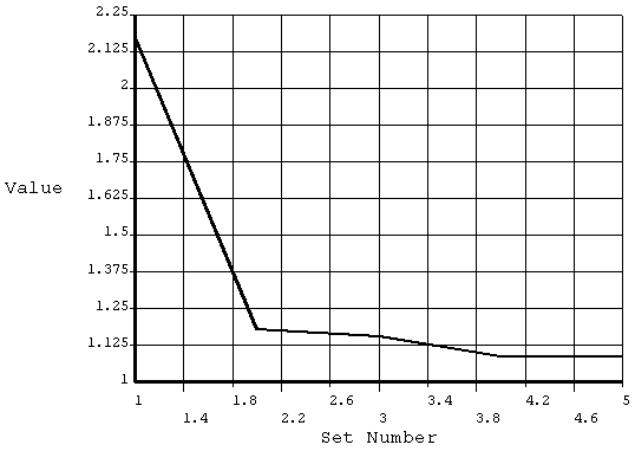
МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Программное обеспечение информационных систем и технологий»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ ИНЖЕНЕРНЫХ КОНСТРУКЦИЙ

Пособие



Минск
БНТУ
2024

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Программное обеспечение информационных систем
и технологий»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ ИНЖЕНЕРНЫХ КОНСТРУКЦИЙ

Пособие для студентов специальности
1-40 05 01 «Информационные системы и технологии
(по направлениям)» направления специальности 1-40 05 01-01
«Информационные системы и технологии (в проектировании
и производстве)»

*Рекомендовано учебно-методическим объединением по образованию
в области информатики и радиоэлектроники*

Минск
БНТУ
2024

УДК 004.4
ББК 32.973.26-018
К64

А в т о р ы:
И. Л. Ковалева, Д. П. Кункевич, В. В. Напрасников, Ю. В. Полозков

Р е ц е н з е н т ы:
кафедра «Информационные технологии автоматизированных систем» УО «Белорусский государственный университет информатики и радиоэлектроники» (зав. кафедрой, канд. физ.-мат. наук, доцент *А. А. Навроцкий*);
ведущий научный сотрудник ОИПИ НАН Беларуси *В. М. Демко*

Ковалева, И. Л.

К64 Информационные технологии в проектировании инженерных конструкций : пособие для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» направления специальности 1-40 05 01 01 «Информационные системы и технологии (в проектировании и производстве)» / И. Л. Ковалева [и др.]. – Минск : БНТУ, 2024. – 82 с.

ISBN 978-985-31-0021-1.

Настоящий материал предназначен для использования в качестве методических указаний при проведении лабораторных работ по дисциплинам: «Компьютерные системы конечно-элементных расчетов», «Основы автоматизированного конструирования», «Оптимизация проектных решений».

УДК 004.4
ББК 32.973.26-018

ISBN 978-985-31-0021-1

ОГЛАВЛЕНИЕ

1. КОМПЬЮТЕРНЫЕ СИСТЕМЫ КОНЕЧНО-ЭЛЕМЕНТНЫХ РАСЧЕТОВ

Программа ANSYS располагает возможностями не только для однократного расчета конструкции, но и для поиска ее оптимального варианта (оптимального проекта). Оптимальным является проект, отвечающий всем предъявляемым требованиям и имеющий минимальные значения определенных показателей, таких, как вес, площадь поверхности, объем, напряжения, собственные частоты и так далее.

В ANSYS доступны **методы** и **средства** оптимизации. Следует сразу указать разницу между ними. **Методы оптимизации (methods)** обеспечивают оптимизацию целевой функции путем варьирования входных параметров. **Средства оптимизации (tools)** обеспечивают получение нескольких наборов выходных параметров (целевая функция, переменные состояния) при изменении входных параметров по заданному закону, оптимизацию целевой функции они не производят.

Перед описанием методики оптимизации проекта в ANSYS необходимо определить некоторые термины.

Переменные проекта (design variables) – это параметры, которые изменяются с целью нахождения оптимального проекта. Для переменных проекта указываются ограничения – минимальное и максимальное значения. Эти значения определяют диапазон изменения переменных проекта. Переменными проекта обычно являются геометрические параметры, такие, как длина, толщина, диаметр или координаты точек. Переменные проекта могут принимать только положительные значения.

Переменные состояния (state variables) – это параметры, на которые наложены ограничения для проекта. Они также называются зависимыми переменными. Как правило, они представляют собой параметры отклика, являющиеся функциями переменных проекта. Переменные состояния могут быть ограничены максимальным и минимальным значениями или иметь только одно из этих ограничений. Примерами переменных проекта являются напряжения, температуры, скорости тепловых потоков, собственные частоты, деформации. Однако переменная состояния не обязательно должна быть вычисляемой величиной, в качестве переменной состояния может быть указан любой параметр.

Целевая функция (objective) – это зависимая переменная, которую требуется минимизировать. Она должна быть функцией переменных проекта, т. е. изменение значений переменных проекта должно изменять значение целевой функции. В оптимизационной задаче может быть определена только одна целевая функция.

Переменные проекта, переменные состояния и целевая функция обобщенно называются **переменными оптимизации (optimization variables)**. Пользователь должен указать, какие параметры в модели являются переменными проекта, переменными состояния и целевой функцией.

Набор параметров проекта, или проект (design set, design) – это набор значений параметров, представляющих какую-либо конфигурацию модели. Как правило, набор параметров проекта характеризуется значениями переменных оптимизации, однако в него включаются все параметры модели, в том числе и те, которые не являются переменными оптимизации.

Возможный проект (feasible design) – это проект, удовлетворяющий всем указанным ограничениям на переменные состояния и переменные проекта. Если хотя бы одно из ограничений не соблюдается, проект называется **невозможным (infeasible design)**.

Наилучший проект (best design) – тот, который удовлетворяет всем ограничениям и обеспечивает минимальное значение целевой функции. Если все проекты являются невозможными, наилучшим является проект, наиболее близкий к тому, чтобы быть возможным, вне зависимости от значения целевой функции.

Замечание. К сожалению, в последних версиях ANSYS разработчики исключили ранее присутствовавший там пункт Design Opt из Main Menu, где удобно было работать с компонентами оптимизационной модели в диалоговом режиме. Такой подход, видимо, объясняется желанием «пересадить» пользователей на ANSYS Worcbehch, что нравится далеко не всем.

В то же время, команды процессора Opt хотя и не документированы, но по-прежнему доступны (такие как OPVAR, OPTYPE, OPANL, OPEXE). Описания этих команд присутствуют, например, в 10 и 11 версиях. Поэтому остается возможность использовать командный файл для создания оптимизационной модели и поиска оптимального решения. В этом пособии излагаются особенности такого подхода.

1.1. Методы и средства оптимизации в ANSYS

1.1.1. Общее описание

Методы оптимизации производят минимизацию целевой функции. В программе доступны два метода: **метод аппроксимации (subproblem approximation method)** и **метод первого порядка (first order method)**. Метод аппроксимации – это метод нулевого порядка, обеспечивающий эффективное решение большинства конструкторских задач. Метод первого порядка основан на оценке чувствительности проекта к изменению определенных факторов и больше подходит для решения задач, требующих высокой точности.

Кроме того, пользователь может применить другой оптимизационный алгоритм, в этом случае алгоритм ANSYS будет пропущен.

Как при использовании метода аппроксимации, так и при использовании метода первого порядка, программа выполняет серию итераций. В течение каждой итерации выполняется расчет начального проекта, оценивается соответствие результатов расчета определенным критериям качества и, при необходимости, осуществляется изменение проекта. Этот процесс продолжается до тех пор, пока не будут выполнены определенные условия.

Переменные состояния и ограничения на переменные проекта используются, чтобы ограничить изменения в проекте, и приводят к оптимизационной задаче с ограничениями. ANSYS преобразует эту задачу в оптимизационную задачу без ограничений. При проведении оптимизации по каждому из методов программа учитывает ограничения, наложенные на переменные состояния, добавляя к целевой функции штрафные функции.

1.1.2. Метод аппроксимации

При использовании метода аппроксимации программа на каждой итерации производит аппроксимацию целевой функции и переменных состояния (методом наименьших квадратов) квадратичными функциями переменных проекта. Для аппроксимации используются значения целевой функции и переменных состояния на предыдущих итерациях (т. е. для предыдущих наборов параметров).

Следует отметить, что в документации к ANSYS не описаны некоторые детали алгоритма построения данных аппроксимаций (вычисление весовых коэффициентов для различных наборов параметров). Кроме того, существуют ограничения на количество одновременно учитываемых при аппроксимации наборов параметров.

После построения аппроксимаций программа преобразует оптимизационную задачу с ограничениями в задачу без ограничений, находит экстремум аппроксимации целевой функции и назначает на следующей итерации значения переменных проекта, соответствующие этому экстремуму. Эта процедура повторяется и на следующих итерациях.

Пользователь может выбирать тип аппроксимирующей функции. Можно использовать линейную функцию, квадратичную функцию без перекрестных членов и квадратичную функцию с перекрестными членами.

Для начала итераций по методу аппроксимации необходимо наличие определенного количества наборов параметров (для построения аппроксимирующей функции). В случае их отсутствия программа создаст их сама, случайным образом варьируя переменные проекта внутри их границ.

Т. к. это случайные наборы параметров, то сходимость может быть медленной. Иногда можно ускорить сходимость, создав несколько возможных наборов параметров. Это может быть сделано путем создания нескольких случайных наборов параметров и исключения всех невозможных наборов параметров. Кроме того, можно создать начальные наборы параметров путем выполнения нескольких одиночных циклов анализа, указывая новые значения переменных проекта перед каждым циклом.

В конце каждого цикла анализа производится проверка сходимости и условий прерывания оптимизации. Задача считается сошедшейся, если текущий, предыдущий и наилучший проекты (наборы параметров) являются возможными и выполнено одно из следующих условий:

– разность значений целевой функции между лучшим возможным проектом и текущим проектом меньше погрешности сходимости целевой функции;

– разность значений целевой функции между двумя последними проектами меньше погрешности сходимости целевой функции;

– разности значений всех переменных проекта между лучшим возможным проектом и текущим проектом меньше их погрешностей сходимости;

– разности значений всех переменных проекта между двумя последними проектами меньше их погрешностей сходимости.

Пользователь может указать погрешности сходимости целевой функции и переменных проекта.

Иногда процедура оптимизации может быть прервана до достижения сходимости. Это происходит в случае выполнения одного из приведенных ниже условий:

– выполнено указанное количество итераций;

– количество последовательных невозможных проектов достигло указанного предела.

Сходимость не всегда означает нахождение глобального минимума. Она означает только то, что был выполнен один из указанных выше критериев. Поэтому именно пользователь должен определить, был ли проект достаточно оптимизирован. Если это не так, то можно выполнить дополнительные итерации.

1.1.3. Метод первого порядка

Метод первого порядка использует информацию о производных зависимых переменных относительно переменных проекта. Этот метод очень точен и хорошо решает задачи с большими диапазонами изменения зависимых переменных, однако требует больших вычислительных ресурсов.

При использовании метода первого порядка программа преобразует оптимизационную задачу с ограничениями в задачу без ограничений, а затем на каждой итерации вычисляет градиент целевой функции по переменным проекта. Для вычисления каждой частной производной программа присваивает небольшое приращение соответствующей переменной проекта, оставляя значения других переменных проекта прежними, и производит расчет конструкции с данным набором параметров.

После вычисления всех частных производных определяется направление поиска экстремума на данной итерации. Следует отметить, что в общем случае поиск осуществляется не в направлении градиента, для определения направления поиска используется более

сложная зависимость. Затем осуществляется линейный поиск экстремума целевой функции по данному направлению.

Пользователь может указать приращения переменных проекта, используемые для вычисления градиента, а также предельное значение шага линейного поиска.

Таким образом, каждая итерация разделяется на набор субитераций, который включает поиск направления и вычисление градиента. В связи с этим одна оптимизационная итерация для метода первого порядка включает в себя несколько циклов анализа.

Найденный таким образом экстремум используется в качестве исходной точки для следующей итерации и так далее.

Итерации продолжаются до тех пор, пока не будет достигнута сходимость или условия прерывания процесса оптимизации. Задача считается сошедшейся, если текущий, предыдущий и наилучший проекты (наборы параметров) таковы, что выполняется одно из следующих условий:

– разность значений целевой функции между лучшим проектом и текущим проектом меньше погрешности сходимости целевой функции;

– разность значений целевой функции между предыдущим проектом и текущим проектом меньше погрешности сходимости целевой функции.

Процедура оптимизации может быть прервана до достижения сходимости. Это происходит в случае, если выполнено максимальное количество итераций, указанное пользователем.

По сравнению с методом аппроксимации, метод первого порядка является более точным. Однако высокая точность метода первого порядка не всегда гарантирует получение наилучшего решения. Для метода первого порядка возможна сходимость при невозможном наборе параметров проекта. В этом случае, скорее всего, был обнаружен локальный минимум, или не существует возможных наборов параметров проекта.

Если это случилось, может быть полезным проведение оптимизации методом аппроксимации, т. к. это средство лучше подходит для исследования всей области варьирования параметров проекта. Кроме того, будет полезным получить случайные наборы параметров проекта, чтобы обнаружить область их возможных значений (если она

существует), а затем перезапустить метод первого порядка, используя возможный набор параметров проекта в качестве начальной точки.

1.1.4. Средства оптимизации в программе ANSYS

В дополнение к двум методам оптимизации, в программе ANSYS доступны пять различных средств оптимизации.

Средства оптимизации используются для определения области варьирования параметров проекта. Они обеспечивают не оптимизацию целевой функции, а автоматическое получение нескольких наборов параметров проекта при определенном законе изменения переменных проекта. Для использования этих средств не требуется наличие целевой функции, однако переменные проекта должны быть определены.

Средства оптимизации, доступные в ANSYS, приведены ниже.

Однократный анализ (single-loop analysis tool) заключается в выполнении расчета одного варианта конструкции с установленными в данный момент переменными проекта.

Случайное варьирование (random tool) выполняет указанное число циклов анализа, устанавливая каждый раз для переменных проекта случайные значения. Появившийся модуль вероятностного анализа является своеобразным расширением этого средства оптимизации.

Сканирование области варьирования параметров (sweep tool) создает заданное количество наборов параметров, поочередно варьируя каждую переменную проекта в исходном наборе параметров через весь диапазон ее изменения. Значения других переменных проекта при этом остаются неизменными.

Факторный анализ (factorial tool) создает наборы параметров, анализируя конструкцию при различных сочетаниях крайних значений переменных проекта.

Градиентный анализ (gradient tool) вычисляет градиенты целевой функции и переменных состояния относительно переменных проекта.

1.2. Выбор оптимальных размеров водолазного кессона

1.2.1. Постановка задачи

Для ремонта опорной конструкции нефтедобывающей платформы используется водолазный кессон. Он необходим, чтобы создать условия для проведения сварки в сухой среде. Его расчетная схема представлена на рис. 1.1.

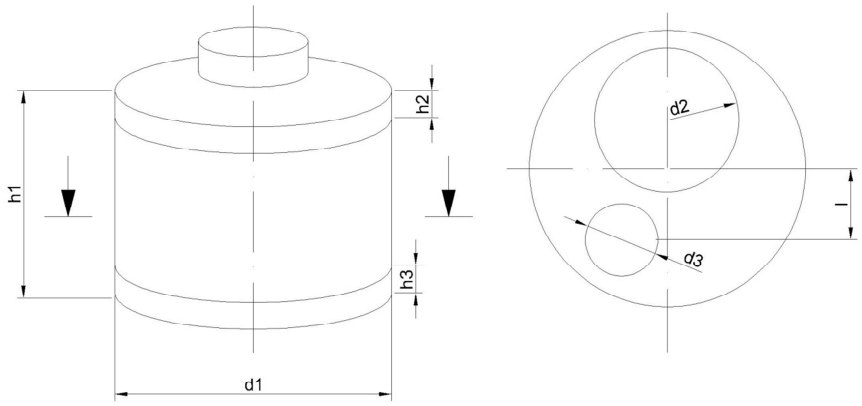


Рис. 1.1. Расчетная схема кессона

Кессон погружается на глубину 6 м, где на него будет действовать гидростатическое давление. Основание и крышка представлены достаточно мощными стальными пластинами толщиной h3 и h2 соответственно, боковая стенка представляет собой тонкий стальной лист толщиной h4. Закрепление кессона осуществляется в нескольких местах. Верхнее отверстие плотно прикрепляется к опорной колонне, снизу кессон прикреплен четырьмя тросами ко дну. В данной модели в качестве нагрузки будем рассматривать только гидростатическое давление воды на кессон.

Конструкция сделана из материала Сталь 45, характеристики которого перечислены ниже.

Модуль юнга равен $2 \cdot 10^{11}$ Па, коэффициент Пуассона равен 0,3.

Пределы прочности для знакопеременных нагрузок:

– при изгибе 135 МПа;

– при растяжении 110 МПа.

Предел текучести – 360 МПа.

Предел выносливости:

– при изгибе 375 МПа;

– при растяжении 220 МПа.

Задача оптимизации ставится так:

Необходимо подобрать параметры h2, h3, h4 так, чтобы минимизировать объем материала конструкции кессона при ограничениях на максимальное напряжение по теории Мизеса в материале. Оно не должно превышать 200 МПа.

1.2.2. Вид командного файла для оптимизации

Замечание.

В этой модели объем (переменная **total_vol**) считается по аналитическим зависимостям.

В общем случае так его посчитать не удастся. Используйте один из следующих способов.

1 способ.

```
/ Расчет с использованием ПОСТПРОЦЕССОРА
/POST1     / Запуск постпроцессора
AVPRIN,0, ,
ETABLE,EVolume,VOLU,
/*
SSUM
/* Объем будет сохранен в переменной Volume
*GET,Volume,SSUM, ,ITEM,EVOLUME
```

2 способ.

```
/ Выполняется в ПРЕПРОЦЕССОРЕ
/PREP7
VADD, ALL / Объединяем полученные объемы в один
/ Вычисляем и отображаем статистику геометрии (объем)
VSUM,TOTAL_VOLUME
/* Объем будет сохранен в переменной TOTAL_VOLUME
*GET,TOTAL_VOLUME,VOLU,ALL,VOLU
```

```
/PREP7
/ Ввод первоначальных значений параметров
*SET,d1,4
*SET,d2,2
*SET,d3,1.5
*SET,h1,3
*SET,h2,0.1
*SET,h3,0.1
*SET,h4,0.004
*SET,h5,0.2
*SET,l,0.7
Lam1=0.7
Lam2=0.3
/ Объем материала кессона
V_1=(d1*d1/4-d2*d2/4)*h3*3.141592
V_2=(d1*d1/4-d3*d3/4)*h2*3.141592
V_3=d1*h4*h1*3.141592
total_vol=V_1+V_2+V_3
/ Построение геометрии объекта
/ задаем точки по координатам
k,1,d1/2,0,0
k,2,0,d1/2,0
k,3,-d1/2,0,0
k,4,0,-d1/2,0
k,5,d1/2,0,h1
k,6,0,d1/2,h1
k,7,-d1/2,0,h1
k,8,0,-d1/2,h1
 / переходим в цилиндрическую СК
CSYS,1
 / Создаем цилиндрические поверхности
a,1,2,6,5,1
a,1+1,2+1,6+1,5+1,1+1
a,1+2,2+2,6+2,5+2,1+2
a,1+3,1,5,8,4
 / переходим в декартову СК
CSYS,0
/отверстие для водолаза
```

```
/ люк
pcirc,d1/2,,,
pcirc,d2/2,,,
 / Перенос поверхности A6 вдоль оси OY
AGEN, ,6, , , ,(d1-d2)/4, , , ,1
 / Вычитаем A6 из A5
ASBA,      5,      6
/ крышка
/ Создаем вспомогательную область A7
pcirc,d1/2,,,
/ Перенос поверхности A7 вдоль оси OZ на h1
AGEN, , 7, , , , ,h1 , , ,1   // ЕЕ НОМЕР СТАЛ A5
/ Создаем область под отверстие
pcirc,d3/2,,, // ЕЕ НОМЕР СТАЛ A6
/ Перенос поверхности A6 вдоль оси OX и оси OY
AGEN, ,6, , , ,(d1-d2)/4, , , ,1
 / Вычитаем A6 из A5
ASBA,      5,      6
/склеиваем поверхности
AGLUE,1,2,3,4,7,8
//Нумерация поверхностей и точек ИЗМЕНИЛАСЬ
/ В результате:
/ A5 – отверстие для трубы на уровне Z = 0
/ A6 – отверстие для водолаза на уровне Z = h2
/ Задаем свойства материала
MPTEMP,1,0
MPDATA,EX,1,,2e11
MPDATA,PRXY,1,,0.3
MPTEMP,,,,,,,,
MPTEMP,1,0
MPDATA,DENS,1,,7850
/ Задаем тип элемента
ET,1,SHELL181
/ Задаем размер конечного элемента
esize,d1/20
/*
/ Номер секции 1 Имя секции h4_DELT
sect,1,shell,,h4_DELT
```

/ *Задаем толщину, равную h4, для боковой стенки*
secdata, h4,1,0.0,3
secoffset,MID
seccontrol,,,, , , ,
/*Наносим сетку на боковую поверхность (от A1 до A4)*
SECNUM, 1
aMESH,1,4
/ *Задаем толщину, равную h2, для верхней крышки*
/ *Номер секции 2 Имя секции h2_DELT*
sect,2,shell,,h2_DELT
/ *Задаем толщину, равную h2*
secdata, h2,1,0.0,3
secoffset,MID
seccontrol,,,, , , ,
/ *Наносим сетку на верхнюю крышку*
SECNUM, 2
aMESH,5
/ *Задаем толщину, равную h3, для нижней крышки*
/ *Номер секции 3 Имя секции h3_DELT*
sect,3,shell,, h3_DELT
/ *Задаем толщину, равную h3*
secdata, h3,1,0.0,3
secoffset,MID
seccontrol,,,, , , ,
/ *Наносим сетку на нижнюю крышку*
SECNUM, 3
aMESH,6
/ *закрепление верхнего отверстия (линии 25, 26, 27, 28)*
DL,25,,Ux,Uy $ DL,26,,Ux,Uy $ DL,27,,Ux,Uy $ DL,28,,Ux,Uy
/ *закрепление точек 5, 6, 7, 8, прилегающих к тросам*
DK,5,ALL $ DK,6,ALL $ DK,7,ALL $ DK,8,ALL
!/GO
/ *Прикладываем давление на верхнюю крышку*
SFA,5,1,PRES,-29400
/GO
*DEL,_FNCNAME
*DEL,_FNCMTID
*DEL,_FNCCSYS

```
/ задаем закон изменения давления с именем Dawlen1
/ на боковую поверхность
*SET,_FNCNAME,'Dawlen1'
*SET,_FNCCSYS,0
*DIM,%_FNCNAME%,TABLE,6,7,1,,,,%_FNCCSYS%
!
! Begin of equation: (3-{Z})*9.8*1000
*SET,%_FNCNAME%(0,0,1), 0.0, -999
*SET,%_FNCNAME%(2,0,1), 0.0
*SET,%_FNCNAME%(3,0,1), 0.0
*SET,%_FNCNAME%(4,0,1), 0.0
*SET,%_FNCNAME%(5,0,1), 0.0
*SET,%_FNCNAME%(6,0,1), 0.0
*SET,%_FNCNAME%(0,1,1), 1.0, -1, 0, 3, 0, 0, 4
*SET,%_FNCNAME%(0,2,1), 0.0, -2, 0, 1, -1, 2, 4
*SET,%_FNCNAME%(0,3,1),   0, -1, 0, 9.8, 0, 0, -2
*SET,%_FNCNAME%(0,4,1), 0.0, -3, 0, 1, -2, 3, -1
*SET,%_FNCNAME%(0,5,1), 0.0, -1, 0, 1000, 0, 0, -3
*SET,%_FNCNAME%(0,6,1), 0.0, -2, 0, 1, -3, 3, -1
*SET,%_FNCNAME%(0,7,1), 0.0, 99, 0, 1, -2, 0, 0
! End of equation: (3-{Z})*9.8*1000
!-->
/GO
/ Прикладываем переменное давление
/ на боковую поверхность (от A1 до A4)
SFA,1,1,PRES,%DAWLEN1%
SFA,2,1,PRES,%DAWLEN1%
SFA,3,1,PRES, %DAWLEN1%
SFA,4,1,PRES, %DAWLEN1%
FINISH
/ Выполняем статический анализ
! solution
/SOL
ANTYPE,0
SOLVE
/ Постпроцессорная обработка результатов
/POST1
SET,FIRST
```

/ *Вычисление макс. напряжений по МИЗЕСУ*
NSORT, S, EQV
*GET, STRESS_MAX, SORT, , MAX
/ *Сохранение файла*
LGWRITE,'KESSON _For_opt','ans','D:\999'
/ *Вход в модуль оптимизации*
/OPT
/ *Открытие файла задачи для оптимизации*
OPANL,'KESSON _For_opt','ans','D:\999\'
!++
/ *Задание параметров оптимизации* /
/ *ПЕРВОЕ: Задание переменных проекта DV's* /

OPVAR,h2,DV,0.05,0.2, , /*толщина верхней крышки h2*
OPVAR,h3,DV,0.05,0.2, , /*толщина нижней крышки h3*
OPVAR,h4,DV,0.002,0.006, , /*толщина боковой стенки h4*

/ *ВТОРОЕ: Задание переменных состояния SV's* /
/ *Задание изменения границ максимальных напряжений*
OPVAR,STRESS_MAX,SV,0,200e6, ,

/ *ТРЕТЬЕ: Задание целевой функции OV* /
/ *В нашем случае это объем материала total_vol*
OPVAR,total_vol,OBJ, , , ,
!++
/ *ЧЕТВЕРТОЕ: Выбираем МЕТОД ПЕРВОГО ПОРЯДКА* /
OPTYPE,FIRST
/ *Команда на запуск оптимизации*
OPEXE
/ *Вывод результатов в текстовом виде*
OPLIST,ALL, ,0
/ *Вывод результатов в графическом виде*
PLVAROPT,h2
PLVAROPT,h3
PLVAROPT,h4
PLVAROPT,STRESS_MAX
PLVAROPT,total_vol

На рис. 1.2 представлено напряженное состояние конструкции при исходных параметрах.

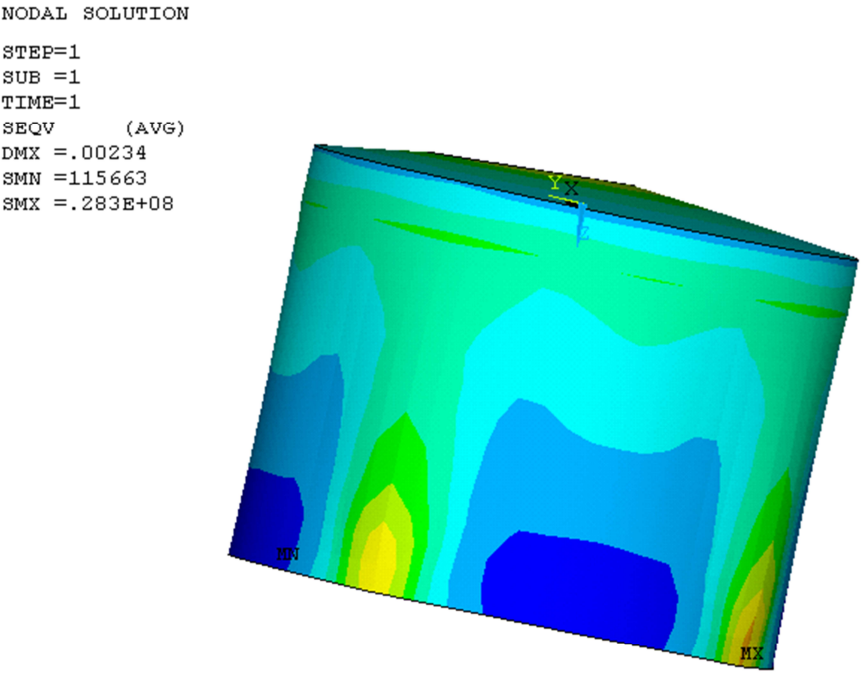


Рис. 1.2. Напряженное состояние конструкции

На рис. 1.3 представлена динамика оптимизации.

Таким образом, объем конструкции может быть уменьшен с 2,1732 м^3 в исходном варианте до 1,0866 м^3 в оптимальном варианте.

```
LIST OPTIMIZATION SETS FROM SET   1 TO SET   5 AND SHOW
ONLY OPTIMIZATION PARAMETERS. (A "*" SYMBOL IS USED TO
INDICATE THE BEST LISTED SET)

                  SET  1        SET  2        SET  3       *SET  4*
                (FEASIBLE)    (FEASIBLE)    (FEASIBLE)    (FEASIBLE)
STRESS_MAX(SV)    0.23261E+08   0.76816E+08   0.76824E+08   0.78308E+08
H2      (DV)      0.10000       0.50000E-01   0.50000E-01   0.50000E-01
H3      (DV)      0.10000       0.52450E-01   0.50000E-01   0.50000E-01
H4      (DV)      0.40000E-02   0.38252E-02   0.38183E-02   0.20000E-02
TOTAL_VOL(OBJ)      2.1732        1.1785        1.1551        1.0866

                  SET  5
                (FEASIBLE)
STRESS_MAX(SV)    0.78308E+08
H2      (DV)      0.50000E-01
H3      (DV)      0.50000E-01
H4      (DV)      0.20000E-02
TOTAL_VOL(OBJ)      1.0866
```

Рис. 1.3. Результаты оптимизации. Символом «*» отмечен наилучший набор параметров

На рис. 1.4 представлена динамика изменения объема в зависимости от номера итерации.

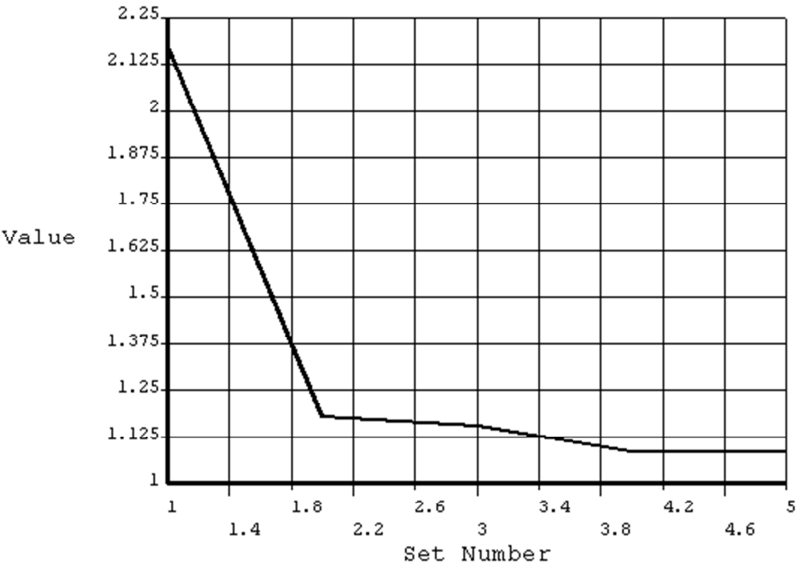


Рис. 1.4. Динамика изменения объема в зависимости от номера итерации

В приведенном выше тексте командного файла специфические команды для оптимизации выделены жирным шрифтом.

В частности, задать другой метод или средство оптимизации можно, поменяв команды.

/ *ЧЕТВЕРТОЕ: Выбираем МЕТОД ПЕРВОГО ПОРЯДКА* /
OPTYPE,FIRST

на следующие их варианты:

Вариант 2.
/ *Средство «Сканирование области варьирования параметров»*
OPTYPE,SWEEP / *Выбор средства решения задачи оптимизации*
/ *Сканирование области варьирования параметров*
OPSWEEP,BEST,5 / *5 evaluations per DV at best design set*
Вариант 3.
/ *Метод квадратичной аппроксимации*
OPTYPE,SUBP / *Выбор метода решения задачи оптимизации*
OPSUBP,30,10, / *Параметры этого метода*
OPEQN,0,3,0,0,0, / *Квадратичные и перекрестные члены учтены*
/ *для целевой функции (первый параметр равен 0)*
/ *Квадратичные и перекрестные члены учтены*
/ *для переменных состояния (второй параметр равен 3)*
Вариант 4.
/ *Метод квадратичной аппроксимации*
OPTYPE,SUBP / *Выбор метода решения задачи оптимизации*
OPSUBP,30,10,
OPEQN,0,0,0,0,0, / *Квадратичные и перекрестные члены учтены*
/ *для целевой функции (первый параметр равен 0)*
/ *Только квадратичные члены учтены*
/ *для переменных состояния (второй параметр равен 0)*
Вариант 5.
/ *Средство «Случайное варьирование»*
OPTYPE,RAND / *Выбор средства решения задачи оптимизации*
OPRAND,20,0, / *Выбор количества итераций*

2. ОСНОВЫ АВТОМАТИЗИРОВАННОГО КОНСТРУИРОВАНИЯ

В данном разделе пособия рассматриваются вопросы, связанные с разработкой конструкторских приложений.

Основными средствами автоматизации конструирования в настоящее время являются системы геометрического моделирования (СГМ): КОМПАС 3D, SolidWorks, NX и т. п. Они помогают быстро визуализировать решение в виде объемной модели, проанализировать его и подготовить конструкторскую документацию.

Однако существует довольно много процедур рутинных и однообразных, но недостаточно однозначных для автоматизации в рамках универсальной системы. Их можно эффективно решать в рамках специализированных приложений при помощи интерфейса программирования, предоставляемого СГМ. Соответствующие методики рассматриваются в данном пособии.

Первая работа посвящена освоению методов автоматизированного проектирования в интерактивном режиме на примере приспособлений для установки деталей на технологических операциях. Следующие работы – автоматизация первой – разработка программных средств формирования аналогичных конструкций.

Выбор СГМ достаточно широк. Принципиальной разницы, какая именно система будет использована для выполнения предлагаемых упражнений, нет. Данное пособие ориентировано на систему SolidWorks.

2.1. Проектирование приспособления для установки и закрепления деталей

2.1.1. Цели и задачи

Цель работы – ознакомление с тем, как происходит конструирование технических объектов с использованием средств автоматизации. Для достижения поставленной цели предлагается выполнить проектирование объекта определенного класса.

Автоматизированное проектирование опирается на традиционные технологии решения конструкторских задач. Однако сущест-

вуют особенности, обусловленные «компьютерной» спецификой процесса. Эти особенности эффективно осваиваются в ходе практического решения проектно-конструкторских задач в **интерактивном режиме**.

2.1.2. Объект проектирования

В качестве «учебного» объекта проектирования предлагаются **приспособления для установки и закрепления деталей** на технологических операциях[1].

Деталь – это **объект оснащения**. Пусть в качестве такового задана деталь Д (рис. 2.1, *а*). В ней необходимо прорезать уступ шириной *B* и глубиной *h*. Это можно сделать инструментом И. Деталь и инструмент надо расположить относительно друг друга соответствующим образом и закрепить. Инструмент устанавливается на шпиндель станка (рис. 2.1, *б*). Для установки же детали и требуется это самое приспособление, состоящее в данном случае из **опор** О, **упора** У и **корпуса** К (рис. 2.1, *в*). Внешний вид такого приспособления приведен на рис. 2.1, *г*.

Опора О и упор У – это **установочные элементы**. Кроме них для фиксации детали в приспособлениях применяют **зажимы** (на рис. 2.1 не показаны). **Установочные** элементы и зажимы – это **функциональные**[2] элементы. Корпус К служит для крепления между собой всех элементов приспособления и установки его на станке[3].

Поверхности детали, которыми она устанавливается в приспособление – «**базы**». В приведенном примере (рис. 2.1, *а*) – поверхность Б1, которой деталь ложится на опору О, и поверхность Б2, которой деталь упирается в упор У.

Поверхности установочных элементов, на которые устанавливается деталь и в которые она упирается, называются «**установочные поверхности приспособления**».

---

[1] Обработка резанием, сборка, сварка и т. п.

[2] Непосредственно реализующие функцию объекта проектирования.

[3] В устройстве верхнего уровня.

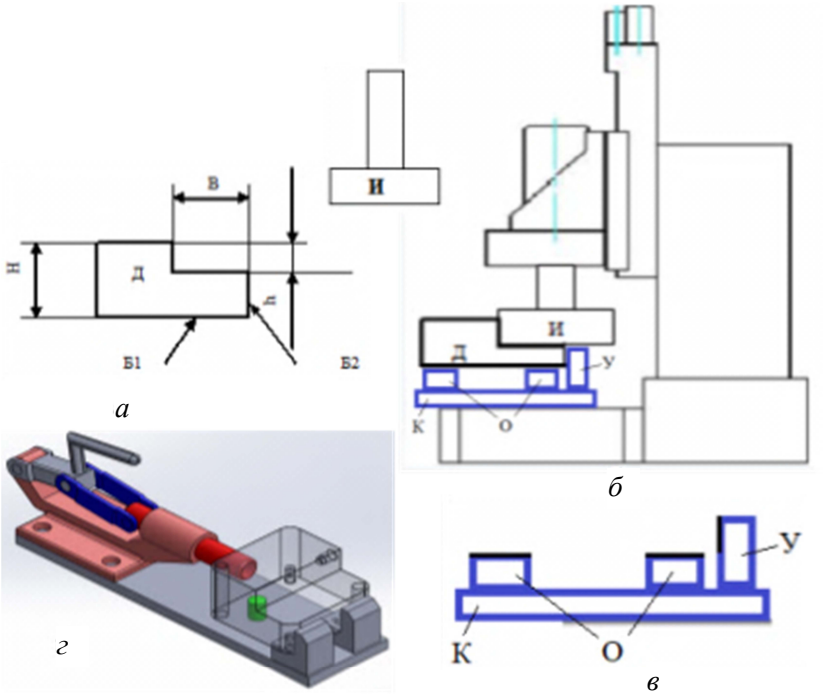


Рис. 2.1. Приспособление для установки и закрепления:
а – деталь; *б* – схема обработки; *в* – установочные элементы приспособления;
г – внешний вид приспособления

2.1.3. Методика выполнения работы

Исходная информация – геометрическая модель детали и множество баз $\boldsymbol{B} = \{\boldsymbol{b}_i\}$, $i = 1...\ n$, где n – количество баз[4] (рис. 2.2).

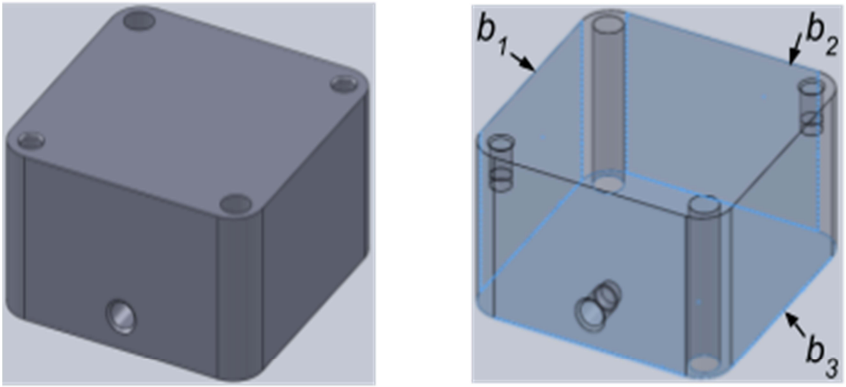
Геометрическая модель – описание, отражающее форму и размеры объекта. Существуют различные виды таких описаний[5], а также форматы их компьютерного представления. Наиболее информативна модель **твердотельная**[6]. Формат в каждой СГМ собственный.

[4] Как правило, три.

[5] Каркасные модели, поверхностные, декомпозиционные, конструктивные и др.

[6] *Solid model.*

Кроме того, существуют специальные форматы обмена моделями между системами. Наиболее распространены IGES[7] и STEP[8].



а *б*

Рис. 2.2. Объект оснащения:
а – внешний вид; *б* – базовые поверхности

Таким образом, для выполнения работы необходим файл детали-объекта оснащения в формате используемой СГМ либо одном из обменных форматов.

Геометрическую модель в обменном формате необходимо **импортировать** в систему, на базе которой выполняется проектирование.

Импортирование – это процедура построения своей геометрической модели по описанию в унифицированном формате. Своя модель – это модель, построенная из объектов классов, определенных в СГМ. Импортирование не является открытием файла, хотя иногда выполняется в контексте именно этой команды. Соответственно, после этой процедуры модель следует **сохранить**.

Работа выполняется в сборочном документе. Соответственно, перед началом работы следует создать документ класса «сборка» (assembly), в который первым компонентом вставить модель объекта оснащения, а затем детали и узлы проектируемого приспособления.

[7] Initial Graphics Exchange Specification.

[8] STandard for Exchange of Product model data – стандарт ISO 10303.

Этапы проектирования приспособления (основные процедуры):

1. **Компоновка** функциональных элементов:

– установочных;

– зажимных (фиксирующих).

2. Формирование корпусной плиты.

3. Согласование функциональных элементов с корпусной плитой.

Ознакомиться с минимально необходимым для выполнения работы набором команд поможет учебное пособие: **«Учебные пособия SolidWorks» → «Введение» → «Упражнение 2 – Сборки»**. Для создания моделей отдельных деталей полезным будет пособие: **«Учебные пособия SolidWorks» → «Введение» → «Упражнение 1 – Детали»**.

2.1.4. Компоновка функциональных элементов

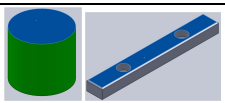
Компоновка функциональных элементов – это их **выбор** и **размещение**.

Выбор определяется **целевыми** поверхностями, их формами и расположением. Размещение – это установление сопряжения между **функциональной** поверхностью элемента и целевой поверхностью. **Целевые** поверхности – поверхности **детали,** на которые направлено действие проектируемого приспособления, – базы, на которые деталь должна быть установлена, и поверхности фиксации. **Функциональная** поверхность – поверхность элемента, которой он непосредственно оказывает свое действие на деталь.

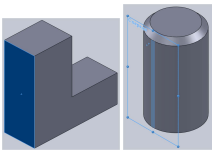
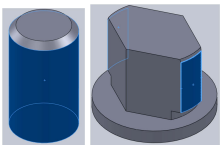
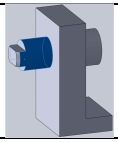
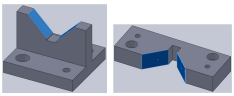
Для каждой базы следует подобрать установочный элемент, руководствуясь приведенной ниже табл. 1.

Таблица 1

Основные классы установочных элементов

Условия применения, характеристики баз	Наименование элемента	Установочная поверхность	Внешний вид, установочные поверхности
1	2	3	4
Плоскость, горизонтальная, широкая	Опора	Плоскость, горизонтальная	

Окончание табл. 1

1	2	3	4
Плоскость вертикальная	Упор	Плоскость вертикальная	
Отверстие вертикальное	Палец	Цилиндр внешний	
Отверстие горизонтальное	Палец выдвижной	Цилиндр внешний	
Внешняя цилиндрическая поверхность, две смежные плоскости	Призма установочная	Две сходящиеся плоскости	

Фиксирующие элементы выбираются и располагаются так, чтобы их усилие F_c было направлено на установочные элементы (рис. 2.3).

Сначала необходимо выбрать подходящую целевую поверхность – ту, на которую будет действовать прижим. Она должна быть такой, чтобы нормаль к ней была направлена на установочные поверхности.

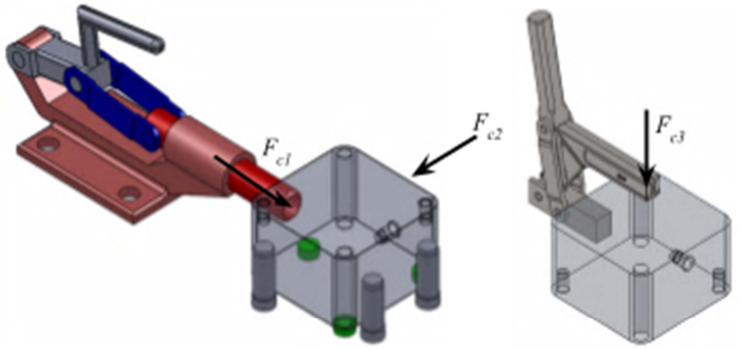


Рис. 2.3. Компоновка фиксирующих элементов

Это, например, вертикальные грани: вектор F_{c1} направлен на два вертикальных упора, а вектор F_{c2} – на одиночный упор. Верхняя горизонтальная грань тоже подойдет – вектор F_{c3} направлен на горизонтальные опоры.

После выбора целевой поверхности выбирается сам элемент. Его действие должно соответствовать направлению усилия: для горизонтального – зажим горизонтального действия, а для вертикального – вертикальный зажим. Выбранные элементы добавляются в конструкцию. Между их функциональными поверхностями и поверхностями целевыми устанавливаются сопряжения.

Элементы, расположенные рядом, можно заменить одним **комбинированным**. Например, горизонтальные опоры и вертикальные упоры (рис. 2.4) можно сгруппировать попарно и заменить каждую пару одним элементом.

Автоматизированное проектирование осуществляется на базе библиотеки элементов. Это означает, что для проектирования следует использовать уже существующий комплект моделей, не делать оригинальные «по месту».

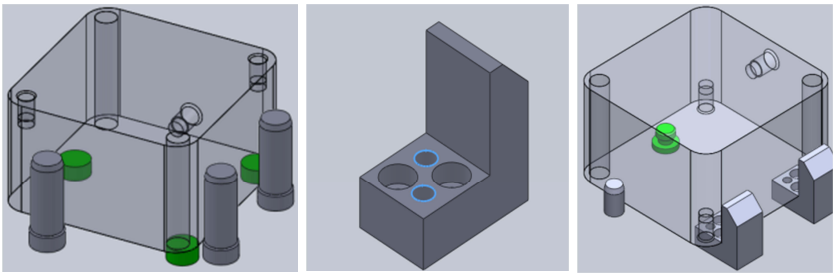


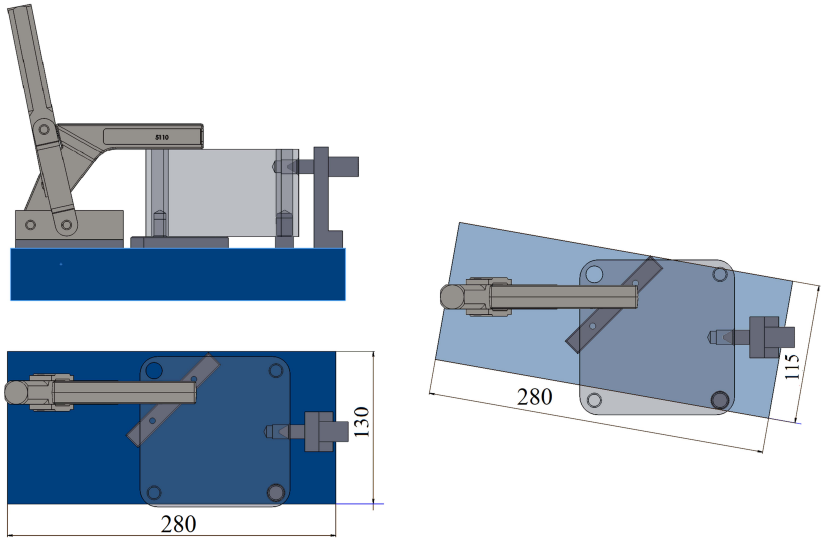
Рис. 2.4. Применение комбинированных элементов

2.1.5. Формирование корпусной плиты

В рамках решения данной задачи определяются **форма**, **размеры** и **расположение** плиты (рис. 2.5).

Корпус – элемент конструкции, на котором монтируются все остальные. В данной работе предлагается использовать прямоугольную плиту. По вертикали она располагается ниже всех элементов приспособления, а в горизонтальной плоскости ее контур охватывает проекции всех этих элементов (рис. 2.5, *а*).

Варьируя расположением, а именно, углом поворота в горизонтальной плоскости, можно уменьшить габариты плиты (рис. 2.5, *б*).



а *б*

Рис. 2.5. Корпусная плита:
а – базовый вариант; *б* – вариант с оптимизированным расположением

Корпусная плита добавляется в конструкцию только после расстановки функциональных элементов. Перед добавлением все имеющиеся элементы сцены рекомендуется зафиксировать. Не рекомендуется устанавливать сопряжения плиты с имеющимися элементами.

2.1.6. Согласование функциональных элементов с корпусом

Согласование – устранение промежутков между элементами конструкции и корпусной плитой, возникающих из-за выравнивания ее (плиты) по вертикали по нижнему элементу. Пример – прижим слева и горизонтальный палец справа (рис. 2.6, *а*).

Варианты решения: **удлинение** и специальный **переходник**.

Удлинение осуществляется за счет увеличения специального параметра, с которым связан соответствующий размер элемента.

Например, параметр *H* горизонтального установочного пальца (рис. 2.6, *б*) определяет расстояние его нижней плоскости[9] от функциональной части. Палец «не дотягивает» до плиты 13 мм. Соответственно, эти 13 мм следует прибавить к параметру *H*.

Если параметрическое удлинение элемента не предусмотрено, между ним и плитой вставляется **переходник** – прямоугольная призма (рис. 2.6, *в*). Длина и ширина определяются размерами опорной поверхности элемента (рис. 2.6, *г*, 90×20), а высота – расстоянием до плиты (7,5, рис. 2.6, *а*, *в*).

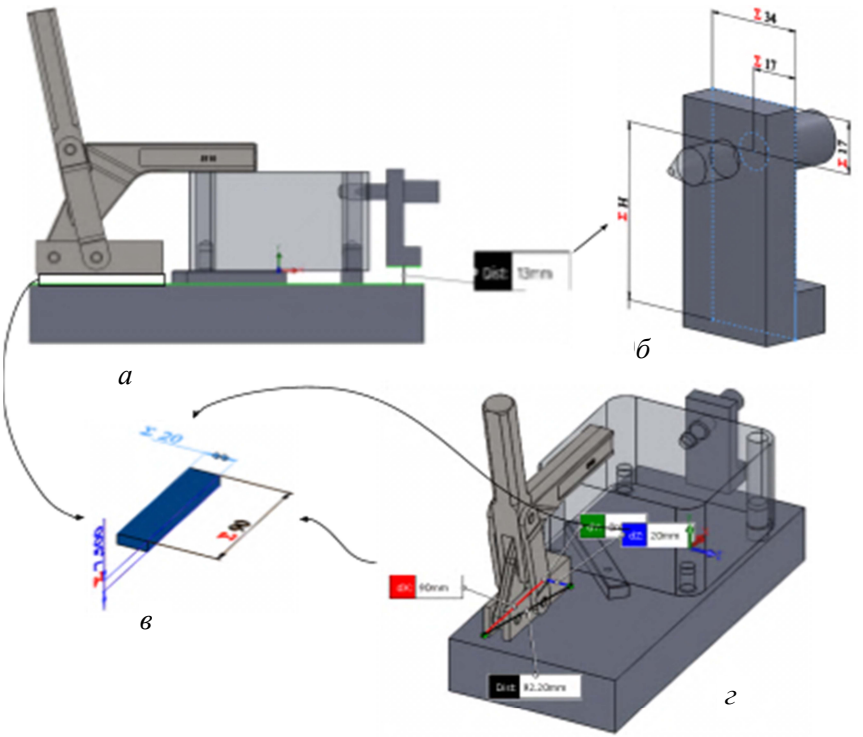


Рис. 2.6. Согласование элементов приспособления с корпусной плитой

[9] Которой он опирается на корпус.

2.2. Автоматизация операций сборки

2.2.1. Цели и задачи

Цель работы – ознакомиться с методами и средствами автоматизированного формирования сборных моделей в системах геометрического моделирования. Для достижения поставленной цели предлагается **разработать процедуру** добавления в текущий документ компонента из внешнего файла.

Для выполнения требуемых построений средства автоматизации проектирования взаимодействуют с СГМ. Последние для этого предоставляют соответствующий интерфейс программирования (Application Programming Interface, API). При этом возникают вопросы, как организовать взаимодействие, какие использовать классы, методы и др. Этим вопросам и посвящена данная работа. На примере достаточно простой операции вставки компонента рассматривается структура приложения, основные компоненты, подбор объектов и др.

2.2.2. Порядок выполнения работы

Перед началом работы следует создать новый документ класса «сборка» – **текущую модель**. СГМ при этом создает объект – экземпляр класса *AssemblyDoc*, в котором определены необходимые для сборочных операций методы.

Также следует подготовить **тестовый пример:** сборочную геометрическую модель, включающую несколько компонентов. Таким примером может быть модель, разработанная в рамках проектирования приспособлений для установки и закрепления деталей.

Разработать процедуру, которая добавит в **текущую модель** те же элементы, что присутствуют в **тестовом примере**.

2.2.3. Методические указания

Прототип процедуры вставки компонентов в сборку можно получить при помощи **макрорекордера** – специального сервиса, поддерживающего протоколирование интерактивных действий[10].

[10] Альтернативный вариант – изучить справочник API.

Освоение методов и средств программирования геометрических построений – сложный, трудоемкий процесс. Необходимо выяснить процедуры и функции, которые следует использовать, в каких классах они определены, что за объекты создаются и т. п. Все это используется базовой системой геометрического моделирования при выполнении штатных команд и, соответственно, может быть записано на каком-либо языке программирования. Получаемые в результате тексты называют ***макрокомандами*** или ***макросами***, а записывающее средство – макрорекордером.

Команды управления макросами доступны в падающем меню «**Инструменты → Макрос**». Кроме того, можно включить соответствующую кнопочную панель.

Макрокоманда создается следующим образом:

– включить запись макроса;

– добавить в текущий документ какой-либо внешний компонент в интерактивном режиме командой «Вставить компоненты» (панель инструментов «Сборка»);

– выключить запись макроса.

При выключении записи макроса поступит запрос на его сохранение.

При сохранении макроса кроме имени файла можно указать его **формат**. В общем случае возможны следующие варианты:

– **Visual Basic for Application** (VBA) – тип файла SW **VBA** *Macro* (**.swp*);

– **Visual Basic. NET** (VB.NET), тип файла – SW **VSTA** VB *Macro* (**.vbproj*);

– ***C#***, тип файла – SW **VSTA** *C# Macro* (**.csproj*).

Два последних поддерживаются средствами **VSTA** – **MS Visual Studio Tools for Application**, которые не всегда доступны. **VBA** – встроенное в SolidWorks инструментальное средство, доступно всегда[11]. Поэтому далее рассматривается именно этот вариант.

Файл, созданный представленным выше образом, открывается для редактирования, причем сделать это можно только в среде создавшей его системы геометрического моделирования (в данном случае – SolidWorks).

[11] Если только не был отключен при установке. В таком случае его можно доустановить.

В начале программы содержится несколько объявлений глобальных переменных, которые поясняются далее:

```
Dim swap As Object
Dim Part As Object
Dim boolstatus As Boolcan
Dim longstatus As Long, longwarnings As Long
```

Затем процедура *Main*. Начинается она с получения ссылок на приложение SolidWorks – переменная *swApp*, и на активный документ – переменная *Part*.

```
Sub main()
Set swApp = Application.SldWorks
Set Part = swApp.ActiveDoc
```

Следующие строки – получение ссылки на модуль инженерного анализа *CosmosWorks*. В данной работе он не используется и эти строки могут быть просто удалены.

```
Dim COSMOSWORKSObj As Object
Dim CWAddinCallbackObj As Object
Set CWAddinCallbackObj = swApp.GetAddlnObject("CosmosWorks.
CosmosWorks")
Set COSMOSWORKSObj = CWAddinCallbackObj.COSMOSWORKS
```

Далее идет открытие файла вставляемого компонента, «*Locator3.SLDPRT*» методом *OpenDoc6*.

Перед этим имя сборки, куда вставляется этот компонент, сохраняется в переменной *AssemblyTitle*, с тем чтобы после открытия вновь сделать эту сборку текущим документом методом *ActivateDoc3*.

```
Dim AssemblyTitle As String
AssemblyTitle = Part.GetTitle
Dim impObj As ModelDoc2
Dim errors As Long
Set tmpObj = swApp.OpenDoc6("D:\Fix\Units\Locator3.SLDPRT", 1,
32, "", errors, longwarnings)
Set Part = swApp.ActivateDoc3(AssemblyTitle, True, 0, errors)
```

Непосредственно вставка компонента в модель выполняется методом *AddComponent5*:

```
Dim swInsertedComponent As Component2
Set InsertedComponent = Part.AddComponent5("D:\Fix\Units\Locator3.SLDPRT", 0, "", False, "", 0, 0, 0)
```

Вместо метода *AddComponent5* можно использовать метод *AddComponent2* – он требует меньше аргументов (только имя и координаты):

```
Set InsertedComponent = Part.AddComponent2("D:\Fix\Units\Locator3.SLDPRT", 0, 0, 0)
```

После вставки компонента его файл, открытый ранее, закрывается:

```
swApp.CloseDoc "D:\Fix\Units\Locator3.SLDPRT"
```

Полные имена файлов, подаваемые в качестве аргументов в методы *OpenDoc6*, *AddComponent5* и *CloseDoc*, можно сократить, если и сборка, и добавляемые компоненты будут в одном каталоге[12].

Цифры в конце имен методов – номера версий. Последующие версии, вероятно, совершеннее предыдущих, однако к первым версиям проще обращаться. Так, например, методу *OpenDoc* передаются только два аргумента – имя файла и его тип. Методу *AddComponent2*[13] также достаточно только имени и аргументов. Таким образом, старые версии методов могут оказаться более эффективными.

Следующий фрагмент – редактирование матрицы, описывающей положение компонента:

```
Dim TransformData() As Double
ReDim TransformData(0 To 15) As Double
```

[12] Файл макрокоманды тоже лучше поместить в этот каталог.

[13] *AddComponent* требует такой же список аргументов, однако он возвращает «логический» результат вставки, в то время, как *AddComponent2* – ссылку на вставленный компонент, которая может быть полезна для последующих операций.

```
    TransformData(0) = 1
    TransformData(1) = 0
    TransformData(2) = 0
    TransformData(3) = 0
    TransformData(4) = 1
    TransformData(5) = 0
    TransformData(6) = 0
    TransformData(7) = 0
    TransformData(8) = 1
    TransformData(9) = 0
    TransformData(10) = 0
    TransformData(11) = 0
    TransformData(12) = 1
    TransformData(13) = 0
    TransformData(14) = 0
    TransformData(15) = 0
    Dim TransformDataVariant As Variant
    TransformDataVariant = TransformData
    Dim swMathUtil As Object
    Set swMathUtil = swApp.GetMathUtiliti()
    Dim swTransform As Object
    Set swTransform = swMathUtil.CreateTransform(TransformData
Variant)
    boolstatus = swInsertedCompontnt.SetTransformAndSolve2(swTrans-
form)
```

В рамках данной работы позиционирование компонентов не предполагается. Поэтому данный фрагмент подробно не рассматривается.

В завершение макроса удаляются объекты (некоторые), создаваемые при его выполнении.

```
    StudyManagerObj = Nothing
    ActiveDocObj = Nothing
    Set CWAddinCallBackObj = Nothing
    Set COSMOSWORKSObj = Nothing

    End Sub
```

2.2.4. Внешнее приложение

Приведенный в предыдущем разделе код исполняется средой VBA – внутренним, как было отмечено, средством SolidWorks. Система геометрического моделирования может также взаимодействовать и с **внешними** приложениями. Структура такого приложения может оставаться абсолютно такой же: те же классы, методы, порядок созданий и обращений и др. Особенность – его необходимо **подключить** к SolidWorks.

Подключиться к внешнему приложению можно, используя *Component Object Model* (COM) – технологический стандарт от компании Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Средства, поддерживающие COM-технологию, определены в пространстве имен *System.Runtime.InteropServices*. В частности, там есть класс *Marshall*, в котором, в свою очередь, определен метод *GetActiveObject*, возвращающий ссылку на активное приложение по его *программному идентификатору*. Ниже приведен *C#*-вариант (сокращенный) рассмотренной ранее *VBA*-процедуры *Main*.

```
…
using System.Runtime.InteropServices;
using SolidWorks.Interop.sldworks;
...
{
classProgram
  {
static void Main(string[] args)
    {
          object _swApp = Marshal.GetActiveObject("sldworks.application");
          ISldWorks swApp = _swApp as ISldWorks;
          object actDoc = swApp .ActiveDoc;
          AssemblyDoc Part = actDoc as AssemblyDoc;
          int errors = 0;
          int longwarnings = 0;
```

```
            swApp .OpenDoc6("D:\Fix\Units\Locator3SLDPRT", 1, 0, "",
            errors, longwarnings);
            Component2 Comp = Part.AddComponent2("D:\Fix\Units\Lo-
            cator3SLDPRT", 0.0, 0.0, 0.0);
            swap .CloseDoc("D:\Fix\Units\Locator3SLDPRT")
    }
  }
}
```

Следует обратить внимание на то, как получается объектная переменная *swApp* – ссылка на приложение SolidWorks. Метод *GetActiveObject* не возвращает объект класса *ISldWorks*. Поэтому сначала создается переменная *_swApp* класса *object*, которая затем явно приводится к классу *ISldWorks*. Аналогичным образом создается переменная *Part* – ссылка на документ.

Далее следует фрагмент кода на языке Python, выполняющий действия, аналогичные рассмотренным ранее.

```
import win32com.client as w32
swApp = w32.Dispatch('sldworks.application')
Part = swApp.ActiveDoc
swApp.OpenDoc('D:\Fix\Units\Locator3SLDPRT', 1)
Comp = Part.AddComponent('D:\Fix\Units\Locator3SLDPRT', 0.0,
0.0, 0.0)
swApp.CloseDoc('D:\Fix\Units\Locator3SLDPRT')
```

Структура кода такая же: подключение модуля *win32com.client*, поддерживающего COM-технологию, затем подключение к SolidWorks (функция *Dispatch*), затем к документу сборки и т. д.

2.2.5. Справка по API

Справочник интерфейса программирования приложений доступен из раздела «Справка» падающего меню SolidWorks. Можно также воспользоваться ресурсом https://help.solidworks.com в Internet[14]

[14] Раздел API в самом конце списка.

или справочным файлом из каталога установки *..\SOLIDWORKS Corp\SOLIDWORKS\api\apihelp.chm*. Информация, необходимая для выполнения работ данного практикума, находится в разделе справочника **SOLIDWORKS API Help** в подразделе пространства имен **SolidWorks.Interop.sldworks** (рис. 2.7).

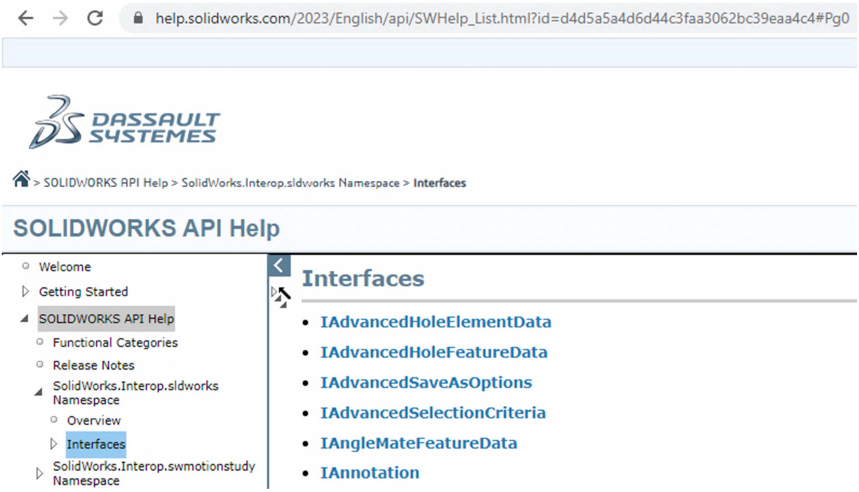


Рис. 2.7. Справочник API SolidWorks

Методам и средствам разработки приложений на базе системы геометрического моделирования SolidWorks посвящены встроенные учебные пособия: раздел падающего меню «Справка» → «**Учебные пособия SolidWorks**» → «**Инструменты повышения производительности**».

2.3. Автоматизация сопряжений

2.3.1. Цель и задачи

Цель работы – освоение эффективных методов автоматизации позиционирования элементов сборочных моделей.

Размещение компонентов конструкции друг относительно друга – одна из важнейших процедур автоматизации конструирования.

Положение каждого компонента описывается специальной матрицей, вычисление элементов которой – задача достаточно сложная.

Современные средства геометрического моделирования поддерживают так называемую ассоциативную сборку: пользователь указывает пары сопрягаемых (ассоциируемых) друг с другом элементов[15], а система сама рассчитывает матрицу положения. Отмеченную возможность можно использовать и программно. Используемые при этом методы и средства являются предметом изучения предлагаемой работы.

Для достижения поставленной цели предлагается разработать процедуру, устанавливающую сопряжения деталей.

2.3.2. Порядок выполнения работы

Перед началом работы необходимо создать новый документ класса «сборка» – **текущую модель**.

В **текущую модель** интерактивно добавляется **тестовая деталь** – первый компонент, с которым будут сопрягаться все остальные.

На тестовой детали **предварительно выделяются** геометрические **элементы** – грани или элементы справочной геометрии – количеством, равным количеству устанавливаемых сопряжений.

Запускается процедура, которая (1) добавит в текущую модель **новые компоненты** (один компонент) и (2) установит сопряжения между этими **новыми компонентами** и **предварительно выделенными элементами тестовой детали**.

Результаты демонстрируются на базе подготовленного заранее **тестового примера**[16]. Т. е. разработанная процедура воспроизводит сопряжения, имеющиеся в этом примере.

2.3.3. Методические указания

Сопряжения устанавливаются методом *AddMate*:

instance.AddMate(MateType, Align, Flip, Dist, Angle),

[15] Граней, ребер, вспомогательных элементов и т. п.

[16] Таким примером может быть модель, разработанная в рамках проектирования приспособлений для установки и закрепления деталей.

где *instance* – экземпляр класса *AssemblyDoc* – документ, в котором создается модель, *MateType* – **тип** устанавливаемого **сопряжения**, *Align* – **тип выравнивания**[17].

Тип сопряжения указывается целочисленным кодом, например, 0 – совпадение плоскостей, 1 – соосность и т. д. (см. справочник).

Тип выравнивания – взаимное расположение **векторов** сопрягаемых поверхностей – нормалей плоскостей, осей цилиндрических поверхностей и т. п. Значение «0» данного аргумента означает сонаправленность, «1» – противоположность. Предусмотрено также значение «–1» – тип выравнивания вновь создаваемого сопряжения определяется автоматически с учетом уже заданных.

Сопрягаемые объекты не передаются в метод аргументами, а должны быть выделены перед его вызовом. Следует отметить, что выделены должны быть **два и только два** объекта и принадлежать они должны **разным компонентам**.

В качестве примера рассматривается взаимное позиционирование пальца и детали (рис. 2.8). Палец должен входить в отверстие до упора буртиком в тыльную сторону детали[18]. Для этого необходимо установить два сопряжения: совпадение плоскостей буртика (*Burt*) с плоскостью детали (*Plan*) и соосность осей пальца (*Pin*) и отверстия (*Hole*).

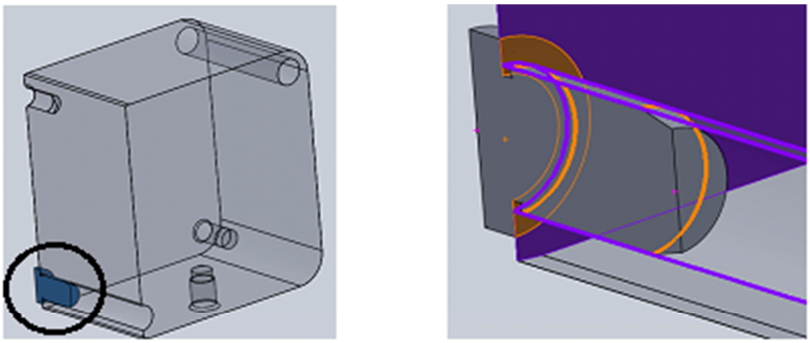


Рис. 2.8. Сопряжение установочного пальца с отверстием

[17] Остальные аргументы в данном контексте несущественны, с ними можно ознакомиться самостоятельно при помощи справочника.

[18] Типовое сочетание установки по цилиндрическим поверхностям и плоским торцам.

Код, устанавливающий такие сопряжения, выглядит следующим образом:

```
instance.ClearSelection
Plan.Select
Burt.Select
instance.AddMate 0, 0, False, 0, 0
instance.ClearSelection
Hole.Select
Pin.Select
instance.AddMate 1, –1, False, 0, 0
```

Метод *ClearSelection* очищает пространство проектирования от случайных выделений, которые могли остаться от предыдущих манипуляций с моделью.

Тип сопряжения должен соответствовать сопрягаемым объектам. Для сопряжения «*совпадение»* (первое обращение к *AddMate*) выделяются две плоскости, для сопряжения «*соосность»* (второе обращение) – два цилиндра. Если выделить цилиндр и плоскость, ни одно из указанных сопряжений не установится.

Тип выравнивания (второй аргумент) для совпадения плоскостей указан 0: плоскости должны упираться друг в друга, соответственно, векторы нормалей направляются в противоположные стороны. Для соосности тип выравнивания указывается –1: система сама выберет значение так, чтобы не было конфликта с установленным ранее совпадением плоскостей.

Сопрягаемые объекты в рамках данной работы можно разделить на две категории. Первая – элементы детали – плоскость *Plan*, и отверстие *Hole*. Эти объекты **предварительно выделены** интерактивно до вызова процедуры сопряжения. Вторая категория – элементы пальца *Pin* и *Burt* – компонента, которого на момент вызова процедуры в документе **еще нет**.

Ссылки на **предварительно выделенные** объекты хранятся в специальном контейнере. В SolidWorks – это свойство документа *SelectionManager*:

Selection_manager = *instance.**SelectionManager***.

Доступ непосредственно к объектам обеспечивает метод *GetSelectedObject* этого контейнера:

*value = Selection_manager.**GetSelectedObject**(Index)*,

где *Index* – порядковый номер объекта в контейнере, который соответствует очередности выделения этого объекта. **Предварительно выделенные объекты** следует переписать в какую-либо структуру данных[19]:

```
Dim InnerStructure As Collection
Set InnerStructure = New Collection
For i = 1[20] To .GetSelectedObjectCount
InnerSructure.Add    Selection_manager.GetSelectedObjectsComponent(i)
Next i
```

Поскольку в данном конкретном случае ожидается только два выделенных объекта, операцию можно упростить. Допустим, первой была выделена плоскость, а затем – отверстие. Тогда инициализация объектов сопряжения со стороны детали будет выглядеть следующим образом:

```
Set Plan = Selection_manager.GetSelectedObjectsComponent(1)
Set Hole = Selection_manager.GetSelectedObjectsComponent(2)
```

Элементы сопряжений со стороны пальца *Burt* и *Pin* должны быть найдены **программно**. Это предполагает знание геометрической модели, изучению которой посвящена отдельная работа. В данном контексте строки кода, где они выделяются – *Burt.Select* и *Pin.Select* – можно закомментировать, а на строках вызова метода *AddMate* определить *паузы* и «кликнуть» нужный объект вручную.

---

[19] Причем сделать это надо в самом начале процедуры, потому что любые манипуляции с документом «очищают» контейнер выделенных объектов и эта информация будет утеряна.

[20] Индексация начинается с 1, не с 0.

2.4. Извлечение информации из геометрических моделей в твердотельном формате

2.4.1. Цель работы и решаемые задачи

Цель работы – освоение методов и средств доступа к элементам твердотельных геометрических моделей и извлечения доступной информации.

Конструкторское проектирование предполагает большое количество исходной информации. Значительная часть этой информации может быть извлечена из геометрических моделей. Разработка соответствующего программного обеспечения способствует повышению эффективности средств автоматизации конструирования.

На уровне отдельных деталей современными системами геометрического моделирования используется твердотельный формат описания и предоставляется широкий набор интерфейсных средств для программной работы с ним.

Для достижения поставленной цели предлагается разработать код, реализующий поиск подходящих геометрических элементов деталей для сопряжения их между собой.

2.4.2. Порядок выполнения работы

В работе, посвященной автоматизации сопряжений, разрабатывалась процедура, которая, в частности, сопрягала предварительно выделенные элементы тестовой детали, уже присутствующей в документе, с **новыми компонентами**, **добавленными** в этот документ **этой же процедурой**. Элементы сопряжений со стороны этих новых компонентов предлагалось выделить интерактивно, установив предварительно паузу для выполнения процедуры. В данной работе предлагается **дополнить** процедуру кодом, который **автоматически** найдет подходящие элементы в моделях **новых компонентов**.

2.4.3. Методические указания

Твердотельная модель включает **конструктивную геометрию** и **граничное представление**.

Конструктивная геометрия (*constructive solid geometry – CSG*) представляет логику геометрической модели, ее структуру, способ построения. Основная ее часть – это множество **объемных фигур**, соединенных бинарными **теоретико-множественными операциями** (рис. 2.9). **Объемная фигура** – фигура, поверхность которой описана аналитически, кинематическим или каркасным способом. **Теоретико-множественные операции** – **объединение** (*union*), **пересечение** (*intersection*), **вычитание** (*subtraction*). То есть непрерывные фигуры комбинируются между собой, образуя комплексные **тела**.

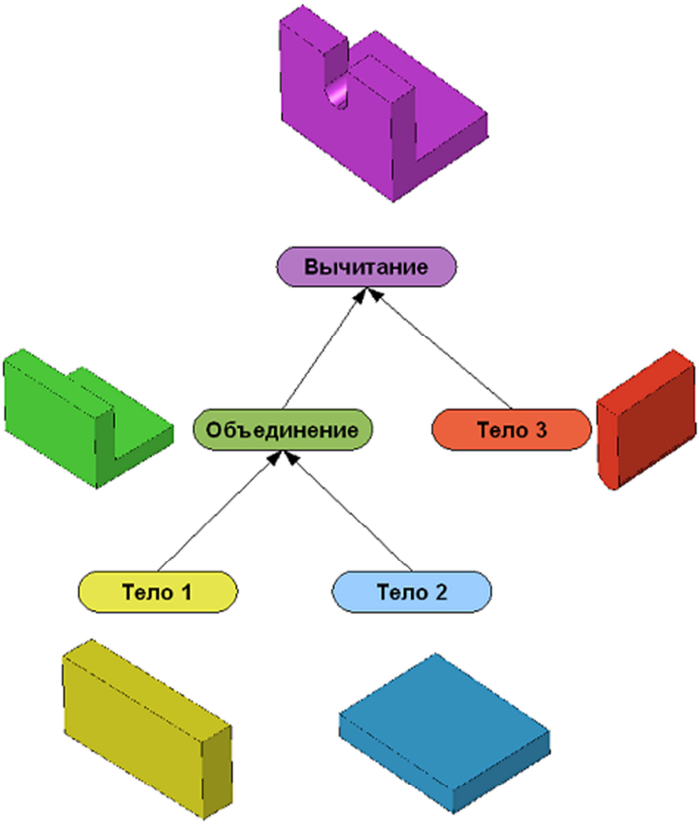


Рис. 2.9. Твердотельная модель

Конструктивная геометрия также включает *вспомогательные* (или справочные, **reference**) элементы – плоскости[21], оси, точки.

В системах геометрического моделирования конструктивная геометрия отображается рядом с графическим полем в виде древовидной структуры, именуемой **Feature manager design tree**. Все элементы этой структуры относятся к конструктивной геометрии и являются **свойствами** – объектами класса *feature*.

Конструктивная геометрия создается пользователем. Каждый ее элемент – это уникальное свойство, имеющее **собственное имя**, по которому может быть найдено в структуре модели.

Граничное представление (*Boundary REPresentation – BREP*) – описание оболочки фигуры, ее **внешних границ**.

Границы **тела** – это **грани**, границы **граней** – это **ребра**, границы **ребер** – **вершины**. Таким образом, граничное представление – это множество граней – объектов класса *face*, ребер – класс *edge* и вершин – класс *vertex*.

Граничное представление генерируется системой исходя из конструктивной геометрии. Элементы граничного представления не имеют индивидуальных отличительных признаков (собственных имен, например). Однако граничное представление однозначно, в отличие от конструктивной геометрии. Другими словами, одну и ту же фигуру можно сконструировать различными способами, но комплект граней будет одинаковым.

Доступ к элементам твердотельной модели.

Доступ к конструктивной геометрии и граничному представлению дает **компонент сборочной модели**. В SolidWorks – это объект класса *Component*. Полный перечень всех компонентов текущей сборки возвращает метод *GetComponents*:

$$Components = instance.GetComponents(True),$$

где *instance* – объект класса *AssemblyDoc*.

Перечень компонентов возвращается в виде одномерного массива, перебрав который, можно выбрать подходящий:

[21] Например, плоскости: «сверху» («top»), «справа» («right»), «спереди» («front»), изначально присутствующие в каждом документе и определяющие, по сути, систему координат.

For i = 1[22] *To . Ubound(Components)*[23]
Процедура_выбора_компонента(Components (i))
Next i

Ссылку на *только что вставленный* компонент возвращает метод *AddComponent* версии 2 и выше:

Set Component = instance.AddComponent2(Name, X, Y, Z).

Свойства конструктивной геометрии – объекты класса *feature* – организованы в структуру, подобную односвязному списку (*Link List*). В классе *Component* определен метод *FirstFeature*, возвращающий, как следует из названия, первое свойство компонента. В классе *feature* есть метод *GetNextFeature*. Используя указанные методы, можно пройти всю конструктивную геометрию и найти нужное свойство по каким-либо признакам.

Если известно точное имя свойства, его можно найти методом *FeatureByName*:

Set theFeature = Component.FeatureByName(<имя_свойства>).

Для получения доступа к элементам граничного представления необходимо сначала извлечь так называемое «тело» модели – объект класса *Body2* компонента:

Set ComponentBody = Component.GetBody.

Грани, ребра, вершины извлекаются из этого контейнера в виде массивов при помощи соответствующих методов. Например, грани – методом *GetFaces*:

CompFaces = CompBody.GetFaces.

Поскольку грани, как отмечалось ранее, собственных имен не имеют, для выбора нужной в той или иной ситуации необходимо анализировать их свойства.

---

[22] Индексация начинается с 1, не с 0.

[23] Функция Basic, возвращающая наибольший индекс массива.

Следует отметить, что часть свойств грани связана с поверхностью, участком которой эта грань является. Это, например, уравнения плоскости, оси, радиусы цилиндрических (конических) поверхностей и т. п. Эти свойства доступны из объекта класса *Surface*, возвращаемого методом *GetSurface* класса *Face*:

SetSurface = Face.GetSurface.

Детально ознакомиться со свойствами элементов конструктивной геометрии и граничного представления предлагается самостоятельно по справочнику по API системы SolidWorks.

Практическое использование геометрической информации.

Для полноценной автоматизации сопряжения установочного пальца с деталью (рис. 2.8) необходимо программно определить объекты *Burt* и *Pin*, соответствующие верхней плоскости буртика пальца и его установочному цилиндру.

Ссылка на сам палец сохраняется при его добавлении в модель:

Set Palec = instance.AddComponent2(Pflec_file_name, 0, 0, 0).

В качестве объекта *Pin* подойдет любая цилиндрическая поверхность[24]. Соответственно, искать можно в граничном представлении:

```
For i = 0 To UBound(Faces)
If  Face(i).GetSurface.IsCylinder Then Set Pin = Face(i)
Next i
```

Объект *Burt* нельзя определить аналогичным образом. Необходимо проанализировать больше информации, например, расположение каждой плоской грани относительно остальных. Однако в качестве элемента сопряжения подойдет вспомогательная плоскость SettingPlan (рис. 2.10). Она является свойством конструктивной геометрии, обладает собственным именем и может быть выделена соответствующим образом:

Set Burt = Palec.FeatureByName(«SettingPlane»).

[24] В общем случае подобное упрощение недопустимо.

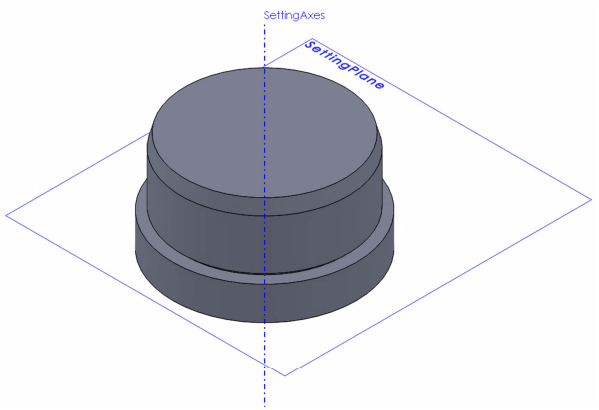


Рис. 2.10. Палец установочный

2.5. Проектирование корпусной плиты

2.5.1. Цель работы и решаемые задачи

Цель работы – освоение эффективных методов и средств получения данных из сборочных геометрических моделей (Assembly Model) и решения с их помощью конструкторских задач.

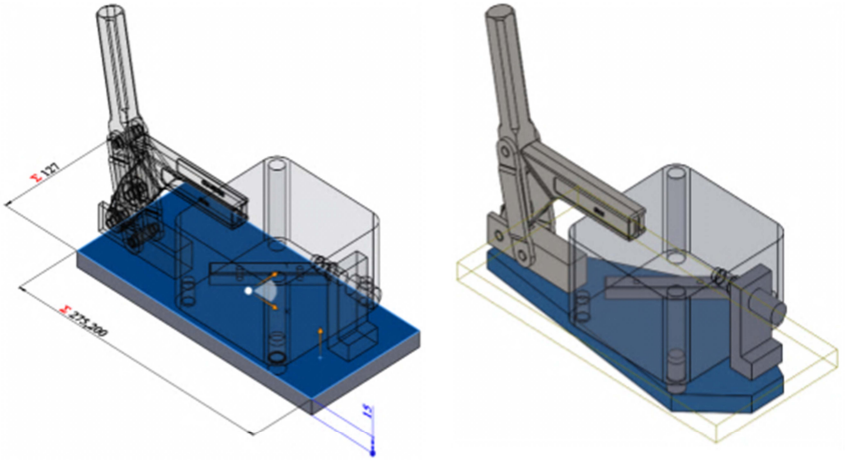
Для достижения поставленной цели предлагается разработать программные средства:

– размещения корпусной плиты прямоугольной формы (рис. 2.11, *а*) относительно компонентов сборки;

– изменения размеров корпусной плиты прямоугольной формы в зависимости от размеров компоновки;

– построения корпусной плиты, ограниченной произвольным выпуклым контуром, определяемым размещением элементов конструкции (рис. 2.11, *б*).

В качестве объекта проектирования в данной работе предлагается **корпусная плита** – элемент конструкции, на котором устанавливаются все остальные ее детали и узлы. То есть форма плиты, ее размеры и расположение определяются уже существующими элементами конструкции, что делает ее подходящим объектом для достижения поставленной цели.



а *б*

Рис. 2.11. Корпусные плиты: прямоугольная (*а*) и контурная (*б*)

2.5.2. Порядок выполнения работы

Для выполнения работы необходима **тестовая сборка** – модель, включающая несколько компонентов. Лучше всего в данном качестве использовать результат проектирования приспособлений для установки и закрепления деталей, удалив из него корпус.

1. Выполнить процедуру, которая добавит в тестовую сборку новый компонент – прямоугольную плиту – и разместит ее ниже всех элементов конструкции. При этом общий геометрический центр элементов должен проецироваться в центр плиты.

2. Выполнить процедуру, которая определит габариты сборки и скорректирует соответствующим образом длину и ширину корпусной плиты.

3. Выполнить процедуру построения корпусной плиты, ограниченной выпуклым ломанным контуром.

2.5.3. Методические указания

Формирование перечня элементов.

Каждый **элемент конструкции** – деталь или узел – представлен в геометрической модели **компонентом** – объектом класса *Compo-*

nent (*Component2*). Перечень этих объектов можно получить методом *GetComponents*. Данный метод определен в классе *AssemblyDoc*. Последний представляет сборочную модель – производный класс по отношению к документу модели. Получить на него ссылку на приложение *SolidWorks*.

```
Dim swApp As Object
Set swApp = Application.SldWorks
Dim Assembly As AssemblyDoc
Set Assembly = swApp.ActiveDoc
Dim Components[25]
Components = Asm.GetComponents(True)
```

Результат метода *GetComponents* – массив объектов класса *Component*. Так, например, для приведенной ниже компоновки (рис. 2.12) будет возвращен массив из четырех элементов.

Расположение компонента.

Можно выделить следующие характеристики, указывающие на то, где находится какой-либо объект: **точка**, **ориентация** относительно этой точки, **область** пространства. Первые две характеристики представляет **привязочная система координат** (**ПСК**, система координат (X1, Y1, Z1), рис. 2.12), третью – **габаритный параллелепипед** (**ГБП**, параллелепипед (min, max), рис. 2.12).

ПСК описывается матрицей положения 4×4:

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & p_1 \\ c_{21} & c_{22} & c_{23} & p_2 \\ c_{31} & c_{32} & c_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (1)$$

где c_{ij} – косинус угла между i-й осью системы координат (СК) сборки и j-й осью собственной СК компонента;

p_i – i-я координата точки начала СК компонента в СК сборки.

[25] Тип данных *Variant*: принимает значения любого типа, удобен для работы с массивами.

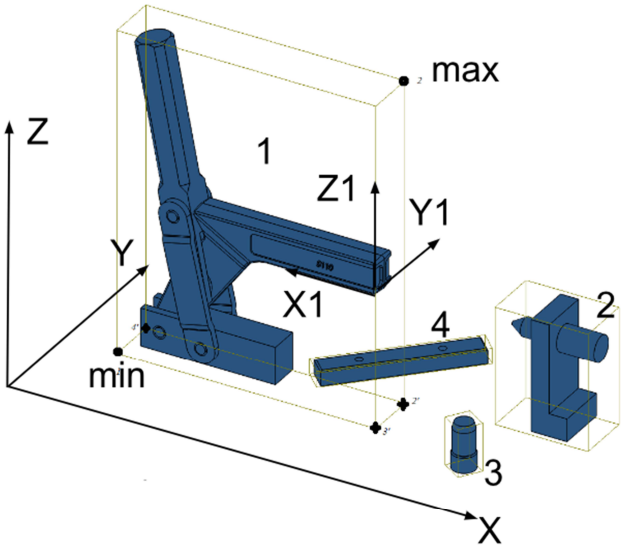


Рис. 2.12. Элементы конструкции

Матрица положения компонента доступна через его свойство *Transform* и представляется объектом класса *MathTransform*.

Свойство *ArrayData* этого класса – элементы матрицы положения в виде массива, одномерного, шестнадцатиэлементного. Метод *GetData* возвращает элементы матрицы в объектном виде: векторы осей и точку начала собственной СК.

ГБП возвращает метод *GetBox*. Он определен как в классе *Component*, представляющем отдельные элементы, так и в классе *AssemblyDoc*, представляющем сборку целиком.

ГП возвращается как массив из 6 элементов – координаты вершин габаритного параллелепипеда – $\boldsymbol{x}_{\min}, \boldsymbol{y}_{\min}, \boldsymbol{x}_{\min}, \boldsymbol{x}_{\max}, \boldsymbol{y}_{\max}, \boldsymbol{z}_{\max}$.

Координаты возвращаются в системе координат (СК) объекта, *из-под которого* были вызваны. Поэтому ГП компонентов сборки необходимо пересчитать из локальных СК элемента в СК сборки.

Преобразование координат из локальной СК компонента в СК сборки выполняется умножением матрицы положения компонента на матрицу-столбец координат:

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & p_1 \\ c_{21} & c_{22} & c_{23} & p_2 \\ c_{31} & c_{32} & c_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

где X, Y, Z – преобразуемые координаты.

Эту операцию можно выполнить процедурой, реализующей соответствующую операцию матричной алгебры, используя свойство *ArrayData* матрицы положения.

Кроме того, пересчитываемые координаты можно представить объектом класса *MathPoint* методом *CreatePoint* из интерфейса *IMathUtility*:

Point = instance.ICreatePoint(ArrayDataIn),

где *instance* – ссылка на объект класса *IMathUtility* (берется из ссылки на приложение *SolidWorks*, подобно ссылке на активный документ, например); *ArrayDataIn* – преобразуемые координаты в виде трехэлементного массива *double*. В классе *MathPoint* определен метод *MultiplyTransform*, который позволяет преобразовать эти координаты:

newPoint = Point.MultiplyTransform(TransformObjIn),

где *TransformObjIn* – матрица положения компонента.

ГП всей конструкции определяется либо методом *GetBox*, вызванным из сборки (объект *AssemblyDoc*), либо вычисляется по координатам ГП отдельных элементов.

Размещение корпусной плиты.

Позиционирование посредством сопряжений для корпусной плиты не подходит. Она предназначена для установки **всех** элементов и со всеми, соответственно, должна быть согласована. Поэтому параметры ее расположения должны быть рассчитаны и переданы в матрицу положения плиты.

Начало собственной СК плиты связывается с центром ее верхней грани (рис. 2.13), а ось Z – с нормалью к этой грани.

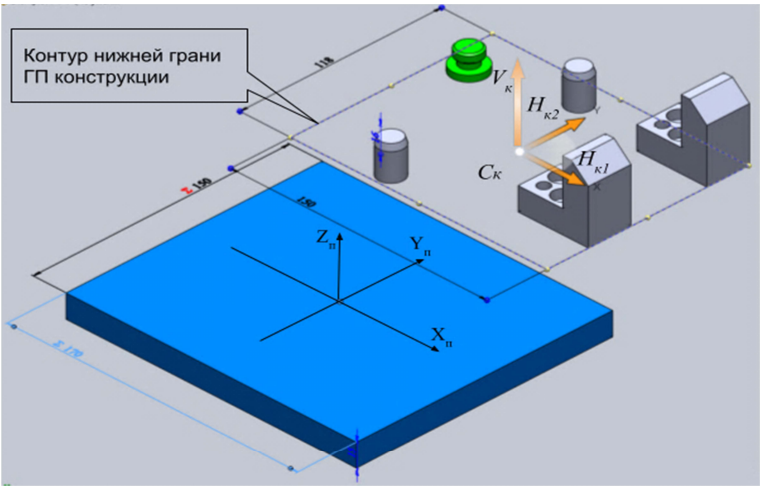


Рис. 2.13. Корпусная плита прямоугольной формы

Для того чтобы разместить прямоугольную плиту относительно элементов конструкции, ее матрицу положения следует заполнить следующим образом:

$$\begin{bmatrix} H_1^x & H_2^x & V_к^x & C_к^{H_1} \\ H_1^y & H_2^y & V_к^y & C_к^{H_2} \\ H_1^z & H_2^z & V_к^z & V_{\min} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

где $V_к$ – вертикальная ось конструкции;

H_1 и H_2 – горизонтальные оси;

$C_к^{H_1}$ и $C_к^{H_2}$ – координаты центра конструкции относительно горизонтальных осей;

$V_{\min}$ – минимальная координата конструкции по вертикальной оси.

Горизонтальные оси – две из осей СК пространства проектирования, X, Y или Z, с которыми совпадает горизонталь конструкции. Соответственно, вертикальная ось – та, которая совпадает с ее вертикалью. Проектирование в системе геометрического моделирования происходит, как правило, так, что вертикальной является одна

из осей главной СК текущего документа. Соответственно, две другие оси – горизонтальные. Исходя из этого, косинусная часть матрицы положения заполняется 0 и 1. Так, например, конструкция на рис. 2.12 выстроена так, что вертикалью является ось *Z*, а оси *X* и *Y* – горизонталями. Соответствующий пример направляющих косинусов показан на рис. 2.14, *а*. Если же вертикалью является ось *Y*[26], матрица заполняется иначе (рис. 2.14, *б*).

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

а *б*

Рис. 2.14. Направляющие косинусы матрицы положения при вертикали конструкции, совпадающей с осью *Z* (*а*), и осью *Y* (*б*)

Центр конструкции $C_к$ – геометрический центр ГП конструкции.

Передача вычисленных значений в модель осуществляется через матрицу положения плиты – свойство *Transform*. Ниже приведены фрагменты соответствующего кода[27].

```
Dim swInsertedComponent As Component2
Set swInsertedComponent=Part.AddComponent5("Korpus.SLDPRT",
0, "", False, "", 0.3, 0.2, 0.3)
...
Dim TransformData() As Double
ReDim TransformData(0 To 15) As Double

Направляющие косинусы СК корпусной плиты
TransformData(0) = 1‘c11 (выражение 1)
TransformData(1) = 0‘c21
```

[26] В SolidWorks пространство проектирования по умолчанию ориентируется именно так.

[27] Записывается макрорекордером при интерактивном выполнении процедуры добавления компонента.

TransformData(2) = 0‘c_{31}
TransformData(3) = 0‘c_{12}
TransformData(4) = 1‘c_{22}
TransformData(5) = 0‘c_{32}
TransformData(6) = 0‘c_{31}
TransformData(7) = 0‘c_{32}
TransformData(8) = 1‘c_{33}

Координаты точки привязки СК корпусной плиты в СК сборки
TransformData(9) = 0.3‘p_1
TransformData(10) = 0.2‘p_2
TransformData(11) = 0.3‘p_3

Коэффициенты масштабирования плиты по осям и общий коэффициент (не меняются)
TransformData(12) = 1
TransformData(13) = 0
TransformData(14) = 0
TransformData(15) = 0

Dim TransformDataVariant As Variant
TransformDataVariant = TransformData
‘ Создание объекта класса MathUtility
Dim swMathUtil As Object
Set swMathUtil = swApp.GetMathUtility()

Создание объекта класса *MathTransform*, который представляет матрицу положения. Массив *TransformDataVariant*, созданный ранее, передается при этом в качестве аргумента:

Dim swTransform As Object
Set swTransform = swMathUtil.CreateTransform ((TransformDataVariant))
*‘*Созданная матрица положения передается в компонент
boolstatus = swInsertedComponent.SetTransformAndSolve2 (swTransform)

Программное редактирование параметров.

Параметризация – мощное средство повышения эффективности геометрического моделирования. Интерактивное редактирование

параметров – достаточно трудоемкая процедура. В некоторых ситуациях ее можно выполнить автоматически. Это касается, например, функциональных размеров конструктивных элементов САПР, значения которых определяются свойствами конструкции, с которыми должен взаимодействовать элемент. В частности, размеры корпусной плиты (рис. 2.15) должны быть таковы, чтобы на нее уместились все элементы конструкции.

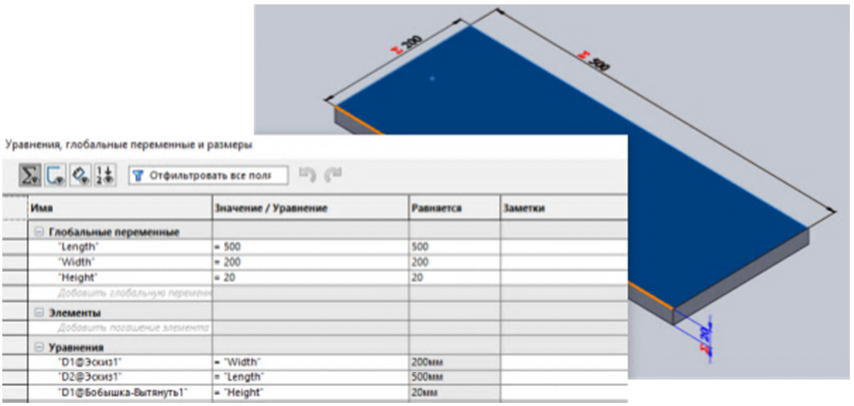


Рис. 2.15. Выражения параметров плиты и размеров

Менеджер выражений (IEquationMgr) – специальный интерфейс для работы с выражениями в системе геометрического моделирования SolidWorks. Именно такими выражениями представлены параметры конструктивных элементов в его геометрической модели. Соответственно, редактирование параметров следует начать с получения доступа к **менеджеру выражений**:

Set ***Equation_Manager*** = *Model_Document.****GetEquationMgr***,

где *Model_Document* – ссылка на модель редактируемого объекта. Если этот объект редактируется в контексте сборки, то он – **компонент**, у которого есть «свой» менеджер выражений, который, в свою очередь, хранится в модели этого компонента. Доступ к последней предоставляет метод *GetModelDoc*:

Set ***Model_Document*** = *Component.****GetModelDoc.***

Доступ непосредственно к выражениям обеспечивается по индексу:

$$E = Equation(i).$$

где i – индекс выражения, его порядковый номер.

Таким образом, чтобы изменить какой-либо параметр, следует перебрать выражения (в цикле, например), найти то, которое этот параметр определяет, и изменить его должным образом. Поисковый признак – имя параметра. Например, чтобы изменить высоту плиты, следует найти строку, начинающуюся со слова Height и изменить ее правую часть.

Формат выражений строковый:

"<*имя_выражения*>=<*выражение*>".

Все имена – как выражения в целом, так и имена переменных в правой части – должны быть заключены дополнительно в кавычки. Например, выражение *D23 = 5.6* D12* в формате выражений SolidWorks будет выглядеть следующим образом: "***D23***"*=5.6**"***D12***". Вставка кавычек в строковое выражение вызывает некоторые затруднения при написании кода. Можно, например, использовать функции, возвращающие символ по его номеру в таблице ASCII-кодов (34):

Char(34) *&* *“D34”* *&* ***Char(34)*** *&* *“=5.6*”* *&* ***Char(34)*** *&* *“D12”* *&* ***Char(34)*** *= ""D23"=5.6*"D12""*

Значения параметров, передаваемые в выражения, вычисляются по данным, извлекаемым из существующей геометрии.

Контур плиты должен соответствовать контуру нижней грани ГП конструкции (рис. 2.13). Соответствующая информация должна была быть получена в ходе выполнения первой работы.

Построение контурной корпусной плиты.

Форма и размеры деталей объектами автоматизированного проектирования становятся достаточно редко. Однако существуют классы деталей, геометрия которых либо унифицирована в целом, либо может быть синтезирована из унифицированных элементов. Это, например, несущие детали – валы, кронштейны, корпусные элементы, в том числе и плиты.

Современные средства геометрического моделирования поддерживают широкий набор средств описания геометрии. В данной работе рассматриваются наиболее простые, но вполне эффективные *контурно-кинематические*[28] технологии, построения геометрических фигур и комбинирование их теоретико-множественными операциями.

Решение поставленной задачи предполагает решение следующих подзадач: определение **контекста** построения фигуры, **создание (построение) контура**, **построение объемной фигуры**.

Фигура должна строиться *в контексте* какой-либо детали – *PartDoc*. Подобные построения выполняются, например, в специальных документах класса «деталь» («part»). Тогда контекст определяется просто активным документом.

Если построение происходит в сборке, как в данной работе, контекстом фигуры должен стать один из ее компонентов. В интерактивном режиме нужный компонент выделяется, а затем включается режим *редактирования компонента*. Практически то же должна проделать процедура, выполняющая построения в автоматизированном режиме.

В первую очередь необходима объектная переменная – ссылка на этот компонент. Такая ссылка может быть сохранена при его вставке в сборку методом *AddComponent*. Можно также перебрать компоненты, уже имеющиеся в сборке, и выбрать нужный, например, по имени.

Если построения надо выполнить в новой детали, ее можно сначала создать методом *InsertNewVirtualPart* класса *ModelDoc2*:

longstatus = swDoc.InsertNewVirtualPart (swFaceOrPlane, swComponent),

где *longstatus* – код результата;

swFaceOrPlane – ссылка на плоскость, к которой будет привязана деталь (может быть *Nothing\null*);

swComponent – переменная класса *Component2*, в которой и будет сохранена ссылка на созданную деталь и которая, соответственно, может быть использована для дальнейших построений.

[28] Протягивание плоского контура вдоль направления на заданное расстояние.

Выбранный компонент должен быть выделен методом *Select*, а метод *EditPart* сборочного документа включит редактирование выделенного компонента:

Component.Select True
Assembly.EditPart

Для построения эскиза в контексте детали необходимо получить доступ к специальному менеджеру, поддерживающему работу с эскизами профилей. В SolidWorks это объект класса *SketchManager*:

Dim sketchMgr As SketchManager
Set sketchMgr = AsseModelDoc.SketchManager,

где *AsseModelDoc* – экземпляр класса *ModelDoc* – родительского класса для всех остальных классов документов. Следует обратить внимание на то, что данный экземпляр – это документ сборки, в которой редактируется компонент, а не документ редактируемого компонента.

В *SketchManager* создается новый эскиз, но для него необходимо заранее выделить плоскость.

В целом эта часть процедуры выглядит следующим образом:

Dim sketchPlane As Object: Set sketchPlane = Component.FeatureByName("Top")
skPlane.Select True
sketchMgr.InsertSketch True

Добавленный эскиз становится текущим – все последующие построения будут выполняться в нем.

Методы, строящие сегменты контура, также определены в *SketchManager*. Например, построение линейного отрезка, *CreateLine*:

sketchMgr.CreateLine Verts(i), Verts(i + 1), 0#, Verts(i + 2), Verts(i + 3), 0#

Аргументами данного метода являются координаты крайних точек отрезка. Они могут быть заранее подготовлены в виде массива, а контур построен в цикле.

Построение объемной фигуры в контексте детали – это создание в ней соответствующего свойства, *feature*. Для работы со свойствами необходим соответствующий менеджер – *FeatureManager*:

Dim featureMgr As FeatureManager
Set featureMgr = AsseModelDoc.FeatureManager

Как и в случае с эскизами, *AsseModelDoc* – ссылка на документ, причем самого верхнего уровня.

В менеджере свойств определены различные методы построения геометрических фигур. В данной работе можно воспользоваться методом *FeatureExtrusion2*, позволяющим построить **объемную** фигуру выдавливанием плоского контура на заданное расстояние, или методом *FeatureCut4*, строящим **полость** в объемной фигуре аналогичным образом:

Dim ConturPlate as Feature
Set ConturPlate = swDoc.FeatureManager.FeatureCut4(список аргументов[29]*)*

или

Set ConturPlate = swDoc.FeatureManager.FeatureExtrusion2(список аргументов)

Метод *FeatureExtrusion2* следует применять, если плита строится в пустой[30] детали.

Методом *FeatureCut4* можно воспользоваться лишь в случае, когда в детали уже есть какая-либо объемная полость – в противном случае не из чего будет вырезать. Например, в конструкцию можно добавить готовую корпусную плиту прямоугольной формы и отрезать контуром внешнюю часть[31].

---

[29] Наборы аргументов у обоих методов велики и с ними предлагается ознакомиться самостоятельно.

[30] Не имеющей еще никаких объемных фигур.

[31] Это управляется специальным аргументом.

3. ОПТИМИЗАЦИЯ ПРОЕКТНЫХ РЕШЕНИЙ

Большинство инженерных задач оптимизации чаще всего многокритериальны. Проектирование реальных объектов осуществляется с учетом большого числа критериев качества (материалоемкости, прочности, долговечности, производительности), которые часто конфликтуют между собой [24]. Как известно, оптимальные решения определяются среди допустимого множества решений, т. е. среди тех решений, которые удовлетворяют всем требованиям и ограничениям.

Одним из важнейших отличий многокритериальных задач от однокритериальных является принципиально другая структура получаемого оптимального решения. Вместо единственной оптимальной точки, как правило, речь идет о поиске целого множества несравнимых решений-альтернатив, известных как Парето-оптимальные точки. Решение считается Парето-оптимальным, если значение любого из критериев можно улучшить лишь в случае ухудшения хотя бы одного из оставшихся критериев.

Выбор окончательного решения из множества Парето-оптимальных точек выполняется на основании различных подходов. Например, реальную многокритериальную задачу часто подгоняют под один самый важный критерий или выполняют свертку критериев, оптимальное решение может находиться так же, как ближайшее к идеальной точке. Кроме того, выбор единственного, окончательного решения может осуществляться и экспертом.

Для того, чтобы повысить информативность экспертов и визуализировать процесс выбора окончательного решения, предлагается разработать приложение Matlab с графическим интерфейсом для решения задач многокритериальной оптимизации на основании нескольких подходов. Использование результатов, полученных с помощью различных подходов, поможет специалисту объективно проанализировать множество Парето-оптимальных точек и выбрать адекватное окончательное решение.

3.1. Основные понятия и определения многокритериальной оптимизации

Согласно [25] в теории многокритериальной оптимизации (МКО) решаются задачи принятия решений одновременно по нескольким

критериям. Задача МКО ставится следующим образом: требуется найти числа $x_1, x_2, \dots, x_n$, удовлетворяющие системе ограничений:

$$g_i(x_1, x_2, \dots, x_m) \le b_i, \quad i = 1, m, \tag{3.1}$$

для которых функции

$$F_k = f_k(x_1, x_2, \dots, x_n), \quad k = 1, K \tag{3.2}$$

достигают максимального значения.

Множество точек $X = (x_1, x_2, \dots, x_n)$, удовлетворяющих системе (3.1), образует *допустимую область* $D \subset R^n$. Элементы множества D называются *допустимыми решениями* или *альтернативами*, а числовые функции f_k, $k = 1, K$ – *целевыми функциями*, или *критериями*, заданными на множестве D. В формулировке задачи (3.1)–(3.2) присутствует K целевых функций. Эти функции отображают множество $D \subset R^n$ в множество $F \subset R^n$, которое называется *множеством достижимости*.

В векторной форме математическую модель МКО (3.1)–(3.2) можно записать следующим образом:

$$f(X) = (f_1(X), \dots, f_2(X)) \to \max, \quad X \in D, \tag{3.3}$$

Здесь $f(X)$ – вектор-функция аргумента $X \in D$.

Если функции $f_1, f_2, \dots, f_K$ достигают максимума в одной и той же точке $X* \in D$, то говорят, что задача (3.3) имеет *идеальное решение*.

Случаи существования идеального решения в многокритериальной задаче крайне редки. В случае отсутствия «идеального решения» в задаче (3.3) происходит поиск *компромиссного решения*. Для всякой альтернативы $X \in D$ вектор из значений целевых функций $(f_1(X), f_2(X), \dots, f_K(X))$ является *векторной оценкой* альтернативы X.

Пусть $X_1, X_2 \in D$. Если для всех критериев $f_1, f_2, \dots, f_K$ имеют место неравенства $f_k(X_2) \ge f_k(X_1)$, $k = 1, K$, причем если хотя бы одно неравенство строгое, то говорят, что решение X_2 предпочтительнее решения X_1. Условие *предпочтительности* принято обозначать в виде $X_2 \succ X_1$.

В задаче МКО точка $X_0 \in D$ *называется оптимальной по Парето*, если не существует другой точки $X \in D$, которая была бы предпочтительнее, чем X_0.

Точки, оптимальные по Парето, образуют множество точек, оптимальных по Парето (множество неулучшаемых или эффективных точек) $D_p \subset D$.

3.2. Подходы к решению многокритериальных задач

Оптимальные решения многокритериальной задачи следует искать только среди элементов множества альтернатив D_p. В этой области ни один критерий не может быть улучшен без ухудшения хотя бы одного из других. Важным свойством множества Парето D_p является возможность выбрать из множества альтернатив D заведомо неудачные, уступающие другим по всем критериям. Обычно решение многокритериальной задачи должно начинаться с выделения множества D_p.

Выбор окончательного решения из множества Парето-оптимальных точек может выполняться экспертом или на основании различных подходов, одним из которых является свертка критериев.

В [26–28] под сверткой критериев понимают любую числовую функцию критериев. Наиболее часто используемой сверткой критериев является линейная свертка, которая заключается в назначении тем или иным способом неотрицательных (а чаще положительных) коэффициентов λ_j, j = 1, … m на множестве X, в сумме дающих единицу (хотя это не обязательно), и последующей максимизации линейной комбинации критериев.

Еще одним видом свертки является свертка на основе идеальной точки. При построении этой свертки могут быть использованы различные метрики. Наиболее часто используются чебышевская метрика, взвешенная чебышевская метрика и метрика на основании Евклидова расстояния.

Перед выполнением свертки необходимо решить задачу «разнородности» критериев. Для этого нужно выполнить нормализацию критериев. В ходе нормализации разнотипные критерии приводятся к единой шкале.

Для выбора окончательного решения из множества Паретовских точек может использоваться метод главного критерия. В этом случае среди всех частных (локальных) критериев выбирается наиболее важный, а остальные переводятся в класс ограничений. Так

многокритериальная задача преобразуется в однокритериальную, которая и решается.

3.3. Разработка приложения с графическим интерфейсом

Цель работы: разработка графического интерфейса в визуальной среде GUIDE пакета Matlab решения задач многокритериальной оптимизации.

3.3.1. Конструирование интерфейса с помощью GUIDE

Для запуска среды GUIDE можно в командной строке задать команду guide, после чего появится окно GUIDE Quick Start, с помощью которого можно либо начать создание нового приложения, либо открыть существующее (рис. 3.1) [29; 30].

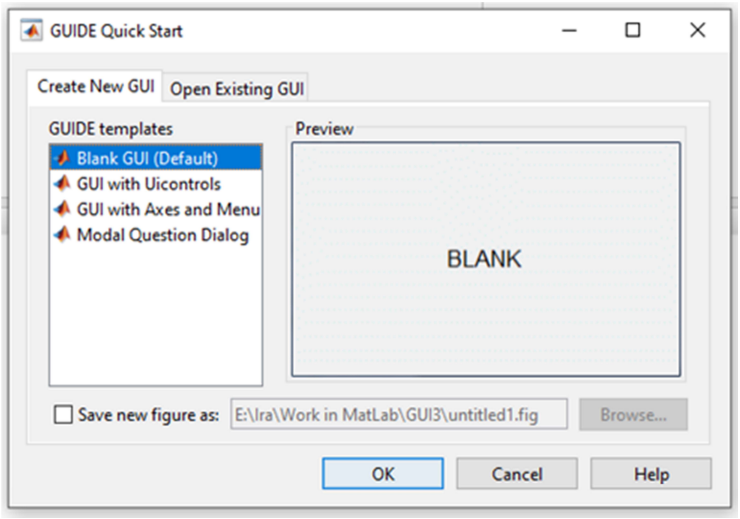


Рис. 3.1. Окно быстрого старта для среды GUIDE

Выберем пустое окно без элементов управления Blank GUI (Default). Появляется редактор среды GUIDE, содержащий основные панели и заготовку приложения. Разместим на нем различные

элементы интерфейса, которые понадобятся для реализации и визуализации результатов решения многокритериальной задачи. Для ввода количества параметров и целевых функций понадобятся два ползуна и два поля для отображения текста (рис. 3.2).

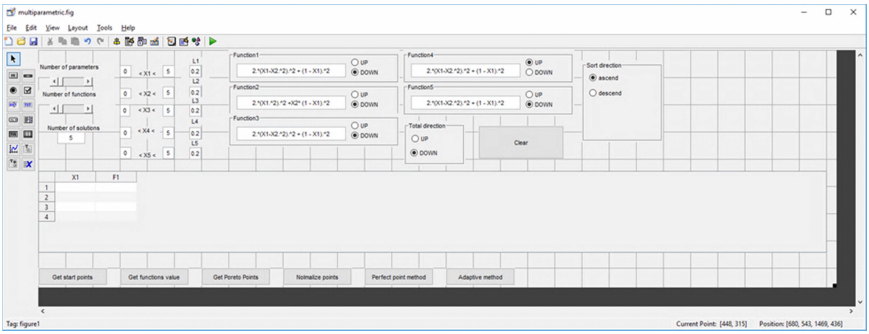


Рис. 3.2. Пример дизайна окна приложения с элементами управления

Далее нужно настроить элементы управления, т. е. задать стандартные значения и определить ключевые слова доступа к ним. Для этого двойным щелчком мыши по любому из объектов можно вызвать инспектор свойств Property Inspector. В свойстве *String* по умолчанию указывается стандартное значение элемента управления. В свойстве *Tag* – строка доступа к элементу управления.

Для ранее добавленных элементов в свойстве *Tag* надо указать: для элементов управления вводом параметров – *textNumberOfParameters, sliderNumberOfParameters*, а для элементов управления вводом функций – *textNumberOfFunctions* и *sliderNumberOfFunctions*. Стандартные значения в свойстве *String* могут быть указаны по желанию разработчика.

Далее добавляются следующие поля:

– поля для ввода минимальных значений параметров (в нашем примере их будет 5), номер поля соответствует номеру параметра. Для каждого из этих полей в свойстве *String* устанавливается значение 0 (минимальное значение по умолчанию), а в свойстве *Tag* – строка доступа к данному полю, согласно шаблону edixMinXn, где *n* – номер параметра (рис. 3.3);

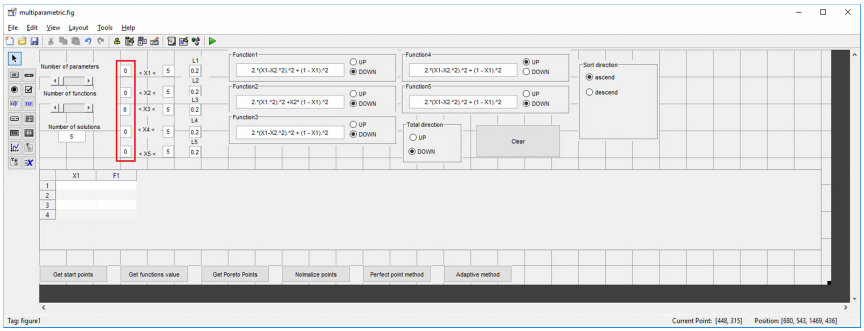


Рис. 3.3. Поля ввода минимальных значений параметров

– поля для ввода максимальных значений параметров (в нашем примере их будет 5), номер поля соответствует номеру параметра. Для каждого из этих полей в свойстве *String* устанавливается значение 0 (минимальное значение по умолчанию), а в свойстве *Tag* – строка доступа к данному полю, согласно шаблону edixMaxXn, где *n* – номер параметра (рис. 3.2). Обозначение полей ввода минимального и максимального значений для каждого параметра выполняется с помощью текстового поля. Для этого в окне инспектора каждого текстового поля в свойстве *String* устанавливаются значения < *Xn* < , а в свойство *Tag* значения *textXn*, где *n* – номер параметра;

– элементы группировки (button group) (в нашем примере их будет 5). В свойствах *Title* для каждого элемента устанавливаются стандартные значения *Functionm*, а в свойствах *Tag* – строки доступа по шаблону *uipanelFunctionm*, где *m* – количество функций (рис. 3.2). Далее в каждый элемент группировки добавляется поле ввода для целевой функции. В окне инспектора для поля ввода в свойстве *Tag* устанавливается значение *editFm*, где *m* – номер функции. Значение в свойстве *String* опциально. Также в каждый элемент группировки добавляются две радиокнопки. Для свойств *String* радиокнопок устанавливается значение UP для одной и DOWN – для другой;

– поле для ввода количества испытаний. В свойстве *String* устанавливается значение 50 (значение по умолчанию), а в свойстве *Tag* значение *editNumberOfSolutions*. А для текстового поля количество испытаний в свойстве *String* – значение *NumberOfSolutions* (рис. 3.2);

– задачи оптимизации могут быть как задачами на минимум, так и задачами на максимум. Конкретный вид задачи можно указать, задав вид обобщенного критерия. Для этого предусмотрен элемент группировки (button group) с двумя радиокнопками. В окне инспектора элемента группировки в свойстве *Title* устанавливается значение *Generalized criterion*, а в свойстве *Tag* – *uipanelGeneralizedCriterion* (рис. 3.2). В свойствах *String* радиокнопок устанавливаются значения UP для одной и DOWN – для другой;

– элемент группировки (button group) с двумя радиокнопками для задания вида сортировки по любому столбцу. В окне инспектора элемента группировки в свойстве *Title* устанавливается значение *Sort direction*, а в свойстве *Tag* – *uipanelSortDirection* (рис. 3.2). В свойствах *String* радиокнопок – значения ASCEND для одной и DESCEND – для другой;

– таблица данных (рис. 3.2). В свойстве *Tag* устанавливается значение table. Таблица данных содержит информацию обо всех этапах принятия решений.

На следующем шаге требуется добавить кнопки для управления данными на форме. Для того, чтобы подписать кнопку на событие, необходимо изменить ее свойство *String*. Также требуется настроить сроки доступа к кнопкам, изменяя значения свойства *Tag*.

В данном случаем выполнены следующие настройки:

– **String** – Get start points, **Tag** – pushbuttonGetStartPoints;
– **String** – Get functions value, **Tag** – pushbuttonGetFunctionsValue;
– **String** – Get Poreto points, **Tag** – pushbuttonGetPoretopoints;
– **String** – Normalize points, **Tag** – pushbuttonNormalizePoints;
– **String** – Perfect point method, **Tag** – pushbuttonPerfectPointMethod;
– **String** – Adaptive method, **Tag** – pushbuttonAdaptiveMethod;
– **String** – Clear, **Tag** – pushbuttonClea.

Все выполненные действия сохраняются, и получается форма, аналогичная представленной на рис. 3.2.

3.3.2. Программирование элементов интерфейса

После сохранения формы с элементами управления в дереве проекта появится файл с тем же именем, которое указывалось при сохранении формы, но с расширением *.m. Это файл с кодом формы.

В данном файле происходит инициализация формы, описываются все функции создания элементов управления и их callback-функции.

Далее требуется создать функции-помощники, которые будут помогать работать с элементами управления. Для этого в файле с кодом формы создается функция *generateColumnName* (рис. 3.4).

```
function generateColunmName(handles, count, type)
allNames = get(handles.table, 'columnname');
columnFormat = {};
newNames = {};
if type == 'X'
    [newNames, columnFormat] =  addNewNames(newNames, type, count, columnFormat);
    [newNames, columnFormat] = addOldNames(newNames,allNames, type, columnFormat);
else
    [newNames, columnFormat] = addOldNames(newNames,allNames, type, columnFormat);
    [newNames, columnFormat] =  addNewNames(newNames, type, count, columnFormat);
end
set(handles.table, 'columnname', newNames, 'columnformat', columnFormat);
```

Рис. 3.4. Фрагмент кода функции *generateColumnName*

Данная функция будет генерировать имена для столбцов параметров и критериев. Принимает 3 параметра: *handles* – это объект формы, в котором находятся все элементы управления; *count* – количество столбцов, для которых требуется задать имена; *type* – тип имени, к которому будет применяться функция. Функция *generateColumnName()* в своей реализации использует вспомогательные функции *addNewNames()* и *addOldName()*. Вспомогательная функция *addNewName()* (рис. 3.5) принимает три параметра: *newNames* – массив новых имен, *type* – тип для генерации нового имени, *count* – количество новых имен – и возвращает массив с добавленными именами.

```
function newNames = addNewNames(newNames, type, count)
for i=1:count
    newNames{end+1} = strcat(type, num2str(i));
end
```

Рис. 3.5. Фрагмент кода функции *addNewName*

Вспомогательная функция *addOldName()* (рис. 3.6) принимает три параметра: *newNames* – массив новых имен; *allNames* – массив всех имен, которые уже есть; *type* – для проверки нового имени в уже существующих – и возвращает массив с добавленными именами.

```
function newNames = addOldNames(newNames, allNames, type)
count  = numel(allNames);
for i=1:count
    bool = isempty(strfind(allNames{i}, type));
    if bool == 1
        newNames{end+1}  = allNames{i};
    end
end
```

Рис. 3.6. Фрагмент кода функции *addOldName*

Следующая функция-помощник – *prepareEdits()* (рис. 3.7). В своей реализации функция отключает все поля ввода и включает поля, которые передаются аргументами. В функции *prepareEdits()* используется стандартная функция для объединения строк – *strcat()*. После этого все обращения к полю объекта выполняются через эту объединенную строку. Также функция *prepareEdits()* использует стандартную функцию для преобразования типа *number* в тип *string* – *num2str()*.

```
function prepareEdits(handles, count, type)
for i=1:5
    s = strcat('edit', type, num2str(i));
    set(handles.(s), 'Enable', 'off');
    if(i <= count)
    set(handles.(s), 'Enable', 'on');
    end
end
```

Рис. 3.7. Фрагмент кода функции *prepareEdits*

Далее разрабатывается функция *getDataFromEdit()* (рис. 3.8). Она возвращает данные, взятые из полей ввода. При этом количество полей, из которых требуется взять значения и тип полей, передается аргументами.

```
function dataArray = getDataFromEdit(handles, count, type)
dataArray = {};
for i=1:count
    dataArray{end+1} =  get(handles.(strcat(type, num2str(i))), 'String');
end
```

Рис. 3.8. Фрагмент кода функции *getDataFromEdit*

Следующая функция-помощник – *getDataFromButtonGroups()* (рис. 3.9). Функция возвращает массив значений активных радиокнопок из нескольких элементов группировки. Доступ к элементам группировки происходит с помощью объединения типа и номера элемента.

```
650   function dataArray = getDataFromButtonGroups(handles, count, type)
651 -   dataArray = {};
652 -   for i=1:count
653 -       dataArray{end+1} = get(get(handles.(strcat(type,num2str(i))), 'SelectedObject'), 'String');
654 -   end
```

Рис. 3.9. Фрагмент кода функции *getDataFromButtonGroups*

Еще одна функция-помощник *addColumnNames()* принимает массив новых имен для столбцов, добавляет их в таблицу и возвращает текущее количество столбцов (рис. 3.10).

```
638   function columnCount = addColumnNames(handles, arrayOfNames)
639 -   columnNames = get(handles.table, 'columnname');
640 -   columnCount = numel(columnNames);
641 -   for i=1:numel(arrayOfNames)
642 -       columnNames{end+1} = arrayOfNames{i};
643 -   end
644 -   set(handles.table, 'columnname', columnNames);
```

Рис. 3.10. Фрагмент кода функции *addColumnNames*

Функция-помощник *getDataFromColumn()* (рис. 3.11) принимает столбец матрицы и извлекает все значения в массив. Если значение в столбце не найдено, то на соответствующее место в новом массиве устанавливается значение, переданное вторым аргументом в функцию.

```
658   function array = getDataFromColumn(column, emptyValue)
659 -   array = [];
660 -   for i=1:numel(column)
661 -       value = cell2mat(column(i));
662 -       if isempty(value)
663 -           array = [array emptyValue];
664 -       end
665 -       array = [array value];
666 -   end
```

Рис. 3.11. Фрагмент кода функции *getDataFromColumn*

Функции-помощники *getIndexOfMin()* и *getIndexOfMax()* (рис. 3.12) принимают массив значений и возвращают индекс минимального или максимального элемента соответственно.

```
668     function indexOfMin = getIndexOfMin(column)
669 -       [value, index] = min(column);
670 -       indexOfMin = index;
671
672     function indexOfMax = getIndexOfMax(column)
673 -       [value, index] = max(column);
674 -       indexOfMax = index;
```

Рис. 3.12. Фрагмент кода функций *getIndexOfMin* и *getIndexOfMax*

Функция *crearData()* (рис. 3.13) разработана для очистки всех прежних данные и подготовки формы для работы с новыми данными.

Она устанавливает пустые значения в таблицу, генерирует имена для столбцов, задает всем кнопкам значение Disable.

```
581     function clearData(handles)
582 -       set(handles.table, 'Data', [[];[]]);
583 -       set(handles.table, 'columnname', []);
584 -       [numberOfSolutions, numberOfParameters, numberOfFunctions] = getDefaultData(handles);
585 -       prepareEdits(handles, numberOfParameters, 'MaxX');
586 -       prepareEdits(handles, numberOfParameters, 'MinX');
587 -       prepareEdits(handles, numberOfFunctions, 'F');
588 -       prepareEdits(handles, numberOfParameters, 'L');
589 -       generateColunmName(handles, numberOfParameters, 'X');
590 -       generateColunmName(handles,numberOfFunctions , 'F');
591 -       set(handles.pushbuttonGetFunctionsValue, 'Enable', 'off');
592 -       set(handles.pushbuttonGetPoretoPoints, 'Enable', 'off');
593 -       set(handles.pushbuttonNormalizePoints, 'Enable', 'off');
594 -       set(handles.pushbuttonPerfectPointMethod, 'Enable', 'off');
595 -       set(handles.pushbuttonAdaptiveMethod, 'Enable', 'off');
```

Рис. 3.13. Фрагмент кода функции *clearData*

Функция *clearData()* включает все вышеописанные функции *prepareEdits()* и *generateColumnName()*. Также функция *clearData()* использует функцию *getDefaultData()* (рис. 3.14). Эта функция возвращает 4 параметра: количество решений, количество функций, количество параметров и вид обобщенного критерия.

```
536     function [numberOfSolutions, numberOfParameters, numberOfFunctions, totalDirection] = getDefaultData(handles)
537 -       numberOfFunctions = get(handles.sliderNumberOfFunctions, 'Value');
538 -       numberOfParameters = get(handles.sliderNumberOfParameters, 'Value');
539 -       numberOfSolutions = str2num(get(handles.editNumberOfSolutions, 'String'));
540 -       totalDirection = get(get(handles.uipanelTotalDirection, 'SelectedObject'), 'String');
```

Рис. 3.14. Фрагмент кода функции *getDefaultData*

После разработки функции *clearData()* ее необходимо применить и проверить. Для этого в методе *multiparametric_OpeningFcn* необходимо вызвать функцию *clearData()* (рис. 3.15).

```
22    function multiparametric_OpeningFcn(hObject, eventdata, handles, varargin)
23 -  handles.output = hObject;
24 -  guidata(hObject, handles);
25 -  set(handles.textValueOfParameters,'String',get(handles.sliderNumberOfParameters, 'Value'));
26 -  set(handles.textValueOfFunctions,'String',get(handles.sliderNumberOfFunctions, 'Value'));
27 -  clearData(handles);
```

Рис. 3.15. Фрагмент кода функции *multiparametric_OpeningFnc*

Также требуется, чтобы при изменении входных данных вся таблица данных очищалась. Для этого в callback-функциях ползунов (slider) необходимо вызвать метод *clearData()*. Добавим данную функцию в *callback sliderNumberOfParameters_Callback* и *sliderNumberOfFunctions_CreateFcn* (рис. 3.16).

```
40    function sliderNumberOfParameters_Callback(hObject, eventdata, handles)
41 -  set(handles.textValueOfParameters, 'String', num2str(get(hObject, 'Value')));
42 -  clearData(handles);
51    function sliderNumberOfFunctions_Callback(hObject, eventdata, handles)
52 -  set(handles.textValueOfFunctions, 'String', num2str(get(hObject, 'Value')));
53 -  clearData(handles);
```

Рис. 3.16. Фрагмент кода функции *sliderNumberOfParameters_Callback* и *sliderNumberOfFunctions_Callback*

Если попробовать запустить программу, то в командном окне появится сообщение об ошибке. Это происходит потому, что при инициализации ползунов (slider) не были указаны минимальное, максимальное значения и шаг. Для указания этих значений в файле с кодом требуется найти функции *sliderNumberOfParameters_CreateFcn* и *sliderNumberOfFunctions_CreateFcn* и добавить в них следующую строку:

set(hObject, 'Max', 5, 'Min', 1, 'Value', 1, 'SliderStep', [1/4 1/4]);

Данная строка означает, что минимальное значение будет равно 1, максимальное – 5, стандартное значение и шаг будут равны 1. Теперь можно проверить работу программы. Если запустить ее, то можно убедиться, что ранее добавленный функционал работает.

Для генерации начальных условий требуется найти callback-функцию, которая называется *pushbuttonGetStartPoints_Callback*. Функция обрабатывает клик по кнопке Get Start Point. В своей реализации с помощью функции-помощника сначала получаются количество решений и количество параметров, а потом вызывается метод *getStartPoints()* (рис. 3.17), который требуется разработать. Данный метод будет возвращать двумерный массив данных. В функции *getStartPoints()* с помощью одной из функций-помощников получаются массивы ограничений для параметров. Далее с помощью стандартной функции *randi* генерируются случайные значения параметров. Параметры объединяются в двумерный массив, количество строк которого равно числу решений, а столбцов – числу параметров. Для удобства доступа к данным в двумерном массиве значения параметров перед записью преобразуются в тип cell. Для объединения двух массивов используется синтаксис [*a*, *b*], где *a* – первый массив, *b* – второй массив. Для добавления новой строчки в двумерный массив используется синтаксис [*a*; *b*], где *a* – предыдущий двумерный массив, *b* – новая строчка двумерного массива.

```
283     function tableData = getStartPoints(numberOfSolutions, numberOfParameters, handles)
284 -     tableData = [];
285 -   for i=1:numberOfSolutions
286 -       newRow = [];
287 -       arrayOfMin = getDataFromEdit(handles, numberOfParameters, 'editMinX');
288 -       arrayOfMax = getDataFromEdit(handles, numberOfParameters, 'editMaxX');
289 -       for j=1:numberOfParameters
290 -           newRow = [newRow, (randi([str2num(arrayOfMin{j}), str2num(arrayOfMax{j})]))];
291 -       end
292 -       tableData = [tableData; newRow];
293 -   end
```

Рис. 3.17. Фрагмент кода функции *getStartPoints*

Для вычисления значений целевых функций для начальных условий используется callback-функция кнопки Get Functions Value. В callback-функции вызывается метод *getFunctionsValue()*, который необходимо разработать. Этот метод возвращает матрицу значений целевых функций (рис. 3.18).

В данной функции с помощью функции-помощника в качестве начальных данных используются количество решений, количество параметров и количество функций. Далее для каждой строки матрицы начальных условий рассчитываются все целевые функции,

полученные при помощи функции-помощника *getFunction()* (рис. 3.19). Метод *getFunction()* также возвращает количество параметров, которые принимает целевая функция.

```
function pushbuttonGetFunctionsValue_Callback(hObject, eventdata, handles)
tableData = get(handles.table, 'Data');
tableData = getFunctionsValue(tableData, handles);
set(handles.table, 'Data', tableData);
set(handles.pushbuttonGetPoretoPoints, 'Enable', 'on');

function tableData = getFunctionsValue(tableData, handles)
[numberOfSolutions, numberOfParameters, numberOfFunctions] = getDefaultData(handles);
for i=1:numberOfSolutions
    for j=1:numberOfFunctions
        [funct, countOfInputParameters] = getFunction(handles, j);
        if(numberOfParameters < countOfInputParameters)
            msgbox('Error! Count of parameters should not be less than count of input parameters.');
            return;
        end
        arrayOfParameters = cell2mat(tableData(i, 1:numberOfParameters));
        value = getValueOfFunction(countOfInputParameters, funct, arrayOfParameters);
        tableData(i,numberOfParameters + j) = {value};
    end
end
```

Рис. 3.18. Фрагмент кода функции *getFunctionsValue*

```
function [funct, countOfInputParameters] = getFunction(handles, which)
funstr=get(handles.(strcat('editF', num2str(which))), 'String');
funct=inline(funstr);
countOfInputParameters = nargin(funct);
```

Рис. 3.19. Фрагмент кода функции *getFunction*

Значение целевой функции рассчитывается с помощью метода *getValueOfFunction()* (рис. 3.20). В каждую целевую функцию подставляется требуемое количество начальных условий.

```
function value = getValueOfFunction(numberOfImputParameters, funct, arrayOfParameters)
switch numberOfImputParameters
    case 1
        value = funct(arrayOfParameters(1));
    case 2
        value = funct(arrayOfParameters(1),arrayOfParameters(2));
    case 3
        value = funct(arrayOfParameters(1),arrayOfParameters(2),arrayOfParameters(3));
    case 4
        value = funct(arrayOfParameters(1),arrayOfParameters(2),arrayOfParameters(3),arrayOfParameters(4));
    case 5
        value = funct(arrayOfParameters(1),arrayOfParameters(2),arrayOfParameters(3),arrayOfParameters(4),arrayOfParameters(5));
end
```

Рис. 3.20. Фрагмент кода функции *getValueOfFunctions*

Если начальных условий не хватает, то программа показывает диалоговое окно с советующей ошибкой.

Для нахождения области компромиссов уже подготовлена кнопка Get Poreto Points. При клике на эту кнопку будет вызываться

функция *getPoretoPoints()* (рис. 3.21). В самом начале функции *getPoretoPoints()* добавляются новые столбцы и получаются стандартные параметры с помощью функций-помощников *addColumnNames()* и *getDefaultData()* соответственно.

```
function pushbuttonGetPoretoPoints_Callback(hObject, eventdata, handles)
tableData = get(handles.table, 'Data');
tableData = getPoretoPoints(tableData, handles);
set(handles.table, 'Data', tableData);
set(handles.pushbuttonNormalizePoints, 'Enable', 'on');

function tableData = getPoretoPoints(tableData, handles)
rowCount = addColumnNames(handles, {'Poreto Points', 'Power'});
[numberOfSolutions, numberOfParameters, numberOfFunctions] = getDefaultData(handles);
tableData = setDefaultRange(tableData, numberOfSolutions, rowCount);
directionOfFunctions = getDataFromButtonGroups(handles, numberOfFunctions, 'uipanelFunction');
for i=1:numberOfSolutions
    for j=i+1:numberOfSolutions
        rowOne = cell2mat(tableData(i, numberOfParameters + 1 : rowCount));
        rowTwo = cell2mat((tableData(j, numberOfParameters + 1 : rowCount)));
        string = compareRow(rowOne, rowTwo, numberOfFunctions, directionOfFunctions);
        tableData = setPoretoPoints(tableData, rowCount, string, i, j);
    end
end
indexInColumn = getIndexOfMax(cell2mat(tableData(:, numberOfParameters + numberOfFunctions + 2)));
point = num2str(cell2mat(tableData(indexInColumn, numberOfParameters + numberOfFunctions + 2)));
tableData(indexInColumn, numberOfParameters + numberOfFunctions + 2) = {markBest(point)};
```

Рис. 3.21. Фрагмент кода функции *getPoretoPoints*

Далее необходимо вызвать метод *sefDefaultRange()* (рис. 3.22), который заполняет добавленные колонки начальными значениями. В дальнейшем это пригодится для проверки области компромиссов и согласия.

```
function tableData = setDefaultRange(tableData, numberOfSolutions, rowCount)
for i=1:numberOfSolutions
    tableData(i, rowCount + 1) = {'K'};
    tableData(i, rowCount + 2) = {'0'};
end
```

Рис. 3.22. Фрагмент кода функции *setDefaultRange*

В методе *getPoretoPoints()* выполняется сравнение строк матрицы значений целевых функций с помощью функции *compareRow()* (рис. 3.23), в которую одним из параметров передается массив направлений целевых функций, полученный функцией-помощником *getDataFromButtonGroups()*.

В функции *compareRow()* для сравнения двух элементов на одной позиции в строке вызывается метод *compareTwoValue()* (рис. 3.24),

который сравнивает два значения по направлению в массиве направлений. Метод возвращает булевское значение или пустую строку, если два значения равны. После сравнения всех значения в двух строках на выходе получается массив значений.

```
function string = compareRow(rowOne, rowTwo, numberOfFunctions, directionOfFunctions)
compareArray = {};
for i=1:numberOfFunctions
    compareArray{end+1} = compareTwoValue(rowOne(i),rowTwo(i), directionOfFunctions{i});
end
if checkOnBool(compareArray, true)
   string = 'isBetter';
elseif checkOnBool(compareArray, false)
    string = 'isWorse';
else
    string = 'areEqual';
end
```

Рис. 3.23. Фрагмент кода функции *compareRow*

```
function bool = compareTwoValue(a, b, type)
if strcmp(type, 'UP')
    if a > b
        bool = true;
    elseif a < b
        bool = false;
    elseif a == b
        bool = '';
    end
else strcmp(type, 'DOWN')
   if a < b
        bool = true;
    elseif a > b
        bool = false;
    elseif a == b
        bool = '';
    end
end
```

Рис. 3.24. Фрагмент кода функции *compareTwoValue*

Если все значения в массиве равны true, значит первая строчка лучше по всем критериям и входит в область компромиссов, а вторая входит в область согласия. Если все значения являются false, то наоборот. В остальных случаях обе точки входят в область компромиссов. Это происходит с помощью функции *checkOnBool()* и *setPoretoPoints()* (рис. 3.25).

```
function bool = checkOnBool(array, type)
bool = true;
for i=1:numel(array)
    if array{i} == ''
        continue;
    elseif array{i} ~= type;
        bool = false;
        return;
    end
end

function tableData = setPoretoPoints(tableData, rowCount, string, i, j)
if strcmp(string,'isBetter')
    if ~strcmp(tableData(i, rowCount + 1), 'C')
        tableData(i, rowCount + 1) = {'K'};
    end
    value = str2num(char(tableData(i, rowCount + 2))) + 1;
    tableData(i, rowCount + 2) = {num2str(value)};
    tableData(j, rowCount + 1) = {'C'} ;
elseif strcmp(string,'isWorse')
    if ~strcmp(tableData(i, rowCount + 1), 'C')
        tableData(j, rowCount + 1) = {'K'};
    end
    tableData(i, rowCount + 1) = {'C'} ;
    value = str2num(char(tableData(j, rowCount + 2))) + 1;
    tableData(j, rowCount + 2) = {num2str(value)};
end
```

Рис. 3.25. Фрагмент кода функции *checkOnBool* и *setPoretoPoints*

Для нормализации данных необходимо вызвать функции *sortPoints()* и *normalizePoints()*. Перед каждой из них требуется добавить новые столбцы в таблицу данных с помощью функции-помощника *addColumnNames()* (рис. 3.26).

```
function pushbuttonNormalizePoints_Callback(hObject, eventdata, handles)
tableData = get(handles.table, 'Data');
[numberOfSolutions, numberOfParameters, numberOfFunctions] = getDefaultData(handles);
[tableData, columnNames] = sortPoints(tableData, numberOfSolutions, numberOfFunctions, numberOfParameters);
addColumnNames(handles, columnNames);
[tableData, columnNames] = normalizePoints(tableData, numberOfSolutions, numberOfFunctions, numberOfParameters);
addColumnNames(handles, columnNames);
set(handles.table, 'Data', tableData);
set(handles.pushbuttonPerfectPointMethod, 'Enable', 'on');
set(handles.pushbuttonAdaptiveMethod, 'Enable', 'on');
```

Рис. 3.26. Фрагмент кода callback-функции кнопки *Normalize Points*

Функция *normalizePoints()* (рис. 3.27) нормализирует результаты, полученные ранее. Для этого от каждого текущего значения отни-

мается минимальное и делится на разницу максимального и минимального значений.

```
function [tableData, columnNames]  = normalizePoints(tableData, numberOfSolutions, numberOfFunctions, numberOfParameters)
arrayOfMin = {};
arrayOfDivider = {};
columnNames = {};
for p=1:numberOfFunctions
    column = tableData( : , numberOfParameters + numberOfFunctions + 2 + p);
    arrayOfMin{end+1} = min(column);
    arrayOfDivider{end+1} = max(column) - min(column);
    columnNames{end+1} = strcat('KN-', num2str(p));
end
for i=1:numberOfSolutions
    for j=1:numberOfFunctions
            value = (tableData(i, numberOfParameters + numberOfFunctions + 2 + j) - arrayOfMin(j)) / arrayOfDivider(j);
            tableData(i, numberOfParameters + numberOfFunctions*2 + 2 + j) = value;
    end
end
```

Рис. 3.27. Фрагмент кода функции *normalizePoints*

Для реализации линейной свертки необходимо извлечь коэффициенты из полей ввода. Также сразу рассчитывается их сумма и, если она не равна единице, то программа генерирует диалоговое окно с соответствующей ошибкой. Далее с помощью функции *getAdaptiveMethodValue()* определяется обобщенный критерий для каждой точки (рис. 3.28).

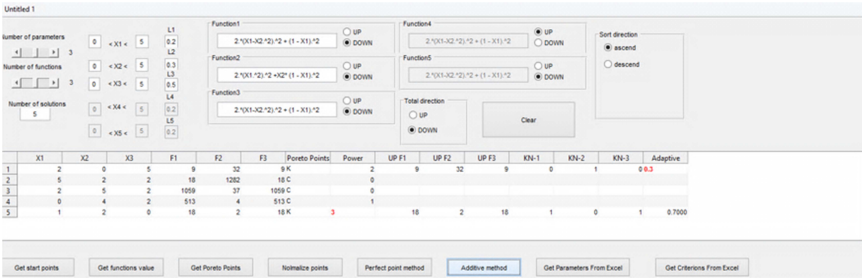
```
function pushbuttonAdaptiveMethod_Callback(hObject, eventdata, handles)
tableData = get(handles.table, 'Data');
rowCount = addColumnNames(handles, {'Adaptive'});
[numberOfSolutions, numberOfParameters, numberOfFunctions] = getDefaultData(handles);
arrayOfLambda = [];
sum = 0;
for j=1:numberOfFunctions
    currLambda = str2num(get(handles.(strcat('editL', num2str(j))), 'String'));
    arrayOfLambda = [arrayOfLambda, currLambda];
    sum = sum + currLambda;
end
if sum ~= 1
    msgbox('sum of lambds should be equal 1');
    return;
end
for i=1:numberOfSolutions
    currPoint = tableData(i, numberOfParameters + numberOfFunctions*2 + 3 : numberOfParameters + numberOfFunctions*3 + 2);
    tableData(i, rowCount + 1) =  getAdaptiveMethodValue(currPoint, numberOfFunctions, arrayOfLambda);
end
set(handles.table, 'Data', tableData);

function value = getAdaptiveMethodValue(currPoint, numberOfFunctions, arrayOfLambda)
value = 0;
for i=1:numberOfFunctions
    value = value + currPoint(i) * arrayOfLambda(i);
end
```

Рис. 3.28. Фрагмент кода функции для линейной свертки

В зависимости от направления обобщенного критерия определяется наилучшее решение.

Результат работы приложения приведен на рис. 3.29. Как видно из таблицы, наименьшее значение обобщенного критерия у точки под номером 1.



	X1	X2	X3	F1	F2	F3	Poreto Points	Power	UP F1	UP F2	UP F3	KN-1	KN-2	KN-3	Adaptive
1	2	0	5	9	32	9 K		2	9	32	9	0	1	0	0.3
2	5	2	2	18	1282	18 C		0							
3	2	5	2	1059	37	1059 C		0							
4	0	4	2	513	4	513 C		1							
5	1	2	0	18	2	18 K	3		18	2	18	1	0	1	0.7000

Рис. 3.29. Результат работы приложения

Список использованной литературы

1. Построение конечно-элементной модели на основе языка APDL : учебно-методическое пособие / В. В. Напрасников [и др.]. – Минск : БНТУ, 2009. – 51 с.

2. Создание конечно-элементной модели для расчета контейнера в процессе прессования порошковой заготовки : лабораторный практикум / В. В. Напрасников [и др.]. – Минск : БНТУ, 2008. – 89 с.

3. Напрасников, В. В. Конечно-элементное моделирование в ANSYS в режиме удаленного доступа к суперкомпьютеру «СКИФ» : учебно-методическое пособие / В. В. Напрасников, А. В. Бородуля, В. А. Кочуров. – Минск : БНТУ, 2008. – 65 с.

4. Моделирование колебаний рамной конструкции на основе метода конечных элементов : учебно-методическое пособие / В. В. Напрасников [и др.]. – Минск : БНТУ, 2010. – 43 с.

5. Наседкин, А. В. Конечно-элементное моделирование на основе ANSYS. Программы решения статических задач сопротивления материалов с вариантами индивидуальных заданий / А. В. Наседкин. – Ростов-на-Дону : УПЛ РГУ, 1998. – 44 с.

6. Конюхов, А. В. Основы анализа конструкций в ANSYS / А. В. Конюхов. – Казань, 2001. – 102 с.

7. Чигарев, А. В. ANSYS для инженеров : справочное пособие / А. В. Чигарев, А. С. Кравчук, А. Ф. Смалюк. – М. : Машиностроение-1, 2004. – 512 с.

8. Басов, К. А. ANSYS : справочник пользователя / К. А. Басов. – М. : ДМК Пресс, 2005. – 640 с.

9. Басов, К. А. Графический интерфейс комплекса ANSYS / К. А. Басов. – М. : ДМК Пресс, 2006. – 25 с.

10. Справочная система ANSYS HELP.

11. ANSYS. Basic Analysis Procedures Guide. Rel. 5.3. / ANSYS Inc. Houston. – 1994.

12. ANSYS. Commands Reference. Rel. 5.3. / ANSYS Inc. Houston. – 1994.

13. ANSYS. Elements Reference. Rel. 5.3. / ANSYS Inc. Houston. – 1994.

14. ANSYS. Theory Reference. Rel. 5.3. Ed. P. Kothnke / ANSYS Inc. Houston. – 1994.

15. ANSYS. Verification Manual. Rel. 5.3. / ANSYS Inc. Houston. – 1994.

16. ANSYS 5.7 Theory Reference / ANSYS Inc. – 2001.

17. ANSYS 5.7 Advanced Analysis Techniques Guide / ANSYS Inc. – 2001.

18. Сергейкин, О. А. Обзор оптимизационных возможностей программы ANSYS [Электронный ресурс] / О. А. Сергейкин. – Режим доступа: http://sergeykin.hotbox.ru

19. Большаков, В. Твердотельное моделирование деталей в CAD-системах: AutoCAD, КОМПАС-3D, SolidWorks, Inventor, Creo: 3D-модели и конструкторская документация сборок / В. Большаков, А. Бочков и Ю. Лячек. – СПб. : Питер, 2015. – 473 с.

20. Большаков, В. П. Основы 3D-моделирования. Изучаем работу в AutoCAD, КОМПАС-3D, SolidWorks, Inventor : учебное пособие для студентов вузов по направлению 211000 «Конструирование и технологии электронных средств» / В. П. Большаков, А. Л. Бочков. – СПб. : Питер, 2013. – 300 с.

21. Тику, Шам SolidWorks 2005 / Шам Тику ; пер. А. Вахитов. – СПб. : Питер, 2006. – 815 с.

22. Пашкевич, М. Ф. Технологическая оснастка : учебник для студентов машиностроительных специальностей вузов / М. Ф. Пашкевич [и др.]. – Минск : Адукацiя i выхаванне, 2002. – 320 с.

23. Гарнаев, А. Ю. Visual Basic .NET: разработка приложений / А. Ю. Гарнаев. – СПб. : БХВ-Петербург, 2002. – 624 с: ил.

24. Соболь, И. М. Выбор оптимальных параметров в задачах со многими критериями / И. М. Соболь, Р. Б. Статников. – М. : Дрофа, 2006. – 175 с.

25. Маркина, М. В. Практикум по решению задач оптимизации в пакете Matlab : учебно-методическое пособие / М. В. Маркина, А. В. Судакова. – Нижний Новгород : Нижегородский госуниверситет, 2017. – 49 с

26. Лотов, А. В. Многокритериальные задачи принятия решений : учебное пособие / А. В. Лотов, И. И. Поспелова. – М.: МАКС Пресс, 2008. – 197 с.

27. Численный анализ и оптимизация / В. М. Волков [и др.]. – Минск : РУП «Белгослес», 2017. – 207 с.

28. Волков В. М. Численные методы для обыкновенных дифференциальных уравнений [Электронный ресурс] : учебно-методиче-

ское пособие для магистрантов специальности 1-31 81 12 «Прикладной компьютерный анализ данных» / В. М. Волков, И. Л. Ковалева. – Минск : БНТУ, 2016.

29. Официальная документация по Matlab [Электронный ресурс]. – Режим доступа: https://www.mathworks.com/help/matlab/.

30. Официальная форум по Matlab [Электронный ресурс]. – Режим доступа: https://www.mathworks.com/matlabcentral/answers/.

Учебное издание

КОВАЛЕВА Ирина Львовна
КУНКЕВИЧ Дмитрий Петрович
НАПРАСНИКОВ Владимир Владимирович и др.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОЕКТИРОВАНИИ ИНЖЕНЕРНЫХ КОНСТРУКЦИЙ

Пособие для студентов специальности
1-40 05 01 «Информационные системы и технологии
(по направлениям)» направления специальности 1-40 05 01-01
«Информационные системы и технологии (в проектировании
и производстве)»

Редактор *А. С. Козловская*
Компьютерная верстка *Н. А. Школьниковой*

Подписано в печать 28.08.2024. Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 4,82. Уч.-изд. л. 3,26. Тираж 100. Заказ 1060.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.