ЛАБОРАТОРНАЯ РАБОТА 10
КОМПЬЮТЕРНЫЕ МОДЕЛИ АНАЛИТИЧЕСКОЙ ГЕОМЕТРИИ
ГЕОМЕТРИЧЕСКИЙ ОБЪЕКТ «ТОЧКА НА ПЛОСКОСТИ»

Компьютерная математика
БГУ, ММФ, 1 курс, КМ
доц. Малевич А.Э.,
доц. Щеглова Н.Л.
ноябрь 2021

Постановка задачи

Требуется спроектировать и построить математическую систему, позволяющую решать задачи в области аналитической и компьютерной геометрии. Среда моделирования – символьный математический пакет *Mathematica*.

1. Проектирование геометрических объектов

Задание 1. Спроектировать в *Mathematica* объекты системы «аналитическая геометрия на плоскости» и функции, позволяющие работать с этими объектами: отображать, вычислять свойства, определять отношения, управлять поведением объектов.

Выполнение задания 1.

В первую очередь определимся, с какими геометрическими объектами мы будем работать при решении задач аналитической геометрии. Для этого обратимся к опыту изучения школьного курса евклидовой геометрии. Как известно, в аксиоматике Гильберта евклидовой геометрии математические понятия точки, прямой и плоскости являются первичными понятиями. На основе первичных понятий определяются луч, отрезок, ломаная линия, полигон. Эти геометрические понятия широко используются при решении задач практического содержания, например, задач компьютерной графики. Особую роль при решении геометрических задач играет понятие вектора.

Каждое из перечисленных выше геометрических понятий будем ассоциировать в компьютерной среде соответствующим одноименным объектом.

`Объект в объектно-ориентированном проектировании` – это сущность, характеризуемая состоянием (информация) и поведением (множество операций) для проверки и изменения этого состояния.

`Базовыми геометрическими объектами плоскости` будем называть объекты точка и прямая. Двумерные объекты отрезок,

многоугольник будем называть **элементарными геометрическими объектами плоскости**.

Вводя в компьютерную среду новый объект, мы должны научить *Mathematica* многому: во-первых, надо научиться **задавать** экземпляр данного объекта, во-вторых надо где-то **сохранять** информацию о данном экземпляре объекта, в-третьих, понадобится **извлекать свойства** объекта, в-четвертых, хотелось бы **иметь** некоторый **образ** объекта. Поэтому дальнейшее выполнение Задания 1 сводится к выполнению более локальных задач, которые мы сформулируем и решим в этом параграфе.

1.1 Внутреннее представление геометрического объекта

Задание 1.1. Сформулировать основные принципы представления геометрического объекта в *Mathematica*.

Выполнение задания 1.1.

Для хранения информации об экземпляре объекта будем использовать понятие внутреннего представления объекта.

Внутреннее представление объекта – это информация об объекте, размещенная в структуре данных, которая используется в моделируемой компьютерной среде.

Чтобы построить внутреннее представление объекта, или, другими словами, дать представление объекта в компьютерной среде, следует выявить информацию об объекте, выбрать структуру данных для ее хранения и разместить информацию об объекте в этой структуре. Внутреннее представление любого объекта далее будем называть коротко: представление объекта.

Понятно, что представление объекта проектируется неоднозначно. Представление объекта зависит от целей моделирования, особенностей компьютерной среды, точки зрения исследователя и т. п.

Рассмотрим задачу представления указанных ранее геометрических объектов в компьютерной среде *Mathematica*. При этом постараемся учесть идеологию символьных математических пакетов.

Известно, что основной структурой данных в *Mathematica* является выражение. Тогда представлением геометрического объекта будем называть выражение, которое имеет строго определенную голову и соответствующие этой голове аргументы

ObjectName[arg1, …, argN]. (1.1)

В отношении выражения (1.1) примем следующее соглашение: голова `ObjectName` представляемого объекта должна быть символом-атомом.

Символом в *Mathematica* называется атомарный (т. е. не составной, не распадающийся на части) поименованный объект, состоящий из

конечной последовательности алфавитно-цифровых знаков, начиная с буквы или $, не содержащей специальных символов [1].

Перечень специальных символов в *Mathematica* оговорен отдельно и включает в себя знаки пунктуации, знаки операций и т. п. Как правило, символами именуют встроенные функции, их аргументы, отвечающие за способ выполнения функции, а также функции, которые задает пользователь.

Кроме того, договоримся, что все вводимые нами символы вида `ObjectName` из (1.1) должны нести смысловую нагрузку и иметь приставку-префикс **`km`**.

Следуя этим соглашениям, для представления в *Mathematica* точки, вектора, прямой будем использовать голову-символ **`kmPoint`**, **`kmVector`**, **`kmLine`**, соответственно.

Спроектируем далее аргументы `arg1, …, argN` представления (1.1). Они должны представлять информацию об экземпляре объекта, необходимую и достаточную для его однозначной идентификации, удобную для вычисления его свойств и определения поведения.

Сначала выявим такую общую информацию, которая необходима для представления любого геометрического объекта: и для точки, и для прямой, и для вектора.

Представление объекта является технической информацией, ее использует разработчик. Внешнему пользователю, пользователю продукта, знать структуру, представляющую объект, необязательно и даже вредно.

Договоримся, что первым аргументом `arg1` в представлении (1.1) будет символ `km`. Для разработчиков символ `km`, будучи первым аргументом выражения с головой **`kmObjectName`**, будет указывать, что данное выражение является представлением, Базой Знаний об объекте, в отличие от выражений-операторов, конструкторов объекта.

Какую еще информацию удобно разместить в представлении объекта? При решении задач мы зачастую именуем точку, вектор, прямую. Поэтому договоримся, что вторым аргументом представления (1.1) будет имя объекта, если объект именован.

Имя логично указывать атомом типа `String`. Например, точку можно именовать `"A"`, прямую – `"L`$_1$`"`. Если объект не именуется, второй аргумент будет представлен выражением вида `""`.

```
"" // Head
String
```

Следующие аргументы `arg3, …, argN` в выражении (1.1) должны представлять базовые знания об экземпляре объекта, информацию для его однозначной идентификации, удобную для вычисления свойств объекта. Эти аргументы мы спроектируем позже, для каждого класса в отдельности.

На основании рассуждений, приведенных выше, детальнее укажем метамодель для представления объектов «точка», «вектор», «прямая». Представление (1.1) будет иметь вид

kmObjectName[km, id_String, __], (1.2)

где голова **kmObjectName** является одним из символов **kmPoint**, **kmVector**, **kmLine**, первый аргумент – маркер, указывающий, что данное выражение является представлением объекта, второй аргумент – имя экземпляра объекта, если оно существует, или пустая строка вида **""** в противном случае.

1.2 Конструкторы геометрического объекта

Задание 1.2. Спроектировать в *Mathematica* возможность задания геометрического объекта различными способами.

Выполнение задания 1.2.

Геометрический объект можно задавать различными способами. При задании объекта пользователь указывает данные, необходимые и достаточные для однозначного определения экземпляра объекта.

Эти данные должны быть обработаны, преобразованы в информацию об объекте, которая, в свою очередь, будет сохранена в соответствующей структуре – так сформируется представление объекта.

Наш замысел таков: пользователь задает экземпляр объекта любым корректным, оговоренным нами, общепринятым математическим способом. *Mathematica* же, получая эти данные, строит представление объекта. Чтобы построить представление объекта на основании указанных данных, создадим **функции-конструкторы объекта**.

Имена функций-конструкторов геометрического объекта мы унифицируем: имя конструктора должно указывать объект, представление которого эта функция строит.

Таким образом, для задания объекта мы будем использовать оператор **kmObjectName[_ _]**. Его проектируем так, чтобы была возможность задания геометрических объектов различными способами. Например, прямую через две различные точки, через точку и направляющий вектор, через точку и угол наклона к оси OX и т.п.

Оператор **kmObjectName[_ _]**, аргументы которого – данные для построения объекта, преобразовывает эти данные во внутреннее представление (1.2). Таким образом, для конкретного объекта его представление и способ задания мы связываем с одним единственным символом. Например, для объекта «точка» – с символом **kmPoint**.

Как же тогда различать представление объекта, тот или иной конструктор для построения представления объекта? Различать по виду аргументов, структуре подвыражений первого уровня. Это мы обсудим далее, на этапе проектирования каждого объекта.

Подводя итог рассуждениям, составим таблицу 1.1, разместив в ней ответы на вопросы, возникающие на начальном этапе проектирования.

Таблица 1.1

Задание и представление объектов

Что сделать?	Реализовать возможность задания объекта	Представить объект
Как сделать?	Создать функции-конструкторы объекта. Для каждого способа задания объекта создается отдельное правило.	Вызвать соответствующий конструктор объекта или записать выражение вида (1.2).

1.3 Свойства геометрического объекта

Задание 1.3. Спроектировать в *Mathematica* возможность извлечения свойств геометрического объекта.

Выполнение задания 1.3.

После того, как мы определились со структурой объектов «точка», «вектор», «прямая» на мета уровне, подумаем о том, какую информацию об объекте мы хотим получать в процессе создания и эксплуатации нашей системы, и как мы будем запрашивать эту информацию.

Какие сущности, позволяющие работать с объектом, мы уже спроектировали? Для каждого объекта мы описали его Представление, а также выявили операторы – Конструкторы объекта, позволяющие по указанным данным строить Представление объекта.

Далее мы спроектируем Запросы для получения свойств объекта.

Как получить свойство объекта? Мы должны это свойство запросить у *Mathematica*, *Mathematica* должна его вычислить и вернуть в качестве ответа на запрос.

Как запросить свойство объекта, что нужно указать? Указать объект, свойство которого запрашиваем, и указать, какое свойство мы запрашиваем.

Как указать объект? Использовать его представление (1.2). Как указать конкретное свойство? Придумать для каждого свойства соответствующий идентификатор.

Таким образом, чтобы сформировать запрос о свойстве объекта, будем использовать выражение вида

```
kmObjectName[km, id_String, __] ["property"],
```  (1.3)

где **"property"** – идентификатор запрашиваемого свойства объекта.

Такой порядок запроса свойства объекта согласуется с нашей концепцией «задать объект – представить объект»: если объект был задан Конструктором, то Конструктор, отработав, вычислил Представление объекта.

Итак, голову выражения (1.3), часть вида **`kmObjectName[km, id_String, _ _]`**, мы получаем одним из двух способов: или построив Представление непосредственно в виде (1.2), или используя один из Конструкторов объекта.

Спроектируем теперь идентификаторы для указания свойств объектов при запросах. Какие свойства наиболее интересны в качестве свойств объекта?

Определимся для каждого объекта отдельно. Каждый из объектов может быть или не быть именованным, и этим фактом мы заинтересуемся, например, при построении образа объекта. Объект «точка», объект «вектор» обязательно имеет координаты, они необходимы при решении различных задач. При исследовании объекта «прямая» удобно иметь ее общее уравнение.

На основании этих рассуждений мы введем следующие идентификаторы свойств объектов: **`"id"`** – для запроса имени объекта, **`"coord"`**– для запроса координат точки или вектора, **`"equ"`**– для запроса общего уравнения прямой на плоскости. Таким образом, запрос на свойства объектов мы будем осуществлять посредством выражений, представленных в таблице 1.2.

Таблица 1.2

Запрос о свойстве объекта

Свойство какого объекта запрашиваем? Какое свойство?	**Выражение, запрашивающее свойство**
Точка, имя	**`kmPoint[_ _] ["id"]`**
Точка, координаты	**`kmPoint[_ _] ["coord"]`**
Вектор, имя	**`kmVector[_ _] ["id"]`**
Вектор, координаты	**`kmVector[_ _] ["coord"]`**
Прямая, имя	**`kmLine[_ _] ["id"]`**
Прямая, общее уравнение	**`kmLine[_ _] ["equ"]`**

Далее, на следующем уровне проектирования, нам следует решить, в каком виде возвращать запрашиваемые свойства. Ответ на этот вопрос мы сформулируем в каждом конкретном случае.

1.4 Графический образ геометрического объекта

Задание 1.4. Спроектировать в *Mathematica* возможность отображения геометрического объекта.

Выполнение задания 1.4.

Помимо возможностей задания геометрического объекта в том или ином виде, его представления, извлечения его свойств, мы спроектируем возможность **отображения геометрического объекта** в *Mathematica*.

Выделим два способа отображения объекта: в протоколе и на рисунке.

Под **отображением объекта в протоколе** будем понимать печать информации об объекте в выходной `Out[]`-ячейке.

Обсудим возможность отображения геометрического объекта на рисунке. Для ее реализации мы научим *Mathematica* воспринимать конкретный геометрический объект как графический примитив двухмерной графики. А именно, мы создадим препроцессор **kmGraphics2D** к существующему в *Mathematica* оператору **Graphics**.

Главной задачей препроцессора будет преобразование всех введенных нами объектов аналитической геометрии на плоскости в графические примитивы двумерной графики, понятные самой *Mathematica.*

2. Геометрический объект «точка на плоскости»

Задание 2. Создать в *Mathematica* подсистему для работы с точкой на плоскости: спроектировать внутреннее представление объекта «точка на плоскости», реализовать способы задания объекта, возможности извлечения его свойств, построения графического образа.

Выполнение задания 2.

Выполнение задания 2 разобьем на этапы, на каждом этапе сформулируем подзадачу, оформив в ее виде *задания 2.xx*. При выполнении заданий будем придерживаться принципов проектирования, сформулированных в параграфе 1.

2.1 Представление и задание объекта «точка на плоскости»

Задание 2.1

Спроектируйте в *Mathematica* внутреннее представление объекта «точка на плоскости».

Выполнение задания 2.1.

Как следует из пункта 1.1, моделируя математическое понятие в компьютерной среде, мы на начальном этапе должны выполнить следующее: во-первых, поставить понятию в соответствие объект, во-вторых, определить информацию об объекте, в-третьих, хранить эту информацию (о заданном экземпляре объекта) в определенной структуре – таким образом мы сформируем представление объекта.

В данном задании речь идет о математическом понятии «точка», которому мы ставим в соответствие объект «точка на плоскости» в среде *Mathematica*. Введем на плоскости прямоугольную декартову систему координат.

Представлением объекта «точка на плоскости» в *Mathematica* будем называть выражение вида

`kmPoint[km, id_String, coord: { _, _ }]` (2.1)

где **`coord`** –координаты точки в прямоугольной декартовой системе координат на плоскости.

Задание 2.2

Спроектируйте в *Mathematica* конструктор объекта «точка на плоскости».

Выполнение задания 2.2.

Рассмотрим способы задания точки. Если на плоскости зафиксирована система координат, то точка однозначно определяется своими координатами. Зачастую рассматривается прямоугольная система координат, иногда удобно рассматривать связанную с ней полярную систему координат, когда полюс находится в точке *(0,0)*, а полярная ось совпадает с положительным направлением оси OX. Кроме того, точка может иметь имя или не быть именованной.

Следуя соглашениям, принятым в пункте 1.2, имя функции-конструктора объекта «точка на плоскости» должно совпадать с головой выражения (2.1), которое является внутренним представлением точки.

Так как точка может быть задана различными способами, то удобно, чтобы конструктор был представлен несколькими глобальными определениями. Правая часть каждого правила-определения будет представлять образец, описывающий конкретный способ задания точки. В левой части укажем выражение, которое сформирует представление (2.1) соответственно способу задания. Левые части глобальных определений для задания точки сведем в таблице 2.1.

Таблица 2.1

Способы задания объекта «точка»

	Без имени	Именованный
Декартовы координаты	`kmPoint[coord:{_,_}]`	`kmPoint[id_String,coord:{_,_}]`
Полярные координаты	`kmPoint[{ϕ_,ρ_},"pol"]`	`kmPoint[id_String,{ϕ_,ρ_},"pol"]`

Задание 2.3

Создайте конструктор объекта «точка на плоскости», требующий указания ее декартовых координат (прямоугольная декартова система координат).

Выполнение задания 2.3

Построим конструктор – оператор **`kmPoint`**, который по указанным декартовым координатам точки строит ее представление вида (2.1). Укажем в качестве единственного аргумента список вида

coord: {x, y}, содержащий координаты точки в прямоугольной декартовой системе координат

```
ClearAll[kmPoint];                                              (2.2)
kmPoint[coord : {_, _}] := kmPoint[km, "", coord];

kmPoint[{1, 2}]
kmPoint[km, , {1, 2}]
```

На первый взгляд, правило (2.2) не имеет никакого практического смысла или значения. Но не будем торопиться!

Задание 2.4

Создайте возможность задания объекта «точка» путем указания ее имени и декартовых координат (прямоугольная декартова система координат).

Выполнение задания 2.4

Определим конструктор, позволяющий задавать точку указанием ее имени и координат.

```
kmPoint[id_String, coord : {_, _}] :=                           (2.3)
  kmPoint[km, id, coord];

kmPoint["A", {1, 2}]
kmPoint[km, A, {1, 2}]
```

Примечательно, что конструктор, конечно же, допускает любое имя точки – так мы его построили

```
myP = kmPoint["моя Точка", {1, 0}]
kmPoint[km, моя Точка, {1, 0}]
```

А если мы имеем координаты точки не в декартовой, а в другой, например, в полярной системе координат? Тогда нам следует добавить еще одно правило для функции-конструктора.

Задание 2.5

Создайте возможность задания объекта «точка на плоскости» посредством указания ее полярных координат (согласованных с прямоугольной декартовой системой координат).

Выполнение задания 2.5

Введем прямоугольную декартову систему координат, связанную с используемой полярной системой координат. Начало координат разместим в полюсе, положительное направление оси *Ox* направим вдоль направления полярной оси. Тогда, следуя формулам перехода от полярной системы координат к декартовой, запишем следующий оператор **kmPoint**, строящий представление (2.1)

```
kmPoint[{ϕ_, ρ_}, "pol"] :=
  kmPoint[km, "", {ρ Cos[ϕ], ρ Sin[ϕ]}];
```

```
kmPoint[{π/3, 2}, "pol"]
kmPoint[km, , {1, √3}]
```

Задание 2.6

Создайте возможность задания объекта «точка» посредством указания ее полярных координат (согласованных с прямоугольной декартовой системой координат) и, возможно, имени точки.

Задание 2.6 выполните самостоятельно.

Таким образом, мы спроектировали представление объекта «точка на плоскости» и обучили *Mathematica* преобразовывать указанные данные об объекте «точка на плоскости» во внутреннее представление этого объекта. При этом точка может быть задана различными способами: или в прямоугольной декартовой системе координат, или в связанной с ней полярной системе координат, быть именованной или существовать без имени.

2.2 Свойства геометрического объекта «точка на плоскости»

Точка, фундаментальное понятие в геометрии, обладает малым количеством свойств. Чтобы указать точку на плоскости, можно, например, нарисовать ее на листке бумаги и написать рядом имя точки. Можно затем работать с точкой, используя это имя. Если же ввести систему координат, то можно использовать еще одно замечательное свойство точки: свойство располагаться в определенном месте. Следовательно, имя точки, координаты точки – та информация, которая отличает ее от других точек. Пожалуй, мы выявили основные свойства точки, на начальном этапе проектирования объекта «точка на плоскости».

Задание 2.7

Напишите оператор, возвращающий имя объекта «точка», если оно существует.

Выполнение задания 2.7

Имя точки является свойством объекта «точка на плоскости», поэтому при выполнении задания мы будем следовать концепции, изложенной в пункте 1.3. В случае, когда точка не имеет имени, оператор будет возвращать нарицательное имя Точка, представленное атомарным объектом **"Точка"**. Признак отсутствия имени точки указан в представлении объекта: в этом случае второй аргумент представления (2.1) имеет вид "".

```
kmPoint[km, id_String, ___]["id"] :=
  If[id == "", "Точка", id];
```

Проверим работу построенной функции

```
kmPoint["A", {1, 2}]["id"]
A
```

Узнаем имя точки, которую мы ассоциировали с символом **myP** при выполнении задания 2.3

```
myP["id"]
моя Точка
```

Запросим имя неименованной точки

```
kmPoint[{5, 2}]["id"]
Точка
kmPoint[{π/3, 2}, "polar"]["id"]
Точка
```

Задание 2.8

Напишите оператор, возвращающий декартовы координаты заданного объекта «точка на плоскости».

Выполнение задания 2.8

Здесь затруднения могут быть лишь при построении образца, находящегося в левой части правила-определения. Следуя соглашениям, описанным в пунктах 1.1-1.3, получаем

```
kmPoint[km, _String, coord_List, ___]["coord"] :=
  coord;
```

```
kmPoint[{π/3, 2}, "polar"]["coord"]
{1, √3}
myP["coord"]
{1, 0}
```

Тестируйте вычисление координат объекта «точка» при всех возможных способах задания точки.

Задание 2.9

Постройте булеву функцию, позволяющую определить, имеет ли представленный объект «точка на плоскости» имя.

Выполнение задания 2.9

Булева функция **NamedQ** возвращает значение **True** в случае, если объект «точка» является именованным объектом, и значение **False** в противном случае.

```
NamedQ[kmPoint[km, id_String, ___]] ^:= id ≠ "";
```

Следует отметить, что при определении данного правила мы использовали оператор **UpSetDelayed**, который ассоциировал правило с единственным аргументом функции **NamedQ**, точнее, с головой этого аргумента, с символом **kmPoint**. Проверим, какие правила на данном этапе проектирования ассоциированы с символом **kmPoint**.

```
?kmPoint
```

```
Global`kmPoint
kmPoint[km, id_String, ___][id] := If[id == , Точка, id]

kmPoint[km, _String, coord_List, ___][coord] := coord

NamedQ[kmPoint[km, id_String, ___]] ^:= id ≠
kmPoint[coord : {_, _}] := kmPoint[km, , coord]

kmPoint[id_String, coord : {_, _}] := kmPoint[km, id, coord]

kmPoint[{ϕ_, ρ_}, polar] := kmPoint[km, , {ρ Cos[ϕ], ρ Sin[ϕ]}]
```

Представляет интерес следующий вопрос: какие списки для символа **kmPoint** построила *Mathematica* и как распределились введенные

глобальные правила по этим спискам? Ответьте на этот вопрос, используя функции **DownValues**, **UpValues**, **SubValues**.

Далее следует тестировать построенную функцию **NamedQ**.

```
NamedQ[myP]
True
NamedQ[kmPoint[{π/3, 2}, "polar"]]
False
```

2.3 Отображение точки на рисунке

Чтобы нарисовать картинку, на которой будет отмечена интересующая нас точка, воспользуемся графическими возможностями системы. А именно, научим *Mathematica* воспринимать объект «точка на плоскости» как графический примитив двумерной графики.

Задание 2.10

Три точки `P1`, `P2`, `P3`, заданные в прямоугольной декартовой системе координат соответственно координатами `(−1, 0)`, `(5, 3)`, `(−2, 2)`. Представьте точки `P1`, `P2`, `P3` в *Mathematica*, каждую как экземпляр объекта «точка на плоскости», и отобразите представленные точки в графической области.

Выполнение задания 2.10

Представим заданные точки в *Mathematica*. Для этого используем конструктор **kmPoint** вида (2.2) и ассоциируем каждую из точек с символами `P1`, `P2`, `P3` соответственно

```
{P1, P2, P3} = kmPoint /@ {{-1, 0}, {5, 3}, {-2, 2}}
{Точка[-1, 0], Точка[5, 3], Точка[-2, 2]}
```

Полюбопытствуем, как выглядит полученный список внутренних представлений заданных точек

```
{P1, P2, P3} // TreeForm
```

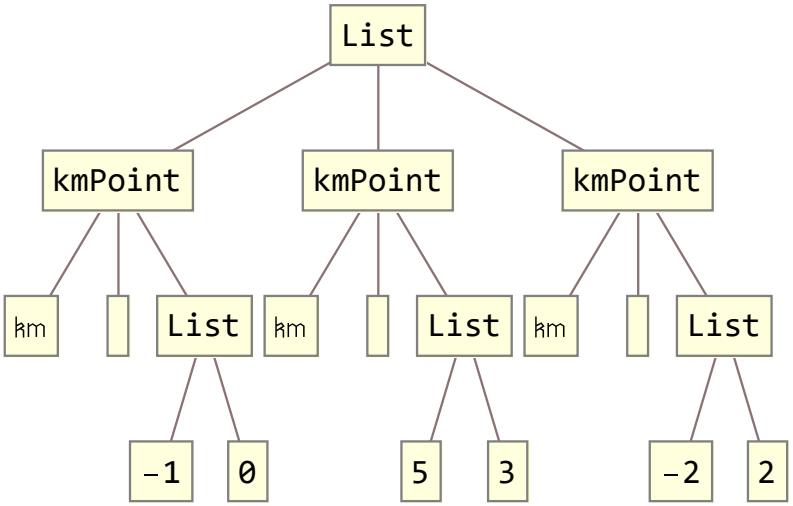


Рис. 2.1 Структура представлений экземпляров объекта «точка»

Обратите внимание, что каждая из представленных точек не имеет имени, атомарный объект `Pi, i = 1,…,3` является лишь символом, с которым ассоциируется представление в *Mathematica* `i`-той точки.

Далее, чтобы построить графический образ точки, используем графический примитив: укажем координаты точки в качестве аргумента функции **`Point`**.

Извлечем координаты из представления каждой точки, формируя запрос **`"coord"`** о свойствах объекта «точка на плоскости»

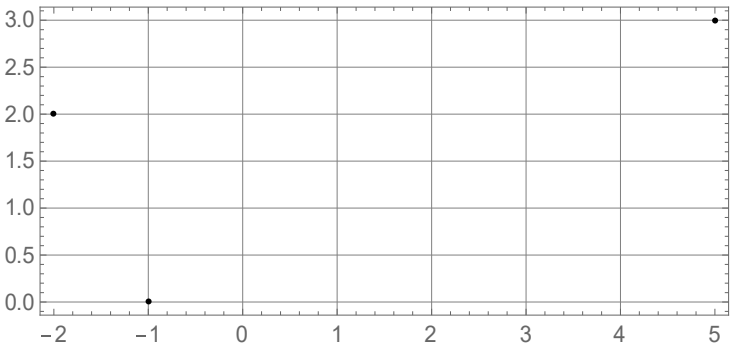
```
#["coord"] & /@ {P1, P2, P3}
{{-1, 0}, {5, 3}, {-2, 2}}
```

Теперь на координаты каждой точки можно подействовать графическим примитивом **`Point`**

```
Point /@ (#["coord"] & /@ {P1, P2, P3})
{Point[{-1, 0}], Point[{5, 3}], Point[{-2, 2}]}
```

Далее изобразим построенные графические объекты, применяя функцию **`Graphics`** и переустанавливая некоторые ее режимы выполнения.

```
Graphics[Point /@ (#["coord"] & /@ {P1, P2, P3}),
 Frame → True, GridLines → Automatic]
```



Оформим построенное выражение в виде глобального правила преобразования

```
ClearAll[kmGraphics2D];
kmGraphics2D[gp_List, opts___] :=
  Graphics[Point /@ (#["coord"] & /@ gp), opts];
```

Вызывая функцию в виде
`kmGraphics2D[{P1, P2, P3}, Frame → True, GridLines → Automatic]`,
убеждаемся, что она возвращает тот же результат, что и в пошаговом случае построения рисунка.

Литература

1. Голубева Л.Л., Малевич А.Э., Щеглова Н.Л. Компьютерная математика. Символьный пакет *Mathematica*. Лаб. практикум в 2 ч. Ч 1. – Минск: БГУ, 2012. – 235 с.