aa, 1, 8) << "  FЎ§a = "
<< krap(bb, 1, 8) << "  h = " << krap(R*aa, 1, 8) << "  F = " <<
krap(t*bb, 1, 8) << std::endl;
               }
          }
          cd /= 1000.;
          vd /= 1000.;
          min = (r - e - R - rnd);
          if (min < 0.)
          {
               ost << "   Требуемого радиуса детали невозможно
достигнуть. Надо ";
               ost << "увеличть разность между начальным радиусом и
эксцентри";
               ost << "ситетом с шероховатостью не менее, чем" <<
std::endl << "на " << -min << " мм." << std::endl;
               ktf(-5000.);
               goto ABC;
          }
          ks = M*R*om*om;
```

```
          t = cd / M;
          s = vd / M / 2.;
          t -= s*s;
          if (t < 0)
          {
               ost << " , В системе нет колебаний. Велика линейная
вязкость" << std::endl;
               ktf(-5000.);
               goto ABC;
          }
          t = sqrt(t);
          t = 2.*M_PI / t;              // Период собсвтенных
колебаний детали
          dt = t / dt;                  // Предварительный шаг по
времени
          s = 1. + 2.*M_PI / om / dt;

          k = (int)s;
          n = sector(k);              // Число элементарных углов
окружности
          dt = 2.*M_PI / om / (double)n;  // Уточненный шаг по
времени
          ost << "Шаг по времени " << dt << " сек." << std::endl;
          i = endic(n);
          ost << "     Обрабатываемая деталь разделена на  " << n <<
ss[i] << std::endl;
          if (flag[1])     //  Пишем исходные параметры
          {
               ost << " □ Шаг по времени          dt  = " << dt << "
сек." << std::endl;
               ost << " Период колебаний детали dt' = " << t << "
сек." << std::endl;
               ost << "  dt'/dt = " << t / dt << "." << std::endl;
          }                                                  //
Обезразмерим

     // Обезразмеривание           // A - матрица линеной системы
          a /= R;                             // P(t) -
фундаментальная матрица
          b /= R;                             // при t = dt
          cd /= M*om*om;                      //
          vd /= M*om;                         //      dt
          r /= R;                             // P0 = ∫P(dt-t)*dt
          e /= R;                             //      0
          v /= R;                             //
          press /= R;
          krak /= R;
          rnd /= R;
          v /= om;
          dt *= om;
          detal(e);
```

```
          detal(r);
          if (a < b + detal(0.) || flag[7])
               a = b + detal(0.);
          for (i = 0; i < nn*nn; i++)
               A[i] = P[i] = P0[i] = 0.;
          for (i = 0; i < nn; i++)
               x[i] = x0[i] = Q[i] = F[i] = 0.;
          A[0] = -vd;              // x' = A*x + Q
          A[1] = 1;              // x = P*x0 + P0*Q
          A[2] = -cd;              // Q - сила резания в начале шага по
времени
          ro = new double[n];     // x' = -vd*x  + y
          syla = new double[n];  // y' = -cd*x      - Q
          glub = new double[n];  // Q = Ka*h + Kb*√h
          N = (long)n;             // (Ka + cd)*h + Kb*√h + cd*(a - ro)
          ro[0] = om;
          ro[1] = R;
          ro[2] = M;
          publ(N, ro, 0, 0, ost);
          for (i = 0, fee = 0.; i < n; i++, fee += dt)
          {
               ro[i] = detal(fee) + rnd*rand(-1., 1.);
               syla[i] = glub[i] = 0.;
          }
          if (flag[0])    // печать начальной конфигурации
               publ(-1, ro, syla, glub, ost);
          if (flag[1])
          {                 // Пишем исходные параметры
               l = dt / om;
               ost << "   Число элементарных участков " << n  <<
std::endl;
               ost << "   Реальный шаг по времени " << l << " сек."
<< std::endl;
               ost << " Все длины отнесены к требуемому радиусу " <<
R << "м детали." << std::endl;
               ost << " Все массы отнесены к массе " << M << " кг
детали." << std::endl;
               ost << " Время умножено на угловую скорость " << om <<
" рад/сек детали." << std::endl;
               ost << "      Безразмерные параметры." << std::endl;
               ost << " Координата режущей кромки резца           "
<< a << std::endl;
               ost << " Координата оси детали                     "
<< x[0] << std::endl;
               ost << " Жесткость пружины детали                  "
<< cd << std::endl;
               ost << " Вязкость пружины детали                   "
<< vd << std::endl;
               ost << " Начальный радиус детали                   "
<< r << std::endl;
```

```
                ost << " Начальный эксцентриситет детали           "
<< e << std::endl;
                ost << " Скорость подачи правого конца             "
<< v << std::endl;
                ost << " Коэффициент силы резания Ka               "
<< aa << std::endl;
                ost << " Коэффициент силы резания Kb               "
<< bb << std::endl;
                ost << " Отладочный коэффициент                    "
<< mu << std::endl;
                ost << " Безразмерный шаг по времени               "
<< dt << std::endl;
                ost << " Относительна точность подачи              "
<< eps << std::endl;
                t = 4.*cd - vd*vd;
                if (t < eps)
                {
                     ost << "  Слишком большая вязкость пружины." <<
std::endl;
                     ktf(-5000.);
                     goto ABC;
                }
                t = sqrt(t);
                ost << " Частота колебаний детали                  "
<< t << std::endl;
                ost << std::endl << "          Матрица A" <<
std::endl;
                for (i = 0; i < nn; i++)
                {
                     ost << "   є";
                     for (j = 0; j < nn; j++)
                          ost << krap(A[i*nn + j], 2, 8);
                     ost << "   є" << std::endl;
                }
           }
           matrkoef(A, P, P0, dt);
           v *= dt;    // v*dt
           v0 = v;
           eps *= v0;

           if (flag[1] == 1)    // Пишем исходные параметры
           {
                ost << std::endl << "                    -1" <<
std::endl;
                ost << "          Матрица  A" << std::endl;
                matrpubl(A, nn, ost);
                ost << std::endl << "          Матрица P" <<
std::endl;
                matrpubl(P, nn, ost);
                ost << std::endl << "          Матрица P0" <<
std::endl;
```

```
                matrpubl(P, nn, ost);
                ost << std::endl;
           }
           /*
           {
                std::ifstream in_d("detal");
                if (!in_d)
                {
                     throw std::string("ERROR! Can't open detal file
for reading.");
                }
                in_d >> t;
                t += 0.1;  // число пар данных
                podacha(t);
                kk = (int)t;
                vrem = new double[kk];
                l = a - b - 1.; // - press; расстояние от резца до
нужной поверхности
                podacha(v0);
                for (i = 0; i < kk; i++)   // остаток пути резца
                {
                     double tmp_d;
                     in_d >> tmp_d;
                     t = l * tmp_d;
                     podacha(t);
                     vrem[i] = t;
                }
                for (i = 0, min = 0.; i < kk + 1; i++)   // подача
                {
                     double tmp_d;
                     in_d >> tmp_d;
                     t = v0 * tmp_d;
                     podacha(t);
                     if (i && i < kk)
                          min += (vrem[i - 1] - vrem[i]) / t;
                }
           }
           if (vrem)
                delete[] vrem;*/
           min *= dt / om;
           ost << "     Предполагаемая продолжительность подачи резца
- ";
           ost << min << " сек." << std::endl;
           if (flag[4])   // Пишем подачу
           {
                for (s = l, h = 0.1*l; s > -3.*h; s -= h)
                {
                     t = podacha(s);
                     t *= R*om / dt;
                     ost << " s = " << s << ", v = " << t <<
std::endl;
```

```
                }
            }
            //  goto ABC;
            // ----------
            Qmax = Qmin = 0.;
            max = 1. + 2.*press;
            min = t = 0.;
            N = 0;
            for (j = 0; max - 1. > press; N++, j = (j + 1) % n, a -=
v0)
            {
                v0 = podacha(a - b - 1.);
                eq(x0, x);  // x0 <- x
                q[0] = x[0] + ro[j] - a;
                q[1] = ktf(x[0] + ro[j] - a);
                eq(Q, q);
                if (flag[3])   // Пишем координаты детали
                    ost << N << ". x = " << (x[0] - b) << ", r0 = "
<< ro[j] << ", a = " << a << "; h = " << Q[0] << ", v = " << (x[1] -
vd*x[0]) << '.' << std::endl;
                t = -vd*x[0] + x[1];
                t = mu*t*t*t;
                F[1] = -Q[1] - t;
                prod(P, x0, x);
                prod(P0, F, x, -2);
                if (Q[0] > 0.)
                    ro[j] -= Q[0];
                if (flag[6])    // Пишем координаты детали, резца и
расстояние от
                {               // резца детали
                    bb = ks*Q[1];
                    t = R*(x[0] + ro[j]);
                    l = R*a;
                    ost << N << ". " << krap(R*x[0], 2, 7) << ", " <<
krap(t, 2, 7) << ", " << krap(l, 2, 7) << ", " << krap(l - t, 2, 7)
<< ", " << krap(bb, 2, 7) << '.' << std::endl;
                }
                if (fabs(x[0]) > krak)
                {
                    t = (double)N / (double)n;
                    l = (double)N*dt / om;
                    ost << std::endl << "  Деталь сломалась на  " <<
N << "-й итерации" << std::endl;
                    ost << "                    на  " << l << "-й
секунде" << std::endl;
                    ost << " на  " << t << "-м обороте" << std::endl;
                    t = ro[j] * R;
                    bb = -ks*q[1];
                    ost << " r = " << t << " мм, глубина  " << (Q[0]
* R) << " мм, центр " << ((x[0] - b)*R) << " мм, ";
                    ost << "сила  " << bb << " Ньют." << std::endl;
```

```
                    t = (a - b - 1.)*R;
                    v0 = podacha(a - b - 1.);
                    v0 *= R*om;
                    ost << " Остаток пути " << t << " мм, подача " <<
v0 << " мм/сек." << std::endl;
                    ktf(-5000.);
                    goto ABC;
               }
               if (!N)
               {
                    Qmax = Qmin = Q[1];
                    ot = tot = N;
               }
               else if (Q[1] > Qmax)
               {
                    Qmax = Q[1];      // Определение наибольшей силы
резания
                    ot = N;
               }
               else if (Q[1] < Qmin)
               {
                    Qmin = Q[1];
                    tot = N;
               }
               if (Q[0] > 0.)
               {
                    if (fl > 0)
                    {
                         t = (double)N / (double)n;
                         fl = 0;
                         if (flag[2])  // Регистрация отскоков и
столкновений
                              ost << " Итерация " << N << ", оборот "
<< krap(t, 3, 4) <<". Есть контакт! d = " << (-R*q[0]) << "." <<
std::endl;
                         knt++;
                    }
               }
               else
               {
                    if (!fl)
                    {
                         t = (double)N / (double)n;
                         fl = 1;
                         if (flag[2])   // Итерация отскоков и
столкновений
                              ost << " Итерация " << N << ", оборот "
<< krap(t, 3, 4) << ". Нет контакта . d =  " << (-R*q[0]) << "." <<
std::endl;
                    }
               }
```

```
                glub[j] = Q[0];             // глубина шлифовки
                syla[j] = Q[1];             // сила шлифовки
                for (i = 1, min = max = ro[0]; i < n; i++)
                {
                     if (ro[i] > max)
                          max = ro[i];
                     else if (ro[i] < min)
                          min = ro[i];
                }
                if (v0 < eps && vv)
                {
                     vv = 0;
                     v = v0 = 0.;
                     t = (double)N / (double)n;
                     l = (double)N*dt / om;
                     ost << "     " << N  << "-я итерация, " << t <<
"-й оборот,  " << l << "-я секунда." << std::endl;
                     ost << "          Подача резца прекращена." <<
std::endl;
                     if (flag[8])  // Прогноз наоставшийся путь
                     {
                          s = podacha(-55.);
                          ost << " Предполагаемое число итераций до
прекращения подачи";
                          ost << " " << (long)s << "." << std::endl <<
" Безразмерны остаток пути " << a - 1 << "." << std::endl;
                     }
                }
                if (N > eob)
                     eob = N;
                if (N < eob)
                {
                     ost << "   Eob = " << eob << ",  N = " << N <<
"." << std::endl;
                     break;
                }
                if (dob)
                {
                     i = N / (long)n;
                     if (N%n == 0 && i == dob)
                     {
                          publ(N, ro, syla, glub, ost);
                          if (bob)
                               dob = 0;
                     }
                }
                if (bob && N%n == 0)
                {
                     t = ro[0] * R;
                     i = (int)(N / n);
```

```
                    ost << "  " << i << "-й оборот,  Ro = " << t <<
", h = " << (Q[0] * R) << std::endl;
                }
                if (_kbhit() && _getch() == 27 || oter > 0 && N >=
oter ||
                    oter < 0 && N >= otermax)
                    break;
          }
          publ(-N, ro, syla, glub, ost);
          Qmax *= ks;
          ost << " Наибольшая сила резания " << Qmax << " Ньют на "
<< ot << "-й итерации ";
          ot /= n;
          ost << "на  " << ot << "-м обороте." << std::endl;
     ABC:
          detal(-1.);
          ktf(-5000.);
          podacha(-100.);
          if (ro) delete[] ro;
          if (syla) delete[] syla;
          if (glub) delete[] glub;
     }
     catch (const std::string &str)
     {
          std::cerr << str << std::endl;
     }
     catch (...)
     {
          std::cerr << "Something bad happend." << std::endl;
     }
}

double ktf(double h)  // Сила резания. h - скорость срезания слоя
{
     static double A = -1., B = -1., mu = -1.;
     double F;
     if (A < -0.5)                 // .
     {                            // x = A·x + Q
          A = h;                     //                  -1
          return(0.);                // x = P(t)·x0 + A ·(P(t) - E)·Q
     }                            // h  - безразмерная глубина
резания
     if (B < -0.5)
     {
          B = h;
          return(0.);
     }
     if (mu < -0.5)
     {
          mu = h;
          return(0.);
```

```
     }                                  // Сила резания
     if (h < -1000.)                     // F = Ka·h + Kb·√h
     {                                  // mu - поджатие
          A = B = mu = -1.;
          return(0.);
     }
     if (h < 0.)
          F = -mu;
     else
          F = A*h + B*sqrt(h);
     return(F);
}

double *regres(int n, double *x, double *y, double *z)
{
     int i, j, k;
     double mx = 0., my = 0., mz = 0., dx = 0., dy = 0., kxy = 0.,
kxz = 0.,
          kyz = 0., u, v, w;
     static double kf[4];
     for (i = 0; i < n; i++)
     {
          mx += x[i];
          my += y[i];
          mz += z[i];
     }
     mx /= (double)n;
     my /= (double)n;
     mz /= (double)n;
     for (i = 0; i < n; i++)
     {
          u = x[i] - mx;
          v = y[i] - my;
          w = z[i] - mz;
          dx += u*u;
          kxy += u*v;
          kxz += u*w;
          dy += v*v;
          kyz += v*w;
     }
     dx /= (double)n;
     dy /= (double)n;
     kxy /= (double)n;
     kxz /= (double)n;
     kyz /= (double)n;
     w = dx*dy - kxy*kxy;
     u = kxz*dy - kyz*kxy;
     v = dx*kyz - kxy*kxz;
     kf[0] = u / w;
     kf[1] = v / w;
     kf[2] = mz - kf[0] * mx - kf[1] * my;
```

```
     for (kf[3] = 0., i = 0; i < n; i++)
          kf[3] += fabs(z[i] - kf[0] * x[i] - kf[1] * y[i]);
     kf[3] /= (double)n;
     return(kf);
}
double *formul(int n, double h, double(*func)(double), double R)
{
     static double q[2];
     int i;
     double t, x, *Fi, *u, *w, *f;
     t = h / (double)n;
     Fi = new double[n];
     u = new double[n];
     w = new double[n];
     for (i = 0, x = t; i < n; x += t, i++)
          Fi[i] = func(x);
     t /= R;
     for (i = 0, x = t; i < n; x += t, i++)
     {
          u[i] = x;
          w[i] = sqrt(x);
     }
     f = regres(n, u, w, Fi);
     q[0] = f[0];
     q[1] = f[1];
     if (Fi) delete[] Fi;
     if (u) delete[] u;
     if (w) delete[] w;
     return(q);
}


int endic(int n)
{
     int l, k, m;
     k = n % 10;
     l = (n / 10) % 10;
     if (l != 1 && k == 1)
          m = 0;      // o£®¤ , αβ®«
     else if (l != 1 && k > 1 && k < 5)
          m = 1;      // o£®¤λ, αβ®«
     else
          m = 2;      // o£®¤,  αβ®«®Ä
     return(m);
}

int sector(int n)
{
     int i;
     for (i = 0; n > 180; i++, n /= 2)
          if (n % 2)
```

```
                n /= 2;
        for (; 360 % n; n++);
        for (; i--; )
                n *= 2;
        return(n);
}

double publ(long N, double *ro, double *strang, double *h,
std::ofstream &ost)
{
        static double om, r, M;
        double ob, fee, df, max, min, t;
        static int n;
        int i = 0, j, l = 0, k = 0;
        if (!strang || !h)
        {
                n = (int)N;
                om = ro[0];
                r = ro[1];
                M = ro[2];
                M *= r*om*om;
                return(0.);
        }
        if (N == -1)
        {
                N = -N;
                i = 1;
        }
        if (N < -1)
        {
                N = -N;
                i = 2;
        }
        df = 360. / (double)n;
        ob = (double)N / (double)n;
        t = 2.0 * M_PI * (double)N / (double)n / om;
        if (i == 1)
                ost << std::endl << "          Первоначальный контур
детали" << std::endl;
        else if (i == 2)
                ost << std::endl << "  Контур детали в конце процесса " <<
std::endl;
        if (N > 0)
                ost << "  Итерация " << N << ", оборот " << ob << ", время
" << t << " сек." << std::endl;
        for (i = 0, fee = 0., min = max = ro[0]; i <= n; i++, fee +=
df)
        {
                j = i%n;
                ost << " " << i << ". " << krap(fee, 4, 4, 'ш') << "
радиус " << krap(ro[j] * r, 2, 6) << " мм ";
```

```
          if (N != -1)
               ost << "глубина  " << krap(h[j], 2, 6) << " мм  сила "
<< krap(strang[j], 3, 3) << " Н" << std::endl;
          if (ro[j] < min)
          {
               min = ro[j];
               l = i;
          }
          else if (ro[j] > max)
          {
               max = ro[j];
               k = i;
          }
     }
     ost << " Rmax = " << krap(max*r, 2, 6) << " мм (угол № " << k
<< "), Rmin = " << krap(min*r, 2, 6) << " мм (угол № " << l << ")."
<< std::endl;
     ost << "           Rmax - Rmin = " << ((max - min)*r) << "мм."
<< std::endl;
     ost << "              Rва = " << r << " мм." << std::endl;
     ost << " Rmax - Rва = " << ((max - 1.)*r) << " мм, Rва - Rmin =
" << ((1. - min)*r) << "  мм." << std::endl;
     t = fabs(max - 1.) + fabs(1. - min);
     return(t);
}

char *krap(double *ss, int *kr, char *fr)
{
     double eps = 1e-11;
     static char sp[10][71], *s;      // Точка десятичного
представления числа
     char st[21];                     // ss[i] будет находиться на
kr[i] - том
     int  i, j, k = 0, k0, l = 0;     // месте текстовой строки s.
Если число
     static int N = 0;                // kr[j] - отрицательно, то
число -kr[j]
                                           // определяет длину
текстовой строки.
     N = (N + 1) % 10;                 // Это число - последнее в
массиве kr.
     for (i = 0, s = sp[N]; i < 70; i++) // fr[i] определяет форму
представления
          s[i] = ' ';                  // числа (l, f, lf, d и
т.п.).
     s[i] = '\0';
     for (j = 0; kr[j] >= 0; j++)
     {
          k0 = k + l;            // следующая ячейка после
записанного числа
```

```
          if ((!fr || fr[j] != 'E' && fr[j] != 'e') && ss[j] > -eps
&& ss[j] < eps)
               ss[j] = 0.;
          if (!fr || fr[0] == ' ' || fr[j] == 'G')
               sprintf(st, "%lg", ss[j]);
          else if (fr[j] == 'F')
               sprintf(st, "%lf", ss[j]);
          else if (fr[j] == 'E')
               sprintf(st, "%le", ss[j]);
          else if (fr[j] == 'D')
               sprintf(st, "%ld", ss[j]);
          else if (fr[j] == 'g')
               sprintf(st, "%g", ss[j]);
          else if (fr[j] == 'f')
               sprintf(st, "%f", ss[j]);
          else if (fr[j] == 'e')
               sprintf(st, "%e", ss[j]);
          else if (fr[j] == 'd')
               sprintf(st, "%d", ss[j]);
          for (i = 0; st[i] != '.' && st[i] != '\0' && st[i] != ' ';
i++);
          l = kr[j] - i;
          if (k0 >= l)     // Между данным числом и предыдущим нет
промежутка
          {
               if (!j)
                    l = k0;
               else if (fr)
                    l = k0 + 1; // Отступаем вправо от предыдущего
числа на одну ячейку
               else if (l <= kr[j - 1] + 1)
               {
                    l = kr[j - 1] + 1;  // Цифры до точки предыдущего
числа сохраняем
                    if (s[l - 1] != ' ' && s[l] != ' ' && s[l - 2] <
'9' && s[l] >= '5')
                         s[l - 2] += 1;
                    s[l - 1] = ' ';
               }
               else if (l == kr[j - 1] + 2)
               {
                    if (s[l - 1] > '5' && s[l - 3] < '9')
                         s[l - 3] += 1;
                    s[l - 2] = s[l - 1] = ' ';
               }
               else
               {
                    if (s[l - 1] > '5' && s[l - 2] < '9')
                         s[l - 2] += 1;
                    s[l - 1] = ' ';
               }
```

```
          }
          for (k = 0; st[k] && st[k] != ' '; k++)
               s[k + l] = st[k];
          s[k + l] = ' ';
     }
     for (i = k + l; i < -kr[j]; i++)
          s[i] = ' ';
     while (i > -kr[j])
          i--;
     s[i] = '\0';
     if (fr[0] == ' ' && (fr[1] == ',' || fr[1] == '.'))
     {                                        //  после последней цифры
          for (i = 0; s[i] != '\0'; i++);     // проставляется символ
fr[1],
          k = i;                                // но число пробелов
вместо
          for (i--; s[i] == ' '; i--);       // недостающих цифр
сохраняется.
          s[k + 1] = '\0';
          s[k] = ' ';
          s[i + 1] = fr[1];
     }
     return(s);
}
char *krap(double ss, int *kr, char *fr)
{
     double eps = 1e-11, x;
     static char sp[10][71], *s;       // Точка десятичного
представления числа
     char st[51], ch = ' ';           // ss будет находиться на
kr[0] - том
     int  i, k = 0, l = 0;            // месте текстовой строки s.
Если число
     static int N = 0;                // kr[1] - отрицательно, то
число -kr[1]
                                           // определяет длину
текстовой строки.
     N = (N + 1) % 10;                    // Это число - последнее
в массиве kr.
     for (i = 0, s = sp[N]; i < 70; i++) // fr[0] определяет форму
представления
          s[i] = ' ';                     // числа (l, f, lf, d и
т.п.).
     s[i] = '\0';
     //  x = (ss < 0.) ? -ss : ss;
     if (kr[1] > 0)
          kr[1] = -kr[1];
     k = -kr[1] - kr[0];
     if (k > 1)
          k--;
     /*
```

```
     for(i = 0; i < k; i++)
     x *= 10.;
     x = (double)((long)(x + 0.5));
     for(i = 0; i < k; i++)
     x /= 10.;
     ss = (ss < 0.) ? -x : x;
     */
     ss = typtab(ss, k);

     if ((!fr || fr[0] != 'E' && fr[0] != 'e') && ss > -eps && ss <
eps)
          ss = 0.;
     if (!fr || fr[0] == 'G')
     {
          sprintf(st, "%.6lf", ss);
          if (ss > 0.0001 || ss < -0.0001 || ss < eps && ss > -eps)
          {
               for (i = 0; st[i] != '.'; i++);
               i += k + 1;
               st[i] = '\0';
               while (st[--i] == '0')
                    st[i] = ' ';
          }
          if (st[i] == '.')
               st[i] = ' ';
     }
     else if (fr[0] == 'F')
          sprintf(st, "%lf", ss);
     else if (fr[0] == 'E')
          sprintf(st, "%le", ss);
     else if (fr[0] == 'D')
          sprintf(st, "%ld", (int)(ss + 0.5));
     else if (fr[0] == 'g')
          sprintf(st, "%g", ss);
     else if (fr[0] == 'f')
          sprintf(st, "%f", ss);
     else if (fr[0] == 'e')
          sprintf(st, "%e", ss);
     else if (fr[0] == 'd')
          sprintf(st, "%d", (int)(ss + 0.5));
     else
     {
          if (ss < 0.0001 && ss > -0.0001 && (ss < -eps || ss > eps))
               sprintf(st, "%.6lf", ss);
          else
               sprintf(st, "%lg", ss);
          ch = fr[0];
     }
     for (i = 0; st[i] != '.' && st[i] != '\0' && st[i] != ' ';
i++);
     l = kr[0] - i;
```

```
     if (0 >= l)     // Целая часть не вмещается до точки
          l = 0;
     for (k = 0; st[k] && st[k] != ' '; k++)
          s[k + l] = st[k];
     s[k + l] = ch;
     for (i = k + l; i < -kr[1]; i++)
          s[i] = ch;
     while (i > -kr[1])
          i--;
     if (s[i] != '.' && s[i] >= '5')
     {
          if (s[i - 1] != '.' && s[i - 1] < '9')
               s[i - 1] += 1;
          else if (s[i - 1] == '.' && s[i - 2] < '9')
               s[i - 2] += 1;
     }
     s[i] = '\0';
     if (fr[0] == ' ' && fr[1] == '-')
     {
          for (i = 0; s[i] != '\0'; i++);
          for (i--; s[i] == ' '; i--)
               s[i] = '\0';
     }
     if (fr[0] == ' ' && (fr[1] == ',' || fr[1] == '.'))
     {                                        //  после последней цифры
          for (i = 0; s[i] != '\0'; i++);    // проставляется символ
fr[1],
          k = i;                              // но число пробелов
вместо
          for (i--; s[i] == ' '; i--);       // недостающих цифр
сохраняется.
          s[k + 1] = '\0';
          s[k] = ' ';
          s[i + 1] = fr[1];
     }
     return(s);
}

char *krap(double x, int p, int q, char ch)
{
     int     i = 0, j, k, flag = 1, rst = 0;
     double  w, z = 1., eps;          // p - резерв числа знаков до
запятой
     static char ss[5][71], *st;      // q - число знаков после
запятой
     static int n = 0;
     n = (n + 1) % 5;
     st = ss[n];
     w = x;
     if (w < 0.)
          w = -w;
```

```
     if (p < 1)
          p = 1;
     if (q < 0)    // если целая часть 0, то должно быть q значащих
цифр
     {
          q = -q;
          rst = 1;
     }
     for (i = 0; i < p + q + 1; i++)
          st[i] = ' ';
     for (; i < 70; i++)
          st[i] = '\0';
     for (i = 0, eps = 1.; i < q + 1; i++)  // Задаем точность: q+1-
й знак
          eps /= 10.;                          // после запятой
     for (k = 1; w > 10.*z - eps; k++)    // Подсчитываем число цифр
          z *= 10.;                            // до запятой
     if (k >= p)      // Если мало места для целой части, мы его
увеличиваем,
     {               // сохраняя длину дробной части. Если ch не
плюс, то
          i = k;         // длина дробной части уменьшается
соответственно.
          if (x < 0.)
               i += 1;
          if (ch != '+')
               q = (q > i - p) ? q - i + p : 0;
          for (j = 0, eps = 1.; j < q + 1; j++)
               eps /= 10.;
          p = i;
     }
     i = p - k;
     if (x < 0. && (rst || w > eps))
          st[i - 1] = '-';
     if (i >= p)
          st[i++] = '0';
     for (; i < p; i++)
     {
          for (k = 0; w > z - eps; k++)
               w -= z;
          st[i] = '0' + k;
          if (k < 9)          // Не все цифры - девятки, поэтому
округление не
               flag = 0;        // увеличит длину массива спереди
          if (k)
               rst = 0;
          z /= 10.;
     }
     if (w > 5.*eps || rst)
     {
          st[p++] = '.';
```

```
          for (i = p; i < p + q; i++)
          {
               for (k = 0; w > z - eps; k++)
                    w -= z;
               st[i] = '0' + k;
               if (k < 9)
                    flag = 0;
               if (k)
                    rst = 0;
               if (!k && rst && eps > 1e-16)
               {
                    eps /= 10.;
                    q++;
               }
               z /= 10.;
          }
          for (i = p; st[i] != '\0'; i++);
          if (flag && w > 5.*eps)
          {
               for (i--; i && st[i] != '.'; i--)
                    st[i] = ' ';
               for (; i && i && st[i] != '-' && st[i] != ' '; i--)
                    st[i] = '0';
               st[i + 1] = '1';
          }
          else if (w > 5.*eps)
          {
               for (i--; i && st[i] == '.' || st[i] == '9'; i--)
                    if (!flag)
                    {
                         if (st[i] == '.')
                              flag = 1;
                         st[i] = ' ';
                    }
                    else
                         st[i] = '0';
               st[i] += 1;
          }
          for (i = p; st[i] != '\0'; i++);
          for (i--; st[i] == '0'; i--)
               st[i] = ' ';
     }
     if (ch != '-' && ch != '+' && ch != 0) // Если число знаков
после за-
     {                                        // пятой оказалось меньше
q, то
          for (i = p - 1; st[i] != '\0'; i++);   // после последней
цифры прос-
          st[i] = ' ';                          // тавляется символ ch
(если
```

```
          for (; st[i - 1] == ' '; i--);       // он не минус), но
число про-
          st[i] = ch;                          // белов вместо
недостающих
     }                                        // цифр сохраняется.
     else if (ch == '-')
     {
          for (i = p - 1; st[i] != ' '; i++);  // Убираем сзади
пробелы
          st[i] = '\0';
     }
     if (x < 0.)
     {
          for (j = 0, k = -1; st[j] != '\0'; j++)
          {
               if (st[j] > '0' && st[j] <= '9')
                    break;
               if (st[j] == '-')
                    k = j;
          }
          if (st[j] == '\0' && k > -1)
               st[k] = ' ';
     }
     return(st);
}

int matrkoef(double *A, double *P, double *P0, double dt, int n)
{
     double E[4], p, q, s, ct, st;  // Потом заменим матрицу A на
     int i, j, k;                   // обратную матрицу
     p = (A[0] + A[3]) / 2.;          // Строим матрицу P
     s = exp(p*dt);
     q = (A[0] - A[3]) / 2.;
     q = -A[1] * A[2] - q*q;
     if (q < 0.)
          return(0);
     q = sqrt(q);
     ct = cos(q*dt);
     st = sin(q*dt);
     P[0] = ct + st*p / q;
     P[1] = st*A[1] / q;
     P[2] = -st*(p*p + q*q) / q;
     P[3] = ct - st*p / q;
     for (i = 0; i < n*n; i++)
          P[i] *= s;
     for (i = 0; i < n*n; i++)       //                  -1
          E[i] = A[i];                 // Строим матрицу A
     q = A[0] * A[3] - A[1] * A[2];
     A[0] = E[3];
     A[1] = -E[1];
     A[2] = -E[2];
```

```
     A[3] = E[0];
     for (i = 0; i < n*n; i++)
          A[i] /= q;
     E[0] = E[3] = 1.;              // Строим матрицу P0
     E[1] = E[2] = 0.;
     for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
               for (k = 0, P0[i*n + j] = 0.; k < n; k++)
                    P0[i*n + j] += A[i*n + k] * (P[k*n + j] - E[k*n +
j]);
     return(1);
}

double matrpubl(double *mtr, int n, std::ofstream & ost)
{
     int i, j;
     for (i = 0; i < n; i++)
     {
          ost << "   e";
          for (j = 0; j < n; j++)
               ost << krap(mtr[i*n + j], 2, 8);
          ost << " e" << std::endl;
     }
     return(1.);
}

double prod(double *P, double *b, double *x, int n)
{
     int i, j;          // x = P·b
     double c;
     if (n > 0)
          for (i = 0; i < n; i++)
               x[i] = 0.;
     else
          n = -n;
     for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
               x[i] += P[i*n + j] * b[j];
     for (i = 0, c = 0.; i < n; i++)
          c += x[i] * x[i];
     c = sqrt(c);
     return(c);
}

double detal(double fee)  // Исходный контур детали
{
     static double e = -1., R = -1.;
     double ro = 0., s;
     if (fee < -0.5)
          e = R = -1.;
     else if (e < -0.5)
```

```
          e = fee;
     else if (R < -0.5)
          R = fee;
     else
     {
          s = e*sin(fee);
          ro = R*R - s*s;
          ro = sqrt(ro);
          ro += e*cos(fee);
     }
     return(ro);
}

double rand(double a, double b)
{
     if (a > b)
     {
          std::swap(a,b);
     }
     return rand() * (b - a) + a;
}


double typtab(double x, int p)
{
     int     i = 0, k, N, y1;
     double  w;
     static char st[71];
     N = 0;
     w = x;
     st[0] = st[1] = ' ';
     if (w < 0.)
          w = -w;
     p += 1;
     while (w > 10. || (int)w > 9)
     {
          w /= 10.;
          N++;
     }
     y1 = (int)w;
     for (k = 0, i = 1; k < N + p; k++)
     {
          sprintf(&st[i + k], "%d", y1);
          w = 10.*w - (double)(10 * y1);
          y1 = (int)w;
          if (k == N)
          {
               i++;
               sprintf(&st[k + i], ".");
          }
     }
```

```
     st[k + i] = '\0';
     if (y1 > 4)
     {
          for (k = 0; st[k] != '\0'; k++);
          for (; k--; )
          {
               if (st[k] == '.')
                    continue;
               if (st[k] == '9')
               {
                    st[k] = '0';
                    continue;
               }
               if (!k)
                    st[k] = 1;
               else
                    st[k] += 1;
               break;
          }
     }
     w = atof(st);
     if (x < 0.)
          w = -w;
     return(w);
}

double eq(double *x, double *p, int n)
{
     while (n--)
          x[n] = p[n];
     return(x[0]);
}

double podacha(double h)
{
     FILE *tb;
     static int n = 0, j = -1;
     int i;
     double v = 0.;
     static double *H, *V, v0;
     if (h < -70.)
     {
          if (H) delete[] H;
          if (V) delete[] V;
          n = 0;
          j = -1;
     }
     else if (h < -50.)
     {
          if (H && V)
               for (i = 1, v = 0.; i < n; i++)
```

```
                    v += (H[i - 1] - H[i]) / V[i];
        }
        else if (!n)
        {
            n = (int)(h + 0.01);
            H = new double[n];    // остаток пути до требуемой границы.
            V = new double[n + 1];    // скорости подачи
        }
        else if (j == -1)
        {
            v0 = h;
            j = 0;
        }
        else if (j < n)
            H[j++] = h;
        else if (j < 2 * n + 1)
            V[j++ - n] = h;
        else if (h > H[0])
            v = v0;
        else
        {
            for (i = 0; i < n && h < H[i]; i++);
            v = V[i];
        }
        return(v);
}
```