# Евклидова геометрия на языке векторной графики ASYMPTOTE

Оригинальная французская версия
*Филиппа Ивальди*
английский перевод
*Оливье Гуибе*
и
*Филиппа Ивальди*

Перевод с английского
Ю.Г. Крячкова

Волгоград
2015

# Euclidean geometry with ASYMPTOTE

Original French Version by
*Philippe Ivaldi*

English Translation by
*Olivier Guibé*
end
*Philippe Ivaldi*

Compiled with asymptote version 2.14svn-r5318
on June 28, 2011

## От автора русского перевода

Язык векторной графики `Asymptote` и интерпретатор для него первоначально задумывался для создания `PostScript`-графики и использования ее в LaTeX документах, однако в итоге получилось, что с его помощью можно создавать рисунки и других графических форматов, а также использовать независимо от LaTeX. Он создан А. Хаммерлиндлом (*Andy Hammerlindl*), Д. Боуменом (*John Bowman*) и Т. Принсом (*Tom Prince*) на основе программы `MetaPost` Дж. Хобби (*John D. Hobby*), которая, в свою очередь, базировалась на программе METAFONT Дональда Кнута (*D.E. Knuth*).

`Asymptote` — это язык программирования высокого уровня, он ориентирован на владеющих математикой, так как многие его функции требуют определенной математической подготовки. Для его освоения не будет лишним некоторый опыт программирования.

`Asymptote` является по сути интерфейсом высокого уровня к языку `PostScript` и претендует стать стандартом для верстки математических фигур так же, как TeX(LaTeX) фактически является стандартом для верстки сложных формул и уравнений.

Использование LaTeX для создания текстовых меток, формул и уравнений обеспечит качество рисунков. По умолчанию `Asymptote` производит выходные файлы формата `eps`. Язык располагает многими дополнительными возможностями, например имеются функции, которые могут создавать новые функции;

Главным сайтом `Asymptote` является сайт:

http://asymptote.sourceforge.net

Интерпретатор `Asymptote` создавался по модульному принципу. Пользователи могут сами создавать `Asymptote` модули, приспосабливая их к своим нуждам. Уже написаны около сорока модулей различного назначения.

Данный перевод сделан, во-первых, для изучения возможностей модуля `geometry.asy`, а во-вторых, для пользователей, желающих научиться иллюстрировать свои тексты хорошей, качественной графикой, но не владеющих в достаточной мере иностранными языками. Ссылки на французский оригинал и его английский перевод находятся также на указанном выше сайте.

Вёрстка перевода сделана с использованием дистрибутива **Ubuntu 14.04 LTS** и **TeX Live 2013**. Коды к некоторым рисункам отличаются от кодов к рисункам в английском переводе и французском оригинале, но эти отличия не принципиальны. Вдумчивый читатель должен понимать необходимость проверки кодов, приведенных к рисункам.

В некоторых рисунках изменены размеры. Однако в целом мы старались сохранять композицию рисунков в тексте и авторский стиль изложения.

Перевод может содержать неточности и даже ошибки. Ответственность за это целиком лежит на переводчике. Заранее благодарен за сообщения об ошибках и замечания, которые могут уточнить перевод. Глубоко признателен П.Б.Ждановичу за поддержку, консультативную помощь и редактирование сложных частей текста перевода.

Ю.Г. Крячков

## Благодарности от автора (Филиппа Ивальди)

Я хотел бы поблагодарить

- Оливье Guibé за многочисленные дискуссии по некоторым вопросам компьютерной алгебры, его одобрение и внимательное выслушивание;
- John Bowman и Andy Hammerlindl, без которых не существовало бы `Asymptote`;
- MB (форум http://forum.mathematex.net/membre3.html), на котором следят за развитием этого расширения, благодаря которому этот модуль исправляют, улучшают и добавляют некоторые особенности.

Филипп Ивальди

**Резюме**

В этом документе рассматривается использование пакета `geometry.asy`, который поможет создавать рисунки на плоскости. В пакете `geometry.asy` определяются некоторые новые типы и процедуры для программного обеспечения `Asymptote`.

Сначала мы приведем список новых типов с их кратким описанием. Затем мы изучим каждый из них и предоставим подробную информацию о связанных с ними процедурах и операторах.

# Содержание

# 1 Введение

## 1.1 Список типов объектов

Пакет *geometry.asy* определяет многие объекты, которые обычно используются в плоской евклидовой геометрии. Подобно тому, как для работы с действительными числами и путями в `Asymptote` используют типы `real` и `path`, пакет *geometry.asy* включает множество структур и функций для работы с объектами евклидовой геометрии. В дальнейшем важно различать объект и его тип. Например, объект «Bivariate Quadratic Equation» является объектом типа `bqe` и содержит объект `a` типа `real[]`; доступ к нему можно получить через код `an_objet_bqe.a`.

Приведем все типы объектов, предоставляемые пакетом *geometry.asy*:

1. **coordsys** (декартова) система координат; этот тип определен в разделе *Системы координат*, а в разделе *Точки и векторы* описано его использование;

2. **point** и **vector** (точка и вектор) рассматриваются в декартовой системе координат. Чтение раздела *Точки и векторы*, в котором используются эти типы, можно опустить, если читатель не хочет использовать другую, нежели по умолчанию, систему координат: в этом случае можно считать, что типы `point` и `vector` те же, что и тип `pair`;

3. **mass** (материальная точка) в декартовой системе координат (см. раздел *Материальные точки*);

4. **line** и **segment** описывают прямую, полупрямую (луч) или сегмент. Тип `segment` является производным от типа `line` и присутствует, чтобы облегчить чтение кода. Эти типы описаны в разделе *Прямые, лучи и сегменты*;

5. **conic** коники любого вида (см. раздел *Коники*).

   Для оптимизации[1] и чтения кода выделяются типы `circle`, `ellipse`, `parabola`, `hyperbola`, `bqe` («Bivariate Quadratic Equation»). Поэтому в соответствии с вашими целями можно выбрать более частные виды коник; конечно, частные виды коник можно привести к типу коники общего вида, как показано в следующем примере:

   ```
   ellipse a_circle=circle ((0,0), 3);
   conic a_conic=a_circle;
   ```

6. **arc** дуга эллипса (см. раздел *Дуги)*;

7. **abscissa** абсцисса на прямой (в общем смысле) или на конике (см. раздел *Абсциссы*);

8. **triangle** треугольник (см. раздел *Треугольники*).

   Код `a_triangle.object` позволяет получить доступ к объектам, которые связаны с треугольником. Эти объекты имеют типы

   - **side** стороны треугольника (см. подраздел *Стороны*);
   - **vertex** вершины треугольника (см. подраздел *Вершины*).

9. **trilinear** трилинейные координаты по отношению к треугольнику (см. подраздел *Трилинейные координаты*).

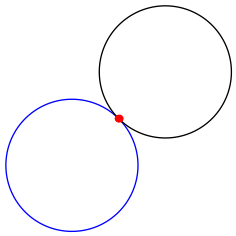
## 1.2 Внутренние вычисления

Вычисления с заданными объектами любого из типов, определенных в пакете *geometry.asy*, выполняются по отношению к сущности объекта, а не по отношению его графическому представлению, которое, в итоге есть `path` или `pair`.

Действительно, следующий код, который вычисляет и рисует пересечение двух касающихся окружностей, работает в пять раз быстрее, чем эквивалентная подпрограмма, которая использует тип `path` вместо типа `circle`.

[1] метод вычисления касательной к окружности отличается в случае гиперболы

```
import geometry;//код к рисунку справа
size(3cm,0);
point O=(0,0);
point O1=(sqrt(2),sqrt(2));
circle cle1=circle(O, 1.0);
circle cle2=circle(O1, 1.0);
draw(cle1,blue); draw(cle2);
dot(intersectionpoints(cle1, cle2),red);
```



### 1.3 Указатель и внешние примеры

Указатель и галерея примеров всех процедур, типов и операторов, определенных в пакете *geometry.asy*, позволяют детально рассмотреть возможности этого модуля:

- Указатель по названию функции: http://www.piprime.fr/files/asymptote/geometry/modules/geometry.asy.index.type.html;
- Указатель по типу функции, там же;
- Галерея примеров, там же.

### 1.4 Приведение типов

Упомянутые выше объекты могут быть обработаны собственными программами (теми, что определены в пакете *geometry.asy*) или основными процедурами ASYMPTOTE благодаря приведению типов. Например, окружность, которая имеет тип `circle`, может быть создана непосредственно стандартной процедурой `draw` благодаря автоматическому приведению типа `circle` к типу `path`. Однако при выполнении кода `dot(a_circle);` будет возвращено сообщения об ошибке, поскольку процедуре `dot` требуется именно тип `path`, поэтому необходимо осуществить явное приведение типа командой `dot((path)a_circle);`

## 2 Системы координат

Если вы не планируете использовать иную, чем по умолчанию, систему координат, этот раздел можно прочитать позже. В этом случае достаточно знать, что пакет *geometry.asy* использует тип `point` вместо типа `pair` и что типы `vector` и `pair` эквивалентны.

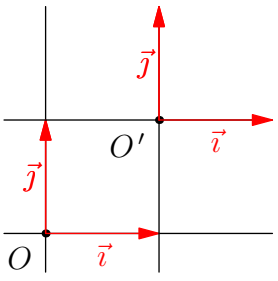
В следующих пунктах мы имеем дело с основными процедурами над декартовой системой координат, а реальное использование системы координат подробно описано в разделе *Точки и векторы*.

### 2.1 Тип coordsys

Пакет *geometry.asy* позволяет определить объекты в любой произвольной декартовой системе координат на плоскости; системы координат имеют тип `coordsys`. Как показано ниже:

- Системой координат по умолчанию является `defaultcoordsys`, она в первую очередь используется в ASYMPTOTE;
- Текущая система координат `currentcoordsys` по умолчанию принимает значение `defaultcoordsys`.

```
import geometry;
unitsize(1.5cm);
show(defaultcoordsys);
show("$O^{\,\prime}$", shift((1,1))*currentcoordsys);
```

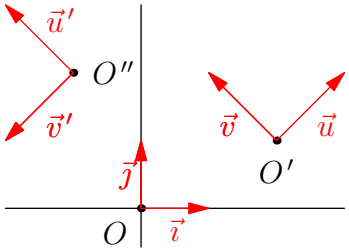


## 2.2 Как определить систему координат

Для определения системы координат можно использовать команду `cartesiansystem` или применить преобразование некоторой системы координат.

Например:

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem(O=(2,1),i=(1,1),j=(-1,1));
show("$O'$","$\vec{u}$","$\vec{v}$",R,xpen=invisible);
show("$O''$","$\vec{u}^{\,\prime}$",
"$\vec{v}^{\,\prime}$", rotate(90)*R, xpen=invisible);
show(defaultcoordsys);
```



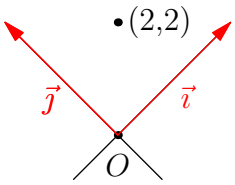

## 2.3 Изменение объекта pair в системе координат

**Примеры этого раздела приведены для информации**. Лучшим методом для определения, изменения, преобразования координат в системе координат является использование типа `point` (см. раздел *Точки и векторы*). Есть два основных способа определения или преобразования координат типа `pair` в системе координат с помощью операторов:

```
pair operator *(coordsys R, pair m)
```

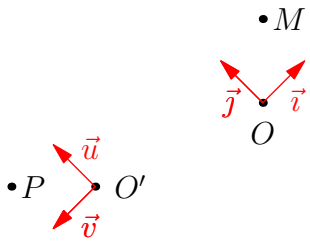
преобразует координаты `m`, приведенные в системе координат `R`, в координаты в системе по умолчанию. Таким образом, в следующем примере точка `M` имеет координаты `(0,5; 0,5)` в `R` и `(2; 2)` в `defaultcoordsys`:

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,1), i=(1,1), j=(-1,1));
pair M=R*(0.5,0.5);
dot("", M);
show(R);
```




```
pair operator /(pair m, coordsys R)
```

преобразует координаты `m`, данные в системе координат по умолчанию, в координаты в системе `R`. Таким образом, в следующем примере точки `M` и `P` имеют одинаковые координаты в репере `R` и, соответственно, в репере `Rp`:




```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,1),i=(1,1),j=(-1,1));
coordsys Rp=cartesiansystem((-2,-1),i=(-1,1), j=(-1,-1));
pair M=R*(1,1); dot("$M$", M);
pair P=Rp*(M/R); dot("$P$", P);
show(R, xpen=invisible);
show("$O'$","$\vec{u}$","$\vec{v}$",Rp,xpen=invisible);
```

## 2.4 Другие процедуры

- `path operator *(coordsys R, path g)`

  Код вида `coordsys*path` возвращает новый путь, который реконструируется из пути `g` в координатной системе `R`. Это обобщает конструкцию `coordsys*pair` для путей.

- `coordsys operator *(transform t, coordsys R)`

Позволяет выполнять код `transform*coordsys`. Отметим, что `shiftless(t)` применяется к `R.i` и `R.j`.

# 3 Точки и векторы

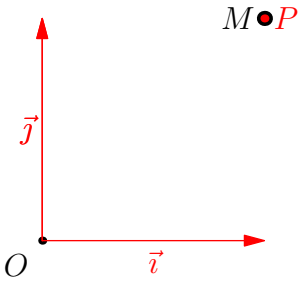
## 3.1 Точки

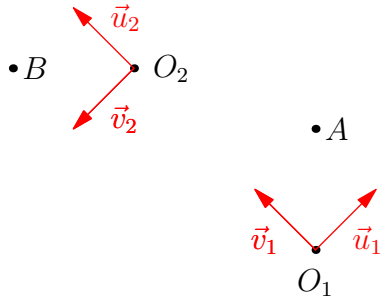
### 3.1.1 Основные принципы

В то время как тип `pair` позволяет задать точку лишь в системе координат по умолчанию, тип `point` позволяет определять точку в любой прямоугольной системе координат (см. тип coordsys); объект типа `point` всегда содержит ссылку на ту систему координат, в которой определена точка.

Благодаря приведению типов, употребление `point` в целом эквивалентно `pair`, если использовать только систему координат по умолчанию. Так в следующем примере `pair M` и `point P` отмечают одну и ту же точку:

```
import geometry;
size(4cm,0);
show(currentcoordsys,xpen=invisible);
pair M=(1,1);
dot("$M$", M, W, linewidth(2mm));
point P=(1,1);
dot("$P$", P, red);
```



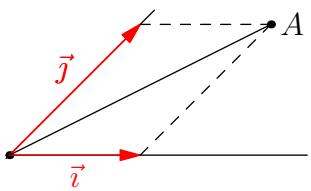




Следующий пример (рисунок слева) показывает влияние изменения текущей системы координат на приведение(casting) типа `pair` к типу `point` для точки `A` и как определить точку в конкретной системе координат процедурой `point point(coordsys R, pair m)` для точки `B`. Далее следует код к рисунку:

```
import geometry;
size(5cm,0);
currentcoordsys=cartesiansystem((3,0), i=(1,1), j=(-1,1));
show("$O_1$", "$\vec{u_1}$", "$\vec{v_1}$", currentcoordsys, xpen=invisible);
point A=(1,1);
dot("$A$", A);
coordsys Rp=rotate(90)*currentcoordsys;
show("$O_2$", "$\vec{u_2}$", "$\vec{v_2}$", Rp, xpen=invisible);
point B=point(Rp, (1,1));
dot("$B$", B);
```

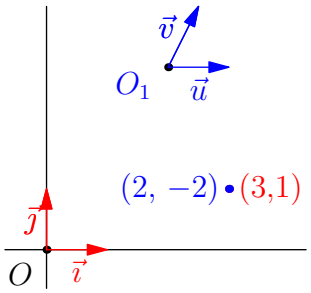
Процедура `point locate(pair m);` позволяет напрямую преобразовать тип `pair` в тип `point` без объявления point:

```
import geometry;
size(5cm,0);
currentcoordsys=cartesiansystem((3,0),
(1,0), (1,1));
show("", currentcoordsys);
point A=(1,1);
dot("$A$", A); draw(locate(0)--A);
draw(locate((0,1))--A, dashed);
draw(locate((1,0))--A, dashed);
```

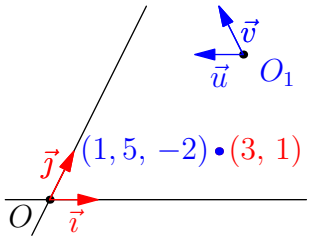



В приведенном ниже примере показано, как преобразовать тип `pair` в тип `point` так, что результат представляет точку и то, как получить её координаты в двух различных системах координат.

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,3),i=(1,0),
j=(0.5,1));
show(currentcoordsys);
show(Label("$O_1$",blue),Label("$\vec{u}$",
blue),Label("$\vec{v}$",blue), R,
xpen=invisible, ipen=blue);
pair A=(3,1); dot("", A, red);
point B=point(R, A/R);dot("", B, W, blue);
```




Процедура `changecoordsys` позволяет легко менять систему координат для данной точки:



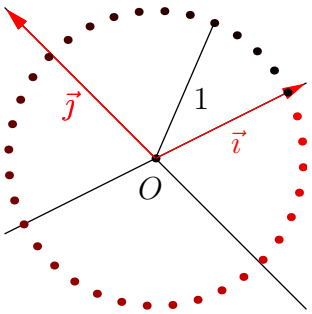

```
import geometry;
size(5cm,0);
currentcoordsys=cartesiansystem((0,0), i=(1,0), j=(0.5,1));
show(currentcoordsys);
coordsys R=cartesiansystem((4,3), i=(-1,0), j=(-0.5,1));
show(Label("$O_1$",blue), Label("$\vec{u}$",align=S,blue),
Label("$\vec{v}$",align=E,blue), R, xpen=invisible, ipen=blue);
point A=(3,1);
dot("", A, red);
point B=changecoordsys(R, A);
dot("", B, W, blue);
```

Как и для типа `pair`, операторы `+, -, *, /` можно использовать для типа `point`.

Заметим, что такая операция над двумя точками, определеными в двух различных системах координат, дает точку, определенную в системе координат по умолчанию `defaultcoordsys;` пользователю выдается предупреждение об этом автоматическом преобразовании.

Указать точку ее полярными координатами можно с помощью кода `pair polar(real r, real angle)` в объекте типа `coordsys`, как показано в следующем примере:

```
import geometry; size(4cm,0);
coordsys R=cartesiansystem(O=(1,2),
                  i=(1,0.5), j=(-1,1));
show(R);
for (int i=0; i < 360; i += 10) {
pen p=(i/360)*red;
dot(point(R, R.polar(1,radians(i))), p);}
point A=point(R, R.polar(1,radians(40)));
draw((string)abs(A), R.O--A);
```




### 3.1.2 Другие процедуры

Теперь, когда определены основные процедуры, касающиеся типа `point`, рассмотрим другие процедуры:

- `point origin(coordsys R=currentcoordsys)`

  Возвращает начальную точку системы координат `R` как точку типа point. Постоянная `point origin` является начальной точкой репера по умолчанию.

- `point point(coordsys R, explicit point M, real m = M.m)`

  Возвращает материальную точку массы `m`, относительные координаты которой в `R` равны значению `M`. Не следует путать эту процедуру с `changecoordsys`.

- `pair coordinates(point M)`

  Возвращает координаты `M` по отношению к своей системе координат.

- `bool samecoordsys (bool warn=true...point[] M)`

  Возвращает значение `true` тогда и только тогда, когда все точки определены в одной и той же системе координат. Если значение `warn=true` и если системы координат различны, то выдается предупреждение.

- `point[] standardizecoordsys (coordsys R = currentcoordsys,`
  `bool warn=true...point[] M)`

  Обобщение процедуры `changecoordsys` на массив точек. Возвращает массив точек в новой системе координат `R`. Если значение `warn=true` и если исходные системы координат точек, входящих в массив, различны, то выдается предупреждение.

- `pair[] operator cast(point[] P)`

  Преобразование массива `point[]` в массив `pair[]`.

- `pair locate(point P)`

  Возвращает координаты `P` в системе координат по умолчанию.

- `point operator *(transform t, explicit point P)`

  Определяет `transform*point`.

  Отметим, что преобразования `scale, xscale`, `yscale` и `rotate` определены в координатной системе по умолчанию, которая, вообще говоря, нежелательна, когда текущая система координат была изменена. Чтобы компенсировать неудобство, можно использовать процедуры `scale(real,point)`, `xscale(real,point)`, `yscale(real,point), rotate(real,point)`, `scale0(real), xscale0(real)`, `yscale0(real)` и `rotate0(real)`, которые описаны в разделе *Преобразования (Часть 1)*.

- `point operator *(explicit point P1, explicit pair p2)`

  Определяет `point*pair`

  Мы считаем, что значением `p2` являются координаты точки `p2` в системе координат, где определяется точка `P1 (point (coordinates(P1)))`

- `bool operator == (explicit point M, explicit point P)`

  Определяет проверку `M == N`, которая возвращает `true`, если и только если `MN < EPS`.

- `bool operator != (explicit point M, explicit point N)`

  Определяет проверку `M != N`, которая возвращает значение `true`, если и только если `MN >= EPS`.

- `real abs(coordsys R, pair m)`

  Возвращает абсолютную величину $|m|$ по отношению к системе координат `R`.

- `real abs (explicit point M)`

  Возвращает абсолютную величину $|M|$ по отношению к своей системе координат.

- `real length (explicit point M)`

  Возвращает абсолютную величину $|M|$ по отношению к своей системе координат.

- `point conj(explicit point M)`

  Возвращает сопряженную точку `M` по отношению к своей системе координат.

- `real degrees (explicit point M, coordsys, R = M.coordsys, bool warn=true)`

  Возвращает угол `M` (в градусах) по отношению к системе координат `R`.

- `real angle(explicit point M, coordsys, R=M.coordsys, bool warn=true)`

  Возвращает угол `M` (в радианах) по отношению к системе координат `R`.

- `bool finite(explicit point p)`

  То же самое, что `finite(pair m)` для того, чтобы избежать вычислений с бесконечными координатами.

- `real dot(point A, point B)`

  Возвращает скалярное произведение `A.B`, вычисленное в системе координат точки `A`.

- `real dot(point A, explicit pair B)`

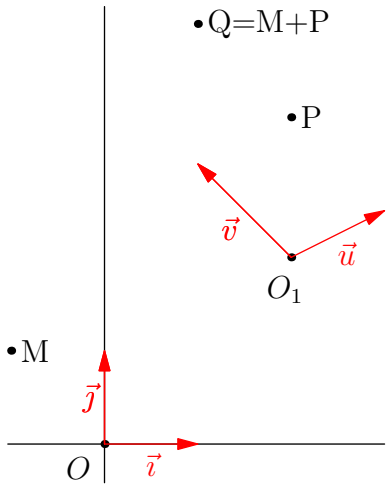
  Возвращает скалярное произведение `A.B`, причем координаты точки `A` сначала преобразуются в координаты в системе по умолчанию. `dot(explicit pair B)` также определено.

## 3.2 Векторы

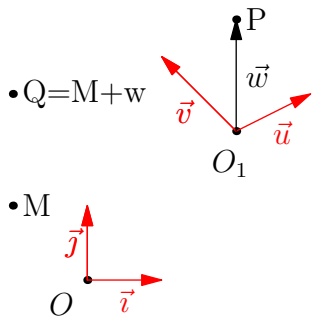
### 3.2.1 Основные принципы

В следующем примере точки M и P определяются относительно двух разных систем координат. Точка $Q = M + P$ получается путем сложения координат M и P **после их преобразования в систему координат по умолчанию**. Таким образом $\overrightarrow{OQ} = \overrightarrow{OM} + \overrightarrow{OP}$.

```
import geometry;
size(5cm,0);
show(currentcoordsys);
coordsys R=cartesiansystem(O=(2,2),
                   i=(1,0.5), j=(-1,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$",
                     R, xpen=invisible);
point M=(-1,1);
dot("M", M);
point P=point(R, (1,1));
dot("P", P);
point Q=M+P; dot("Q=M+P", Q);
```




Если мы определяем вектор $\vec{w}$ как `vector w=vector(R, P.coordinates);` (или проще `w=P;`) тогда мы имеем $\vec{w} = \overrightarrow{O_1P}$, и код `point Q=M+w` определяет $Q$ так, что $\overrightarrow{OQ} = \overrightarrow{OM} + \overrightarrow{O_1P}$. Приведем пример:

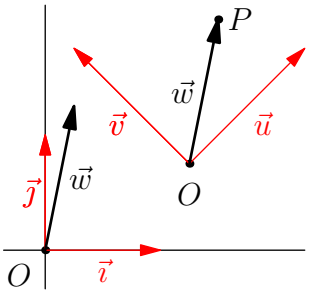



```
import geometry;
size(4cm,0);
show(currentcoordsys, xpen=invisible);
coordsys R=cartesiansystem(O=(2,2),i=(1,0.5),j=(-1,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$", R,
                                    xpen=invisible);
point M=(-1,1); dot("M", M);
point P=point(R, (1,1)); dot("P", P);
vector w=P; show("$\vec{w}$", w);
point Q=M+w; dot("Q=M+w", Q);
```

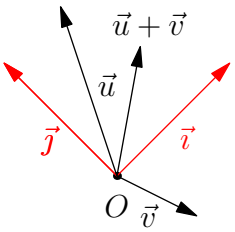
Объект `u` типа `vector` ведёт себя так же, как и объект типа `point`, но его преобразование в `pair` или в `point M` относительно системы координат по умолчанию таково, что $\vec{u} = \overrightarrow{OM}$.

Смотри ниже пример, который использует процедуру `pair locate(vector v)`:

```
import geometry;
size(4cm,0);
currentcoordsys=cartesiansystem((1.25,0.75), i=(1,1),
                                          j=(-1,1));
coordsys Rp=currentcoordsys;
coordsys R=defaultcoordsys;
show(R); show("$O'$","$\vec{u}$","$\vec{v}$", Rp,
                                      xpen=invisible);
point P=(0.75,0.5); dot("$P$",P); vector w=P;
pen bpp=linewidth(bp);
draw("$\vec{w}$",origin()--origin()+w, W,bpp,
                                        Arrow(3mm));
draw("$\vec{w}$", origin--locate(w), E, bpp,
                                        Arrow(3mm));
```

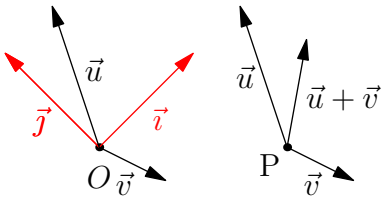



Процедуры, которые описаны для типа `point`, также доступны для типа `vector`. См. ниже некоторые простые примеры:




```
import geometry;
size(5cm,0);
currentcoordsys=cartesiansystem((0,0),i=(1,1),j=(-1,1));
show(currentcoordsys, xpen=invisible);
vector u=(0.5,1), v=rotate(-135)*u/2;
show("$\vec{u}$", u);
show("$\vec{v}$", v);
show(Label("$\vec{u}+\vec{v}$",EndPoint), u+v);
```

```
import geometry;
size(5cm,0);
currentcoordsys=cartesiansystem((0,0),i=(1,1),
                                        j=(-1,1));
show(currentcoordsys, xpen=invisible);
vector u=(0.5,1), v=rotate(-135)*u/2;
show("$\vec{u}$", u); show("$\vec{v}$", v);
point P=(1,-1); dot("P", P, SW);
draw(Label("$\vec{u}$",align=W),P--(P+u), Arrow);
draw("$\vec{v}$", P--(P+v), Arrow);
draw("$\vec{u}+\vec{v}$",P--(P+(u+v)), Arrow);
```



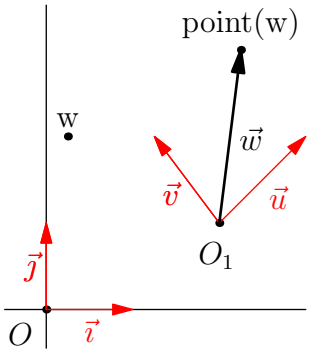

### 3.2.2 Преобразование «вектор/точка»

В процессе приведения объект `point` можно преобразовать в объект `vector` и. взаимно, объект `vector` может быть преобразован в объект `point`. Возможно, я повторюсь, но важно понять, что разницы между `point` и `vector` не существует, если вы не используете другую систему координат, нежели по умолчанию.

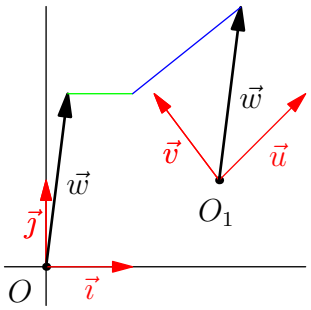
В следующем примере обратите внимание на различие между `dot(w)` и `dot(point(w))`, во втором случае вектор преобразуется в точку, на которую «указал» этот вектор в системе координат R, в то время как в первом случае `vector` преобразуется в `pair`, как описано выше.

Заметим, что можно написать `point M=w;` вместо `point M=point(w);`.

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,1), i=(1,1),
                                    j=(-0.75,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$", R,
                                xpen=invisible);
show(currentcoordsys);

vector w=vector(R, (1,1));
show("$\vec{w}$", w, linewidth(bp), Arrow(3mm));
dot("w", w, N); dot("point(w)", point(w), N);
```



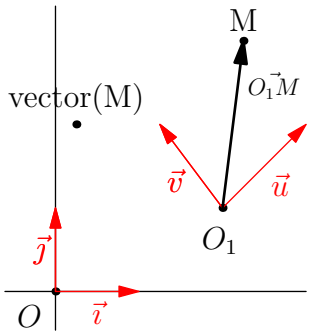
Внимательный читатель поймет следующий пример:




```
import geometry;
size(4cm,0); pen bpp=linewidth(bp);
coordsys R=cartesiansystem((2,1), i=(1,1),
                                  j=(-0.75,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$", R,
                                xpen=invisible);
show(currentcoordsys); vector w=vector(R, (1,1));
show("$\vec{w}$", w, bpp, Arrow(3mm));
show("$\vec{w}$", locate(w), bpp, Arrow(3mm));
draw((1,2)--locate(w), green);
draw((1,2)--point(w), blue);
```

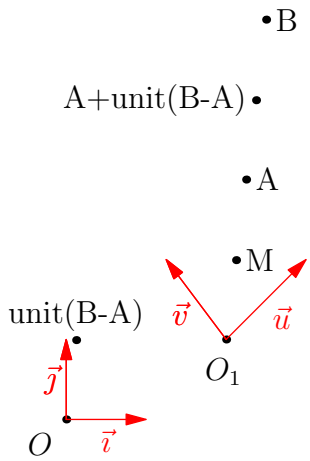
Обратной к процедуре `point point(explicit vector)` является `vector vector(point)`

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,1), i=(1,1),
                               j=(-0.75,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$", R,
                           xpen=invisible);
show(currentcoordsys);
point M=point(R, (1,1)); dot("M", M, N);
dot("vector(M)", vector(M), N);
show(Label(scale(0.75)*"$\vec{O_1M}$",
Relative(0.75)), M, linewidth(bp), Arrow(3mm));
```




Наконец, читатель должен обратить внимание на процедуру `vector unit(point)` и процедуру `vector unit(vector)`, которые всегда возвращают объект `vector`. Таким образом в следующем примере поведение `point P=unit(B-A);` не удивительно, в то время как `dot(unit(B-A))` может вызывать сомнения.

```
import geometry;
size(4cm,0);
coordsys R=cartesiansystem((2,1), i=(1,1),
                               j=(-0.75,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$", R,
                           xpen=invisible);
show(currentcoordsys, xpen=invisible);
point A=point(R, (1,1)); dot("A", A);
point B=point(R, (2,2));
dot("B", B); point M=unit(B-A); dot("M", M);
dot("unit(B-A)", unit(B-A), N);
dot("A+unit(B-A)", A+unit(B-A), W);
```




### 3.2.3 Другие процедуры

Как уже упоминалось, все подпрограммы, посвященные типу `point` также можно использовать с типом `vector`. Упомянем легко понятную обычную процедуру:

```
bool collinear(vector u, vector v)
```

# 4 Материальные точки

## 4.1 Основные принципы

Объект «точка» может обладать массой, которая присваивается полем `a_point.m` и существует процедура, позволяющая вычислять центр масс множества точек. «Замечательно!» Нет... не совсем.

- Если определена точка $M = P1 + P2;$, то масса точки $M$ есть сумма масс $P1$ и $P2$, что приятно, однако если $M$ определена как $M = P/2$, то координаты $M$ – это половины координат $P$, но масса остается неизменной. Таким образом, чтобы определить точку, имеющую половину массы от данной, мы **должны** написать:

  ```
  point P=point((1,1), 3); // point of mass 3

  point Q=P;
  Q.m=P.m/2;
  ```

  что может очень скоро вызвать проблемы.

- Также проблематично указывать массу каким-либо согласованным способом, используя, в то же время, процедуры `dot` и `label`.

Руководствуясь, по большей части, указанными причинами, в пакете *geometry.asy* определяется новый тип «mass», который наилучшим образом ведет себя в качестве «материальной точки». Например, $mass\, M = object\_mass/2$ определяет материальную точку $M$ с теми же координатами, что и у $M$, но вдвое меньшей массы, а выражение $point\, M = object\_mass/2$ ведет себя так же, и напрямую конвертируется в `point`.

Подпрограммы `mass mass(explicit point)` и `point point(explicit mass)` позволяют легко переключаться между двумя типами: `mass` и `point`; элегантный способ выполнить деление над массой объекта типа `point` состоит в следующем:

```
point P=point((1,1), 3); // Точка массы 3

point Q=mass(P)/2; // Деление массы; координаты не изменяются
```

Деление координат объекта типа `mass` производится аналогично:

```
mass P=mass((1,1), 3); // Вес равен 3

mass Q=point(P)/2; // Деление координат; вес не изменяется.
```

## 4.2 Другие процедуры

Благодаря приведению типов, все свойства типа `point` имеют место для типа `mass`, учитывая уже подмеченные ньюансы. Ниже приводится список других подпрограмм, связанных с материальными точками:

- `mass mass(coordsys R, explicit pair p, real m)` Возвращает объект типа `mass` веса `m`, чьи координаты в `R` есть `p`.
- `mass mass(point M, real m)` Переводит точку `M` в материальную точку массы `m`.
- `point(explicit mass m)` Переводит материальную точку типа `mass` в точку типа `point`.
- `mass mass(explicit point P)` Переводит материальную точку типа `point` в точку типа `mass`.
- `mass masscenter(... mass[] M)` Вычисляет центр масс массива `M`. Благодаря приведению `point[]` к `mass[]`, эта подпрограмма, заметьте, работает с массивом типа `point[]`.
- `string defaultmassformat;`

  Формат по умолчанию для построения подписи, в которой указывается масса. Его значением по умолчанию является следующее:

  ```
  "$\left(%L;%.4g\right)$"
  ```

где вместо %L будет подставлена метка со значением массы. Следующий пример поясняет эту подпрограмму:


```
import geometry;
size(4cm,0);
mass M=mass((0,0), 1); dot("M", M);
defaultmassformat="$%L(%.4g)$";
dot("M", M+(1,0));
```

- `string massformat(string format=defaultmassformat,string s, mass M)`

  Возвращает строку, которая форматирована командой `format`, в которой `format` передается в качестве параметра, где %L заменяется на $s$ и вес точки $M$.

```
import geometry;
write(massformat(s="foo", mass((0,0),1000)));
// Return (foo; 1000)

write(massformat("%L_%e", "foo", mass((0,0),1000)));
// Return foo_1×10^3
```

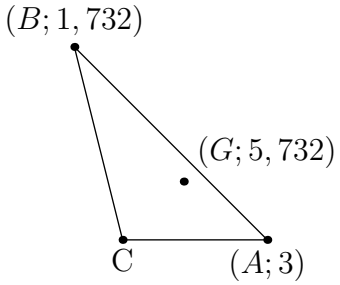
- `void label(picture pic=currentpicture, Label L, mass M,`
  `align align=NoAlign, string format=defaultmassformat,`
  `pen p=nullpen, filltype filltype=NoFill)`

  Рисуется метка точки $M$, возвращаемая `massformat(format,L,M)`, вместе с координатами точки $M$.

- `void dot(picture pic=currentpicture, Label L, mass M, align align=NoAlign,`
  `string format=defaultmassformat, pen p=currentpen)`

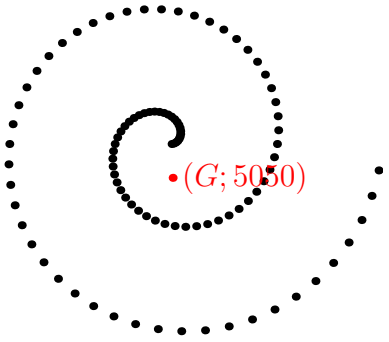
  Рисует точку $M$ и метку, возвращаемую при вызове `massformat(format,L,M)`.

В завершение данного раздела, укажем три примера, использующих некоторые из упомянутых подпрограмм.

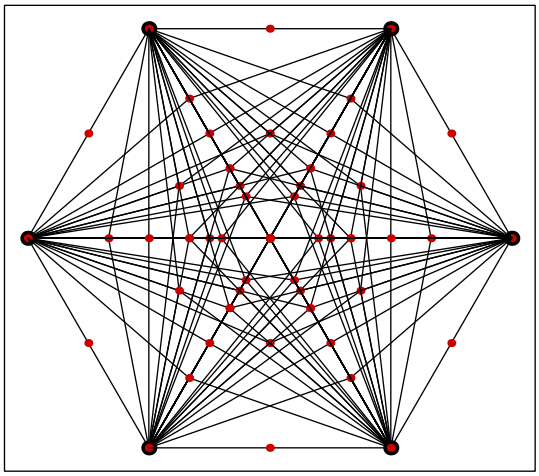



```
import geometry;
size(4cm,0);
mass A=mass((1,0), 3);
mass B=mass((0,1), sqrt(3));
point C=(0.25,0); // C has the default weight:1.
dot("B", B, N); dot("C", C, S); dot("A", A, S);
draw(A--B--C--cycle);
dot("G", masscenter(A,B,mass(C)), 2NE);
```

```
import geometry;
size(5cm,0);
int n=50;
mass[] M;
real m, step=360/n;
pair dir;
for (int i=0; i < 2*n; ++i)
{
  dir=dir(i*step);
  m=i+1;
  M.push(mass(m*dir, m));
  dot(locate(M[i]));
}
dot("G",masscenter(... M), red);
```



Следующий пример показывает, как можно построить все частичные центры для $n$ точек; каждый центр соединен с точками, из которых он возник.



```
import geometry;
size(7cm,0);
int[][] parties(int n) {
int[][] oi;
void loop(int[] arr, int i) {
oi.push(arr);
for (int j=i; j < arr.length; ++j) {
int[] tt=copy(arr);
tt[j]=1;
loop(tt, j+1);}}
loop(sequence(new int(int n){return 0;}, n), 0);
return oi;}
int n=6;
real step=360/n;
point[] M;
for (int i=0; i < n; ++i) {
M[i]=(dir(i*step), 1);// слово point удалено как лишнее
dot(M[i],linewidth(2mm));}
int[][] part=parties(n); int l=part.length;
point[][] group=new point[l][];
for (int i=0; i < l; ++i)
for (int j=0; j < n; ++j)
if(part[i][j] == 1) group[i].push(M[j]);
point[][] partbar=new point[l][2];
for (int i=0; i < l; ++i) {
if(group[i].length > 0) partbar[i][0]=masscenter(...group[i]);
for (int j=0; j < group[i].length; ++j)
draw(group[i][j]--partbar[i][0]);
if(group[i].length > 0) dot(partbar[i][0], 0.8*red);}
```

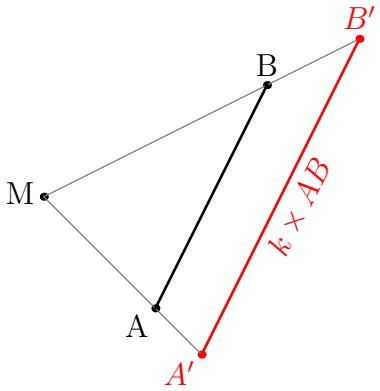
# 5 Преобразования (Часть 1)

В дополнение к обычным аффинным преобразованиям в пакете *geometry.asy* определяются и другие преобразования плоскости. Некоторые из них ведут себя специфически по отношению к используемой текущей системе координат. Таким образом, этот раздел разбит на два подраздела. В первом из них мы изучаем преобразования, которые не зависят от текущей системы координат. Во втором мы детализируем сведения о преобразованиях, зависящих от системы координат.

## 5.1 Преобразования, не зависящие от системы координат

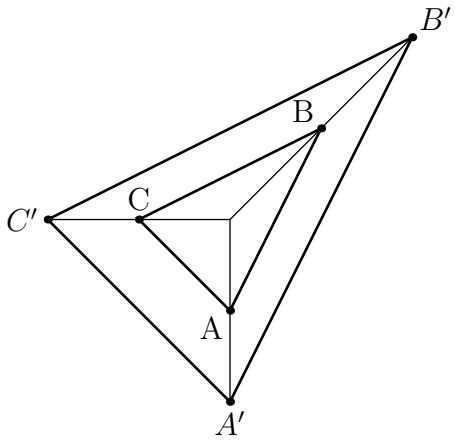
- `transform scale(real k, point M)`

  Гомотетия относительно центра $M$ с коэффициентом $k$.




```
import geometry;
size(5cm,0);
pen bpp=linewidth(bp); real k=sqrt(2);

point A=(0,0); dot("A", A, SW);
point B=(1,2); dot("B", B, N);
point M=(-1,1);
dot("M", M, -dir(M--A,M--B));

point Ap=scale(k, M)*A;
dot("A'", Ap, SW, red);
point Bp=scale(k, M)*B;
dot("B'", Bp, N, red);

draw(M--Ap, grey); draw(M--Bp, grey);
draw(A--B, bpp);
draw(rotate(unit(Bp-Ap))*"$k\times AB$",
Ap--Bp, bpp+red);
```

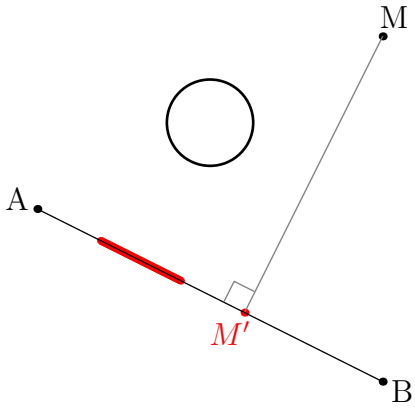
```
import geometry;
size(5cm,0);
pen bpp=linewidth(bp);
point A=(0,0);
dot("A", A, SW);
point B=(1,2);
dot("B", B, NW);
point C=(-1,1);
dot("C", C,N);
path g=A--B--C--cycle;
draw(g, bpp);
point M=(0,1);
path gp=scale(2, M)*g;
draw(gp, bpp);
for (int i=0; i < 3; ++i)
draw(M--point(gp,i));
dot("A'", point(gp,0), SW);
dot("B'", point(gp,1), NW);
dot("C'", point(gp,2), N);
```




- `transform projection(point A, point B)`

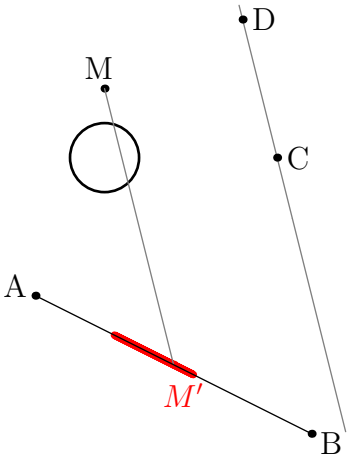
  Ортогональная проекция на прямую $(AB)$

```
import geometry;
size(6cm);
point A=(2,2); point B=(4,1);
point M=(4,3);
path cle=shift(3,2.5)*scale(.25)*unitcircle;
draw(cle, linewidth(bp));
transform proj=projection(A,B);
point Mp=proj*M;
draw(proj*cle, 1mm+red);
dot("A",A,unit(A-B)); dot("B",B,unit(B-A));
dot("M", M, unit(M-Mp));
dot("M'", Mp, unit(Mp-M), red);
draw(M--Mp, grey); draw(A--B);
markrightangle(M,Mp,A, grey);
```




- `transform projection(point A, point B, point C, point D, bool safe=false)`

  Возвращает проекцию параллельно прямой $(CD)$ на прямую $(AB)$. Если значение `safe` есть `true`, и если линия $(AB)$ параллельна линии $(CD)$, то возвращается тождественное преобразование. Если значение `safe` есть `false`, и если линия $(AB)$ параллельна линии $(CD)$, то возвращается гомотетия с центрм $O$ и бесконечным коэффициентом.




```
import geometry;
size(6cm);
point A=(2,2); point B=(4,1);
point C=(3.75,3);
point D=(3.5,4); point M=(2.5,3.5);
path cle=
shift(2.5,3)*scale(0.25)*unitcircle;
draw(cle, linewidth(bp));
draw(line(C,D), grey);

transform proj=projection(A,B,C,D);
point Mp=proj*M;

draw(proj*cle, 1mm+red);
dot("A", A, unit(A-B));
dot("B", B, unit(B-A));
dot("C", C);
dot("D", D);
dot("M", M, unit(M-Mp));
dot("M'", Mp, 2*unit(Mp-M), red);
draw(M--Mp, grey);
draw(A--B);
```

- `transform scale(real k, point A, point B, point C, point D, bool safe=false)`

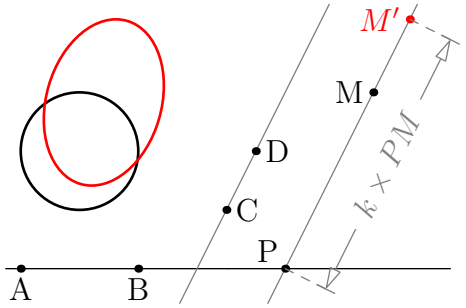
  Возвращается аффинное преобразование: косое сжатие с коэффициентом $k$, к оси $(AB)$, в направления $(CD)$.

  Если $M$ — точка евклидовой плоскости, то ее образ $M'$ определяется как $P + k\overrightarrow{PM}$, где точка $P$ является проекцией точки $M$ вдоль $(CD)$ на $(AB)$.

  Если значение `safe true`, и если линия $(AB)$ параллельна линии $(CD)$, то возвращается тождественное преобразование.

  Если значение `safe false`, и если линия $(AB)$ параллельна линии $(CD)$, то возвращается гомотетия с центрм $O$ и бесконечным коэффициентом. (Далее следует код и рисунок)

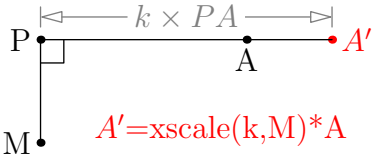
```
import geometry;
size(6cm,0);
pen bpp=linewidth(bp); real k=sqrt(2);
point A=(0,0), B=(2,0), C=(3.5,1);
point D=(4,2), M=(6,3);
path cle=shift(1,2)*unitcircle;
draw(cle, bpp);
draw(line(A,B));
draw(line(C,D), grey);
transform dilate=scale(k,A,B,C,D);
draw(dilate*cle, bpp+red);
point Mp=dilate*M;
point P=intersectionpoint(line(A,B),
line(M,Mp)); draw(line(P,M), grey);
dot("A", A, S); dot("B", B, S);
dot("C", C);dot("D", D);
dot("M", M, W); dot("P", P, NW);
dot("M'", Mp, W, red);
distance("$k\times PM$", P, Mp, 6mm,
grey, joinpen=grey+dashed);
```




## 5.2 Преобразования, зависящие от системы координат

- `transform xscale(real k, point M)`

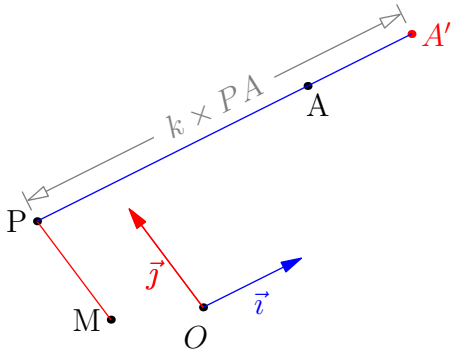
  Аффинное преобразование «сжатие с коэффициентом $k$ в направлении оси $(Ox)$ к прямой, проходящей через точку $M$ и параллельной оси $(Oy)$».




```
import geometry;
size(5cm,0);
real k=sqrt(2);
point A=(1,2); dot("A", A, S);
point M=(-1,1); dot("M", M, W);
point Ap=xscale(k, M)*A; dot("A'", Ap, red);
label("A'=xscale(k,M)*A", (0.75,1.125), red);
point P=extension(A, Ap, M, M+N);
dot("P", P, W); draw(M--P); draw(P--Ap);
perpendicularmark(P, dir(-45));
distance("$k\times PA$", P, Ap, -3mm, grey);
```

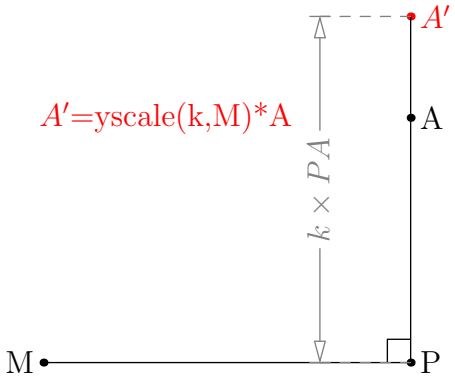
Тот же пример, в произвольной системе координат:

```
import geometry; size(6cm,0);
currentcoordsys=cartesiansystem((2,1),
                    i=(1,0.5), j=(-0.75,1));
show(currentcoordsys, ipen=blue, jpen=red,
                              xpen=invisible);
real k=sqrt(2); point A=(2,1.25);
point M=(-0.75,0.25); dot("M", M, W);
point Ap=xscale(k, M)*A;
dot("A'", Ap, red); dot("A", A, I*unit(A-Ap));
point P=intersectionpoint(line(A,Ap),
                                line(M,M+N));
dot("P", P, W); draw(M--P, red); draw(P--Ap,
                                       blue);
distance("$k\times PA$", P, Ap, -3mm, grey);
```




- `transform yscale(real k, point M)`

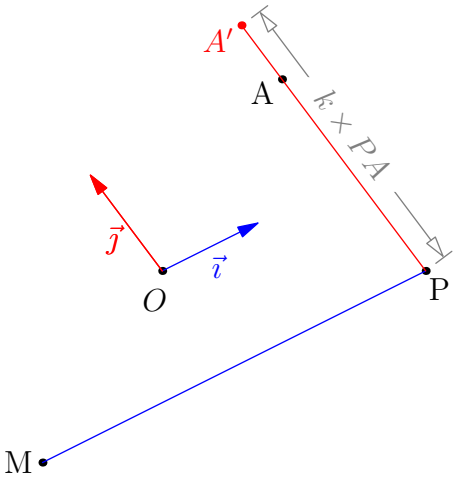
  Аффинное преобразование «сжатие с коэффициентом $k$ в направлении оси $(Oy)$ к прямой, проходящей через точку $M$ и параллельной оси $(Ox)$».




```
import geometry;
size(6cm,0);
real k=sqrt(2);
point A=(2,1);
point M=(-1,-1); dot("M", M, W);
point Ap=yscale(k, M)*A;
dot("A'", Ap, red); dot("A", A,
                        I*unit(A-Ap));
label("A'=yscale(k,M)*A", (0,1), red);
point P=intersectionpoint(line(A,Ap),
                          line(M,M+E));
dot("P", P); draw(M--P); draw(P--Ap);
perpendicularmark(P, dir(135));
distance("$k\times PA$",P,Ap,-12mm,
                    grey,grey+dashed);
```

Тот же пример, в произвольной системе координат:

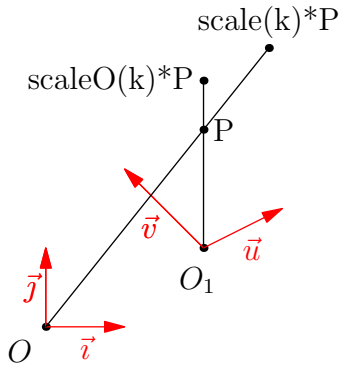
```
import geometry; size(6cm,0);
currentcoordsys=cartesiansystem((2,1),
              i=(1,0.5), j=(-0.75,1));
show(currentcoordsys, ipen=blue,
            jpen=red, xpen=invisible);
real k=sqrt(2); point A=(2,1);
point M=(-2,-1); dot("M", M, W);
point Ap=yscale(k, M)*A;
dot("A'", Ap, -I*unit(A-Ap), red);
           dot("A", A, -I*unit(A-Ap));
point P=intersectionpoint(line(A,Ap),
                          line(M,M+E));
dot("P", P, locate(unit(A-Ap)));
   draw(M--P, blue); draw(P--Ap, red);
distance("$k\times PA$", P, Ap, 3mm,grey);
```




- `transform scale0(real x)`

  Возвращает гомотетию коэффициентом $x$ и центром «начало текущей системы координат». Это преобразование эквивалентно `scale(x, origin())`.

  В следующем примере можно увидеть разницу между `scale(k)*P` и `scale0(k)*P`.




```
import geometry; size(4.5cm,0);
currentcoordsys=cartesiansystem((2,1),
                      i=(1,0.5), j=(-1,1));
show("$O_1$", "$\vec{u}$", "$\vec{v}$",
          currentcoordsys, xpen=invisible);
show(defaultcoordsys, xpen=invisible);
real k=sqrt(2); point P=(1,1); dot("P", P);
point P1=scale(k)*P, P2=scale0(k)*P;
dot("scale(k)*P", P1, N);
dot("scale0(k)*P", P2, W);
draw((0,0)--locate(P1)); draw(origin()--P2);
```

- `transform xscale0(real x)`

  Эквивалентно `xscale(x, origin())` (см. `xscale(real, point)`).

- `transform yscale0(real x)`

  Эквивалентно `yscale(x, origin())` (см. `yscale(real, point)`).

- `transform rotate0(real angle)`

  Эквивалентно `rotate(angle, origin())`.

# 6 Прямые линии, лучи и сегменты

## 6.1 Тип line

Объект типа `line` предназначен для описания прямой, полупрямой (луча) или сегмента, в зависимости от значения `bool extendA, extendB`, которые можно получить кодами `line.extendA` и `line.extendB`. Полные описания методов и свойств типа `line` доступны на сайте:
http://www.piprime.fr/files/asymptote/geometry/modules/geometry.asy.index.type.html.

### 6.1.1 Прямая, определенная двумя точками, основные процедуры

- `line line(point A, bool extendA=true, point B, bool extendB=true)`

  Определяет объект типа `line`, а, точнее, прямую, проходящую через две точки `A` и `B` и направленную от `A` до `B`. Если значение `extendA` — `true`, то прямая выходит за точку `A`. Объект типа `line` принадлежит к той системе координат, в которой определены точки `A` и `B`. Если это не так, точки автоматически переопределяются в существующей системе координат и выдается предупреждающее сообщение.

- `line Ox(coordsys R=currentcoordsys)`

  Возвращает ось `x` системы координат `R`.

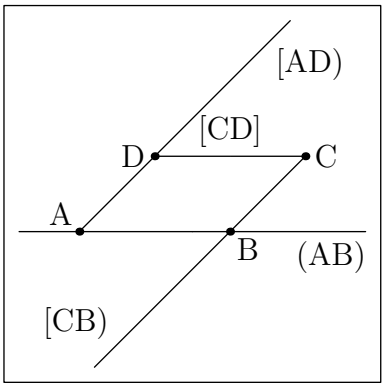
  Определена подпрограмма `line Oy(coordsys R=currentcoordsys)`. Константы `Ox` и `Oy` определены в системе координат по умолчанию.

- `void draw(picture pic=currentpicture, Label L=, line l,`
  `bool dirA=l.extendA, bool dirB=l.extendB, align align=NoAlign, pen p=currentpen,`
  `arrowbar arrow=None, Label legend=, marker marker=nomarker)`

  Чертит на рисунке `pic` прямую `l` без изменения размера изображения для корректного положения метки `Label` и чтобы переменная `linemargin` не являлась отрицательной.

  Логическое значение параметров `dirA` и `dirB` служат для проверки, чертится ли бесконечная часть прямой.

  Заметим, что действительная переменная `linemargin` позволяет контролировать поле между границей изображения и начерченой прямой. Значение по умолчанию равно 0, отрицательное значение будет менять размер изображения.




```
import geometry;
size(6cm,0);
linemargin=2mm;
point A=(0,0), B=(2, 0), C=(3,1), D=(1,1);
dot("A", A, NW); dot("B", B, SE); dot("C", C);
dot("D", D, W);

line AB=line(A, B);
line CB=line(C, false, B);
line CD=line(C, false, D, false);
line AD=line(A, false, D);

draw("(AB)", AB); draw("[CB)", CB);
draw(Label("[CD]",Relative(0.5),align=N), CD);
draw("[AD)", AD); draw(box((-1,-2),(4,3)));
```

- `void show(picture pic=currentpicture, line l, pen p=red)`

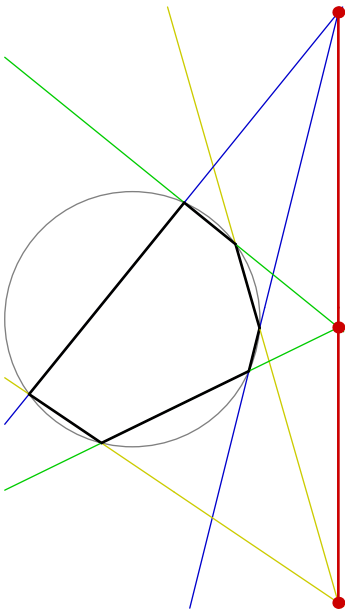
  Строит на рисунке `pic` точки, которые используются для определения прямой `l`, а также направление линии и ортогональное направление.

- `point intersectionpoint(line l1, line l2)`

  Возвращает точку пересечения прямых `l1` и `l2`. Если прямые не пересекаются или если существует бесконечное число точек пересечения, то подпрограмма возвращает точку с координатами $(\infty, \infty)$. Заметим, что если две прямые определены в двух разных системах координат, точка пересечения определяется в системе координат по умолчанию и выдается предупреждающее сообщение.

Следующий пример является иллюстрацией знаменитой теоремы Паскаля, которая утверждает: «Если шестиугольник вписан в коническое сечение, то три точки, в которых пересекаются пары противоположных сторон, лежат на одной прямой.» (в этом примере коническим сечением является окружность).

```
import geometry;
size(6.5cm,0);
draw(unitcircle, grey);
point[] P;
  real[] a=new real[]{0, 20, 60, 90, 240, 280};
real cor=24.0036303043338;
for (int i=0; i < 6; ++i) {
P.push((Cos(a[i]-cor),Sin(a[i]-cor)));
}
pen[] p=new pen[] {0.8*blue, 0.8*yellow,
                                   0.8*green};
line[] l;
for (int i=0; i < 6; ++i) {
l.push(line(P[i],P[(i+1)%6]));
draw(l[i], p[i%3]);
draw(P[i]--P[(i+1)%6], linewidth(bp));
}
point[] inter;
for (int i=0; i < 3; ++i) {
  inter.push(intersectionpoint(l[i],l[(i+3)%6]));
dot(inter[i], 1.5*dotsize()+0.8*red);
}
draw(line(inter[0],inter[1]), bp+0.8*red);
shipout(bbox(2mm));
```



- `point[] intersectionpoints(line l, path g)`

  Возвращает массив всех точек пересечения прямой `l` и пути `g`

### 6.1.2 Прямые, определенные уравнениями

- `line line(coordsys R=currentcoordsys, real a, real b, real c)`

  Возвращает прямую с уравнением $ax + by + c = 0$ в системе координат $R$.

- `line line(coordsys R=currentcoordsys, real slope, real origin)`

  Возвращает прямую с направлением `slope` и точкой пересечения `origin` с осью $y$ в системе координат $R$.

### 6.1.3 Прямые и параллелизм

- `line parallel(point M, line l)`

  Возвращает прямую, проходящую через точку $M$ параллельно $l$.

- `line parallel(point M, explicit vector dir)`

  Возвращает прямую, проходящую через точку $M$ в направлении `dir`.

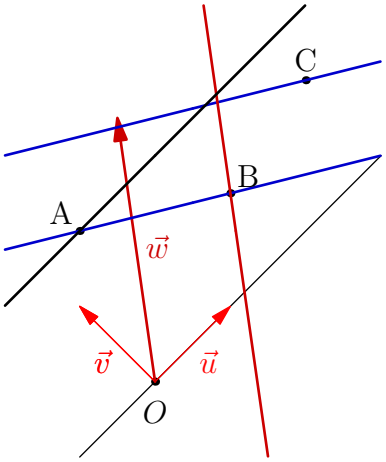
- `line parallel(point M, explicit pair dir)`

  Возвращает прямую, проходящую через точку $M$ в направлении `dir` в текущей системе координат.

- `bool parallel(line l1, line l2, bool strictly=false)`

  Возвращает `true` если `l1` параллельна `l2` (строго, если значение `strictly` `true`).

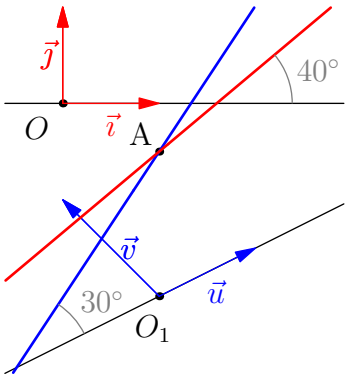
```
import geometry;
size(5cm,0);
coordsys R=cartesiansystem((1,-2),i=(1,1),j=(-1,1));
show("$O$","$\vec{u}$","$\vec{v}$",R,ypen=invisible);
pen bpp=linewidth(bp);
point A=(0,0), B=(2, 0.5), C=(3,2);
vector w=vector(R, (1.5,2));
line AB=line(A,B);
dot("A", A, NW); dot("B", B, NE);
dot("C", C, N);
show("$\vec{w}$", w, bpp+0.8*red, Arrow(3mm));
draw(AB, bpp+0.8*blue);
draw(parallel(C, AB), bpp+0.8*blue);
draw(parallel(B, w), bpp+0.8*red);
draw(parallel(A, R.i), bpp);
draw(box((-1,-3),(4,3)), invisible);
```




### 6.1.4 Прямые и углы

- `line line(real a, point A=point(currentcoordsys,(0,0)))`

  Возвращает прямую, проходящую через точку `A` под углом `a` к оси x в той системе координат, в которой определена точка `A`. Процедура `line(point,real)` также определена.




```
import geometry;
size(5cm,0);
coordsys R=cartesiansystem((1,-2), i=(1,0.5),
                                    j=(-1,1));
show("$O_{1}$", "$\vec{u}$", Label("$\vec{v}$",
         align=E), R, ipen=blue, ypen=invisible);
show(defaultcoordsys, ypen=invisible);
point A=point(R,(1,1)); dot("A", A, NW);

line l=line(A, 30);
draw(l, bp+blue);
markangle("$30^\circ$", Ox(R), l, grey);

A=changecoordsys(defaultcoordsys, A);
line ll=line(A, 40);
draw(ll, bp+red);
markangle("$40^\circ$", Ox, ll, grey);
draw(box((-0.6,-2.8), (2,-0.3)), invisible);
```

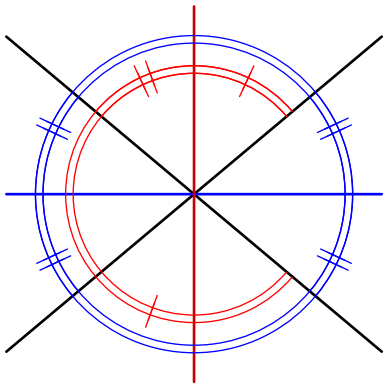
- `line bisector(line l1, line l2, real angle=0, bool sharp=true)`

  Возвращает образ биссектрисы угла между двумя ориентированными прямыми `l1` и `l2`, которая получена поворотом `l1` на угол `angle` вокруг точки пересечения `l1` и `l2`.

  Если значение `sharp=true`, возвращается внутренняя биссектриса. Заметим, что возвращаемая прямая наследует систему координат, в которой определяется прямая `l1`.

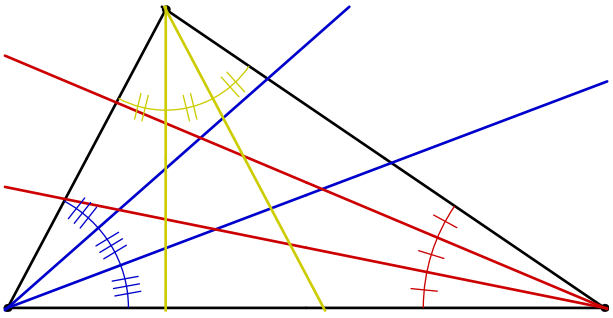
```
import geometry;
size(5cm,0);
point A=(0,0), B=(2*Cos(40),2*Sin(40)); line l1=line(A,B);
draw(l1, linewidth(bp));
line l2=rotate(100,A)*l1;
draw(l2, linewidth(bp));
line bis=bisector(l1,l2); draw(bis, bp+blue);
line Bis=bisector(l1,l2,false); draw(Bis, bp+0.8*red);
markangleradiusfactor *= 4;
marker mark2=StickIntervalMarker(2, 1, red, true);
```

```
markangle(2, l1, l2, red, mark2);
markangle(2, reverse(l2), reverse(l1), radius=-markangleradius(),
red, mark2);
markangleradiusfactor *= 3/2;
marker mark1=StickIntervalMarker(2, 2, blue, true);
markangle(2, l1, reverse(l2), radius=-markangleradius(),
blue, mark1);
markangle(2, reverse(l1), l2, radius=-markangleradius(),
blue, mark1);
draw(box((-1,-1),(1,1)), invisible);
```



- `line sector(int n=2, int p=1, line l1, line l2, real angle=0, bool sharp=true)`

  Возвращает изображение $p$-й линии, из числа тех, которые делят угол между ориентированными прямыми $l1$ и $l2$ на $n$ равных частей, полученной поворотом вокруг точки пересечения прямых $l1$ и $l2$ на угол `angle`. Если значение sharp истинно, эта процедура выполняется над острым углом. Заметим, что возвращаемая прямая наследует систему координат, в которой определяется $l1$. См. ниже пример деления угла на три равные части.



```
import geometry;
size(8cm,0);
point A=(0,0), B=(3,0), C=(0.795,1.5);
dot(A); dot(B); dot(C);
pen pb=0.8*blue, pr=0.8*red, py=0.8*yellow, bpp=linewidth(bp);
line AB=line(A,B), AC=line(A,C), BC=line(B,C);
draw(AB, bpp); draw(AC, bpp); draw(BC, bpp);
line bA1=sector(3,AB,AC), bA2=sector(3,2,AB,AC);
line bB1=sector(3,AB,BC), bB2=sector(3,2,AB,BC);
line bC1=sector(3,AC,BC), bC2=sector(3,2,AC,BC);
draw(bA1, bpp+pb); draw(bA2, bpp+pb);
draw(bB1, bpp+pr); draw(bB2, bpp+pr);
draw(bC1, bpp+py); draw(bC2, bpp+py);
markangleradiusfactor *= 8;
markangle(BC, reverse(AB), pr, StickIntervalMarker(3,1,pr,true));
markangleradiusfactor /= 3;
```

```
markangle(reverse(AC), reverse(BC), py, StickIntervalMarker(3,2,py,true));
markangleradiusfactor *= 3/2;
markangle(AB, AC, pb, StickIntervalMarker(3,3,pb,true));
```

- `line perpendicular(point M, line l)`

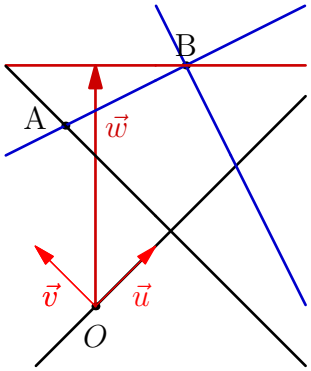
  Возвращает прямую, проходящую через `M` и перпендикулярную к прямой `l`.

- `line perpendicular(point M, explicit vector normal)`

  Возвращает прямую, проходящую через `M` перпендикулярно направлению (или вектору нормали) `normal`.

- `line perpendicular(point M, explicit pair normal)`

  Возвращает прямую, проходящую через `M` перпендикулярно направлению `normal` в текущей системе координат `currentcoordsys`.




```
import geometry;
size(4cm,0);
pen bpp=linewidth(bp);
coordsys R=cartesiansystem((0.5,-2), i=(1,1), j=(-1,1));
show("$O$","$\vec{u}$","$\vec{v}$", R, xpen=bpp,
ypen=invisible);
point A=(0,1), B=(2,2);
vector w=vector(R, (2,2)); line AB=line(A,B);
dot("A", A, 2*dir(165)); dot("B", B, N);
show(Label("$\vec{w}$",Relative(0.75)), w, bp+0.8*red,
Arrow(3mm));
draw(AB, bp+0.8*blue);
draw(perpendicular(B, AB), bp+0.8*blue);
draw(perpendicular(B, w), bp+0.8*red);
draw(perpendicular(A, R.i), bpp);
draw(box((-1,-3),(4,3)), invisible);
```

- `real angle(line l, coordsys R=coordsys(l))`

  Возвращает радианную меру угла, в промежутке $(-\pi;\pi]$, соответствующую системе координат `R` и ориентированной прямой `l` то есть угол между ориентированной прямой и осью `x`.

- `real degrees(line l, coordsys R=coordsys(l))`

  Возвращает градусную меру угла, в промежутке $[0;360)$, соответствующую системе координат `R` и ориентированной прямой `l` то есть угол между ориентированной прямой и осью `x`.

- `real sharpangle(line l1, line l2)`

  Возвращает радианную меру острого ориентированного угла в промежутке $(-\frac{\pi}{2};\frac{\pi}{2}]$ между `l1` и `l2`.

- `real sharpdegrees(line l1, line l2)`

  Возвращает градусную меру угла, в промежутке $(-90;90]$ между прямыми `l1` и `l2`.

- `real angle(line l1, line l2)`

  Возвращает радианную меру ориентированного угла в промежутке $(-\pi;\pi]$ между ориентированными прямыми `l1` и `l2`.

- `real degrees(line l1, line l2)`

  Возвращает градусную меру ориентированного угла в промежутке $(-180;180]$ между ориентированными прямыми `l1` и `l2`.

### 6.1.5 Прямые и операторы

- `line operator *(transform t, line l)`

  Дает код `transform*line`.

- `line operator /(line l, real x)`

  Дает код `line/real`. Возвращает «прямую», проходящую через `l.A/x` и `l.B/x`. Код `line operator *(real x, line l)` также определен.

- `line operator *(point M, line l)`

  Дает код `point*line`. Возвращает «прямую», проходящую через `unit(M)*l.A` и `unit(M)*l.B`.

- `line operator +(line l, vector u)`

  Дает код `line+vector`. Возвращает образ прямой `l` при переносе на вектор `u`. Код `line operator -(line l, vector u)` также определен.

- `line[] operator ^^(line l1, line l2)`

  Дает код `line ^^line`. Возвращает массив `new line[] l1,l2`.

- `bool operator ==(line l1, line l2)`

  Дает тест `line == line`. Возвращает `true`, если и только если прямые `l1` and `l2` равны.

- `bool operator !=(line l1, line l2)`

  Дает тест `line != line`. Возвращает `false`, если и только если прямые `l1` and `l2` равны.

- `bool operator @(point M, line l)`

  Дает код `point @ line`. Возвращает `true`, если и только если точка `M` принадлежит объекту `l`.

### 6.1.6 Другие процедуры

В этом разделе мы опишем процедуры, которые касаются линий. Некоторые из них позволяют получить абсциссы точек, принадлежащих объектам типа `line`.

- ```
  void draw(picture pic=currentpicture,Label[] L=new Label[], line[] l,
            align align=NoAlign, pen[] p=new pen[], arrowbar arrow=None,
                      Label[] legend=new Label[], marker marker=nomarker)
  ```

  Рисует каждую прямую, определенную в массиве `line[] l`, соответствующим пером массива `pen[] p`. Если `p` не указано, используется текущее перо.

- ```
  void draw(picture pic=currentpicture,Label[] L=new Label[], line[] l,
            align align=NoAlign, pen p, arrowbar arrow=None,
                      Label[] legend=new Label[], marker marker=nomarker)
  ```

  Рисует каждую прямую, определенную в массиве `line[] l` одним и тем же пером `p`.

- `real distance(point M, line l)`

  Возвращает расстояние от точки `M` до прямой `l`. Функция `real distance(line l,point M)` также определена.

- `bool sameside(point M, point P, line l)`

  Возвращает `true` если и только если `M` и `P` находятся по одну сторону от `l`.

- `point[] sameside(point M, line l1, line l2)`

  Возвращает массив из двух точек: первая — проекция `M` на `l1` вдоль `l2` и вторая — проекция `M` на `l2` вдоль `l1`.

- `coordsys coordsys(line l)`

  Возвращает систему координат, в которой определена прямая `l`.

- `line changecoordsys(coordsys R, line l)`

  Возвращает «прямую», описывающую `l` в системе координат `R`.

- `line reverse(line l)`

  Возвращает «прямую», описывающую `l` с противоположной ориентацией.

- `line extend(line l)`

  Возвращает «полную прямую» которая содержит данный луч или сегмент.

- `line complementary(explicit line l)`

  Если `l` есть луч, то возвращается луч, дополнительный к `l`.

- `bool concurrent(... line[] l)`

  Возвращается `true`, если и только если все прямые массива `line[] l` проходят через одну точку.

- `bool perpendicular(line l1, line l2)`

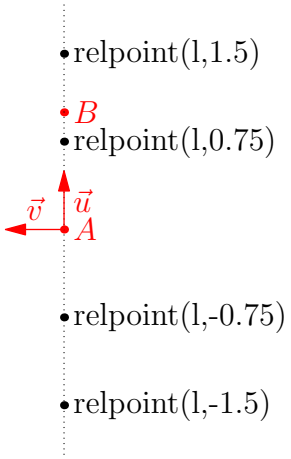
  Возвращается `true`, если и только если прямые `l1` и `l2` перпендикулярны.

- `point point(line l, real x)`

  Возвращает точку между `l.A` и `l.B` как точку с кодом `point(l.A - - l.B,x)`.

- `point relpoint(line l, real x)`

  Возвращает точку с относительной абсциссой `x` в отношении ориентированного отрезка `[AB]`. Иными словами, коды `relpoint(l,x)` и `l.A+x*vector(l.B-l.A)` эквивалентны.

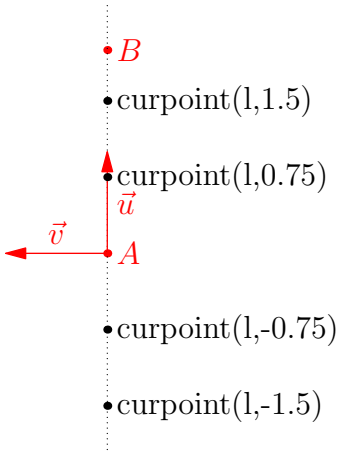



```
import geometry;
size(0,6cm);
point A=(0,0), B=(0,2);
line l=line(A,B); show(l);
dot("relpoint(l,0.75)", relpoint(l,0.75));
dot("relpoint(l,-0.75)", relpoint(l,-0.75));
dot("relpoint(l,1.5)", relpoint(l,1.5));
dot("relpoint(l,-1.5)", relpoint(l,-1.5));
addMargins(bmargin=5mm);
```

- `point curpoint(line l, real x)`

  Возвращает точку с абсциссой `x` соответствующей системе координат `(l.A ; l.u)`. Другими словами, коды `curpoint(l,x)` и `l.A+x*unit(l.B-l.A)` эквивалентны.

```
import geometry;
size(0,6cm);
point A=(0,0), B=(0,2);
line l=line(A,B); show(l);
dot("curpoint(l,0.75)", curpoint(l,0.75));
dot("curpoint(l,-0.75)", curpoint(l,-0.75));
dot("curpoint(l,1.5)", curpoint(l,1.5));
dot("curpoint(l,-1.5)", curpoint(l,-1.5));
addMargins(bmargin=5mm);
```



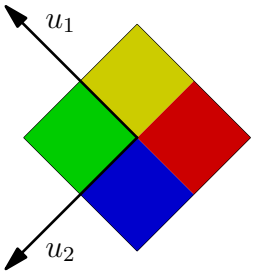
### 6.1.7 Прямые и маркеры

- `void markangle(picture pic=currentpicture, Label L="", int n=1,`
  `real radius=0, real space=0, line l1, line l2, arrowbar arrow=None,`
  `pen p=currentpen, margin margin=NoMargin, marker marker=nomarker)`

  Эта процедура отмечает круговой дугой (дугами) ориентированный угол между прямыми l1 и l2. Дуга(дуги) чертятся против часовой стрелки, если radius является положительным, в противном случае — по часовой стрелке.

- `void perpendicularmark(picture pic=currentpicture, line l1, line l2,`
  `real size=0, pen p=currentpen, int quarter=1,`
  `margin margin=NoMargin, filltype filltype=NoFill)`

  Отмечает прямым углом в точке пересечения `l1` и `l2` в четверти quarter против часовой стрелки, первая четверть определена векторами `l1.u` и `l2.u`.




```
import geometry;
size(3.5cm,0);
transform t=rotate(135);
line l1=t*line((0,0),E); line l2=t*line((0,0),N);

perpfactor *=3.5;
perpendicularmark(l1,l2, Fill(0.8*green));
perpendicularmark(l1,l2, quarter=2, Fill(0.8*blue));
perpendicularmark(l1,l2, quarter=3, Fill(0.8*red));
perpendicularmark(l1,l2, quarter=4, Fill(0.8*yellow));

pen bpp=linewidth(bp); position pos=Relative(0.75);
show(Label("$u_1$",pos), l1.u, bpp, Arrow(3mm));
show(Label("$u_2$",pos,align=SE), l2.u, bpp, Arrow(3mm));
show("", -l1.u, invisible); show("", -l2.u, invisible);
```

## 6.2 Тип segment

Как заявлено во *Введении*, тип `segment` является сегментом прямой, зависит от типа `line`. Почти все процедуры в отношении объектов типа `line` пригодны для объекта типа `segment` и наоборот.

Однако заметим, что, когда рисуется segment, значение переменной `addpenline` добавляется к используемому перу `pen`. Значение по умолчанию этой переменной равно `squarecap` для правильного оформления концов. Отсюда следует, что код `draw(a_segment, dotted);` не будет производить пунктирный сегмент.

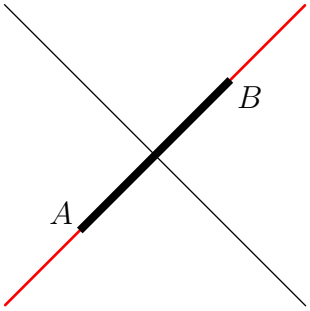
Есть три решения для обхода этой проблемы:

1. применить код `draw(a_segment,roundcap+dotted);` вместо кода `draw(a_segment, dotted);`
2. присвоить переменной `addpenline` значение `nullpen`
3. сообщить автору geometry.asy, что вы не согласны со значением по умолчанию для `addpenline`;

Наконец, подобно тому, как мы можем переходить между типами `point` и `mass`, объекты типа `line` и `segment` также могут быть конвертированы из одного в другой, например при помощи записи кода:

- `segment s=an_obj_line;`
- `draw(segment(an_obj_line));`
- `draw(line(an_obj_segment));`

Следующий пример является тому иллюстрацией.




```
import geometry;
size(4cm,0);
point A=SW, B=NE;
label("$A$", A, NW); label("$B$", B, SE);
line l=line(A,B);
draw(l, bp+red);

segment s=l;
draw(s, linewidth(3bp));
draw(line(rotate(90,midpoint(s))*s));
draw(box(2*A,2*B), invisible);
```

Помимо подпрограмм, определенных для линейных объектов, у нас также есть некоторые специальные процедуры, которые работают с объектом `segment`:

- `segment segment(point A, point B)`

  Возвращает сегмент прямой линии с концами A и B.

- `point midpoint(segment s)`

  Возвращает середину сегмента s

- `line bisector(segment s, real angle=0)`

  Возвращает срединный перпендикуляр к сегменту `s` и углом angle.

- `line[] complementary(explicit segment s)`

  Возвращает две полупрямые, которые содержат сегмент `s`, с конечными точками `s.A` и `s.B`.

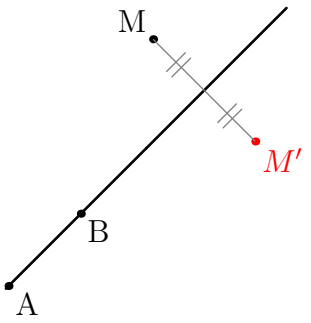
# 7 Аффинные преобразования (Часть 2)

Некоторые преобразования, которые определяются для точек в разделе *Аффинные преобразования (Часть 1)* могут быть также определены для прямых.

- `transform reflect(line l)`

  Возвращает симметрию относительно прямой `l`.

```
import geometry;
size(5cm,0);
point A=origin, B=NE, M=2*B+N;
dot("A", A, I*unit(A-B)); dot("B", B, I*unit(A-B));
line AB=line(A,B);
draw(AB, linewidth(bp));
transform reflect=reflect(AB);
point Mp=reflect*M;
dot("M",M, unit(M-Mp)); dot("M'", Mp,unit(Mp-M), red);
draw(segment(M,Mp), grey,StickIntervalMarker(2,2,grey));
```

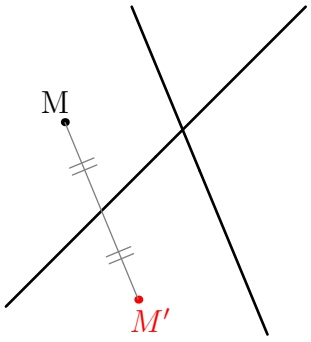



- `transform reflect(line l1, line l2, bool safe=false)`

  Возвращает симметрию (косую) относительно прямой `l1` в направлении прямой `l2`.

  Если значение `safe` есть `true` и если прямые `l1` и `l2` параллельны, то процедура возвращает тождественное отображение.

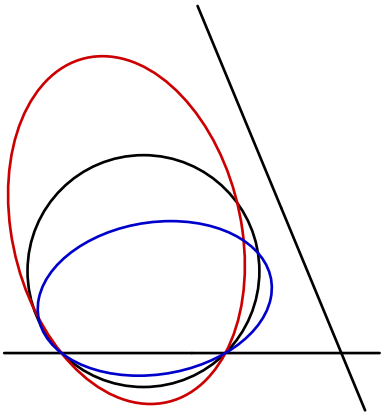
```
import geometry;
size(4.5cm,0);
line AB=line(origin, NE), CD=line(2*NE+N, 2*NE+SE);
draw(AB, linewidth(bp)); draw(CD, linewidth(bp));
transform reflect=reflect(AB,CD);

point M=1.75*NE+0.5N, Mp=reflect*M;
dot("M",M, unit(M-Mp)); dot("M'", Mp, unit(Mp-M), red);
draw(segment(M,Mp), grey, StickIntervalMarker(2,2,grey));
draw(box((1,1), (2.2,2.2)), invisible);
```



- `transform scale(real k, line l1, line l2, bool safe=false)`

  Возвращает аффинное преобразование с коэффициентом `k` к оси `l1` и направлением `l2` (см. определение в Разделе 5.1, такое аффинное преобразование называется "affinity"(«косое сжатие»)).

  Если значение `safe` есть `true` и если прямые `l1` и `l2` параллельны, то процедура возвращает тождественное отображение.



```
import geometry;
size(6.5cm,0);
pen bpp=linewidth(bp);
line AB=line(origin, E), CD=line(2*NE+N,
                                 2*NE+SE);
draw(AB, bpp); draw(CD, bpp);
transform dilatation=scale(1.5,AB,CD);
path cle=shift(NE)*unitcircle;
draw(cle,bpp);
draw(dilatation*cle, 0.8*red+bpp);
draw(inverse(dilatation)*cle, 0.8*blue+bpp);
draw(box((-0.5,-0.5), (2.75,3)), invisible);
```

- `transform projection(line l)`

  Возвращает ортогональную проекцию на прямую `l`.

- `transform projection(line l1, line l2, bool safe=false)`

  Возвращает параллельную проекцию вдоль прямой `l2` на прямую `l`.

  Если значение `safe` есть `true` и если прямые `l1` и `l2` параллельны, то обычно возвращает тождественное отображение.

- `transform vprojection(line l, bool safe=false)`

  Возвращает проекцию на `l` вдоль оси `y`.

  Это равносильно `projection(l,line(origin,point(defaultcoordsys,S)),safe)`.

  Если значение `safe` есть `true` и если прямая `l` параллельна оси `y`, то возвращает тождественное отображение.

- `transform hprojection(line l, bool safe=false)`

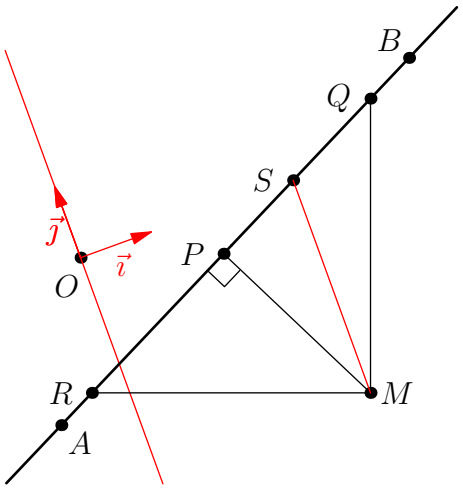
  Возвращает проекцию на `l` вдоль оси `x`.

  Это равносильно `projection(l,line(origin,point(defaultcoordsys,E)),safe)`.

  Если значение `safe` есть `true` и если прямая `l` параллельна оси `x`, то возвращает тождественное отображение.

Предлагаемый рисунок иллюстрирует различные виды проекций.

```
import geometry;
size(7.5cm,0); dotfactor*=1.5;
currentcoordsys=rotate(20)*defaultcoordsys;
show(currentcoordsys, xpen=invisible, ypen=red);
point A=(-1,-3), B=(5,2);point M=(3,-3);
line l1=line(A,B); draw(l1, linewidth(bp));
dot("$A$", A, SE); dot("$B$", B, NW);
point P=projection(l1)*M; dot("$P$", P, 2W);
markrightangle(l1.A, P, M);draw(M--P);
point Q=vprojection(l1)*M;dot("$Q$", Q, 2W);
draw(M--Q);dot("$M$", M);
point R=hprojection(l1)*M;
dot("$R$", R, 2W); draw(M--R);
point S=projection(l1,line((0,0),(0,1)))*M;
dot("$S$", S, 2W); draw(M--S, red);
draw(box((-1,-4),(5,5)), invisible);
```



# 8 Коники

## 8.1 Тип conic

### 8.1.1 Описание

Пакет *geometry.asy* определяет тип `conic` для создания конических сечений. Хотя его вполне можно использовать как таковой, он скорее предназначен для работы внутри расширения. Мы предпочтем использовать непосредственно типы `circle`, `ellipse`, `parabola` и `hyperbola`, их мы опишем позже.

Посмотрите на структуру типа и определенные в ней аргументы:

```
struct conic { real e, p, h; point F; line D; }
```

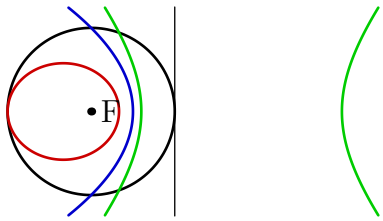
- `e` — экцентриситет;
- `F` есть фокус и `D` ассоциированная с ним директриса;
- `h` — расстояние от `F` до `D`;
- `p` — фокальный параметр, определенный равенством p=he.

Вот две основных процедуры, определяющие коники:

1. `conic conic(point F, line l, real e)`

   Возвращает конику, определенную фокусом F, ассоциированной с ним директрисой l и эксцентриситетом e;

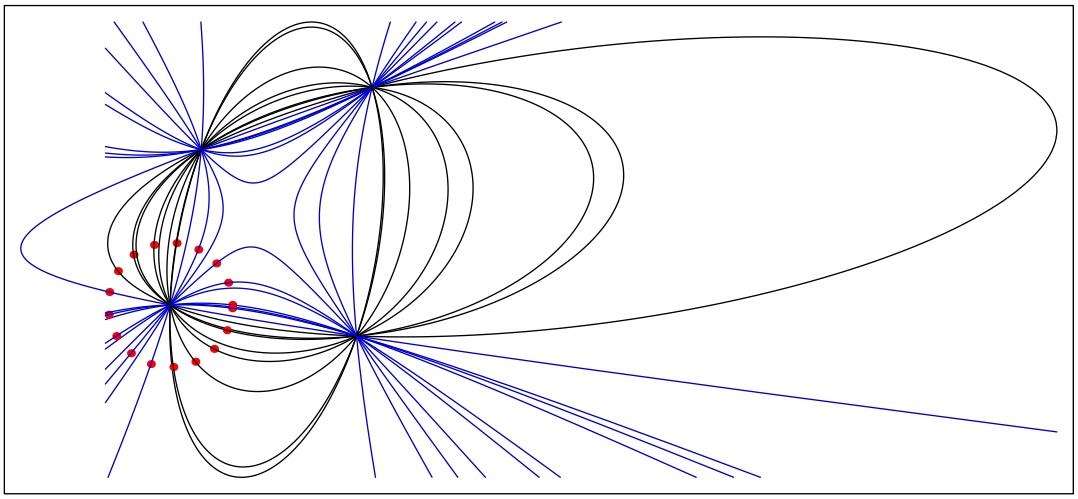
   Вот пример использования:




```
import geometry;
size(6cm,0);
point F=(0,0); dot("F", F);
line l=line((1,0),(1,1));
draw(l);
pen[] p=new pen[] {black,red,blue,green};
for (int i=0; i < 4; ++i) {
conic co=conic(F,l,0.5*i);
draw(co, bp+0.8*p[i]);
}
draw(box((-1,-1.25), (3.5,1.25)), invisible);
```

2. `conic conic(point M1, point M2, point M3, point M4, point M5)`

   Возвращает невырожденную конику, определяемую пятью точками `M1, M2, M3, M4, M5`.



```
import geometry;//Код к рисунку
size(13cm,0);
point B=(1.75,3), C=(-1,2), D=(-1.5,-0.5), F=(1.5,-1);
for (int i=0; i < 360; i += 21) {
point A=shift(D)*dir(i);
dot(A,red);
conic co=conic(A,B,C,D,F);
draw(co, co.e < 1 ? black : 0.8*blue);
}
shipout(bbox(2mm));
```

Следует отметить, что также можно определить конику по ее уравнению в определенной системе координат (смотрите в подразделе *Уравнения коник*) и что другие пути для определения коник осуществляются подпрограммами, относящимися к определенным типам коник, которые могут быть конвертированы в тип `conic`, (смотрите в подразделе *Коники, приведение типов*).

### 8.1.2 Основные процедуры

Следующие функции могут замещать объект типа `conic` одним из типов `circle`, `ellipse`, `parabola` или `hyperbola`, за исключением случаев, когда ключевое слово `explicit` предшествует типу `conic` в определении подпрограммы. Следует отметить, что, в дополнение к функциям, описанным в этом разделе, существуют процедуры, возвращающие абсциссу точки, принадлежащей объекту типа `conic`.

- `conic changecoordsys(coordsys R, conic co)`

  Возвращает конику `co` относительно координатной системы `R`.

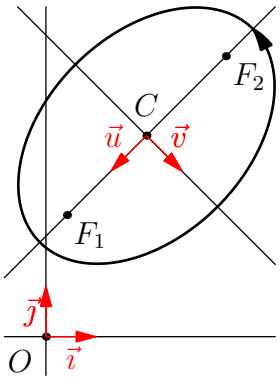
- `coordsys coordsys(conic co)`

  Возвращает систему координат, в которой определена коника `co`

- `coordsys canonicalcartesiansystem(explicit conic co)`

  Возвращает каноническую систему координат коники `co`. Существуют также процедуры:
  `canonicalcartesiansystem(ellipse)`, `canonicalcartesiansystem(hyperbola)`,
  `canonicalcartesiansystem(parabola)`.

  Следующий пример служит иллюстрацией в случае эллипса.



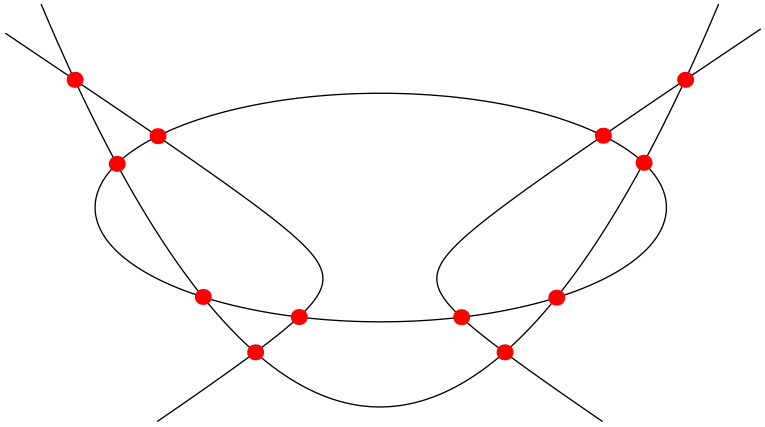

```
import geometry;
size(4cm,0);
show(defaultcoordsys);
ellipse el=ellipse((2,4),3,2,45);
dot("$F_1$", el.F1, dir(-45));
dot("$F_2$", el.F2, dir(-45));
draw(el, linewidth(bp), Arrow(3mm));
show("$C$", "$\vec{u}$" ,"$\vec{v}$",
canonicalcartesiansystem(el));
shipout(bbox(2mm,white));
```

- `int conicnodesnumber(conic co, real angle1, real angle2, bool dir=CCW)`

  Возвращает число узлов используемых для преобразования коники `co` в путь между углами `angle1` и `angle2` в заданном направлени `dir`.

- `point[] intersectionpoints(conic co1, conic co2)`

  Возвращает в форме массива точки пересечения двух коник `co1` и `co2`. (см рисунок)



```
import geometry; size(10cm); conic co[];//Код к рис.8.2
co[0]=conic((-4.58,1.25), line((-5.45545,1.25), (-5.45545,2.12287)), 0.9165);
draw(co[0]);
co[1]=conic((0,-1),line((0,-3.5),(-1,-3.5)),1); draw(co[1]);
co[2]=conic((-1.2,0), line((-5/6,0),(-5/6,-1)),1.2); draw(co[2]);
dotfactor *= 2;
for (int i=0; i < 3; ++i)
for (int j=i+1; j < 3; ++j)
dot(intersectionpoints(co[i],co[j]), red);
addMargins(lmargin=10mm,bmargin=10mm);
```

- `point[] intersectionpoints(line l, conic co)`

  Возвращает в форме массива точки пересечения прямой `l` с коникой `co`.

  Программа `intersectionpoints(conic,line)` также определена.

- `point[] intersectionpoints(triangle t, conic co, bool extended=false)`

  Возвращает в форме массива точки пересечения треугольника `t` с коникой `co`. Если опция `extended` принимает значение `true`, стороны треугольника рассматриваются как прямые; (Смотри раздел *Треугольники*). Процедура `intersectionpoints(conic,triangle,bool)` также определена.

### 8.1.3 Операторы

Как в предыдущих процедурах, операторы, описанные здесь, можно использовать для замены объекта типа `conic` на типы `circle, ellipse, parabola` или `hyperbola`.

- `bool operator @(point M, conic co)`

  Выдает код `point @ conic`.

  Возвращает `true` если и только если точка `M` принадлежит объекту `co`.

- `conic operator *(transform t, conic co)`

  Выдает код `transform*conic`.

- `conic operator +(conic co, explicit point M)`

  Выдает код `conic+point`.

  Возвращает изображение коники `co` при переносе на вектор $\overrightarrow{OM}$.

  Процедура `-(conic,explicit point)` также определена.

- `conic operator +(conic co, explicit pair m)`

  Выдает код `conic+pair`

  Возвращает изображение коники `co` при переносе на вектор $\overrightarrow{Om}$; `m` представляет координаты точки, определенные в той системе координат, в которой определена коника. Процедура `-(conic,explicit pair)` также определена.

- `conic operator +(conic co, explicit vector u)`

  Выдает код `conic+vector`.

  Возвращает изображение коники `co` при переносе на вектор $\vec{u}$.

  Процедура `-(conic,explicit vector)` также определена.

### 8.1.4 Уравнения коник

Тип `bqe`, или *Bivariate Quadratic Equation*, позволяет создать объект, представляющий собой уравнение коники в данной системе координат. Его структура такова:

```
struct bqe
        { real[] a; coordsys coordsys; } где
```

- `a` есть упорядоченный набор из шести коэффициентов уравнения коники, заданного в форме

$$a[0]x^2 + a[1]xy + a[2]y^2 + a[3]x + a[4]y + a[5] = 0$$

- `coordsys` есть система координат, в которой определено это уравнение.

Приведем список процедур, использующих объекты типа `bqe`:

- `bqe bqe(coordsys R=currentcoordsys, real a,real b,real c, real d, real e, real f)`

  Возвращает объект типа `bqe`, представленный уравнением $ax^2+bxy+cy^2+dx+ey+f = 0$ относительно системы координат $R$.

- `bqe changecoordsys(coordsys R, bqe bqe)`

  Возвращает объект типа `bqe` относительно системы координат `R` и представляющий ту же конику, какая представлена параметром `bqe`. Эта процедура позволяет изменить систему координат в *bivariate quadratic equation*.

- `bqe bqe(point M1, point M2, point M3, point M4, point M5)`

  Возвращает уравнение коники, проходящей через пять точек `M1, M2, M3, M4` и `M5`. Если точки определены относительно одной и той же системы координат, возвращает уравнение относительно этой же системы координат; иначе — уравнение относительно системы координат по умолчанию `defaultcoordsys`

- `string conictype(bqe bqe)`

  Возвращает тип коники, представленный `bqe`. Возможны возвращаемые значения `degenerated, ellipse, parabola` и `hyperbola`.

- `bqe equation(explicit conic co)`

  Возвращает, в виде объекта типа `bqe`, уравнение коники `co`.

  Ещё есть процедуры `equation(ellispe)`, `equation(parabola)` и `equation(hyperbola)`.

- `bqe canonical(bqe bqe)`

  Возвращает уравнение коники, представленной `bqe` относительно канонической системы координат рассматриваемой коники.

- `conic conic(bqe bqe)`

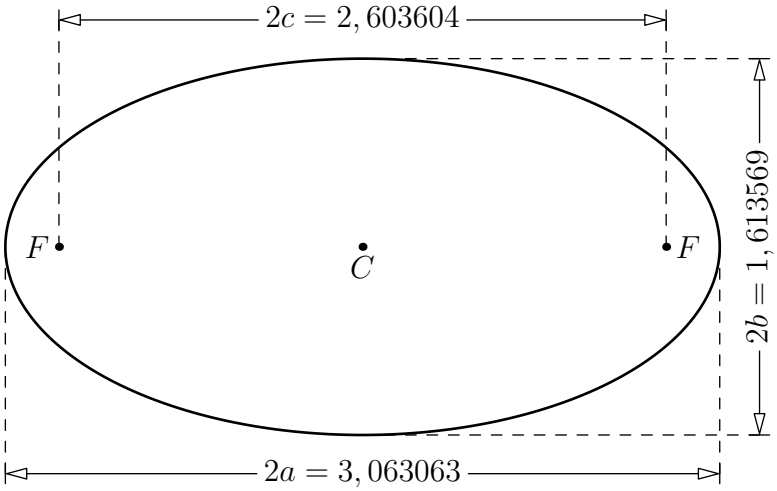
  Возвращает конику, представленную `bqe`.

### 8.1.5 Коники, приведение типов

Как уже отмечалось в предыдущих разделах, тип conic может инициализировать объект, представляющий конику. Однако его возможно, и часто рекомендуется, преобразовать в объект, представляющий любую конику определенного типа, для того, чтобы использовать специфические свойства и подпрограммы.

Конкретным видом коники является `circle` — частный случай типов `ellipse, parabola` и `hyperbola`; эти типы описаны ниже.

Так, в примере, приведенном ниже, коника `co` определяется фокусом и соответствующей директрисой с эксцентриситетом меньше единицы. Поскольку это коника представляет собой эллипс, можно присвоить переменной типа `ellipse` переменную `co` для того, чтобы извлечь его размеры.




```
import geometry;
size(10cm);
point F=(-1,0); line D=line(N,S);
conic co=conic(F, D, 0.85); dot("$F$", F); draw(co, linewidth(bp));
ellipse el=co; dot("$C$", el.C, S);
distance(format("$2c=%f$", el.c),el.F1, el.F2, 3cm, joinpen=dashed);
distance(format("$2a=%f$", el.a), relpoint(el,0), relpoint(el,0.5), 3cm,
joinpen=dashed);
distance(format("$2b=%f$", el.b), relpoint(el,0.25), relpoint(el,0.75), 5.25cm,
joinpen=dashed);
dot("$F'$", el.F2, W);
```

С точки зрения внутреннего функционирования пакета *geometry.asy* некоторые процедуры, которые применяются к объекту типа `conic` по сути эквивалентны процедурам для конкретных коник; это позволяет оптимизировать некоторые расчеты. И, наоборот, процедуры для конкретного типа коник используют базовые процедуры для любой коники.

Конкретные виды коник описаны ниже.

## 8.2 Окружности

### 8.2.1 Основные процедуры

Кроме процедур для объектов типа `conic` есть некоторые другие процедуры, определенные для объекта типа `circle`:

- `circle circle(explicit point C, real r)`

  Возвращает окружность радиуса `r` с центром `C`.

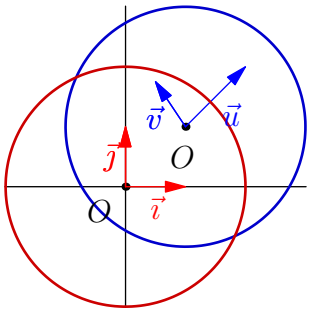
  Заметим, что программа `circle circle(pair C, real r)` переопределяет и возвращает объект типа `circle`, центр которой представляют координаты точки в существующей системе координат `currentcoordsys`.

Следующий пример показывает разницу между кодом `circle((0,0),R)`,

который определяет синюю окружность в текущей системе координат и кодом

`circle(point(defaultcoordsys,(0,0)), R)`, который определяет красную окружность в системе координат по умолчанию.

Очевидно, если переменная `currentcoordsys` не изменяется, коды эквивалентны.




```
import geometry;
size(6cm,0);
currentcoordsys=cartesiansystem((1,1), i=(1,1),
                                  j=(-0.5,.75));
show("$O'$","$\vec{u}$", "$\vec{v}$",
   currentcoordsys, ipen=blue, xpen=invisible);
show(defaultcoordsys);
point O=(0,0);
real R=2.0;
circle C=circle(O, R);
draw(C, bp+0.8*blue);
circle Cp=circle(point(defaultcoordsys,(0,0)),R);
draw(Cp, bp+0.8*red);
```

- `circle circle(point A, point B)`

  Возвращает окружность диаметром `AB`

- `circle circle(point A, point B, point C)`

  Возвращает окружность, проходящую через три точки `A`, `B` и `C`. Равносильным является код `circle circumcircle(point A, point B, point C)`.

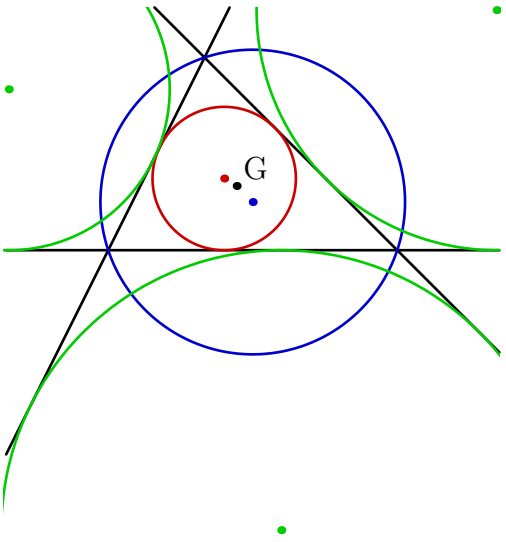
- `circle incircle(point A, point B, point C)`

  Возвращает окружность, вписанную в треугольник `ABC`.

- `circle excircle(point A, point B, point C)`

  Возвращает окружность, вневписанную около треугольника `ABC`, касающуюся стороны `(AB)`.

В примере, приведенном ниже, мы видим использование программы `clipdraw`, которая рисует путь, ограничивая размер окончательного рисунка.



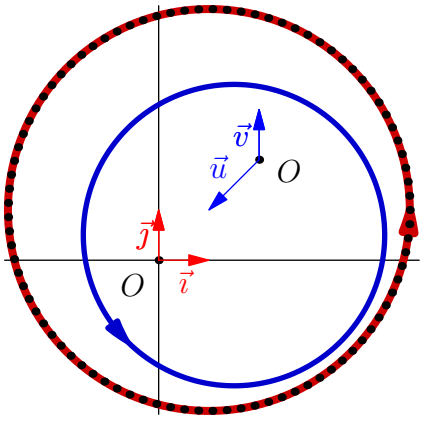

```
import geometry;
size(7cm);
green=0.8green; blue=0.8blue; red=0.8red;
pen bpp=linewidth(bp);
point A=(-1,0), B=(2,0), C=(0,2);
draw(line(A,B),bpp); draw(line(A,C),bpp);
draw(line(B,C),bpp);
circle cc=circle(A,B,C);
draw(cc, bp+blue); dot(cc.C, blue);
circle ic=incircle(A,B,C);
draw(ic, bp+red); dot(ic.C, red);
circle ec=excircle(A,B,C);
clipdraw(ec, bp+green); dot(ec.C, green);
ec=excircle(A,C,B);
clipdraw(ec, bp+green); dot(ec.C, green);
ec=excircle(C,B,A);
clipdraw(ec, bp+green); dot(ec.C, green);
dot("G", centroid(A,B,C), NE);
```

Используя специальные процедуры для геометрии треугольника, можно достичь того же результата более элегантным способом, см. раздел *Треугольники*.

### 8.2.2 От типа circle к типу path

Выбор объекта типа `circle` как типа `path` осуществляется в соответствии со следующими правилами:

- путь циклический, проходится против часовой стрелки.
- первый узел пути возвращается процедурой `pair point(path g, real t)` при `t=0`, является точкой пересечения окружности с полупрямой, проходящей через центр параллельно оси системы координат, в которой определена окружность;
- количество точек пути зависит от радиуса окружности; оно рассчитывается программой `int circlenodesnumber(real r)`, которая зависит от переменной `circlenodesnumberfactor`.




```
import geometry;
size(6cm,0);
currentcoordsys=cartesiansystem((2,2), i=(-1,-1),
j=(0,1));
show("$O'$","$\vec{u}$", "$\vec{v}$",
currentcoordsys, ipen=blue, xpen=invisible);
show(defaultcoordsys);
point O1=(.5,-1);//вставлено дополнительно
circle C=circle(O1,3.0);//вместо следующей строки
circle C=circle((0.5,-1), 3);//неверно
draw(C, 2bp+0.8*blue, Arrow(3mm));
circle Cp=circle(point(defaultcoordsys,(1,1)),4);
draw(Cp, dotsize()+0.8*red, Arrow(3mm));
dot((path)Cp);
```

### 8.2.3 Операторы

Помимо операторов, применяющихся к объектам типа `conic`, приведем список специальных операторов, определенных для объектов типа `circle`.

- `circle operator *(real x, explicit circle c)`

  Создает код `real*circle`. Возвращает окружность с центром `c` и радиусом `x` time that of c. Также определен оператор `circle operator /(explicit circle c, real x)`.

- `real operator ^(point M, explicit circle c)`

  Создает код `point^circle`. Возвращает степень точки `M` относительно окружности `c`.

- `bool operator @(point M, explicit circle c)`

  Создает код `point @ circle`. Возвращает `true` если и только если точка `M` расположена вне `c`.

- `ellipse operator cast(circle c)`

  Позволяет перейти от `circle` к `ellipse`. Так же определен переход от `ellipse` к `circle`.

  Обратите внимание, что нет оператора `operator *(transform t, circle c)`; однако после выбора оператора `ellipse operator *(transform t, ellipse el)` можно использовать следующий код: `transform*circle`.

  Таким образом код `scale(2)*circle` возвращает объект типа `ellipse` и можно написать код `circle=scale(2)*circle`, в то время как код `circle=xscale(2)*circle` выдает ошибку.

### 8.2.4 Другие процедуры

Помимо процедур, применяемых к объектам типа `conic`, приведем список конкретных процедур для объектов типа `circle`.

- `point radicalcenter(circle c1, circle c2)`

  Возвращает проекцию радикальной оси двух окружностей, `c1` и `c2`. Системой координат, в которой определена возвращаемая точка, является система окружности `c1`.

- `point radicalcenter(circle c1, circle c2, circle c3)`

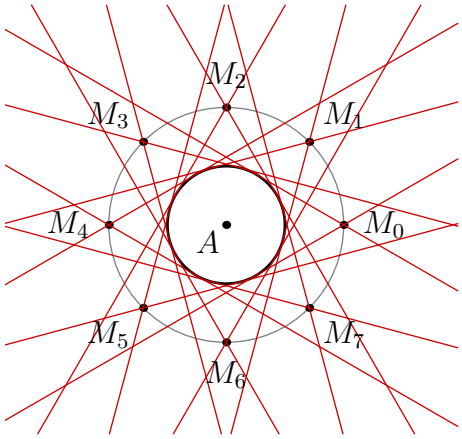
  Возвращает радикальный центр трех окружностей.

- `line radicalline(circle c1, circle c2)`

  Возвращает радикальную ось двух окружностей.

- `line[] tangents(circle c, point M)`

  Возвращает все касательные к окружности `c`, проходящие через точку `M`.



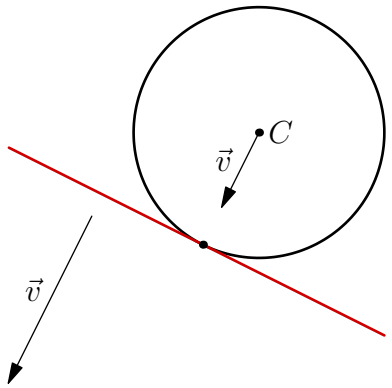

```
import geometry;
size(7.5cm,0);
point A=(2.5,-1);
dot("$A$", A, SW);
circle C=circle(A,1);
draw(C, linewidth(bp));
path Cp=shift(A)*scale(2)*unitcircle;
draw(Cp, grey);
for (int i=0; i < 360; i+=45) {
point M=relpoint(Cp, i/360);
dot(format("$M_{%f}$", i/45), M,
                          2*unit(M-A));
draw(tangents(C, M), 0.8*red);
}
addMargins(10mm,10mm);
```

- `line tangent(circle c, point M)` Возвращает касательную к окружности `c` в точке пересечения `c` с лучом, выходящем из центра окружности и проходящим через `M`. Точку касания можно получить процедурой `point(circle c, point M)`.

- `line tangent(circle c, explicit vector v)`

  Возвращает касательную к окружности `c` в точке пересечения `c` с лучом, выходящим из центра окружности в направлении вектора $\vec{v}$.

  Точка касательной может быть получен процедурой `point(circle c, vector v)`.




```
import geometry;
size(5cm);
circle cle=circle((point)(2,1),1.5);
draw(cle, linewidth(bp));
dot("$C$", cle.C);
vector v=(-1,-2);
show("$\vec{v}$",v);
line tgt=tangent(cle,v);
draw(tgt, bp+0.8*red);
draw("$\vec{v'}$",cle.C--(cle.C+tgt.v),Arrow);
dot(point(cle,v));
```

- `line tangent(circle c, abscissa x)`

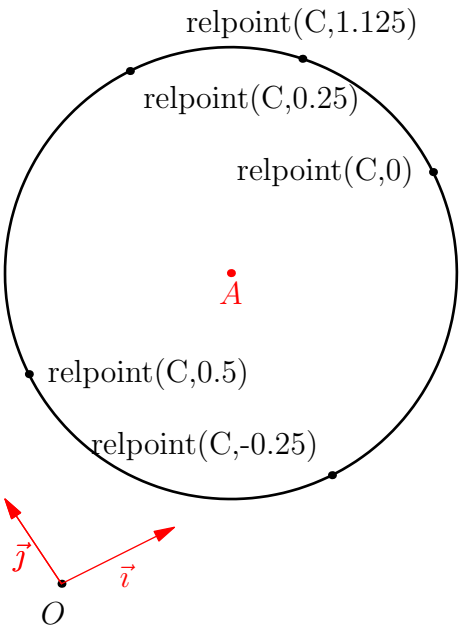
  Возвращает касательную к окружности `c` в точке, абсцисса которой равна `x`.

- `point point(explicit circle c, real x)`

  Возвращает точку на окружности `c`, которую можно получить как точку, возвращаемую кодом `point((path)c,x)`.

- `point relpoint(explicit circle c, real x)`

  Возвращает точку на окружности `c`, соответствующую `x` time “the perimeter of c”.

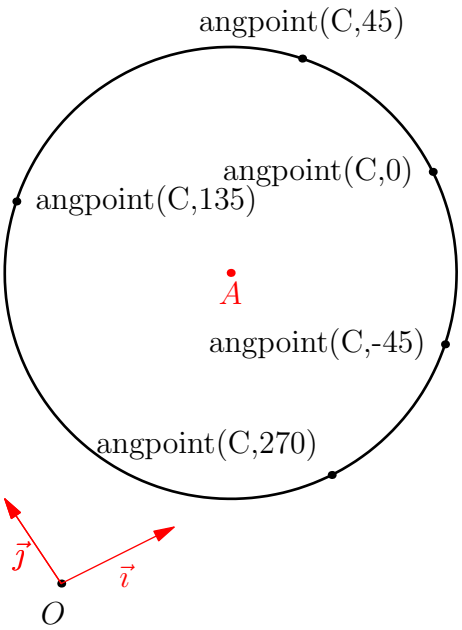



```
import geometry;
size(6cm,0);
currentcoordsys=cartesiansystem((0,0),
            i=(1,0.5), j=(-0.5,.75));
show(currentcoordsys, xpen=invisible);
point A=(2.5,2);
dot("$A$", A, S, red);
real R=2;
circle C=circle(A,R);
draw(C, linewidth(bp));
dot("relpoint(C,0)", relpoint(C,0), 2W);
dot("relpoint(C,0.25)",
                relpoint(C,0.25), 2SE);
dot("relpoint(C,0.5)",
                 relpoint(C,0.5), 2E);
dot("relpoint(C,-0.25)",
               relpoint(C, -0.25), 2NW);
dot("relpoint(C,1.125)",
                relpoint(C, 1.125), 2N);
```

- `point angpoint(explicit circle c, real x)`

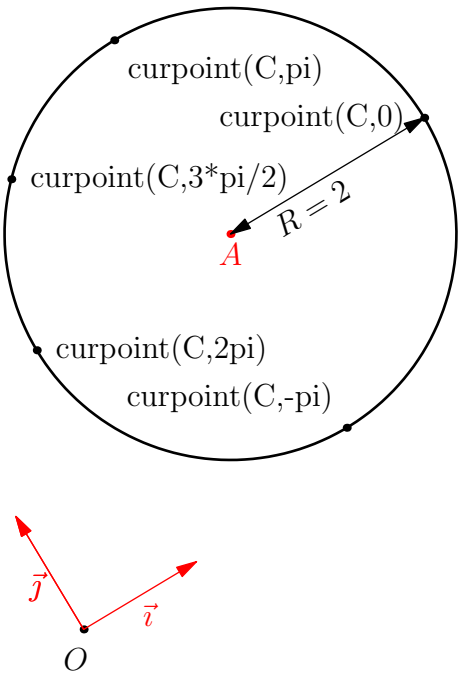
  Возвращает точку на окружности `c`, соответствующую углу в `x` градусов.

```
import geometry;
size(6cm,0);
currentcoordsys=cartesiansystem((0,0),
           i=(1,0.5), j=(-0.5,.75));
show(currentcoordsys, xpen=invisible);
point A=(2.5,2); dot("$A$", A, S, red);
real R=2;
circle C=circle(A,R);
draw(C, linewidth(bp));
dot("angpoint(C,0)",
                 angpoint(C,0), 2W);
dot("angpoint(C,45)",
                angpoint(C,45), 2N);
dot("angpoint(C,135)",
               angpoint(C,135), 2E);
dot("angpoint(C,270)",
              angpoint(C, 270), 2NW);
dot("angpoint(C,-45)",
               angpoint(C, -45), 2W);
```




- `point curpoint(explicit circle c, real x)`

  Возвращает точку окружности $c$, криволинейная абсцисса которой равна $x$.







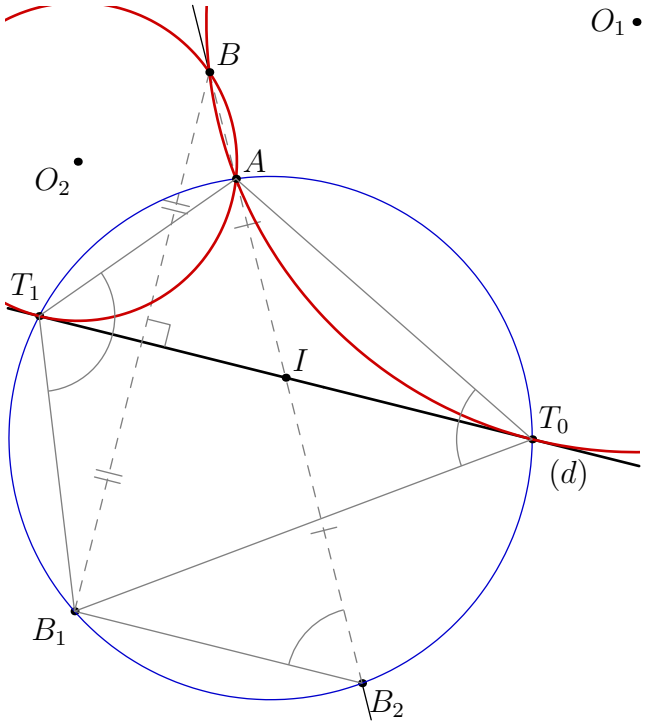
```
import geometry;
size(6cm,0); real R=2;
currentcoordsys=cartesiansystem((0,0), i=(1,0.5),
j=(-0.5,1));
show(currentcoordsys, xpen=invisible);
point A=(2.5,2); dot("$A$", A, S, red);
circle C=circle(A,R); draw(C, linewidth(bp));
draw(rotate(A-point(C,0))*("$R="+(string)R+"$"),
A--point(C,0), S, Arrows);
dot("curpoint(C,0)", curpoint(C,0), 2W);
dot("curpoint(C,pi)", curpoint(C,pi), 2SE);
dot("curpoint(C,3*pi/2)", curpoint(C,3*pi/2), 2E);
dot("curpoint(C,-pi)", curpoint(C, -pi), 2NW);
dot("curpoint(C,2pi)", curpoint(C, 2*pi), 2E);
```

Есть другие подпрограммы, определённые для объекта типа `circle`, они доступны через процедуры, использующих тип `ellipse`.

В заключение этого раздела отметим, что можно использовать объект типа `circle` для инверсии. См. раздел *Инверсии* для получения дополнительной информации.

Вот некоторые примеры использования процедур, описанных ранее:

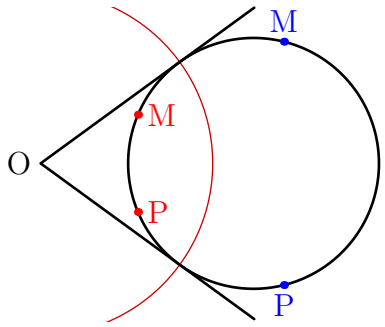
- Построим две окружности, проходящие через точки `A` и `B` и касающиеся данной прямой `d`.




Далее предлагаем код, который определяет этот рисунок.

```
import geometry;
size(10cm,0);
pen bpp=linewidth(bp);
line l=line(origin,(1,-0.25)); draw("$(d)$", l, bpp);
point A=(1,1.5), B=(0.75,2.5);
line AB=line(A,B);
point B1=reflect(l)*B, I=intersectionpoint(l,AB), B2=rotate(180,I)*B;
dot("$I$", I, NE); dot("$B_1$", B1, SW); dot("$B_2$", B2, SE);
draw(B--B1, grey+dashed, StickIntervalMarker(2,2,grey));
markrightangle(B,midpoint(B--B1),I, grey);
draw(B--B2, grey+dashed, StickIntervalMarker(2,1,grey));
draw(complementary(segment(B,B2)));
circle C=circle(A,B1,B2); draw(C, 0.8*blue);
point[] T=intersectionpoints(l,C);
dot("$T_0$",T[0], NE); dot("$T_1$",T[1], N+NW);
circle C1=circle(A,B,T[0]), C2=circle(A,B,T[1]);
clipdraw(C1, bpp+0.8*red); clipdraw(C2, bpp+0.8*red);
dot("$O_1$", C1.C, W); dot("$O_2$", C2.C, SW); dot("$A$",
                                                A, NE); dot("$B$", B, NE);
draw(A--T[0]--B1, grey); markangle(A,T[0],B1, grey);
draw(A--T[1]--B1, grey); markangle(B1,T[1],A, grey);
draw(B2--B1, grey); markangle(A,B2,B1, grey);
```

- Вот построение окружности, которая ортогональна окружности инверсии и которая проходит через две точки и инверсные к ним (если они лежат на одной окружности).




```
import geometry;
size(5cm,0); currentpen=linewidth(bp);
point O=origin, M=(2,1), P=(2,-1);
dot("O", O, W);
inversion t=inversion(2,O);
point Mp=t*M, Pt=t*P;
circle C=circle(M,P,Mp); draw(C);
dot("M", M, N, blue); dot("P", P, S, blue);
dot("M'", Mp, red); dot("P'", Pt, red);
circle Ct=circle(t); clipdraw(Ct, 0.8*red);
point[] T=intersectionpoints(C,Ct);
draw(line(O,false,T[0]));
draw(line(O,false,T[1]));
```

- Даны три окружности: $\mathscr{C}_1$, $\mathscr{C}_2$ и $\mathscr{C}_3$, такие что $r_3 < r_1$ и $r_3 < r_2$. Построить окружность, одновременно касающуюся этих трех данных окружностей.

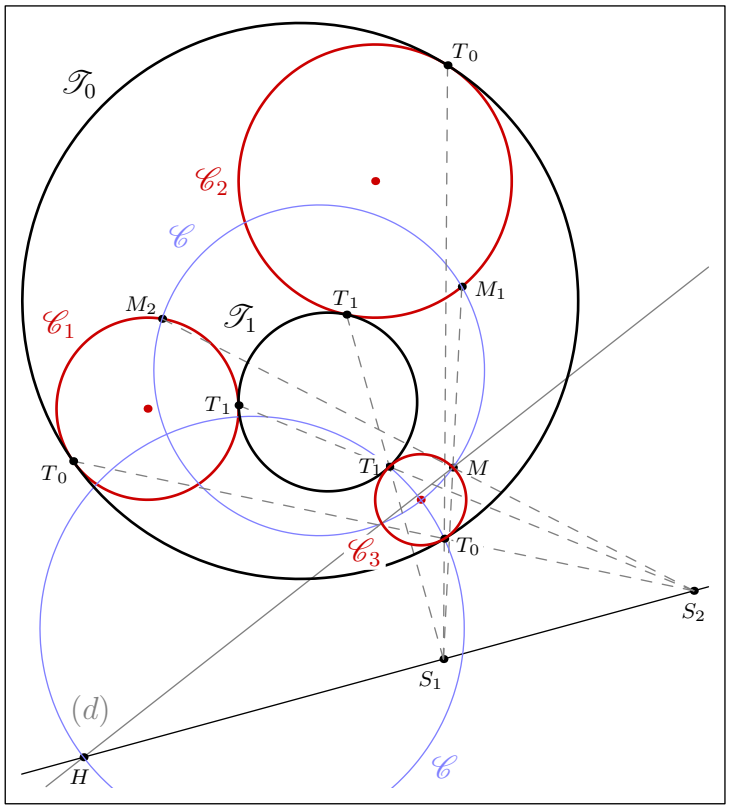
  Принцип построения показан на рисунке ниже; он выглядит следующим образом:

  - Если $S_1$ и $S_2$ являются инверсиями с положительным радиусом, которые преобразуют $\mathscr{C}_2$ в $\mathscr{C}_3$ и $\mathscr{C}_1$ в $\mathscr{C}_3$ соответственно;
  - Учитывая, что точка $M$ на окружности $\mathscr{C}_3$ является образом точек $M_1$ и $M_2$ при $S_1$ и $S_2$ соответственно;
  - Если $\mathscr{C}$ есть окружность, проходящая $M, M_1$ и $M_2$ ;
  - Радикальная ось $(d)$ окружностей $\mathscr{C}_3$ и $\mathscr{C}$ пересекает прямую, проходящей через центры инверсий $(S_1S_2)$ в точке $H$; Если $\mathscr{C}'$ есть окружность с диаметром $[HO_3]$, где $O_3$ является центром $\mathscr{C}_3$;

  тогда:

  - окружность $\mathscr{C}'$ пересекает $\mathscr{C}_3$ в двух точках $T_0$ и $T_1$ ;
  - окружность, проходящая через $T_0$ и через ее образы $T_0'$ и $T_0''$ при инверсиях $T_0$ относительно $S_1$ и $S_2$ соответственно, является одним из решений;

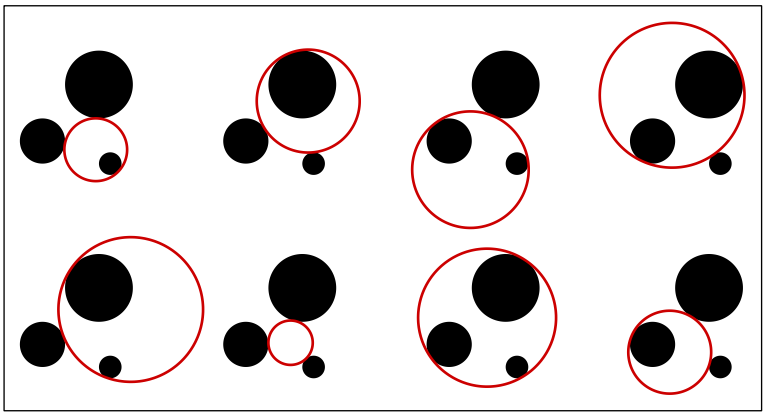
– окружность, проходящая через $T_1$ и через ее образы $T_1'$ и $T_1''$ при инверсиях $T_1$ относительно $S_1$ и $S_2$ соответственно, является другим решением;




```
import geometry;
size(8cm,0); usepackage("mathrsfs");
currentpen=fontsize(8); pen bpp=linewidth(bp);
circle C1=circle((0,0),2), C2=circle((5,5), 3), C3=circle((6,-2),1);
draw(Label("$\mathscr{C}_1$",Relative(0.375)), C1, bp+0.8*red);
draw("$\mathscr{C}_2$", C2, bp+0.8*red);
dot(C1.C, 0.8*red); dot(C2.C, 0.8*red); dot(C3.C, 0.8*red);
inversion S1=inversion(C2,C3), S2=inversion(C1,C3);
dot("$S_1$", S1.C, 2S+W); dot("$S_2$", S2.C, 2S);
line cl=line(S1.C,S2.C); draw(cl);
point M=relpoint(C3,0.125), M2=S2*M, M1=S1*M;
dot("$M$", M, 2*E); dot("$M_2$", M2, NW); dot("$M_1$", M1, 2*dir(-10));
draw(segment(S2.C,M2), dashed+grey); draw(segment(S1.C,M1), dashed+grey);
circle C=circle(M,M2,M1); draw(Label("$\mathscr{C}$",
Relative(0.375)), C, lightblue);
line L=radicalline(C,C3); draw("$(d)$", L, grey);
point H=intersectionpoint(L,cl); dot("$H$", H, 2*dir(260));
circle Cp=circle(H,C3.C);
clipdraw(Label("$\mathscr{C'}$", Relative(0.9)), Cp, lightblue);
point[] T=intersectionpoints(Cp,C3);
point[][] Tp= new point[][] {{S2*T[0], S1*T[0]},{S2*T[1], S1*T[1]}};
draw(S2.C--Tp[0][0], dashed+grey); draw(S1.C--Tp[0][1], dashed+grey);
draw(S2.C--Tp[1][0], dashed+grey); draw(S1.C--Tp[1][1], dashed+grey);
dot(Label("$T_0$",UnFill), T[0], 2*dir(-20));
dot(Label("$T_1$",UnFill), T[1], W);
dot("$T''_0$", Tp[0][0], SW); dot("$T'_0$", Tp[0][1], NE);
dot("$T''_1$", Tp[1][0], W); dot("$T'_1$", Tp[1][1], N);
draw(Label("$\mathscr{T}_0$", Relative(0.375)),
                              circle(T[0],Tp[0][0],Tp[0][1]), bpp);
```

```
draw(Label("$\mathscr{T}_1$", Relative(0.375)),
                                circle(T[1],Tp[1][0],Tp[1][1]), bpp);
draw(Label("$\mathscr{C}_3$",Relative(0.625),UnFill), C3, bp+0.8*red);
```

- Учитывая четыре возможных комбинации для инверсий $S_1$ и $S_2$ , мы получаем восемь случаев касания к трем данным окружностям:



```
import geometry;
size(10cm,0); int shx=18;
circle C1=circle((0,0),2), C2=circle((5,5), 3), C3=circle((6,-2),1);
picture disc;
fill(disc,(path)C1); fill(disc,(path)C2); fill(disc,(path)C3);
transform tv=shift(S), th=shift(E);
int k=0, l=0;
for (int i=0; i < 2 ; ++i)
for (int j=0; j < 2; ++j) {
picture[] tpic; tpic[0]=new picture; tpic[1]=new picture;
add(tpic[0], disc); add(tpic[1], disc);
inversion S1=inversion(C2,C3, sgnd(i-1)), S2=inversion(C1,C3, sgnd(j-1));
line cl=line(S1.C,S2.C);
point M=relpoint(C3,0.125), M2=S2*M, M1=S1*M;
circle C=circle(M,M2,M1);
line L=radicalline(C,C3);
point H=intersectionpoint(L,cl);
circle Cp=circle(H,C3.C);
point[] T=intersectionpoints(Cp,C3);
point[][] Tp= new point[][] {{S2*T[0], S1*T[0]},{S2*T[1], S1*T[1]}};
draw(tpic[0], circle(T[0],Tp[0][0],Tp[0][1]), bp+0.8*red);
draw(tpic[1], circle(T[1],Tp[1][0],Tp[1][1]), bp+0.8*red);
add(tv^(shx*(i+1))*th^(shx*(l))*tpic[0]);
l=(l+2)%4; ++k;
add(tv^(shx*(i+1))*th^(shx*(l+1))*tpic[1]);
}
```

## 8.3 Эллипсы

Тип `ellipse` никого не удивит; он позволяет создать объект, представляющий собой эллипс. Так как тип `circle` есть частный случай типа `ellipse`, можно создать окружность как эллипс нулевого эксцентриситета и, обратно, при обращении эксцентриситета эллипса в нуль получить окружность. Наконец, поскольку существует взаимно-однозначное соответствие между объектами типа `ellipse` и теми, которые имеют тип `conic` с эксцентриситетом строго меньше, чем 1, объекты типа `ellipse` наследуют набор процедур и операторов для типа `conic`.

### 8.3.1 Основные процедуры

Вот список других процедур, определенных для типа `ellipse`.

- `ellipse ellipse(point F1, point F2, real a)`

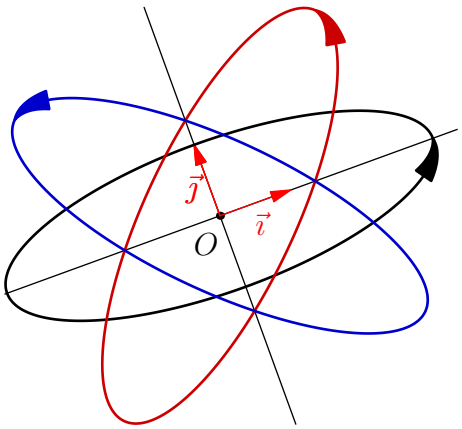
  Возвращает эллипс с фокусами `F1, F2` и большой полуосью `a`.

- `ellipse ellipse(point F1, point F2, point M)`

  Возвращает эллипс с фокусами `F1, F2`, проходящий через точку `M`.

- `ellipse ellipse(point C, real a, real b, real angle=0)`

  Возвращает эллипс с центром `C`, большой полуосью `a`, расположенной по направлению `dir(angle)` и малой полуосью `b`.




```
import geometry;
size(6cm);
currentcoordsys=rotate(20)*defaultcoordsys;
show(currentcoordsys);
ellipse e0=ellipse((0,0), 3, 1);
draw(e0, linewidth(bp), Arrow);
ellipse e1=ellipse((0,0), 3, 1, 45);
draw(e1, bp+0.8*red, Arrow);
ellipse e2=ellipse((0,0), 1, 3, 45);
draw(e2, bp+0.8*blue, Arrow);
shipout(bbox(2mm));
```

### 8.3.2 От типа ellipse к типу path

Приведение типа `ellipse` к типу `path` осуществляется в соответствии со следующими правилами:

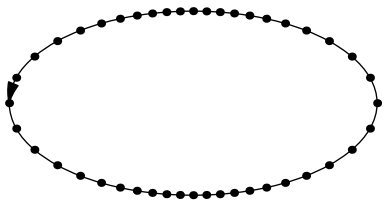
- Путь `path` должен быть циклом, с ориентацией против часовой стрелки;
- первая точка, которая возвращается процедурой `pair point(path g, real t)` при `t=0`, есть точка пересечения фокальной полупрямой `[F1 F2)` с эллипсом;
- число узлов пути зависит от длин осей эллипса; оно вычисляется функцией

  `int ellipsenodesnumber(real a, real b)`, зависящей от переменной

  `ellipsenodesnumberfactor`;

- узлы пути определены в полярных координатах с углами относительно центра эллипса и равномерно распределены по всему интервалу `[0 ; 360)`.

```
import geometry;
size(5cm,0);
ellipsenodesnumberfactor=50;
ellipse e=ellipse(origin, 4, 2, 180);
draw(e, Arrow);
dot((path)e);
```



### 8.3.3 Другие процедуры

Помимо процедур, применяемых к объектам типа `conic`, приведем список процедур, специфических для объектов типа `ellipse`.

- `real centerToFocus(ellipse el, real a)`

  Позволяет конвертировать угол с вершиной в центре эллипса в угол с вершиной в первом фокусе.

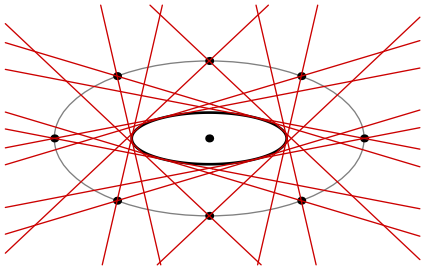
  Процедура `real focusToCenter(ellipse,real)` также определена.

- `real arclength(ellipse el, real angle1, real angle2, bool direction=CCW, polarconicroutine polarconicroutine=currentpolarconicroutine)`

  Возвращает длину дуги эллипса `el`, которая определяется от угла `angle1` к `angle2` (в градусах) в направлении `direction`. Возможным значением `polarconicroutine` является `arcfromfocus`, которое является значением по умолчанию для `currentpolarconicroutine`, или для `arcfromcenter`; в первом случае углы берутся относительно первого фокуса, во втором случае они берутся относительно центра эллипса.

- `line[] tangents(ellipse el, point M)`

  Возвращаются всевозможные касательные к эллипсу `el`, проходящие через точку `M`.

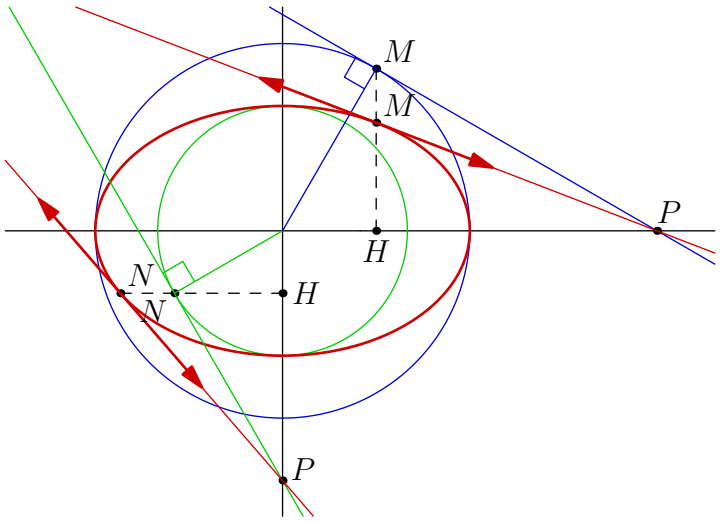


```
import geometry; size(5.5cm,0);
point A=(2.5,-1); dot(A);
ellipse C=ellipse(A,3,1);
draw(C,linewidth(bp));
path Cp=
shift(A)*xscale(2)*scale(3)*unitcircle;
draw(Cp, grey);
for (int i=0; i < 360; i+=45) {
point M=relpoint(Cp, i/360); dot(M);
draw(tangents(C, M), 0.8*red);
}
addMargins(8mm,8mm);
```

- `line tangent(ellipse el, abscissa x)`

  Возвращается касательная к эллипсу `el` в его точке с абсциссой `x`.

  В следующем примере иллюстрируется определение эллипса как образа окружности при дилатации и связанные с этим свойства касательных.




```
import geometry; size(10cm,0); draw(Ox()^^Oy()); real a=3, b=2;
circle C=circle(origin,a), Cp=circle(origin,b);
draw(C, 0.8*blue); draw(Cp, 0.8*green);
transform T=scale(b/a,Ox(),Oy()), Tp=scale(a/b,Oy(),Ox());
ellipse e=T*C; draw(e, bp+0.8*red);
point H=(a/2,0), Hp=(0,-b/2); dot("$H$", H, S); dot("$H'$", Hp);
line L=line(H,false,H+N), Lp=line(Hp,false,Hp+W);
point M=intersectionpoints(L,C)[0], NN=intersectionpoints(Lp,Cp)[0];
point Mp=T*M, NNp=Tp*NN; L=segment(H,M); Lp=segment(Hp,NNp);
```

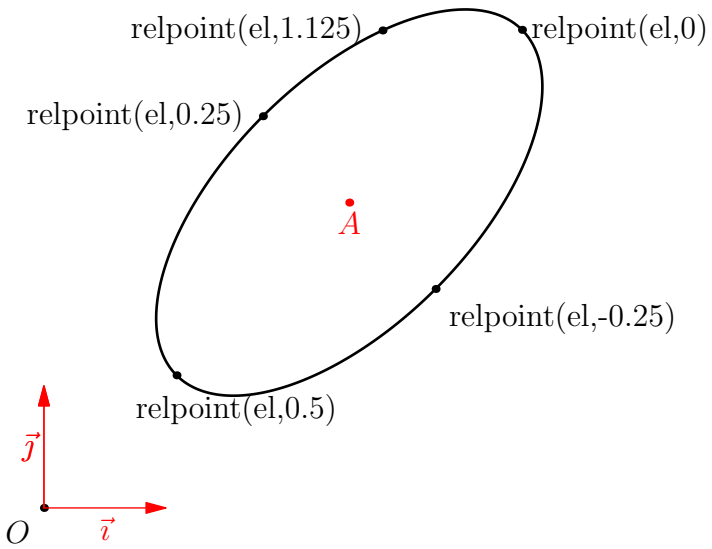
```
dot("$M$", M, NE); dot("$M'$", Mp, NE); dot("$N$", NN, SW); dot("$N'$", NNp, NE);
draw(L, dashed); draw(Lp, dashed);
segment SS=segment(origin,M), SSp=segment(origin,NN);
draw(SS, 0.8*blue); draw(SSp, 0.8*green);
line tgM=tangents(C, M)[0]; point P=intersectionpoint(tgM,Ox());
draw(tgM, 0.8*blue); dot("$P$", P, dir(60));
line tgN=tangents(Cp, NN)[0]; point Pp=intersectionpoint(tgN,Oy());
draw(tgN, 0.8*green); dot("$P'$", Pp, dir(30));
perpendicularmark(tgM,SS, 0.8*blue);
perpendicularmark(tgN,SSp, quarter=2, 0.8*green);
line tgMp=line(P, Mp), tgNp=line(Pp, NNp);
draw(tgMp, 0.8*red); draw(tgNp, 0.8*red);
draw(Mp+2tgMp.u--Mp-2tgMp.u, bp+0.8*red, Arrows(3mm));
draw(NNp+2tgNp.u--NNp-2tgNp.u, bp+0.8*red, Arrows(3mm));
addMargins(5mm,5mm);
```

- `point point(explicit ellipse el, real x)`

  Возвращается точка эллипса `el` в виде объекта типа `pair`, как при использовании кода `point((path)el,x)`.

- `point relpoint(explicit ellipse el, real x)`

  Возвращается точка эллипса `el`, которая соответствует части `x` от периметра `el`.

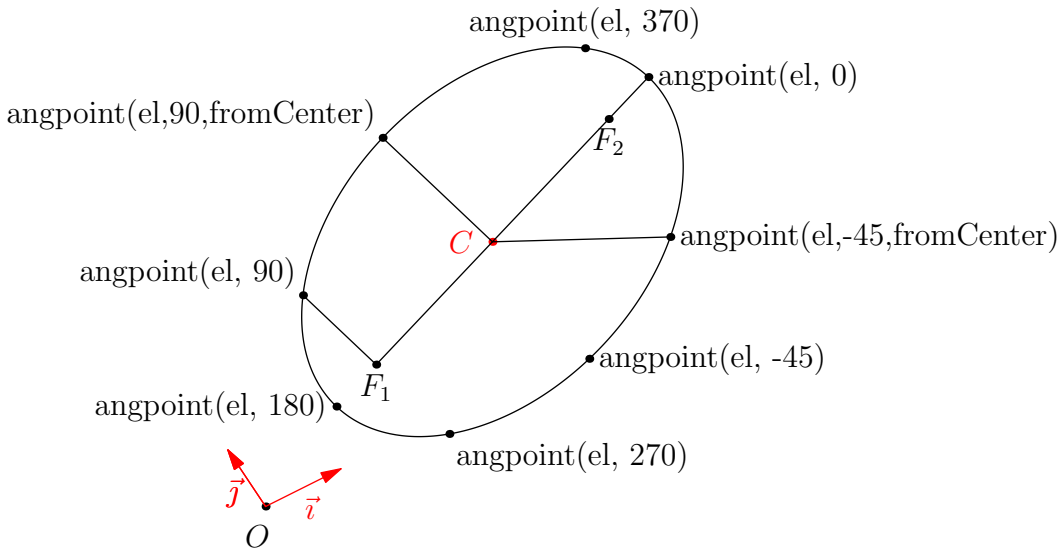



```
import geometry; size(8cm,0);
show(currentcoordsys, xpen=invisible);
point A=(2.5,2.5); dot("$A$", A, S, red);
ellipse el=ellipse(A,2,1,45);
draw(el, linewidth(bp));
dot("relpoint(el,0)", relpoint(el,0), 2S);
dot("relpoint(el,0.25)", relpoint(el,0.25), 2S);
dot("relpoint(el,0.5)", relpoint(el,0.5), 2S+E);
dot("relpoint(el,-0.25)",
                        relpoint(el, -0.25), 2SW);
dot("relpoint(el,1.125)", relpoint(el, 1.125), 2W);
```

- `point angpoint(explicit ellipse el, real x, polarconicroutine polarconicroutine=currentpolarconicroutine)`

Возвращает точку эллипса `el`, соответствующую углу в `x` градусов относительно центра, если `polarconicroutine=fromCenter`,

и относительно первого фокуса, если `polarconicroutine=fromFocus`.

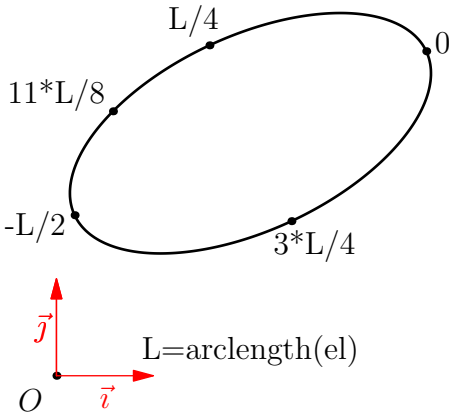



```
import geometry; size(10cm,0);
currentcoordsys=cartesiansystem((0,0),i=(1,0.5),j=(-0.5,.75));
show(currentcoordsys, xpen=invisible);
ellipse el=ellipse((4,2),3,2,20);
draw(el); dot("$C$",el.C,2W,red); dot("$F_1$",el.F1,S); dot("$F_2$",el.F2,S);
point P=angpoint(el, 0); dot("angpoint(el, 0)", P,E);
draw(el.F1--P);
point M=angpoint(el, 90); dot("angpoint(el, 90)", M,NW); draw(el.F1--M);
dot("angpoint(el, 180)", angpoint(el,180), W);
dot("angpoint(el, 270)", angpoint(el,270), SE);
dot("angpoint(el, 370)", angpoint(el,370), NE);
dot("angpoint(el, -45)", angpoint(el,-45), SE);
point P=angpoint(el, 90, fromCenter); dot("angpoint(el,90,fromCenter)", P,NW);
point Q=angpoint(el, -45, fromCenter); dot("angpoint(el,-45,fromCenter)", Q,S);
draw(el.C--P); draw(el.C--Q);
```

- `point curpoint(explicit ellipse c, real x)`

Возвращает точку эллипса `c`, имеющей криволинейную (curvilinear) абсциссу `x`.

```
import geometry; size(7cm,0);
show(currentcoordsys, xpen=invisible);
ellipse el=ellipse((2,2),2,1,25);
draw(el, linewidth(bp));
real L=arclength(el);
dot("0", curpoint(el,0), dir(25));
dot("L/4", curpoint(el,L/4), dir(115));
dot("3*L/4", curpoint(el,3*L/4), -dir(115));
dot("-L/2", curpoint(el, -L/2), -dir(25));
dot("11*L/8",curpoint(el,11*L/8),dir(145));
label("L=arclength(el)",(2,0.25));
```



## 8.4 Параболы

Тип `parabola` позволяет создать параболу. Так как существует взаимно-однозначное соответствие между объектами типа `parabola` и объектами типа `conic` с эксцентриситетом равным 1, объекты типа `parabola` наследуют набор процедур и операторов для типа `conic`.

### 8.4.1 Основные процедуры

Процедуры, доступные для определения параболы:

- `parabola parabola(point F, line l)`

  Возвращает параболу с фокусом `F` и директрисой `l`.

- `parabola parabola(point F, point vertex)`

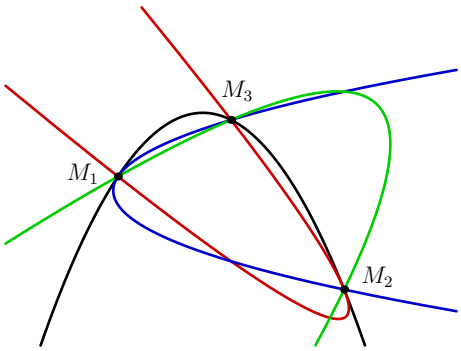
  Возвращает параболу с фокусом `F` и вершиной `vertex`.

- `parabola parabola(point F, real a, real angle)`

  Возвращает параболу с фокусом `F`, с latus rectum `a` ( хорда, проходящая через фокус параллельно директрисе) и углом `angle` между осью абсцисс и осью параболы.

- `parabola parabola(point M1, point M2, point M3, line l)`

  Возвращает параболу, проходящую через точки `M1, M2` и `M3`, директриса которой параллельна прямой `l`.




```
import geometry;
size(9cm,0);
draw(box((-2,-3),(6,3)), invisible);
point M1=(0,0), M2=(4,-2), M3=(2,1);
pen[] p=new pen[] {black,red,blue,green};
parabola P;
for (int i=0; i < 4; ++i) {
P=parabola(M1,M2,M3,rotate(45*i)*Ox());
draw(P, bp+0.8*p[i]);
}
dot(scale(0.75)*"$M_1$", M1, 2*dir(175));
dot(scale(0.75)*"$M_2$", M2, 2*dir(25));
dot(scale(0.75)*"$M_3$", M3, 2*dir(80));
shipout(bbox(2mm));
```

### 8.4.2 От типа parabola к типу path

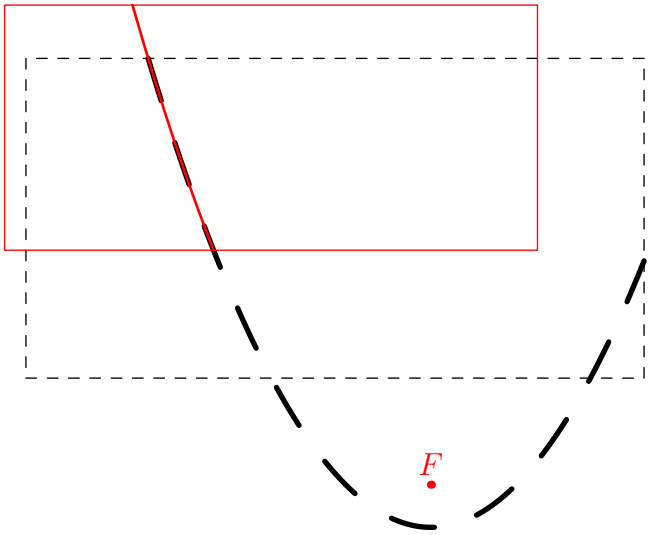
Приведение объекта `P` типа `parabola` к типу `path` осуществляется в соответствии со следующими правилами:

- Путь должен иметь ориентацию против часовой стрелки;
- Путь должен содержаться, по возможности:

  1. в текущем изображении, если переменные `P.bmin` и `P.bmax`, типа `pair`, не были изменены;
  2. в прямоугольнике `box((P.bmin),box(P.bmax))` в остальных случаях.

  Так, в следующем примере, когда делается первое преобразование в путь, область изображения обозначается пунктиром, и путь может не содержатся в этом прямоугольнике.

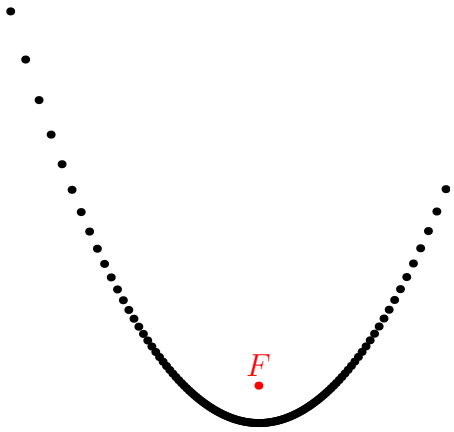
  Во время второго преобразования изменение переменных `p.bmin` и `p.bmax` переопределяет область конвертации, которая укладывается в красный прямоугольник с соответствующей частью параболы.




```
import geometry;
size(8.5cm);
point F=(2,-1.5);
dot("$F$",F,N,red);
parabola p=parabola(F,0.2,90);
draw(box((.1,-1),(3,.5)),dashed);
draw((path)p, 2*bp+dashed);
p.bmin=(0,-0.4);
p.bmax=(2.5,0.75);
draw(box(p.bmin,p.bmax), red);
draw((path)p, bp+red);
```

- Число узлов пути зависит от углов (в градусах), вершины которых находятся в фокусе, до концов пути и вычисляются функцией `int parabolanodesnumber(parabola p, real angle1, real angle2)`, зависящей от переменной `parabolanodesnumberfactor;`
- Узлы на пути, определенные в полярных координатах углами, вершины которых находятся в фокусе параболы и равномерно распределены на всем интервале, концы которого возвращает функция `real[] bangles(picture pic=currentpicture, parabola p`

```
import geometry;
size(6cm);
point F=(2,-1.5);
dot("$F$",F,N,red);
parabola p=parabola(F,0.2,90);
draw(box((0.6,-1.75),(3,0.5)),
              invisible);
parabolanodesnumberfactor=50;
dot((path)p);
shipout(bbox(2mm,invisible));
```

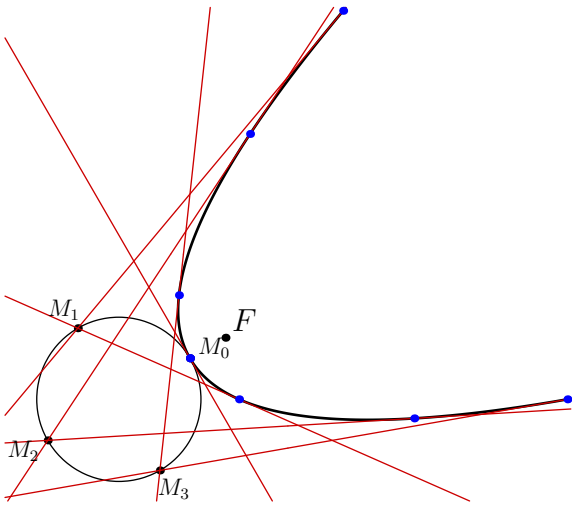



### 8.4.3 Другие процедуры

Кроме процедур, применяемые к объектам типа `conic`, приводим список специальных процедур для объектов типа `parabola`.

- `line[] tangents(parabola p, point M)`

  Возвращает все касательные к параболе `p`, проходящие через точку `M`.




```
import geometry; size(7cm,0);
point F=(0,0); dot("$F$", F, NE);
parabola p=parabola(F, 0.1, 30);
draw(p, linewidth(bp));
point C=shift(2*(p.V-p.F))*p.V;
circle cle=circle(C, 0.2);draw(cle);
for (int i=0; i < 360; i+=90) {
point M=C+0.2*dir(i+30);
dot(scale(0.75)*("$M_"+(string)(i/90)
+"$"),M, unit(M-C));
line[] tgt=tangents(p, M);
draw(tgt, 0.8*red);
for (int i=0; i < tgt.length; ++i) {
dot(intersectionpoints(p, tgt[i]),
blue);} }
```

- `line tangent(parabola p, abscissa x)`

  Возвращает касательную к параболе `p` в ее точке с абсциссой `x`.

- `point point(explicit parabola p, real x)`

  Возвращает точку на `p`, которую можно получить с помощью кода `point((path)p,x)`.

- `point relpoint(explicit parabola p, real x)`

  Возвращает точку на `p`, которую можно получить с помощью кода `relpoint((path)p,x)`.

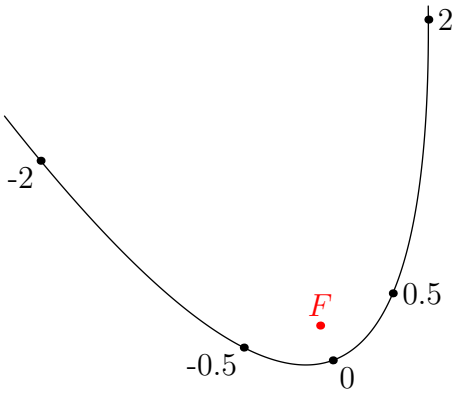
- `point angpoint(explicit parabola p, real x)`

  Возвращает точку на `p` по углу в `x` градусов.

- `point curpoint(explicit parabola p, real x)`

  Возвращает точку на `p`, криволинейная (curvilinear) абсцисса которой равна `x`, считая от вершины параболы.

```
import geometry; size(6cm);
point F=(1,-1.5); dot("$F$",F,N,red);
parabola p=parabola(F,0.2,110); draw(p);
dot("0",curpoint(p,0),SE);
dot("0.5",curpoint(p,0.5));
dot("-0.5",curpoint(p,-0.5),SW);
dot("-2",curpoint(p,-2),SW);
dot("2",curpoint(p,2),E);
shipout(bbox(2mm));
```



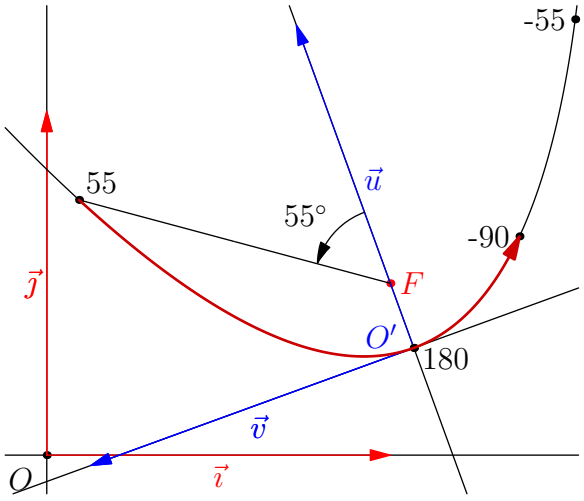

- Можно получить дугу параболы в виде пути с помощью функции

  `path arcfromfocus(conic co, real angle1, real angle2, int n=400, bool direction=CCW)`

  Хотя эта функция доступна для любого вида коник, ее использование действительно интересно для параболы и гиперболы; дуги эллипса имеют определенный тип, описанный в разделе *Дуги*. Вот пример, иллюстрирующий использование процедуры `arcfromfocus` для параболы.

```
import geometry;
size(8cm);
show(currentcoordsys);
point F=(1,0.5); dot("$F$",F,E,red);
parabola p=parabola(F,0.2,110); draw(p);
```

```
coordsys Rp=canonicalcartesiansystem(p);
show(Label("$O'$",align=NW+W,blue), Label("$\vec{u}$",blue),Label("$\vec{v}$",blue),
Rp, ipen=blue);
dot("180", angpoint(p,180), dir(-30));
point P=angpoint(p,55); dot("55",P,NE);
segment s=segment(F,P); draw(s);
line l=line(F,F+Rp.i);
line l1=line(F,P);
markangle("$"+(string)degrees(l,l1)+"^\circ$",l,l1,Arrow);
//markangle("$"+(string)degrees(l,s)+"^\circ$",l,s,Arrow);
//old variant
dot("-55", point(arcfromfocus(p,-55,-55,1),0), W);
dot("-90", point(arcfromfocus(p,-90,-90,1),0), W);
draw(arcfromfocus(p,55,-90), bp+0.8*red, Arrow(3mm));
```




## 8.5 Гиперболы

Тип `hyperbola` позволяет создать гиперболу. Так как существует взаимно-однозначное соответствие между объектами типа `hyperbola` и объектами типа `conic` с эксцентриситетом больше 1, объекты типа `hyperbola` наследуют набор процедур и операторов для типа `conic`.

### 8.5.1 Основные процедуры

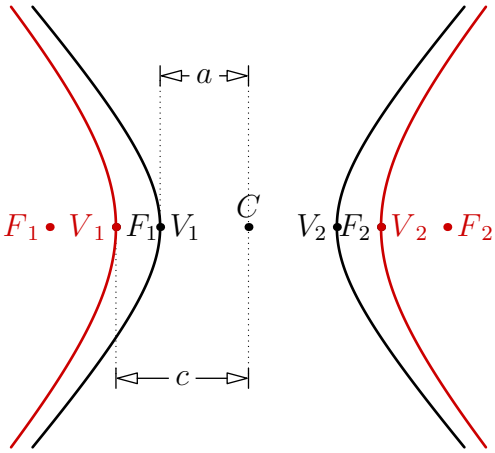
Процедуры, доступные для определения гиперболы:

- `hyperbola hyperbola(point P1, point P2, real ae, bool byfoci=byfoci)`

  Если `byfoci=true`: возвращается гипербола с большой полуосью `ae` и фокусами `P1` и `P2`;

  Если `byfoci=false`: возвращается гипербола с эксцентриситетом `ae` и вершинами `P1` и `P2`;

  Для большей ясности, определены константы `byfoci` и `byvertices`. Их значения, соответственно, `true` и `false`.




```
import geometry; size(6cm);
pen Red=0.8*red; point P1=(-3,0), P2=(3,0);
draw(box((-5,-5),(5,5)), invisible);
hyperbola Hf=hyperbola(P1,P2,2);
draw(Hf, linewidth(bp)); dot("$C$", Hf.C, N);
dot("$F_1$", Hf.F1); dot("$F_2$", Hf.F2, W);
dot("$V_1$", Hf.V1, E); dot("$V_2$", Hf.V2, W);
distance("$a$", Hf.C, Hf.V1, 2cm,
                                  joinpen=dotted);
distance("$c$", Hf.C, Hf.F1, -2cm,
                                  joinpen=dotted);
hyperbola Hv=hyperbola(P1,P2,1.5,byvertices);
draw(Hv, bp+Red);
dot("$V'_1$",Hv.V1, W, Red);
dot("$V'_2$",Hv.V2, Red);
dot("$F'_1$",Hv.F1, W, Red);
dot("$F'_2$",Hv.F2, Red);
shipout(bbox(2mm));
```

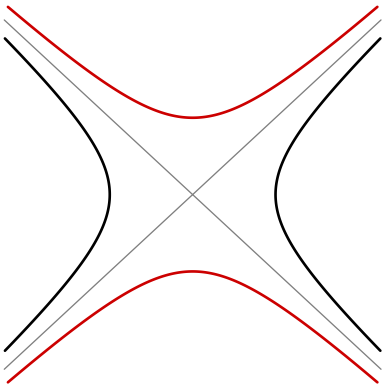
- `hyperbola hyperbola(point C, real a, real b, real angle=0)`

  Возвращает гиперболу с центром `C`, большой полуосью `a` вдоль `C-C+dir(angle)` и малой полуосью `b`.

- `hyperbola conj(hyperbola h)`

  Возвращает гиперболу, сопряженную с гиперболой `h`.

```
import geometry;
size(6cm);
point P1=(-3,0), P2=(3,0);
draw(box((-5,-5),(5,5)), invisible);
hyperbola H=hyperbola(P1,P2,2.2);
draw(H, linewidth(bp));
draw(H.A1^^H.A2, grey);
draw(conj(H), bp+0.8*red);
shipout(bbox(2mm));
```



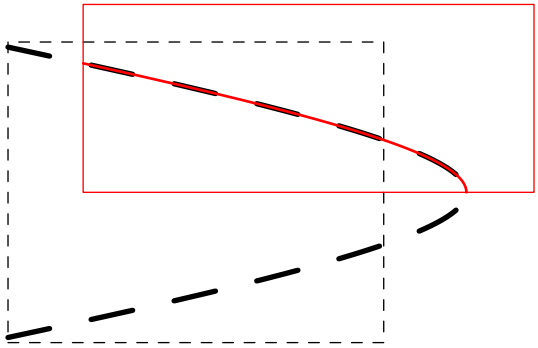
### 8.5.2 От типа hyperbola к типу path

Приведение объекта `H` типа `hyperbola` к типу `path` осуществляется в соответствии со следующими правилами:

- Путь должен быть ветвью гиперболы с фокусом `H.F1` с тригонометрической ориентацией;
- Путь должен содержаться, по возможности:
  1. в текущем изображения, если переменные `H.bmin` и `H.bmax`, типа `pair`, не были изменены;
  2. в прямоугольнике `box((H.bmin),box(H.bmax))` в остальных случаях.

  Так, в следующем примере, когда делается первое преобразование в путь, область изображения обозначается пунктиром, и путь может не содержатся в этом прямоугольнике.

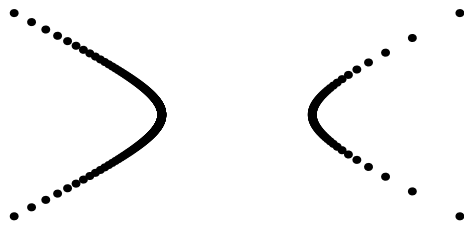
  Во время второго преобразования изменение переменных `H.bmin` и `H.bmax` переопределяет область конвертации, которая укладывается в красный прямоугольник с соответствующей частью гиперболы.



```
import geometry;
size(7cm,0);
point P1=(-3,0), P2=(3,0);
hyperbola H=hyperbola(P1,P2,2.95);
draw(box((-6,-1),(-3.5,1)), dashed);
draw((path)H, 2*bp+dashed);
H.bmin=(-5.5,0);
H.bmax=(-2.5,1.25);
draw(box(H.bmin,H.bmax), red);
draw((path)H, bp+red);
```

- Число узлов пути зависит от углов (в градусах), вершины которых находятся в фокусе, до концов пути и вычисляются функцией `int hyperbolanodesnumber(hyperbola p, real angle1, real angle2)`, зависящей от переменной `hyperbolanodesnumberfactor;`
- Узлы на пути, определенные в полярных координатах углами, вершины которых находятся в фокусе гиперболы `H.F1` и равномерно распределены на всем интервале, концы которого возвращает функция `real[][] bangles(picture pic=currentpicture, hyperbola p)`.

```
import geometry;
size(6cm,0);
point P1=(-3,0), P2=(3,0);
draw(box((-8,-4),(8,4)),
                 invisible);
dot((path)hyperbola(P1,P2,2.7));
hyperbolanodesnumberfactor=30;
dot((path)hyperbola(P2,P1,2.7));
```



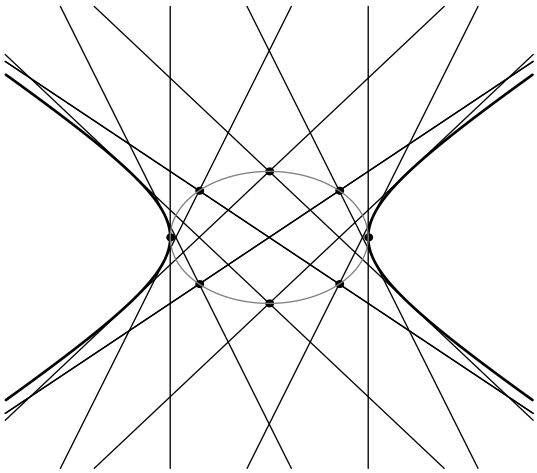
### 8.5.3 Другие процедуры

Кроме процедур, применяемые к объектам типа `conic`, приводим список специальных процедур для объектов типа `hyperbola`.

- `line[] tangents(hyperbola h, point M)`

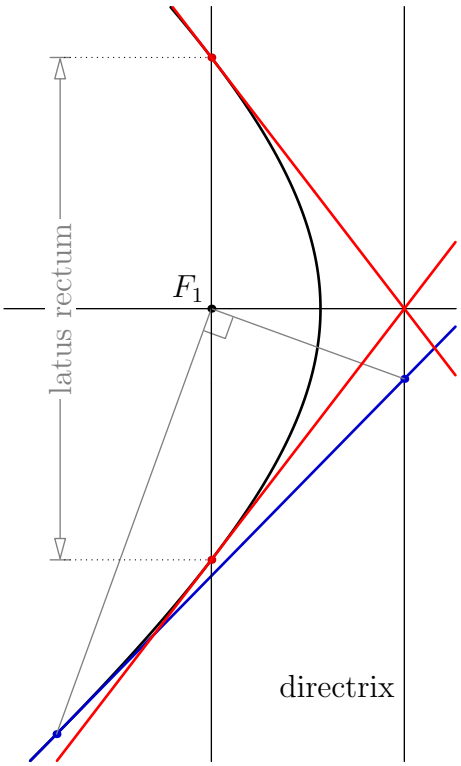
  Возвращает все касательные к гиперболе `p`, проходящие через точку `M`.

```
import geometry;
size(5cm,0);
draw(box((-5,-3),(5,3)), invisible);
hyperbola h=hyperbola(origin,1.5,1);
draw(h, linewidth(bp));

for (int i=0; i < 360; i +=45 )
  {
    point M=(1.5*Cos(i), Sin(i));
    dot(M);
    draw(tangents(h,M));
  }
draw(ellipse(origin,1.5,1), grey);
```



- `line tangent(hyperbola h, abscissa x)`

  Возвращает касательную к гиперболе `h` в ее точке с абсциссой `x`.




```
import geometry; size(0,10cm);
pen bl=0.8blue, re=0.8*red;
draw(box((-2.25,-1.5),(-0.75,1)),
                          invisible);
hyperbola h=hyperbola(origin,1.2,1);
draw((path)h, linewidth(bp));
draw("directrix", h.D1);
dot("$F_1$", h.F1, NW);
line axis=line(h.F1,h.F2);draw(axis);
point M=point(h,angabscissa(70));
dot(M, bl);
line tgt=tangent(h,angabscissa(70));
draw(tgt, bp+bl);
point P=intersectionpoint(tgt,h.D1);
dot(P, bl);
draw(P--h.F1--M, grey);
markrightangle(P,h.F1,M, grey);
line lr=perpendicular(h.F1, axis);
draw(lr);
point[] plr=intersectionpoints(h,lr);
dot(plr, re);
distance(Label("latus rectum",
       Fill(white)), plr[0], plr[1],
                 -2cm, grey, dotted);
for (int i=0; i < 2; ++i) {
draw(tangents(h,plr[i])[0], bp+red); }
```

- `point point(explicit hyperbola h, real x)`

  Возвращает точку на `h`, которую можно получить с помощью кода `point((path)h,x)`.

- `point relpoint(explicit hyperbola h, real x)`

  Возвращает точку на `h`, которую можно получить с помощью кода `relpoint((path)h,x)`.

- `point angpoint(explicit hyperbola h, real x, polarconicroutine polarconicroutine=currentpolarconicroutine)`

  Возвращает точку на `h` по углу в `x` градусов, с вершиной в центре гиперболы, если `polarconicroutine=fromCenter` или с вершиной в фокусе, если `polarconicroutine=fromFocus`

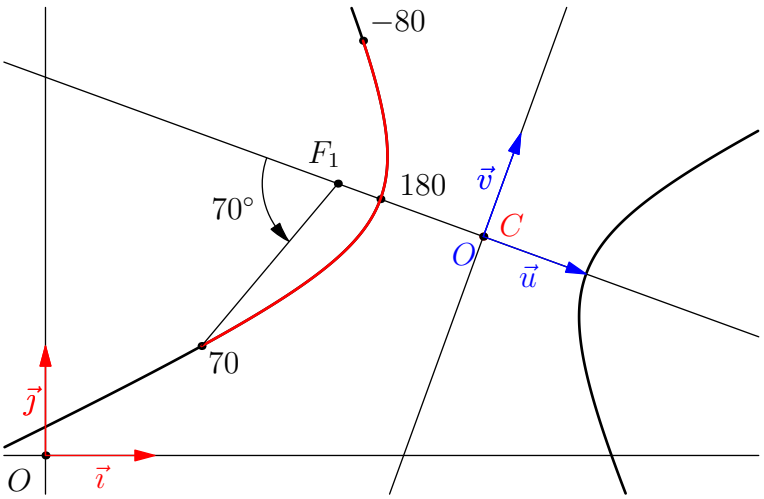
  Далее приведем два примера.

- Можно получить дугу гиперболы в форме пути, используя следующие подпрограммы:

  1. `path arcfromfocus(conic co, real angle1, real angle2, int n=400, bool direction=CCW)`

     Хотя эти подпрограмы доступны для всех типов коник, их интересно использовать для парабол и гипербол. Для дуг эллипса существует отдельный тип, описанный в разделе *Дуги*.

     Вот пример, показывающий использование подпрограммы `arcfromfocus` для гиперболы:



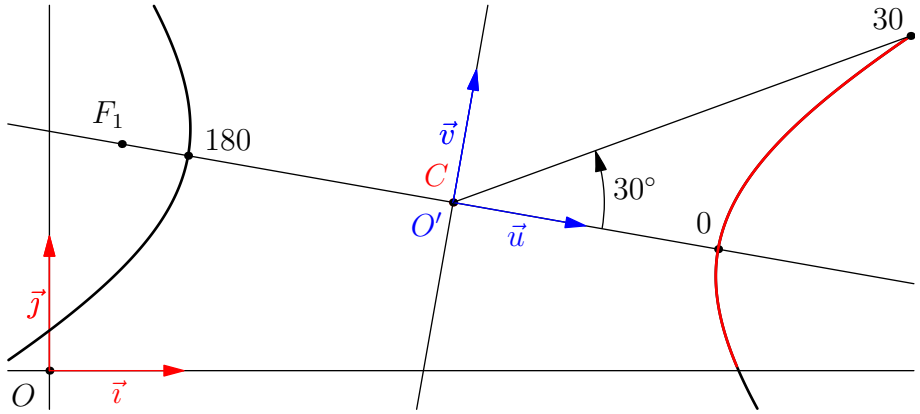

```
import geometry; size(10cm,0);
point C=(4,2); dot("$C$", C, E+NE, red);
hyperbola H=hyperbola(C,1,1,-20); draw(H, linewidth(bp));
coordsys R=currentcoordsys; show(R);
coordsys Rp=canonicalcartesiansystem(H);
show(Label("$O'$",align=SW,blue), Label("$\vec{u}$",blue),
     Label("$\vec{v}$",blue), Rp, ipen=blue);
dot("$180$", angpoint(H,180), N+2E);
dot("$-80$", angpoint(H,-80), NE);
point P=angpoint(H,70); dot("$70$", P, SE);
draw(arcfromfocus(H,70,-80), bp+red);
segment s=segment(H.F1,P); draw(s);
line l=line(H.F1,H.F1-Rp.i);
line l1=line(H.F1,P);
dot("$F_1$", H.F1, N+NW);
markangle("$70^\circ$",l,l1,Arrow);
addMargins(rmargin=3cm);
```

2. `path arcfromcenter(hyperbola h, real angle1, real angle2, int n=hyperbolanodesnumber(h,angle1,angle2), bool direction=CCW)`

Вот пример, показывающий использование подпрограммы `arcfromcenter` для гиперболы:

```
import geometry;
size(12cm);
coordsys R=currentcoordsys;
show(R);

point C=(3,1.25);
dot("$C$", C, 2*dir(120), red);
hyperbola H=hyperbola(C, 2, 1.5, -10);
draw(H, linewidth(bp));

coordsys Rp=canonicalcartesiansystem(H);
show(Label("$O'$", align=SW,blue), Label("$\vec{u}$",blue),
Label("$\vec{v}$",blue), Rp, ipen=blue);
dot("$0$", angpoint(H,0,fromCenter), 2*dir(120));
dot("$180$", angpoint(H,180,fromCenter), 2*dir(30));
draw(arcfromcenter(H,-20,30), bp+red);
dot("$F_1$", H.F1, N+NW);
point P=angpoint(H,30,fromCenter);
```

```
dot("$30$", P, NW);
segment s=segment(C, P);
draw(s);
markangle("$30^\circ$", Ox(Rp), s, radius=2cm, Arrow);
```




# 9 Дуги

Тип `arc` позволяет инициализировать ориентированную дугу эллипса. Основная подпрограмма для определения такой дуги указана ниже.

```
arc arc(ellipse el, real angle1, real angle2,
polarconicroutine polarconicroutine=polarconicroutine(el), bool direction=CCW)
```

Возвращает дугу эллипса `el` от угла `angle1` (в градусах) до `angle2` в направлении `direction` и заданную относительно первого фокуса, если `polarconicroutine=fromFocus` и относительно центра эллипса, если `polarconicroutine=fromCenter`.

Процедура `polarconicroutine polarconicroutine conic(co)`, используемая для определения значения по умолчанию параметра `polarconicroutine`, возвращает в данном случае `fromCenter`, если `co` представляет собой окружность, или `currentpolarconicroutine`, которая по умолчанию равна `fromFocus`, если `co` представляет собой эллипс.

Важно отметить, что, когда рисуются дуги, значение переменной `addpenarc` добавляется к используемому перу. По умолчанию эта переменная установлена в `squarecap`, что обычно выпрямляет концы, а это делает отображение пунктирной дуги неэффективным.

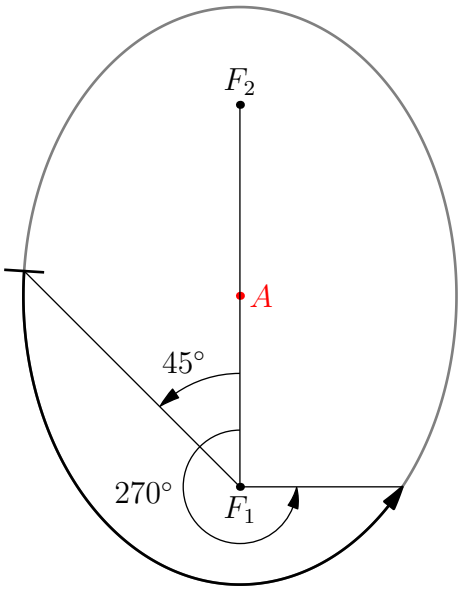
Для обхода этой проблемы существует три решения:

1. использовать команду `draw(a_arc, roundcap+dotted)`; вместо `draw(a_arc, dotted);`
2. установить значение `addpenarc` в `nullpen`.
3. связаться с автором пакета *geometry.asy* чтобы сообщить ему о вашем несогласии со значением по умолчанию `addpenline`;

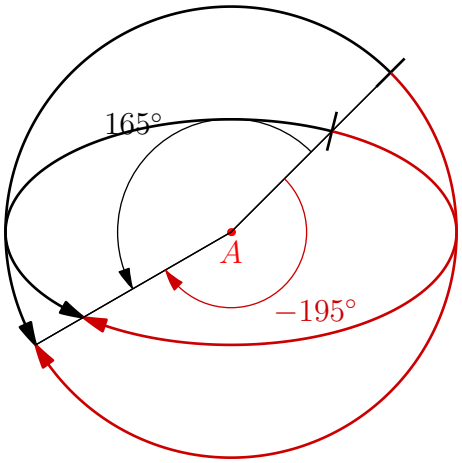
Вот несколько примеров, иллюстрирующих употребление процедуры
`arc(ellipse,real,real,polarconicroutine,bool)`

- В следующем примере показано, как получить дугу эллипса, где углы приводятся от первого фокуса, что соответствует поведению по умолчанию. Обратите внимание на использование функции `markarc` которая будет описана далее.

```
import geometry;
size(6cm,0);
real a=2, b=1.5;
point A=(1,1);
dot("$A$", A, red);
ellipse EL=ellipse(A,a,b,90);
draw(EL, bp+grey);
dot("$F_1$", EL.F1, S);
dot("$F_2$", EL.F2, N);
draw(EL.F1--EL.F2);
arc AE=arc(EL, 45, 270);
draw(AE, linewidth(bp),
                    Arrow(3mm), BeginBar);
point Bp=point(AE, 0), Ep=relpoint(AE,1);
draw(EL.F1--Bp); draw(EL.F1--Ep);
markangle(format("$%0g^\circ$",AE.angle1),
EL.F2,EL.F1,Bp, radius=1.5cm, Arrow);
markangle(Label(format("$%0g^\circ$",
          AE.angle2), Relative(0.35)),
EL.F2, EL.F1, Ep, radius=0.75cm, Arrow);
```



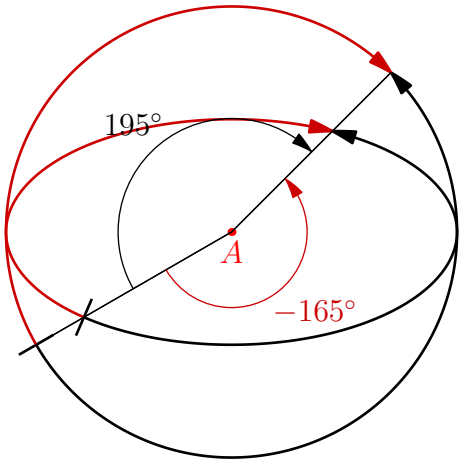

- В следующем примере показано влияние параметров `polarconicroutine` и `direction`. Обратите внимание на использование функции `degrees(arc)`, которая будет описана далее.




```
import geometry; size(8cm,0);
real a=2, b=1;
point A=(1,1); dot("$A$",A,S,red);
ellipse EL=ellipse(A,a,b);
arc AE=arc(EL, 45, 210, fromCenter);
draw(AE, linewidth(bp), Arrow(3mm),
                                BeginBar);
arc AEp=arc(EL, 45, 210, fromCenter, CW);
draw(AEp, bp+0.8*red, Arrow(3mm));
circle C=circle(A,a);
arc AC=arc(C, 45, 210);
draw(AC, linewidth(bp), Arrow(3mm),
BeginBar);
arc ACp=arc(C, 45, 210, CW);
draw(ACp, bp+0.8*red, Arrow(3mm));
markarc(format("$%0g^\circ$",degrees(AC)),
AC, radius=1.5cm, Arrow);
markarc(format("$%0g^\circ$",degrees(ACp)),
ACp, markpen=0.8*red, Arrow);
```

- Пример ниже показывает предыдущий код, меняющий местами углы 45° и 210° .

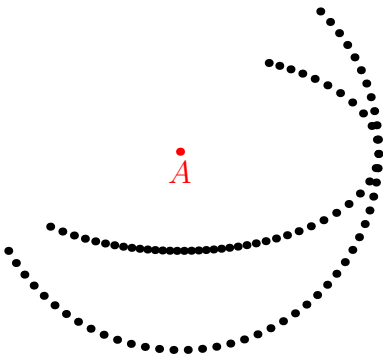
```
import geometry; size(8cm,0);
real a=2, b=1;
point A=(1,1); dot("$A$",A,S,red);
ellipse EL=ellipse(A,a,b);
arc AE=arc(EL, 210, 45, fromCenter);
draw(AE, linewidth(bp), Arrow(3mm),
BeginBar);
arc AEp=arc(EL, 210, 45, fromCenter, CW);
draw(AEp, bp+0.8*red, Arrow(3mm));
circle C=circle(A,a);
arc AC=arc(C, 210, 45);
draw(AC, linewidth(bp), Arrow(3mm),
                                BeginBar);
arc ACp=arc(C, 210, 45, CW);
draw(ACp, bp+0.8*red, Arrow(3mm));
markarc(format("$%0g^\circ$",degrees(AC)),
AC, radius=1.5cm, Arrow);
markarc(format("$%0g^\circ$",degrees(ACp)),
ACp, markpen=0.8*red, Arrow);
```




## 9.1 От типа arc к типу path

Приведение объекта `A` типа `arc` к типу `path` осуществляется в соответствии со следующими правилами:

- путь ориентирован в направлении `A.direction`;
- число узлов вычисляется функцией `int arcnodesnumber(explicit arc a)`, зависящей от переменной `ellispenodesnumberfactor`;
- узлы на пути определены в полярных координатах с углами, вершины которых находятся в первом фокусе или в центре эллипса, в зависимости от значения `A.polarconicroutine` и равномерно распределены в необходимом интервале.

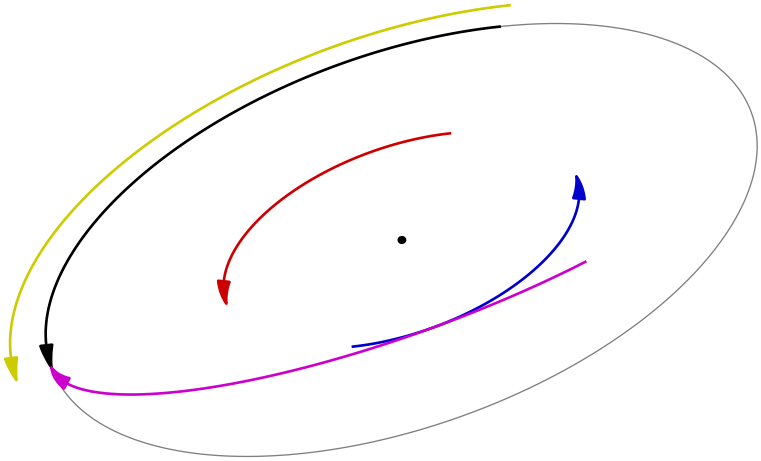



```
import geometry;
size(5cm,0);
ellipsenodesnumberfactor=100;
point A=(1,1); dot("$A$",A,S,red);
ellipse EL=ellipse(A,2,1);
dot((path)arc(EL, 210, 45, fromCenter));
circle C=circle(A,2);
dot((path)arc(C, 210, 45));
```

## 9.2 Операторы

- `arc operator *(transform t, explicit arc a)`

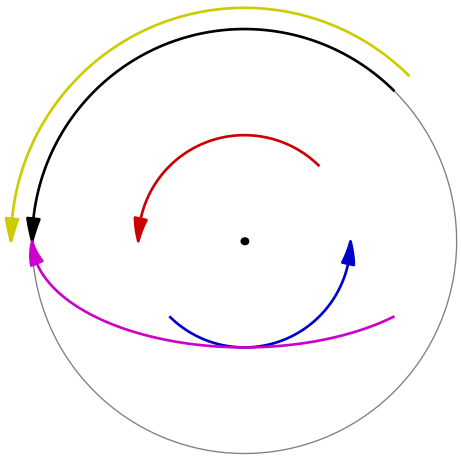
  Дает код `transfomr*arc`, поведение которого ничуть не удивляет. В следующем примере разноцветные дуги — образы черной дуги при аффинных преобразованиях.



```
import geometry; size(10cm,0);
currentcoordsys=rotate(20)*defaultcoordsys;
point C=(1,1); dot(C);
ellipse el=ellipse(C,2,1); draw(el, grey);
arc AE=arc(el, 45, 180, fromCenter); draw(AE, linewidth(bp), Arrow(3mm));
draw(scale(0.5,C)*AE, bp+0.8red, Arrow(3mm));
draw(scale(-0.5,C)*AE, bp+0.8blue,Arrow(3mm));
draw(scale(1.1,C)*AE, bp+0.8*yellow, Arrow(3mm));
transform t=scale(-0.5,line(el.F1,el.F2), line(S,N));
draw(t*AE, bp+0.8(red+blue), Arrow(3mm));
```

Тот же пример для дуг окружности:

```
import geometry; size(6cm,0);
point C=(0,0); dot(C);
ellipse el=circle(C,2); draw(el, grey);
arc AE=arc(el, 45, 180, fromCenter);
draw(AE, linewidth(bp), Arrow(3mm));
draw(scale(0.5,C)*AE, bp+0.8red,
                          Arrow(3mm));
draw(scale(-0.5,C)*AE, bp+0.8blue,
                          Arrow(3mm));
draw(scale(1.1,C)*AE, bp+0.8*yellow,
                            Arrow(3mm));
transform t=scale(-0.5,Ox(), Oy());
draw(t*AE, bp+0.8(red+blue),
                          Arrow(3mm));
```

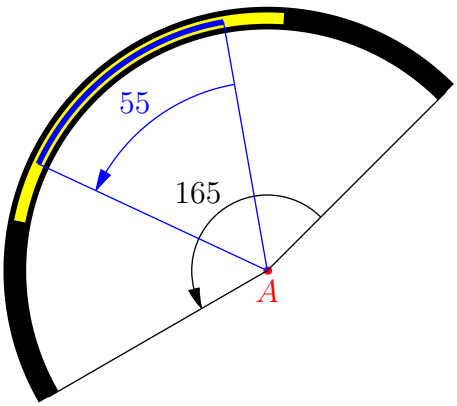


- `arc operator *(real x, explicit arc a)`

  Дает код `real*arc`.

  Возвращает дугу с углами `a.angle1-(x-1)*degrees(a)/2` и `a.angle2+(x-1)*degrees(a)/2`. Оператор `/(explicit arc,real)` тоже определен.

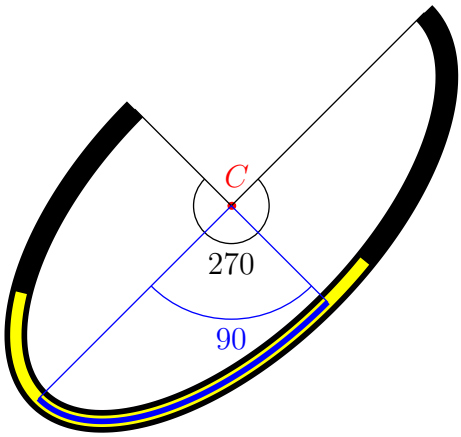
  В следующем примере желтые дуги получаются умножением черной дуги на 0,5 и голубая дуга получается делением ее на 3.




```
import geometry; size(8cm,0);
point A=(1,1); dot("$A$",A,S,red);
arc C=arc(circle(A,2), 45, 210);
draw(C,linewidth(3mm));
markarc(format("%0g",degrees(C)),
                          C, Arrow);
draw(0.5*C,1.5mm+yellow);
arc Cp=C/3;
draw(Cp, 0.75mm+blue);
markarc(format("%0g",degrees(Cp)),
radius=25mm, Cp, blue, Arrow);
```

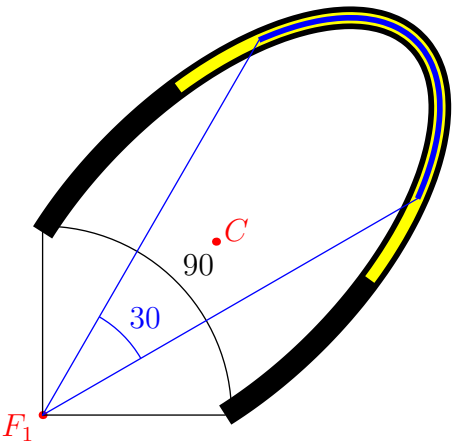
Аналогично для дуги эллипса, определенной относительно центра эллипса.

```
import geometry; size(6cm,0);
point C=(1,1);
dot("$C$", C, 2*dir(80), red);
arc a=arc(ellipse(C,2,1,45),90,0,
                          fromCenter);
draw(a, linewidth(3mm));
markarc(format("%0g", degrees(a)),
radius=-0.5*markangleradius(), a);
draw(0.5*a, 1.5mm+yellow);
arc ap=a/3;
draw(ap, 0.75mm+blue);
markarc(format("%0g", degrees(ap)),
radius=1.5*markangleradius(),ap,blue);
```




Наконец, в следующем примере, дуги определяются относительно первого фокуса эллипса:

```
import geometry; size(8cm,0);
point C=(1,1);
dot("$C$", C, dir(30), red);
arc a=arc(ellipse(C,2,1,45), -45, 45);
draw(a, linewidth(3mm));
dot("$F_1$", a.el.F1, dir(210), red);
markarc(format("%0g", degrees(a)),
radius=2.5*markangleradius(), a);
draw(0.5*a, 1.5mm+yellow);
arc ap=a/3;
draw(ap, 0.75mm+blue);
markarc(format("%0g", degrees(ap)),
radius=1.5*markangleradius(),ap,blue);
```




- `arc operator +(explicit arc a, point M)`

  Дает код `arc+point`, равносильный коду `shift(point)*arc`.

  Еще определены операторы: `-(explicit arc,point)`, `+(explicit arc,vector)` и `-(explicit arc,vector)`.

- `bool operator @ (point M, arc a)`

  Дает код `point @ arc`. Возвращает `true` если и только если точка `M` принадлежит дуге `a`.

- `arc operator *(inversion i, segment s)`

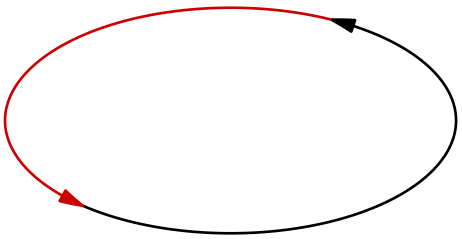
  Дает код `inversion*segment`. Возвращает образ сегмента `s` при инверсии i; смотрите примеры `inversion*segment` в разделе *Инверсии*.

## 9.3 Другие процедуры

В дополнение к функциям, описанным в этом разделе, добавляются процедуры выяснения расположения точки на объекте типа `arc`, описанные в разделе *Абсциссы*.

- `arc complementary(arc a)`

  Возвращает дополнительную дугу к дуге `a`.



```
import geometry;
size(6cm,0);
ellipse EL=ellipse(origin,2,1);
arc AE=arc(EL, 210, 45, fromCenter);
draw(AE, linewidth(bp), Arrow(3mm));
draw(complementary(AE),
                bp+0.8*red, Arrow(3mm));
```

- `arc reverse(arc a)`

  Возвращает дугу, реверсную к дуге `a`, аналогично процедуре `reverse(path)`.

- `real degrees(arc a)`

  Возвращает угол ориентированной дуги в градусах в интервале $[-360; 360]$.

  Функция `angle(arc)` определяется и в радианах.

- `real arclength(arc a)`

  Возвращает длину дуги `a`.

- ```
  void markarc(picture pic=currentpicture,
               Label L=, int n=1, real radius=0, real space=0,
               arc a, pen sectorpen=currentpen, pen markpen=sectorpen,
               margin margin=NoMargin, arrowbar arrow=None, marker marker=nomarker)
  ```

  Позволяют отмечать угол `a` дугой окружности.

  Параметр `sectorpen` является пером, которое используется для отметки сегмента, который связывает центр или фокус дуги с ее концами.

  Параметр `markpen` является пером, которым чертится дуга окружности, которая, в свою очередь, может быть отмечена параметром `marker`. Примеры использования уже давались.

- `point[] intersectionpoints(arc a1, arc a2)`

  Возвращает точки пересечения двух дуг в виде массива. Процедуры пересечения объекта `arc` с другими объектами, определенными в пакете *geometry.asy* также определены; например

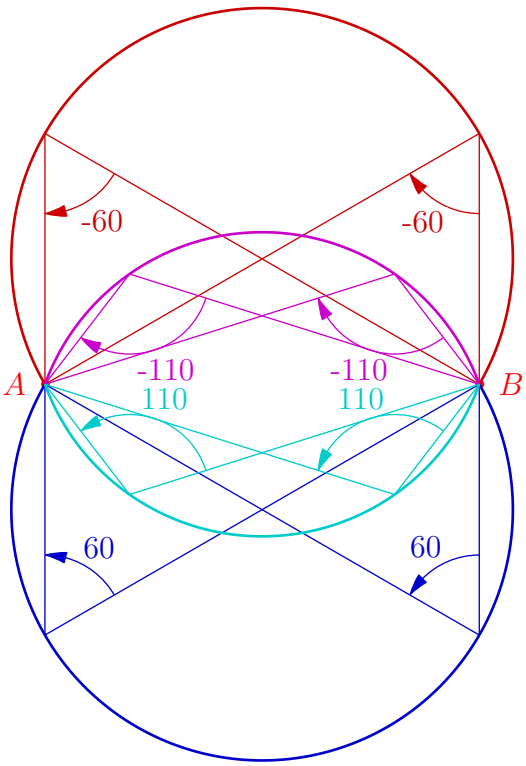
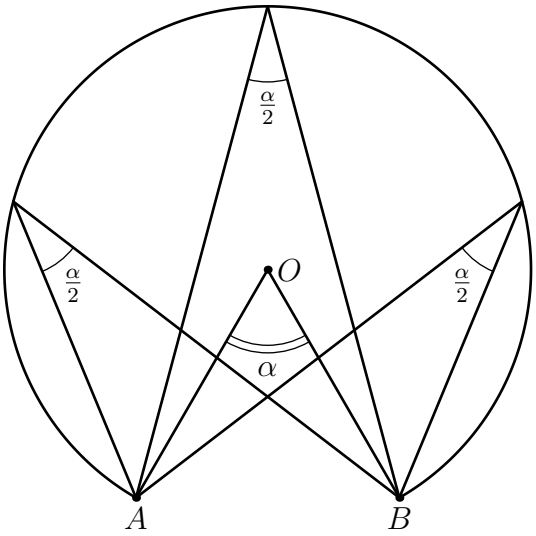
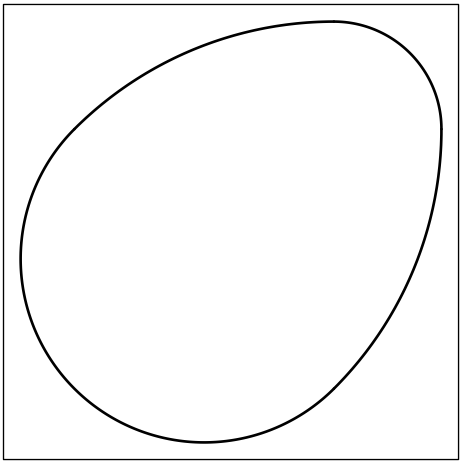
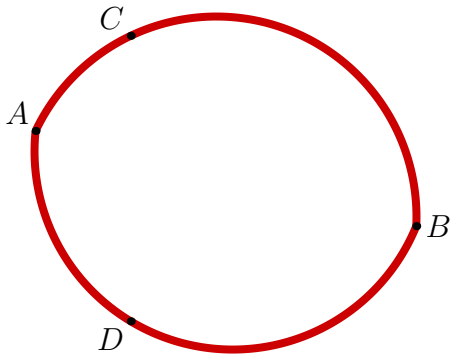
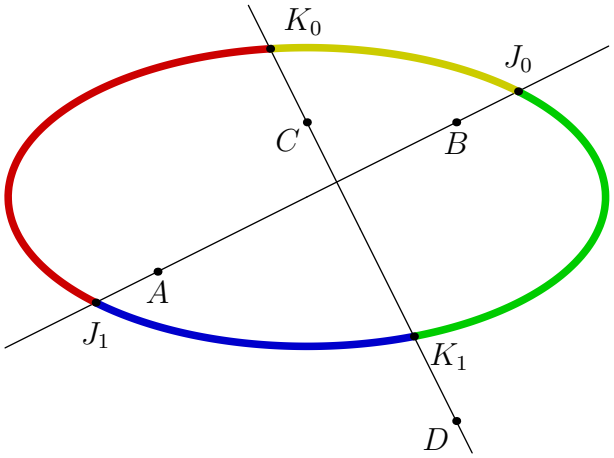
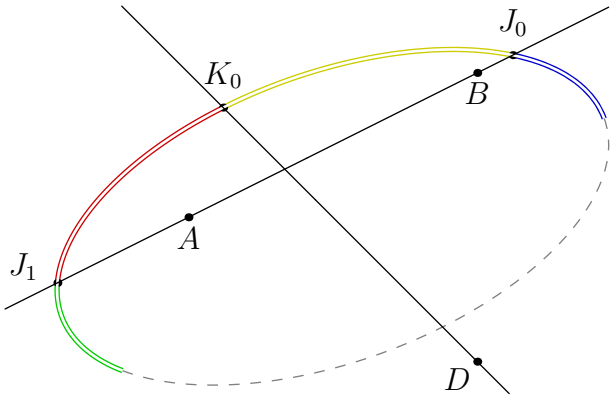
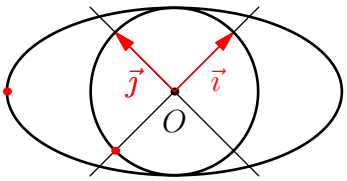
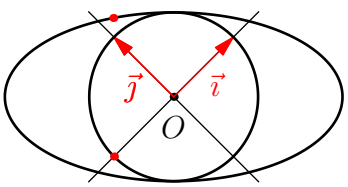
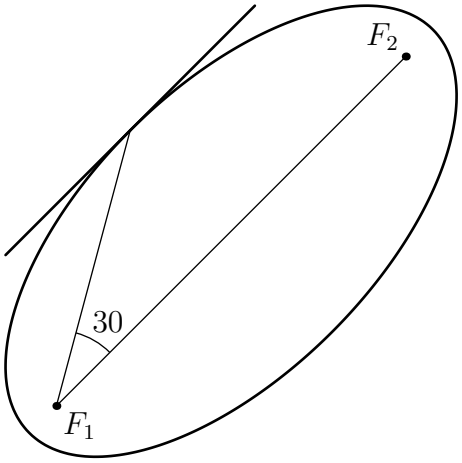
  `intersectionpoints(conic co, arc a)`,

  `intersectionpoints(arc a, conic co)`,

  `intersectionpoints(line l, arc a)` и т.д.

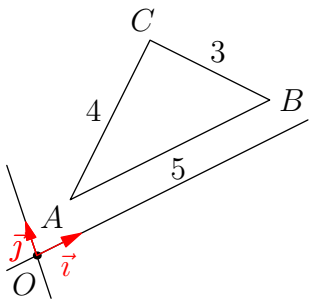
- `arc arcsubtended(point A, point B, real angle)`

  Возвращает дугу с углом `angle`, опирающуюся на сегмент `[AB]`.

  Код `a_arcsubtended.C` позволяет восстановить центр дуги, это же можно сделать процедура `point arcsubtendedcenter(point A, point B, real angle)`.
  ```




```
import geometry; size(7cm,0);
point A=(-1,0), B=(1,0);
dot("$A$", A, 2W, red);
dot("$B$", B, 2E, red);
real[] angles=new real[]
      {60, 110, -60, -110};
pen[] p=new pen[] {red, blue+red,
                   blue, cyan};
int i=0;
for(real a:angles) {
arc arcsubtended=arcsubtended(A,B,a);
draw(arcsubtended, bp+0.8*p[i]);
for (int j=0; j < 2; ++j) {
point M=relpoint(arcsubtended,
                        0.25+0.5*j);
draw(A--M--B, 0.8*p[i]);
real gle=degrees(B-M)-degrees(A-M);
markangle(Label(format("%0g",-gle),
UnFill), B, M, A, radius=sgn(-gle)*30,
                  Arrow, 0.8*p[i]); }
++i; }
```

```
import geometry; size(7cm,0);
point A=(-1,0), B=(1,0);
dot("$A$", A, S); dot("$B$", B, S);
pen bpp=linewidth(bp);
arc Ac=arcsubtended(A,B,30);
draw(Ac, bpp);
dot("$O$", Ac.el.C);
markarc("$\alpha$", Ac, n=2,
radius=1cm, sectorpen=bpp,
          markpen=currentpen);
for (int i=0; i < 3; ++i) {
point M=relpoint(Ac, 0.25+0.25*i);
draw(M--A^^M--B, linewidth(bp));
markangle("$\frac{\alpha}{2}$",
                       A, M, B); }
```




- `arc arccircle(point A, point B, real angle, bool direction=CCW)`

  Возвращает дугу окружности, с центром `A`, от точки `B` к образу `B` при повороте вокруг `A` на угол `angle` в направлении `direction`.



```
import geometry;
size(6cm);
point A=(-1,1), B=(1,-1);
point M=(A+B)/2;
point P=rotate(90,M)*B;
arc A1=arccircle(A,B,45),
        A2=arccircle(B,A,-45,CW),
A3=arccircle(P,relpoint(A2,1),-90,CW),
A4=arccircle(M,A,180);
draw(A1^^A2^^A3^^A4, linewidth(bp));
shipout(bbox(2mm));
```

- `arc arccircle(point A, point M, point B)`

  Возвращает дугу окружности $AB$, проходящую через точку `M`

```
import geometry;
size(6cm);
point A=(-1,0), B=(3,-1), C=(0,1), D=(0,-2);
draw(arccircle(A,C,B), dotsize()+0.8*red);
draw(arccircle(A,D,B), dotsize()+0.8*red);
dot("$A$", A, NW); dot("$B$", B, E);
dot("$C$", C, NW); dot("$D$", D, SW);
```




- `arc arc(ellipse el, point M, point N, bool direction=CCW)`

  Возвращает дугу эллипса `el`, в направлении `direction`, с концами `M` и `N` которые должны принадлежать `el`.




```
import geometry; size(8cm);
point A=(-1,0), B=(1,1), C=(0,1), D=(1,-1);
dot("$A$",A,S); dot("$B$",B,S); dot("$C$",C,SW); dot("$D$",D,SW);
ellipse el=ellipse((0,0.5),2,1);
line l1=line(A,B), l2=line(C,D); draw(l1); draw(l2);
point[] J=intersectionpoints(l1,el), K=intersectionpoints(l2,el);
```

```
draw(arc(el, J[0],K[0]), 1mm+0.8yellow); draw(arc(el, K[0],J[1]), 1mm+0.8red);
draw(arc(el, J[1],K[1]), 1mm+0.8blue); draw(arc(el, K[1],J[0]), 1mm+0.8green);
dot("$J_0$", J[0], 2N); dot("$J_1$", J[1], 2S);
dot("$K_0$", K[0], 2NE) ; dot("$K_1$", K[1], 2dir(-35));
```

- `arc arc(ellipse el, explicit abscissa x1, explicit abscissa x2, bool direction=CCW)`

  Эта процедура ведет себя так же, как и предыдущие процедуры, но точки определяются абсциссами относительно эллипса.

- `arc arc(explicit arc a, point M, point N)`

  Возвращает часть дуги `a` между `M` и `N`.




Далее приведем код к рисунку.

```
import geometry; size(8cm);
point A=(-1,0), B=(1,1), C=(0,0), D=(1,-1);
dot("$A$",A,S); dot("$B$",B,S); dot("$D$",D,SW);
arc c=arc(ellipse(C,2,1,20), 0, 270); draw(complementary(c),dashed+grey);
line l1=line(A,B), l2=line(C,D);
point[] J=intersectionpoints(l1,c), K=intersectionpoints(l2,c);
draw(arc(c,J[0],K[0]), 2bp+0.8yellow); draw(arc(c,K[0],J[1]), 2bp+0.8red);
draw(arc(c,J[1],relpoint(c,1)), 2bp+0.8green);
draw(arc(c,point(c,0),J[0]), 2bp+0.8blue);
dot("$J_0$",J[0],2N); dot("$J_1$",J[1],N+2W); dot("$K_0$",K[0],2N);
draw(c, bp+white); draw(l1^^l2);
```

- `arc arc(arc el, explicit abscissa x1, explicit abscissa x2)`

  Эта процедура ведет себя так же, как и предыдущие процедуры, но точки определяются абсциссами относительно эллипса.

- `arc inverse(real k, point A, segment s)`

  Возвращает образ сегмента `s` при инверсии с центром `A` и радиусом `k`;

  смотри примеры `inversion*segment` в разделе *Инверсии*.

- `line tangent(explicit arc a, point M)`

  Возвращает касательную к `a` в точке `M` на `a`.

- `line tangent(explicit arc a, abscissa x)`

  Возвращает касательную к `a` в точке с абсциссой `x` заданной относительно `a`.

# 10 Абсциссы

Тип `abscissa` позволяет создать абсциссу на объекте типа `line`, `segment`, `conic` и `arc`. Структура объекта типа `abscissa` такова:

```
struct abscissa { real x; int system;
                  polarconicroutine polarconicroutine;
                                    abscissa copy() {...} }
```

Здесь `x` является значением абсциссы, параметр `system` представляет собой тип абсциссы:

```
0   для абсциссы как части длины пути;
1   для криволинейной абсциссы;
2   для угловой абсциссы;
3   для абсциссы относительно узлов пути.
```

Для лучшей читаемости кода определены следующие константы:

```
int relativesystem=0, curvilinearsystem=1, angularsystem=2, nodesystem=3; ,
```

Процедура `polarconicroutine` позволяет указать точку отсчета ( reference center ) в случае угловой абсциссы; возможны значения `fromCenter` и `fromFocus`; наконец, `abscissa copy()` возвращает полную копию абсциссы.

## 10.1 Определение абсцисс

Есть много процедур для определения абсциссы по ее типу. После того, как абсциссы определены, можно восстановить точку объекта по этой абсциссе процедурой `point(object,abscissa)`.

- `abscissa relabscissa(real x)`

  Возвращает абсциссу `x` как часть от длины пути. Нужно отметить, что код

  `point(object,relabscissa(x))`, равносилен `relpoint(object,x)`.




```
import geometry; size(4.5cm);
currentcoordsys=rotate(45)*defaultcoordsys;
show(currentcoordsys);
abscissa rel=relabscissa(0.5);
ellipse el=ellipse(origin(),2,1,-45);
draw(el,linewidth(bp));
circle c=circle(origin(),1);
draw(c,linewidth(bp));
dot(point(el,rel), red);
dot(point(c,rel), red);
```

- `abscissa curabscissa(real x)`

  Возвращает криволинейную абсциссу `x`. Нужно заметить, что код

  `point(object,curabscissa(x))` эквивалентен `curpoint(object,x)`.

```
import geometry; size(4.5cm);
currentcoordsys=rotate(45)*defaultcoordsys;
show(currentcoordsys);
abscissa cur=curabscissa(pi);
ellipse el=ellipse(origin(),2,1,-45);
draw(el,linewidth(bp));
circle c=circle(origin(),1);
draw(c,linewidth(bp));
dot(point(el,cur), red);
dot(point(c,cur), red);
```



- `abscissa angabscissa(real x, polarconicroutine polarconicroutine=currentpolarconicroutine)`
  Возвращает угловую абсциссу `x`. Нужно отметить, что код `point(object,angabscissa(x))` равносилен коду `angpoint(object,x)`.




```
import geometry;
size(6cm);
abscissa x=angabscissa(30);
ellipse el=ellipse(origin(),2,1,45);
draw(el,linewidth(bp));
point M=point(el,x);
draw(M--el.F1--el.F2);
dot("$F_1$", el.F1, SE);
dot("$F_2$", el.F2, NW);
markangle((string)x.x, el.F2, el.F1, M);
draw(tangent(el,x), linewidth(bp));
shipout(bbox(2mm));
```

- `abscissa nodabscissa(real x)`
  Возвращает абсциссу `x` относительно узлов пути. Необходимо отметить, что код `point(object,nodabscissa(x))`, равносилен коду `point(object,x)`.

## 10.2 Получение абсциссы точки

Программы для получения абсциссы точки, принадлежащей данному объекту, имеют такие же имена, как описано в предыдущем разделе. Следующие функции возвращают соответственно: относительную абсциссу, криволинейную абсциссу, угловую абсциссу и «узловую абсциссу» в точке `M`, принадлежащей указанному объекту.

**Относительная абсцисса**

- `abscissa relabscissa(line l, point M)`
- `abscissa relabscissa(ellipse el, point M)`
- `abscissa relabscissa(arc a, point M)`

**Криволинейная абсцисса**

- `abscissa curabscissa(line l, point M)`
- `abscissa curabscissa(ellipse el, point M)`
- `abscissa curabscissa(parabola p, point M)`

**Угловая абсцисса**

- `abscissa angabscissa(circle c, point M)`
- `abscissa angabscissa(ellipse el, point M, polarconicroutine polarconicroutine=currentpolarconicroutine)`
- `abscissa angabscissa(hyperbola h, point M, polarconicroutine polarconicroutine=currentpolarconicroutine)`
- `abscissa angabscissa(parabola p, point M)`

**«Узловая » абсцисса**

- `abscissa nodabscissa(line l, point M)`
- `abscissa nodabscissa(ellipse el, point M)`
- `abscissa nodabscissa(parabola p, point M)`

### 10.3 Операторы

```
abscissa operator +(real x, explicit abscissa a)
```

Возвращает копию абсциссы `a` как `x+a.x`.

Еще определены операторы:

```
operator + (explicit abscissa a, real x)
operator - (real x, explicit abscissa a)
operator - (explicit abscissa a, real x)
operator - (explicit abscissa a)
operator * (real x, explicit abscissa a)
operator * (explicit abscissa a, real x)
operator / (real x, explicit abscissa a)
operator / (explicit abscissa a, real x)
```

## 11 Треугольники

### 11.1 Структура

Структура `triangle` (треугольник) более сложнна, чем предыдущие, уже разобранные в данном документе. Она действительно определяет новые типы, создающие объекты, непосредственно связаные с треугольником. Кроме того, эти объекты обладают ссылками на ассоциированный треугольник. Иными словами, объект `TR` типа `triangle` содержит в качестве элемента структуры объект `VA` типа `vertex` (вершина), который обозначается `TR.VA` (вершина $A$) и объект `TR.VA` содержит объект `t` типа `triangle` со значением `TR`; отсюда следует, что значением `TR.VA.t` является `TR`.

Так как объект типа `vertex` является вершиной треугольника, то довольно просто написать функцию, которая возвращает, например, внутреннюю биссектрису угла треугольника: для этого достаточно придать объекту тип `vertex`, поскольку этот объект содержит ссылку на треугольник, в котором он является вершиной.

Полная структура `triangle` подробно излагается здесь.

Вот упрощенная версия структуры `triangle`:

```
struct triangle { restricted point A, B, C;
        struct vertex { int n; triangle t; }
            restricted vertex VA, VB, VC;
        struct side { int n; triangle t; }
        side AB, BC, CA, BA, AC, CB; }
```

В этой структуре можно выделить:

1. `A, B` и `C` представляют точки, обозначающие вершины треугольника;
2. `struct vertex` определяет структуру *вершина*, которая позволяет создать экземпляр объекта, представляющего собой вершину треугольника. Поскольку мы должны манипулировать этой структурой в очень разных случаях, полезно знать ее свойства.

   Свойство `n` позволяет связать вершины с точками типа `point`, обозначающими эти вершины:

   если `n = 1`, вершина связана с точкой `A`;

   если `n = 2`, вершина связана с точкой `B`;

   если `n = 3`, вершина связана с точкой `C`;

   если `n = 4`, вершина связана с точкой `A`;

и т.д.

Значением свойства `t` является: «объект типа `triangle` с принадлежащими ему вершинами».

Более детальное использование этой структуры смотри в подразделе *Вершины треугольника*. `VA`, `VB` и `VC` представляют вершины треугольника абстрактным способом, в отличие от объектов `point A`, `B` и `C`, обозначающими вершины; смотри подраздел *Вершины треугольника*.

3. `struct side` определяет структуру *сторона*, которая позволяет создать экземпляр объекта, представляющего сторону в треугольнике. Поскольку мы должны манипулировать этой структурой, полезно знать ее свойства.

   Свойство `n` позволяет связать объект типа `side` со сторонами треугольника;

   Если `n = 1`, сторона представляет `AB`, направлена от `A` к `B`;

   Если `n = 2`, сторона представляет `BC`, направлена от `B` к `C`;

   Если `n = 3`, сторона представляет `CA`, направлена от `C` к `A`;

   Если `n = 4`, сторона представляет собой `AB`;

   и т.д.

   Если `n` отрицательно, ориентация меняется на противоположную.

   Значением свойства `t` является: «объект типа `triangle` с принадлежащими ему сторонами».

   Программы для сторон описаны в подразделе *Стороны треугольника*.

   `AB`, `BC`, `CA`, `BA`, `AC` и `CB` представляют стороны треугольника абстрактным образом, в отличие от объектов `line`: `line(TR.AB)`, `line(TR.BC)`, `line(TR.CA)`, которые являются прямыми, обозначающими стороны треугольника `TR`; Смотри подраздел *Стороны треугольника*.

## 11.2 Определение и черчение треугольника

Этот раздел посвящен описанию основных процедур, определяющих и рисующих треугольник. Другие подпрограммы будут предоставлены по мере чтения.

- ```
  void label(picture pic=currentpicture, Label LA="$A$",
             Label LB="$B$", Label LC="$C$",
             triangle t,
             real alignAngle=0,
             real alignFactor=1,
             pen p=nullpen, filltype filltype=NoFill)
  ```

  Рисует подписи `LA`, `LB` и `LC` около вершин треугольника `t`.

  Параметры `alignAngle` и `alignFactor` позволяют изменять выравнивание.

- ```
  void show(picture pic=currentpicture,
            Label LA="$A$", Label LB="$B$", Label LC="$C$",
            Label La="$a$", Label Lb="$b$", Label Lc="$c$",
            triangle t, pen p=currentpen, filltype filltype=NoFill)
  ```

  Рисует треугольник $t$, размещает подписи к вершинам треугольника и указывает длины сторон. При использовании данной процедуры полезно указывать t.A, t.B и t.C.

- ```
  void draw(picture pic=currentpicture, triangle t,
            pen p=currentpen, marker marker=nomarker)
  ```

  Рисует треугольник $t$; стороны изображаются отрезками.

- ```
  void drawline(picture pic=currentpicture, triangle t, pen p=currentpen)
  ```

  Рисует треугольник $t$; стороны изображаются прямыми.

- ```
  triangle triangle(point A, point B, point C)
  ```

  Возвращает треугольник с вершинами A, B и C.
  ```

- `triangle triangleabc(real a, real b, real c, real angle=0, point A=(0,0))`

  Возвращает треугольник $ABC$ такой, что $BC = a, AC = b, AB = c$ и $(\vec{i}; \overrightarrow{AB})$ =`angle`.




```
import geometry;
size(4cm);
currentcoordsys=cartesiansystem(i=(1,0.5),
                                j=(-0.25,.75));
show(currentcoordsys);
triangle t=triangleabc(3,4,5, (1,1));
show(La="3", Lb="4", Lc="5", t);
```

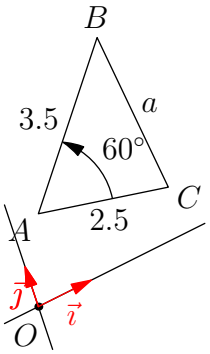
- `triangle triangleAbc(real alpha, real b, real c, real angle=0, point A=(0,0))`

  Возвращает треугольник $ABC$, в котором

$$\left(\overrightarrow{AB}; \overrightarrow{AC}\right) = \texttt{alpha}, \quad AC = b, \quad AB = c$$

  и $(\vec{i}; \overrightarrow{AB})$ =`angle`.

```
import geometry;
size(5cm);
currentcoordsys=cartesiansystem(i=(1,0.5),
                          j=(-0.25,.75));
show(currentcoordsys);
triangle t=triangleAbc(-60,2.5,3.5,
                       angle=45,(0.5,2));
show(Lb="2.5", Lc="3.5",t);
markangle("$60^\circ$",t.C,t.A,t.B, Arrow);
```



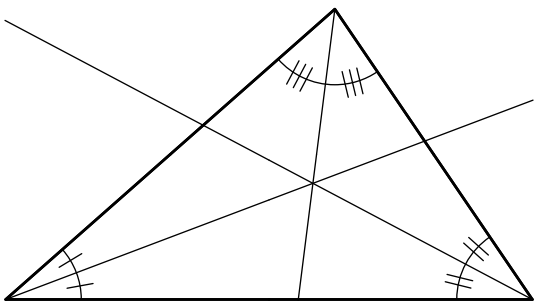

- `triangle triangle(line l1, line l2, line l3)`

  Возвращает треугольник со сторонами `l1`, `l2` и `l3`.

## 11.3 Вершины треугольника

Пусть `t` — объект типа `triangle`. Его свойства `t.VA`, `t.VB` и `t.VC` имеют тип `vertex` и описывают вершины треугольника `t`. Пакет *geometry.asy* допускает вызов процедур с передачей вершин треугольника в качестве параметров, без явного указания треугольника, к которому они относится.

Например, в следующем коде процедура `line bisector(vertex V, real angle=0)` возвращает образ внутренней биссектрисы, проходящей через `V`, при повороте вокруг центра `V` на угол поворота `angle`.



```
import geometry; size(7cm);
triangle t=triangleabc(4,5,6);
drawline(t, linewidth(bp));
line ba=bisector(t.VA), bb=bisector(t.VB);
line bc=bisector(t.VC); draw(ba^^bb^^bc);
markangle(t.AB, t.AC,
                StickIntervalMarker(2,1));
markangle(t.BC, t.BA,
                StickIntervalMarker(2,2));
markangle(t.CA, t.CB,
                StickIntervalMarker(2,3));
```

Вот несколько основных процедур и операторов для объектов типа `vertex`.

- `point operator cast(vertex V)`

  Позволяет привести `vertex` к типу `point`.

- `point point(explicit vertex V)`

  Возвращает объект типа `point` соответствующий объекту `V` типа `vertex`. Код `point(V)` эквивалентен коду `(point)V`, осуществляющему приведение `vertex` к типу `point`.

- `vector dir(vertex V)`

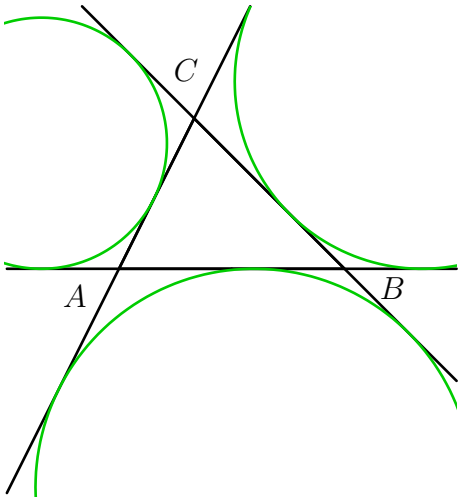
  Возвращает единичный вектор внутренней биссектрисы угла при вершине `V` и направленного наружу из вершины `V` треугольника. Эта процедура особенно полезна для расположения меток (подписей) в вершинах треугольника.

  Другие процедуры с объектом типа `vertex` в качестве параметра описаны в в следующем разделе, вместе с подпрограммами для треугольников.

## 11.4 Стороны треугольника

Пусть `t` — объект типа `triangle`. Его свойства `t.AB, t.BC, t.CA, t.BA, t.AC` и `t.CB`, типа `side` представляют стороны треугольника `t`. Пакет *geometry.asy* позволяет выполнять процедуры над сторонами треугольника, переданными в качестве параметров, без явного указания треугольника, к которому они относятся.

Как пример в следующем коде процедура `circle excircle(side s)` возвращает вневписанную окружность треугольника, которая относится к `s` и касается `s`.




```
import geometry;
size(6cm,0);
triangle t=triangle((-1,0), (2,0), (0,2));
drawline(t, linewidth(bp));
label(t,alignFactor=4);
clipdraw(excircle(t.AB), bp+0.8green);
clipdraw(excircle(t.BC), bp+0.8green);
clipdraw(excircle(t.AC), bp+0.8green);
draw(box((-2.5,-3), (3.5,3.5)), invisible);
```

Вот некоторые процедуры и основные операторы для объектов типа `side`:

- `line operator cast(side side)`

  Осуществляет приведение `side` к типу `line`.

- `line line(explicit side side)`

  Возвращает объект типа `line`, соответствующий объекту `side` типа `side`. Код `line(S)` эквивалентен коду `(line)S`, который рассматривает `side` как `line`.

- `segment segment(explicit side side)`

  Возвращает объект типа `segment`, соответствующий объекту `side` типа `side`. Код `segment(S)` эквивалентен коду `(segment)S`, который рассматривает `side` как `segment`.

- `side opposite(vertex V)`

  Возвращает противолежащую сторону к вершине `V` треугольника с вершиной `V`.

- `vertex opposite(side side)`

  Возвращает противолежащую вершину к стороне `side` в треугольнике, которому принадлежит `side`.

Другие процедуры с объектами типа `side` в качестве параметра описываются вместе с подпрограммами, касающимися треугольников.

## 11.5 Операторы

Единственным оператором, который может быть применен к `triangle` является

- `triangle operator *(transform T, triangle t)`

который делает возможным код `transform*triangle`.

## 11.6 Другие процедуры

- `point orthocentercenter(triangle t)`

  Возвращает ортоцентр треугольника `t`

- `point foot(vertex V)`

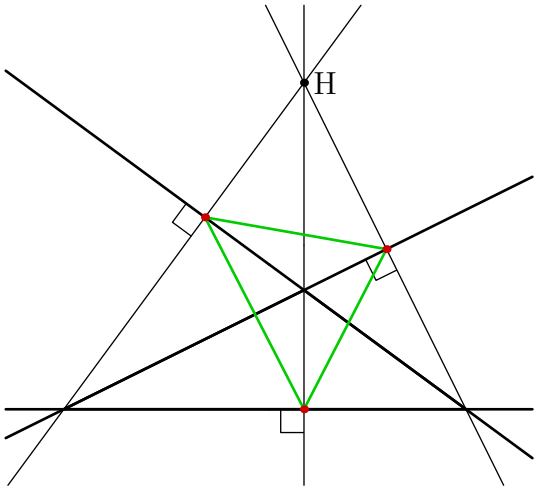
  Возвращает основание высоты из вершины `V`. Есть еще процедура `point foot(side side)`.

- `line altitude(vertex V)`

  Возвращает высоту из вершины `V`. Есть еще процедура `line altitude(side side)`.

- `triangle orthic(triangle t)`

  Возвращает ортоцентрический треугольник для `t`; его вершинами служат основания высот треугольника `t`.




```
size(8cm);
import geometry;
triangle t=triangleabc(3,4,6);
drawline(t, linewidth(bp));
line hc=altitude(t.AB), hb=altitude(t.AC);
line ha=altitude(t.BC); draw(hc^^hb^^ha);
dot("H", orthocentercenter(t));
perpendicularmark(t.AB,hc,quarter=-1);
perpendicularmark(t.AC,hb,quarter=-1);
perpendicularmark(t.BC,ha);
triangle ort=orthic(t);
draw(ort,bp+0.8*green); dot(ort, 0.8*red);
addMargins(1cm,1cm);
```

- `point midpoint(side side)`

  Возвращает середину стороны `side`

- `point centroid(triangle t)`

  Возвращает центр тяжести (центроид) треугольника `t`

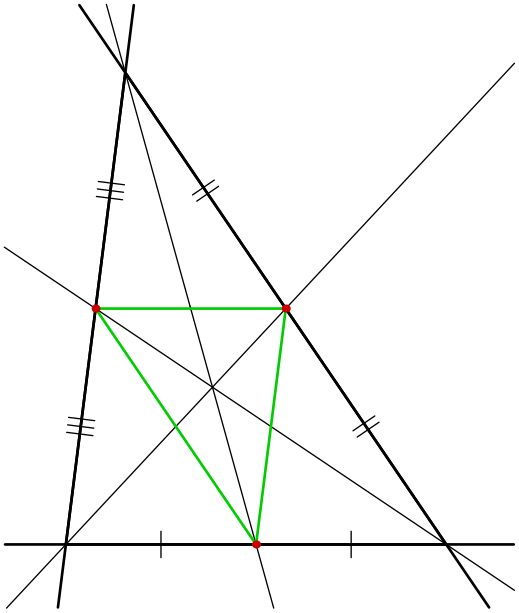
- `line median(vertex V)`

  Возвращает медиану из вершины `V`. Еще определена процедура `line median(side side)`

- `triangle medial(triangle t)`

  Возвращает медиальный треугольник к треугольнику `t`. (Его вершинами являются середины сторон треугольника `t`)

```
size(8cm);
import geometry;
triangle t=triangleabc(6,5,4);
drawline(t, linewidth(bp));
line ma=median(t.VA), mb=median(t.VB);
line mc=median(t.VC); draw(ma^^mb^^mc);
draw(segment(t.AB),
          StickIntervalMarker(2,1));
draw(segment(t.BC),
          StickIntervalMarker(2,2));
draw(segment(t.CA),
          StickIntervalMarker(2,3));
triangle med=medial(t);
draw(med,bp+0.8*green);
dot(med, 0.8*red);
addMargins(1cm,1cm);
```



- `triangle anticomplementary(triangle t)`

  Возвращает антикомплементарный треугольник треугольника `t` (`t` является медиальным треугольником для антикомплементарного).

- `line bisector(vertex V, real angle=0)`

  Возвращает изображение внутренней биссектрисы при вершине `V` при повороте вокруг `V` на угол `angle`. Примеры уже приводились.

- `point bisectorpoint(side side)`

  Возвращает точку пересечения стороны `side` с внутренней биссектрисой угла, противоположного стороне `side`.

- `line bisector(side side)`

  Возвращает срединный перпендикуляр к стороне `side`.

- `point circumcenter(triangle t)`

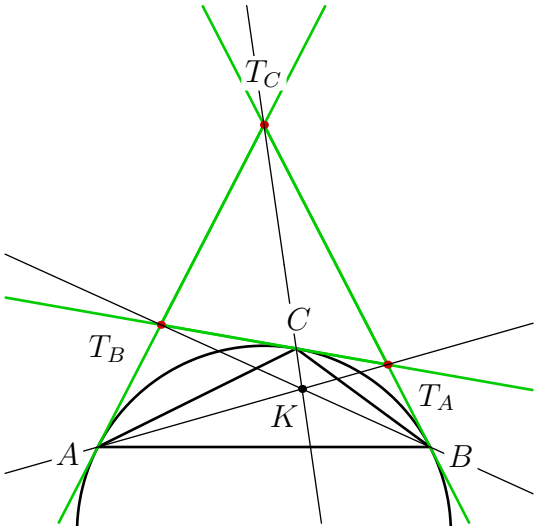
  Возвращает центр описанной окружности треугольника `t`.

- `circle circle(triangle t)`

  Возвращает описанную окружность треугольника `t`. Еще есть `circumcircle(triangle t)`.

- `triangle tangential(triangle t)`

  Возвращает касательный треугольник к треугольнику `t`. Он образован касательными к описанной окружности треугольника `t` в его вершинах.




```
size(7cm); import geometry;
triangle t=triangleabc(3,4,6);
draw(t, linewidth(bp));
clipdraw(circle(t), linewidth(bp));
triangle itr=tangential(t);
drawline(itr, bp+0.8*green);
dot(itr, 0.8*red);
line syma=line(itr.A,t.A),
             symb=line(itr.B,t.B);
line symc=line(itr.C,t.C);
draw(syma^^symb^^symc);
dot("$K$",
intersectionpoint(syma,symb), 2*dir(-120));
label(t,alignFactor=2,UnFill);
label("$T_A$","$T_B$","$T_C$",
         itr, alignFactor=4, UnFill);
addMargins(1cm,1cm);
```

- `point incenter(triangle t)`

  Возвращает центр окружности, вписанной в `t`.

- `real inradius(triangle t)`

  Возвращает радиус окружности, вписанной в `t`.

- `circle incircle(triangle t)`

  Возвращает окружность, вписанную в `t`.

- `triangle intouch(triangle t)`

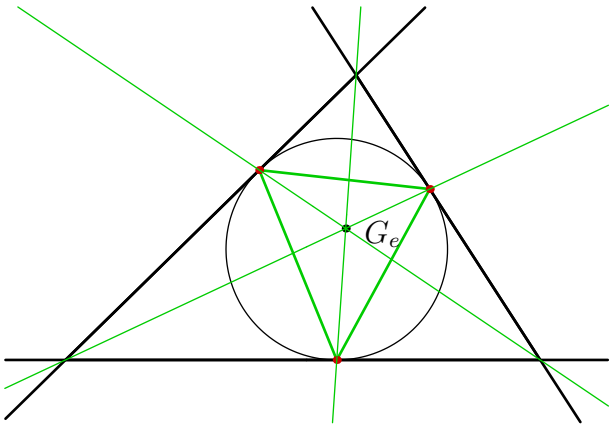
  Возвращает контактный треугольник, вершинами которого являются точки, в которых вписанная окружность касается данного треугольника `t`.

- `point intouch(side side)`

  Возвращает точку на стороне `side`, в которой вписанная окружность касается стороны `side`.

- `point gergonne(triangle t)`

  Возвращает точку Жергона треугольника `t`.




```
size(8.5cm,0);import geometry;
triangle t=triangleabc(5,6,7);
drawline(t, linewidth(bp));
draw(incircle(t));
triangle itr=intouch(t);
draw(itr,bp+0.8*green);
dot(itr, 0.8*red);
point Ge=gergonne(t);
dot("$G_e$", Ge, 2*dir(-10));
draw(line(Ge,t.A), 0.8*green);
draw(line(Ge,t.B), 0.8*green);
draw(line(Ge,t.C), 0.8*green);
addMargins(1cm,1cm);
```

- `point excenter(side side)`

  Возвращает центр вневписанной окружности к стороне `side`

- `real exradius(side side)`

  Возвращает радиус вневписанной окружности к стороне `side`.

- `circle excircle(side side)`

  Возвращает вневписанную кружность, касающуюся стороны `side`.

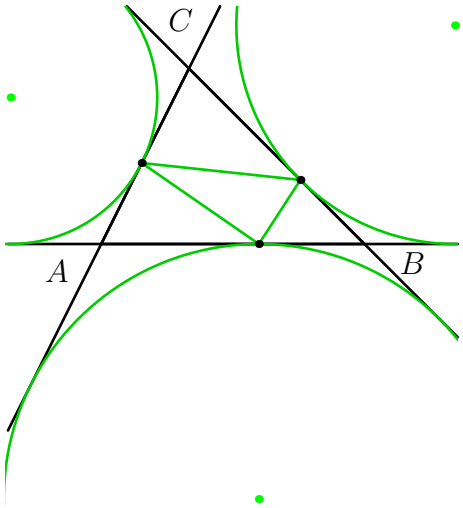
- `triangle extouch(triangle t)`

  Возвращает «внеконтактный» треугольник треугольника `t`, вершинами которого служат точки касания сторон с вневписанными окружностями.

- `point extouch(side side)`

  Возвращает точку касания стороны `side` с вневписанной окружностью, `excircle(side)`.

```
import geometry; size(6cm,0);
triangle t=triangle((-1,0),
                    (2,0), (0,2));
drawline(t, linewidth(bp));
label(t,alignFactor=4);
circle c1=excircle(t.AB),
              c2=excircle(t.BC);
circle c3=excircle(t.AC);
clipdraw(c1, bp+0.8green);
clipdraw(c2, bp+0.8green);
clipdraw(c3, bp+0.8green);
dot(c1.C^^c2.C^^c3.C, green);
draw(extouch(t), bp+0.8green, dot);
```




- `point symmedian(triangle t)`

  Возвращает точку пересечения симмедиан (называемую точкой Lemoine) треугольника `t`.

- `point symmedian(side side)`

  Возвращает точку симмедианы стороны `side`.

- `line symmedian(vertex V)`

  Возвращает симмедиану, проходящую через вершину `V`.

```
import geometry; size(10cm,0);
triangle t=triangle((-1,0), (2,0), (0,2));
drawline(t, linewidth(bp));
label(t,alignFactor=2, alignAngle=90);
triangle st=symmedial(t);
draw(st, bp+0.8green);
label("$A'$", "$B'$", "$C'$", st, alignAngle=45, 0.8green);
line mA=median(t.VA);
draw(mA, blue); dot("$M_A$",midpoint(t.BC), 1.5E, blue);
draw(segment(t.BC), bp+blue, StickIntervalMarker(2,2,blue));
line bA=bisector(t.VA);
draw(bA, grey); dot("$B_A$", bisectorpoint(t.BC));
line sA=symmedian(t.VA);
draw(sA, 0.8*green);
draw(symmedian(t.VB), 0.8*green);
```

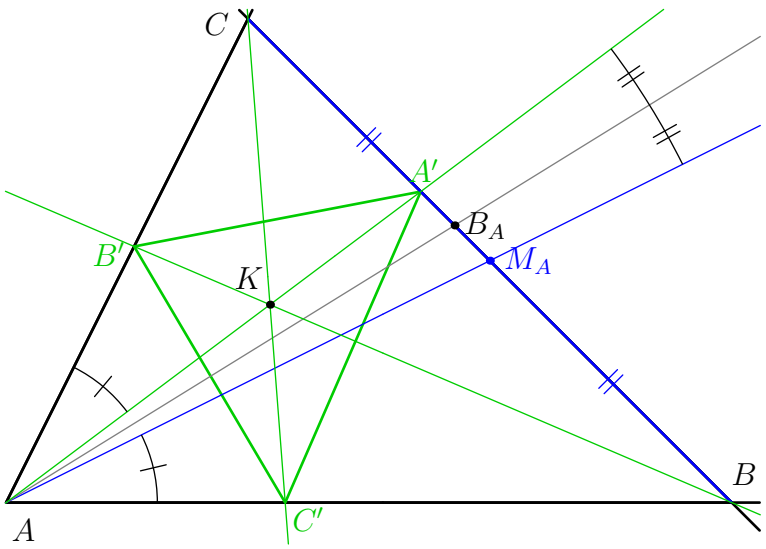



Рис. 11.1:

```
draw(symmedian(t.VC), 0.8*green);
point sP=symmedian(t);
dot("$K$", sP, 2*dir(125));
markangle(sA, t.AC, radius=2cm, StickIntervalMarker(1,1));
markangle(t.AB, mA, radius=2cm, StickIntervalMarker(1,1));
markangle(mA, sA, radius=10cm, StickIntervalMarker(2,2));
```

- `point cevian(side side, point P)`

  Возвращает точку Чевы для точки `P`, принадлежащую стороне `side`.

- `triangle cevian(triangle t, point P)`

  Возвращает треугольник Чевы для точки `P`.

- `line cevian(vertex V, point P)`

  Возвращает прямую Чевы, проходящую через точку `P` и вершину `V`.

  На рисунке 11.2 иллюстрируется утверждение: *Если $\triangle A'B'C'$ является треугольником Чевы для $\triangle ABC$, то $\triangle A''B''C''$, вершины которого симметричны $A'$ , $B'$ и $C'$ относительно середин соответствующих сторон $\triangle ABC$, также является треугольником Чевы для $\triangle ABC$.*

```
import geometry; size(10cm,0);
triangle t=triangleabc(5,6,7); label(t); draw(t, linewidth(bp));
point P=0.6*t.B+0.25*t.C; dot("$P$", P, dir(60), 0.8*red);
triangle C1=cevian(t, P);
label("$A^{\prime}$","$B^{\prime}$","$C^{\prime}$", C1, 0.8*red);
draw(C1, bp+0.8*red, dot);
draw(t.A--C1.A, 0.8*red); draw(t.B--C1.B, 0.8*red); draw(t.C--C1.C, 0.8*red);
point Ma=midpoint(t.BC), Mb=midpoint(t.AC), Mc=midpoint(t.BA);
dot("$M_1$",Ma,-dir(t.VA)); dot("$M_2$",Mb,-dir(t.VB)); dot("$M_3$",Mc,-dir(t.VC));
draw(t.A--Ma^^t.B--Mb^^t.C--Mc, grey); dot("$G$", centroid(t), 2*dir(-65), grey);
point App=rotate(180,Ma)*C1.A, Bpp=rotate(180,Mb)*C1.B, Cpp=rotate(180,Mc)*C1.C;
draw(C1.A--App, 0.8*green, StickIntervalMarker(2,1,0.8*green));
draw(C1.B--Bpp, 0.8*green, StickIntervalMarker(2,2,0.8*green));
draw(C1.C--Cpp, 0.8*green, StickIntervalMarker(2,3,0.8*green));
triangle C2=triangle(App,Bpp,Cpp);
```

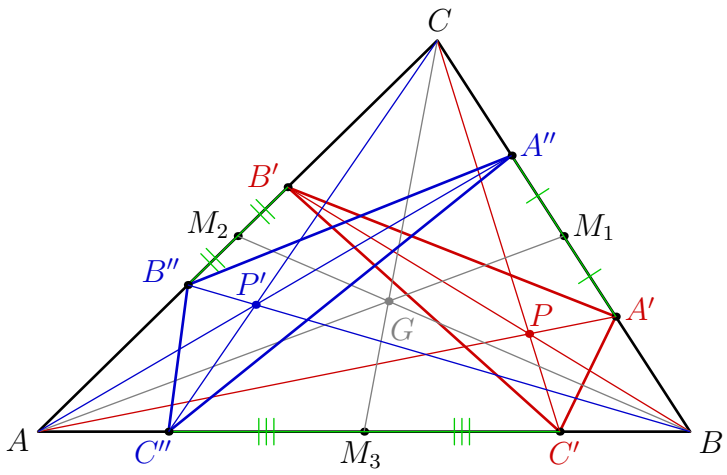



Рис. 11.2:

```
label("$A^{\prime\prime}$","$B^{\prime\prime}$","$C^{\prime\prime}$", C2, 0.8*blue);
draw(C2, bp+0.8*blue, dot);
segment sA=segment(t.A,C2.A), sB=segment(t.B,C2.B);
point PP=intersectionpoint(sA,sB);
dot("$P^{\prime}$", PP, dir(100), 0.8*blue);
draw(sA, 0.8*blue); draw(sB, 0.8*blue); draw(segment(t.C,C2.C), 0.8*blue);
shipout(bbox(2mm));
```

- `line isotomic(vertex V, point M)`

  Возвращает изотомическую прямую (isotomic line), проходящую через `V`, относительно `M` в треугольнике с вершиной `V`.

- `point isotomicconjugate(triangle t, point M)`

  Возвращает точку, изотомически сопряженную с `M` относительно `t`.

- `point isotomic(side side, point M)`

  Возвращает точку пересечения изотомической прямой относительно `M`, в треугольнике со стороной `side`, со стороной `side`.

- `triangle isotomic(triangle t, point M)`

  Возвращает треугольник, вершинами которого служат точки пересечения изотомических линий относительно `M` в `t` со сторонами `t`. Так, на рис. 11.2 треугольник $\triangle A''B''C''$ является изотомическим для $\triangle ABC$ относительно `P`.

  На рис. 11.3 треугольник получен с использованием процедуры `isotomic`.

```
import geometry; size(10cm,0);
triangle t=triangleabc(5,6,7); label(t); draw(t, linewidth(bp));
point P=0.6*t.B+0.25*t.C; dot("$P$", P, dir(60), 0.8*red);
draw(segment(isotomic(t.VA,P))^^segment(isotomic(t.VB,P))^^segment(isotomic(t.VC,P)),
0.8*blue);
draw(segment(cevian(t.VA,P))^^segment(cevian(t.VB,P))^^segment(cevian(t.VC,P)),
0.8*red);
triangle t1=cevian(t,P); label("$P_1$", "$P_2$", "$P_3$", t1); draw(t1, bp+0.8*red);
triangle t2=isotomic(t,P); label("$Q_1$", "$Q_2$", "$Q_3$", t2);
draw(t2, bp+0.8*blue);
dot("$Q$", isotomicconjugate(t,P), dir(100), 0.8*blue);
```

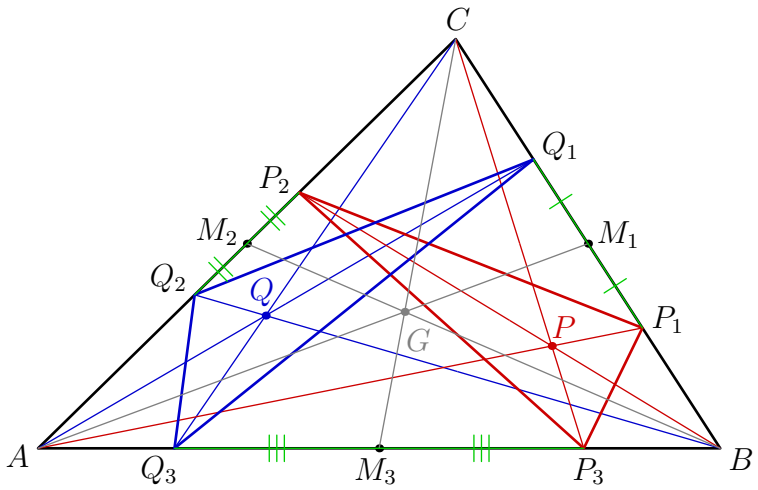



Рис. 11.3:

```
point Ma=midpoint(t.BC), Mb=midpoint(t.AC), Mc=midpoint(t.BA);
dot("$M_1$",Ma,-dir(t.VA)); dot("$M_2$",Mb,-dir(t.VB)); dot("$M_3$",Mc,-dir(t.VC));
draw(t.A--Ma^^t.B--Mb^^t.C--Mc, grey); dot("$G$", centroid(t), 2*dir(-65), grey);
draw(t1.A--t2.A, 0.8*green, StickIntervalMarker(2,1,0.8*green));
draw(t1.B--t2.B, 0.8*green, StickIntervalMarker(2,2,0.8*green));
draw(t1.C--t2.C, 0.8*green, StickIntervalMarker(2,3,0.8*green));
```

- `point isogonalconjugate(triangle t, point M)`

  Возвращает точку, изогонально сопряженную к `M` в `t`.

- `point isogonal(side side, point M)`

  Возвращает точку пересечения изогональной прямой к `M`, со стороной `side`.

- `triangle isogonal(triangle t, point M)`

  Возвращает треугольник, вершинами которого служат точки точки пересечения сторон `t` с изогональными прямыми к `M` в `t`.

  В следующем примере иллюстрируется свойство: Педальные треугольники двух изогональных точек `P` и `Q` принадлежат одной окружности с центром в середине сегмента `[P Q]`.

```
import geometry; size(10cm,0);
triangle t=triangleabc(5,6,7); draw(t, linewidth(bp));
point P=0.5*t.B+0.3*(t.C-t.B); dot("$P$", P, N, 0.8*red);
point Q=isogonalconjugate(t,P); dot("$Q$", Q, dir(-30));
point Q1=projection(t.AB)*Q; segment sq1=segment(Q,Q1);
point Q2=projection(t.BC)*Q; segment sq2=segment(Q,Q2);
point Q3=projection(t.AC)*Q; segment sq3=segment(Q,Q3);
draw(sq1); draw(sq2); draw(sq3);
dot("$Q_1$", Q1, SE); dot("$Q_2$", Q2); dot("$Q_3$", Q3, NW);
point P1=projection(t.AB)*P; segment sp1=segment(P,P1);
point P2=projection(t.BC)*P; segment sp2=segment(P,P2);
point P3=projection(t.AC)*P; segment sp3=segment(P,P3);
draw(sp1, 0.8*red); draw(sp2, 0.8*red); draw(sp3, 0.8*red);
dot("$P_1$",P1,SW,0.8*red); dot("$P_2$",P2,0.8*red); dot("$P_3$",P3,NW,0.8*red);
perpendicularmark(t.AB,sq1); perpendicularmark(t.BC,sq2);
perpendicularmark(reverse(t.AC),sq3); perpendicularmark(t.AB,sp1, red);
perpendicularmark(t.BC,sp2, red); perpendicularmark(reverse(t.AC),sp3, red);
```

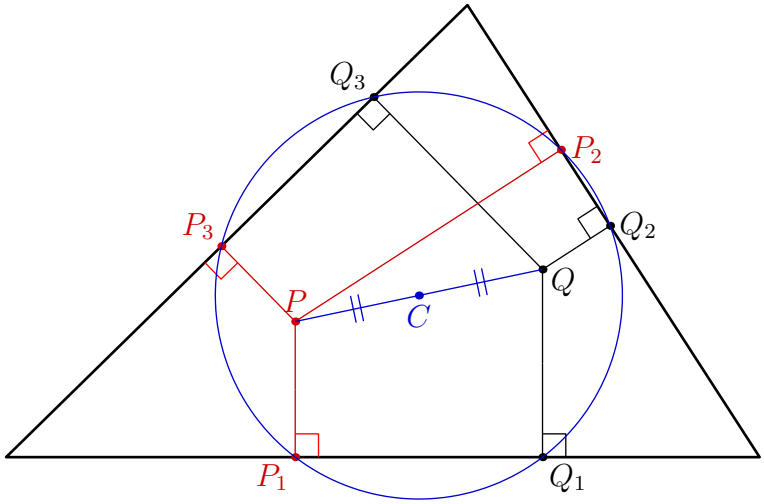



Рис. 11.4:

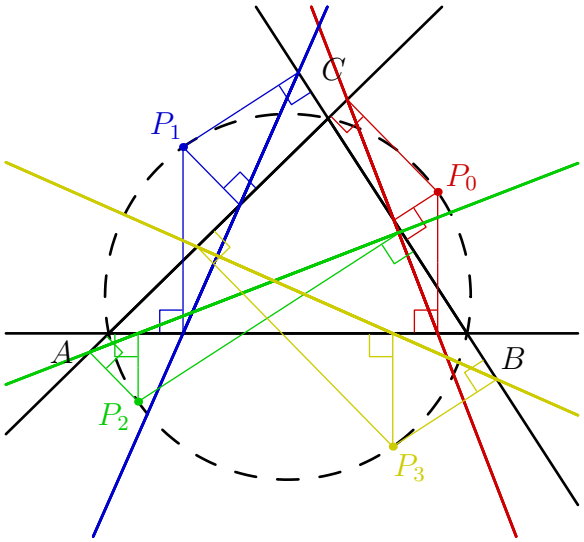
```
circle C=circle(Q1,Q2,Q3); draw(C, 0.8*blue);
draw(segment(Q,P), 0.8*blue, StickIntervalMarker(2,2, 0.8*blue));
dot("$C$", C.C, S, 0.8*blue);
```

- `triangle pedal(triangle t, point M)`

  Возвращает педальный треугольник точки `M` в `t`.

- `line pedal(side side, point M)`

  Возвращает прямую, проходящую через `M` и через основание перпендикуляра из `M` к прямой `side`. В следующем примере показано несколько прямых Симсона (Simson); Заметим, что использование методов `t.side(int)` и `t.vertex(int)` позволяет обращаться к сторонам и вершинам треугольника `t` по их номерам.

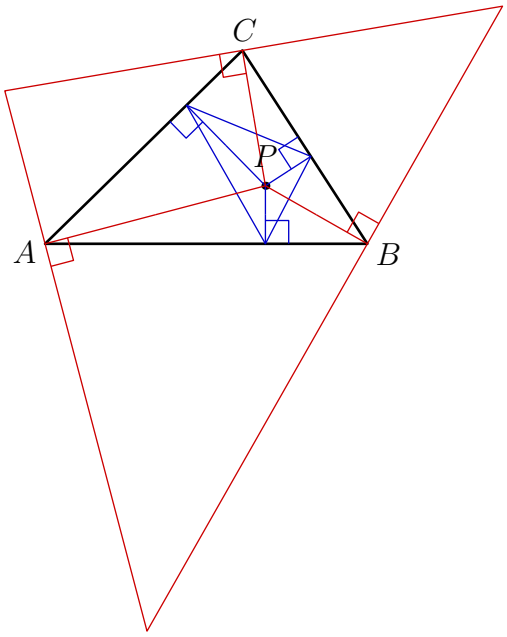



```
import geometry; size(8cm,0);
triangle t=triangleabc(5,6,7);
label(t, alignFactor=4);
drawline(t, linewidth(bp));
circle C=circle(t); draw(C, bp+dashed);
pen[] p=new pen[] {0.8*red,0.8*blue,
0.8*green, 0.8*yellow};
for (int i=0; i < 4; ++i) {
real x=35+i*90; point P=angpoint(C,x);
dot("$P_"+(string)i+"$",P,dir(x),p[i]);
for (int j=1; j < 4; ++j) {
segment Sg=segment(pedal(t.side(j),P));
draw(Sg,p[i]);
markrightangle(P,Sg.B,t.vertex(j),p[i]);
}
drawline(pedal(t,P), bp+p[i]);
}
addMargins(1cm,1cm);
shipout(bbox(2mm));
```

- `triangle antipedal(triangle t, point M)`

  Возвращает треугольник, педальным к которому относительно `M` является `t`.

```
import geometry; size(7cm,0);
triangle t=triangleabc(5,6,7);
label(t); draw(t, linewidth(bp));
point P=0.5*t.B+0.3*t.C;
dot("$P$", P, 2*dir(60));
triangle Pt=pedal(t,P);
currentpen=0.8*blue; draw(Pt);
segment psA=segment(P,Pt.A);
segment psB=segment(P,Pt.B);
segment psC=segment(P,Pt.C);
draw(psA); draw(psB); draw(psC);
perpendicularmark(t.BC,psA);
perpendicularmark(t.CA,psB);
perpendicularmark(t.AB,psC);
triangle APt=antipedal(t, P);
currentpen=0.8*red; draw(APt);
segment apsA=segment(P,t.A);
segment apsB=segment(P,t.B);
segment apsC=segment(P,t.C);
draw(apsA); draw(apsB); draw(apsC);
perpendicularmark(APt.BC,apsA);
perpendicularmark(APt.CA,apsB);
perpendicularmark(APt.AB,apsC);
currentpen=0.8*black;
shipout(bbox(2mm));
```




## 11.7 Трилинейные координаты

Тип `trilinear`, структура которого приведена далее, позволяет инициализировать объект, представляющий трилинейные координаты `a:b:c` по отношению к треугольнику `t`.

```
struct trilinear
       { real a,b,c; triangle t; }
```

Определение трилинейных координат `a:b:c` относительно треугольника `t` производится при помощи функции:

```
trilinear trilinear(triangle t, real a, real b, real c)
```

Трилинейные координаты точки можно найти функцией:

```
trilinear trilinear(triangle t, point M)
```

Еще возможно определение трилинейных координат с помощью triangle center function $f$ и трех параметров `a, b` и `c`, используя функцию:

```
trilinear trilinear(triangle t, centerfunction f,
real a=t.a(), real b=t.b(), real c=t.c())
```

где тип `centerfunction` есть действительная функция трех действительных переменных.

Приведение объекта типа `trilinear` к типу `point` можно сделать двумя способами: функцией `point(trilinear)` или синтаксическим приведением `(point) trilinear`.

Например, трилинейные координаты изотомически сопряженной точки определяем функцией `isotomicconjugate`:

```
point isotomicconjugate(triangle t, point M)
{
trilinear tr=trilinear(t,M);
return point(trilinear(t,1/(t.a()^2*tr.a),1/(t.b()^2*tr.b),1/(t.c()^2*tr.c)));
}
```

# 12 Инверсии

Тип `inversion`, структура которого приведена ниже, позволяет инициализировать инверсию с центром `C` и радиусом `k`

```
struct inversion
       {
            point C; real k;
       }
```

## 12.1 Определение инверсии

Для определения инверсий служат следующие подпрограммы и операторы:

- `inversion inversion(real k, point C)`

  Возвращает инверсию с центром `C` и радиусом `k`. Также существует функция `inversion(point C, real k)`.

- `inversion inversion(circle c1, circle c2, real sgn=1)`

  – Если `sgn` отлично от нуля, функция возвращает инверсию, знак радиуса которой совпадает со знаком `sgn` и которая переводит `c1` в `c2`;

  – Если `sgn` равно нулю, функция возвращает инверсию, с центром в основании радикальной оси и оставляющей инвариантными каждую из двух окружностей `c1` и `c2`.

  Пример использования этой процедуры уже приводился.

- `inversion inversion(circle c1, circle c2, circle c3)`

  Возвращает инверсию, оставляющую инвариантными окружности `c1, c2` и `c3`.

- `circle operator cast(inversion i)`

  Позволяет привести объект типа `inversion` к типу `circle`. Возвращаемая окружность является главной(базисной) окружностью инверсии `i`.

  Это же делает функция `circle circle(inversion i)`.

- `inversion operator cast(circle c)`

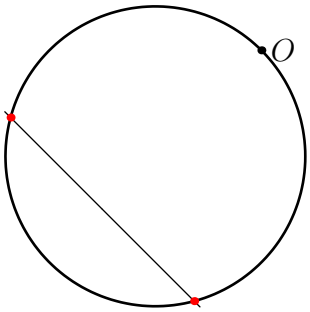
  Задает окружность `circle` как базисную для `inversion`. Возвращаемая инверсия оставляетокружность `c` инвариантной, с центром в центре `c` и радиусом, равным радиусу `c`. Это же делает процедура `inversion inversion(circle c)`.

## 12.2 Применение инверсии

Следующие операторы реализуют действие `inversion*object`, которое возвращает образ `object` при `inversion`.

- `point operator *(inversion i, point P)`
- `circle operator *(inversion i, line l)`
- `circle operator *(inversion i, circle c)`
- `arc operator *(inversion i, segment s)`
- `path operator *(inversion i, triangle t)`

Следует отметить, что образом окружности при инверсии может быть прямая. В этом случае возвращается окружность `C` с бесконечным радиусом и свойством `C.l`, типа `line`, что является корректным; процедура признающая эту окружность, как параметр, использует `C.l`, как показано на следующем примере.



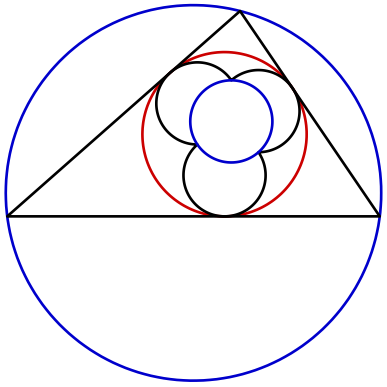

```
settings.outformat="pdf";
import geometry;
size(4cm);point C0=(0,0);
circle C=circle(C0,1);
draw(C, linewidth(bp));
point O=dir(45);
dot("$O$",O);
inversion inv=inversion(3.0,O);
circle Cp=inv*C;
draw(Cp);
dot(intersectionpoints(C,Cp), red);
```

## 12.3 Примеры

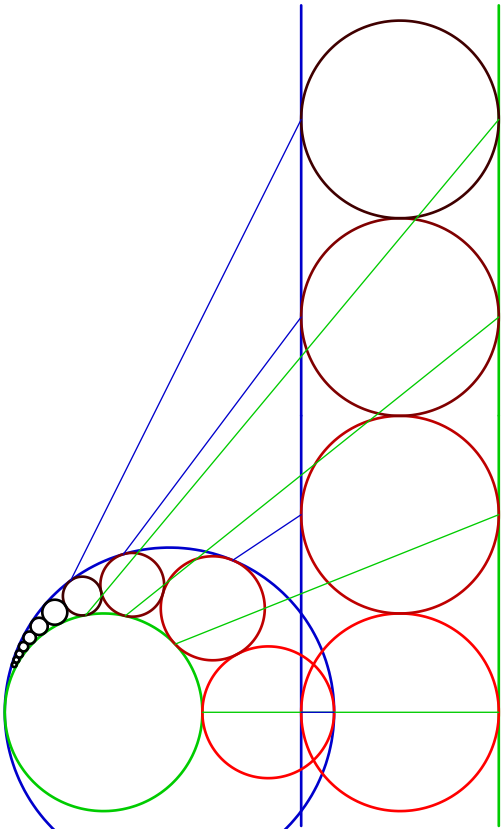
Примеры использования инверсии уже приводились, вот некоторые другие.

Мы начнем с иллюстрации использования окружности, вписанной в треугольник, как базисной окружности инверсии:

```
import geometry;
size(5cm,0);
triangle t=triangleabc(4,5,6);
circle C=circumcircle(t),
             inC=incircle(t);
draw(inC, bp+0.8*red);
draw(C, bp+0.8*blue);
draw(t, linewidth(bp));
draw(inC*t, linewidth(bp));
draw(inC*C, bp+0.8*blue);
```

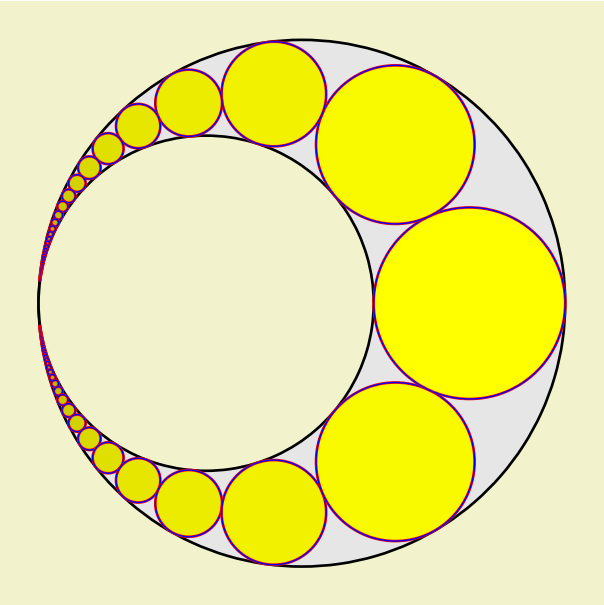


Ниже приводится конструкция цепочки Паппа (Pappus).



```
settings.outformat="pdf";
import geometry;
size(7cm,0);
point I=(-4,0);
dot(I,invisible);
inversion inv=inversion(10.0,I);
line l1=line((-1,0),(-1,1)),
l2=line((1,0),(1,1));
draw(l1, bp+0.8*blue);
draw(l2, bp+0.8*green);
clipdraw(inv*l1,bp+0.8*blue);
clipdraw(inv*l2,bp+0.8*green);
int n=10;
for (int i=0; i <= n; ++i) {
circle C=circle((point)(0,2*i),1);
circle Cp=inv*C;
draw((path)Cp,bp+(1-abs(i/4))*red);
if(abs(i) < 4){
draw((path)C,bp+(1-abs(i/4))*red);
draw((1,2*i)--inv*(1,2*i),0.8*green);
draw((-1,2*i)--inv*(-1,2*i),0.8*blue);}}
addMargins(2mm,2mm);
```

Можно легко получить хорошее представление:



```
settings.outformat="pdf";
import geometry;
size(8cm);
inversion inv=inversion(1,(-4.5,0));
path g1=inv*line((-1,0),(-1,1)),
g2=inv*line((1,0),(1,1));
fill(g1,lightgrey);
draw(g1,linewidth(bp));
unfill(g2);
draw(g2,linewidth(bp));
int n=40;
for (int i=-n; i <= n; ++i){
circle g=inv*circle((point)(0,2*i),1.0);
fill((path)g,(1-(abs(i)/n))*yellow);
draw(g,bp+red);
draw(g,blue);
}
shipout(bbox(5mm,
        Fill(rgb(0.95,0.95,0.8))));
```

Следующий рисунок, вариант, где линии не параллельны:

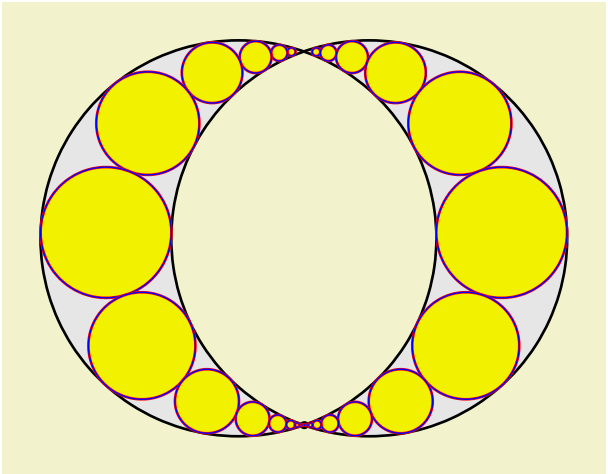


Рис. 12.1:

```
import geometry; size(8cm,0);
point P=(0,-4.5); dot(P); inversion inv=inversion(1,P);
line l1=line((0,0),(1,0.35)), l2=line((0,0),(-1,0.35));
path g1=inv*l1, g2=inv*l2;
fill(g1^^g2,evenodd+lightgrey); draw(g1,linewidth(bp)); draw(g2,linewidth(bp));
for (int i:new int[]{-1,1}) {
point P=(i*0.1,0); triangle t=triangle(shift(P)*vline,l1,l2); int n=15;
for (int j=0; j <= n; ++j) {
circle C=excircle(t.AB);
t=triangle(shift(angpoint(C,(i-1)*90))*vline,l1,l2);
circle Cp=inv*C; path g=Cp; fill(g,0.95*yellow); draw(g,bp+red); draw(g,blue); }}
shipout(bbox(5mm,Fill(rgb(0.95,0.95,0.8))));
```

На следующем рисунке представлен образ шахматной доски $12 \times 12$ при инверсии, центр которой совпадает с центром доски и с радиусом 1.

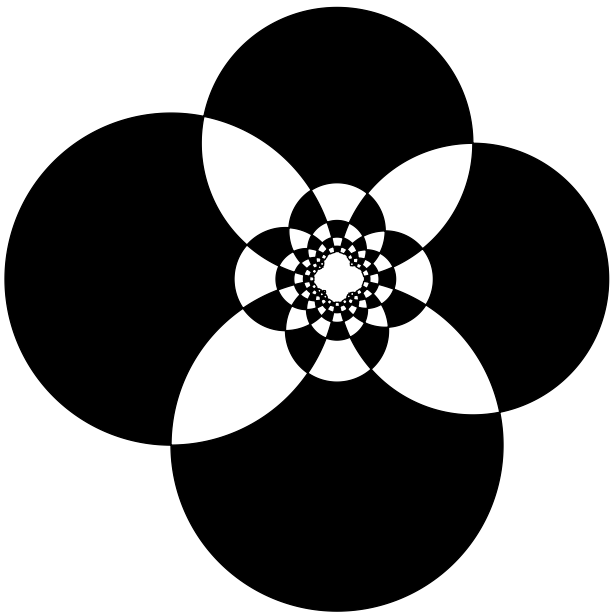


Рис. 12.2:

```
import geometry;
size(8cm,0);
int n=12; segment[] S;
inversion inv=inversion(1,(n/2+0.45,n/2+0.45));
transform tv=shift(0,1), th=shift(1,0);
for (int i=0; i < n; ++i)
for (int j=0; j < n; ++j) {
for (int l=0; l < 4 ; ++l)
S[l]=segment(point(tv^i*th^j*unitsquare,l), point(tv^i*th^j*unitsquare,(l+1)%4));
path g;
for (int l=0; l < 4; ++l) g=g--(path)(inv*S[l]);
g=g--cycle;
if((i+j)%2 == 0) draw(g); else fill(g);
}
```

# Предметный указатель