Задача 3

Условный оператор + функции + классы + перечисления + декомпозиция задачи (разбиение на подзадачи) + формальный подход.

На изображниях ниже (для каждого варианта свое) приводится часть координатной плоскости ($-10 \le x \le 10,\ -10 \le y \le 10$) с графиками функций (парабола, линия) и фигурами (круг, прямоугольник). Размер одной клетки сетки – 1 (единица). Таким образом параметры функций и фигур можно однозначно определить из рисунка.

Области, на которые разбивается плоскость графиками и контурами фигур, закрашены разными цветами (белый, серый, желтый, оранжевый, голубой, зеленый). Необходимо реализовать программу, которая для точки (x, y) определяет цвет области, в которую данная точка попадает. Будем считать, что если точка (x, y) попадает на линию графика функции или контура фигуры, то правильным ответом будет цвет любой соседней области. Также стоит заметить, что некоторые комбинации линий предполагают их пересечение за пределами изображенного на рисунке фрагмента плоскости, поэтому, чтобы исключить неопределенность ответа, допустимыми значениями (x, y) являются только такие, которые попадают в приведенное изображение.

В программе должна быть реализована следующая функция (статический метод), которая для точки (x, y) возвращает цвет точки на изображении:

```
public static SimpleColor getColor(double x, double y) {
    // реализовать самостоятельно
    ...
}
```

Как видно из заголовка функции, она должна возвращать значение типа SimpleColor – перечисления (enum) вида (описывается в отдельном файле: New → Java Class → Enum):

```
/**
 * Перечисление для цветов
 */
public enum SimpleColor {
    BLACK,
    WHITE,
    GRAY,
    RED,
    ORANGE,
    YELLOW,
    GREEN,
    BLUE;
```

```
}
```

Для декомпозиции и упрощений задачи необходимо создать отдельные классы (в отдельных файлах), которые будут описывать встречающиеся геометрические фигуры и будут позволять определить положение точки относительно данной фигуры. Ниже приводится описание класса для линии и для «горизонтальной» параболы (для варианта 1):

```
/**
 * Линия вида y = a * (x - x0) + y0
 * (можно обойтись без y0, но с ним удобнее;
 *  будем считать, что линия всегда наклонная)
 */
public class Line {
    public double x0;
    public double y0;
    public double a;

    public Line(double x0, double y0, double a) {
        this.x0 = x0;
        this.y0 = y0;
        this.a = a;
    }

    /**
     * Проверяет, находится ли точка (x, y) выше линии
     */
    public boolean isPointAboveLine(double x, double y) {
        return y > a * (x - x0) + y0;
    }
}
```

«Горизонтальная» парабола:

```
/**
 * "Горизонтальная" парабола вида x = a * (y - y0) + x0
 */
public class HorizontalParabola {
    public double x0;
    public double y0;
    public double a;

    public HorizontalParabola(double x0, double y0, double a) {
        this.x0 = x0;
        this.y0 = y0;
        this.a = a;
    }

    /**
     * Проверяет, находится ли точка (x, y) справа
```

```
     * (сверху, если повернуть изобюражение на 90 градусов
     * против часовой стрелки)
     * от параболы
     */
    public boolean isPointRightOfParabola(double x, double y) {
        return x > a * Math.pow(y - y0, 2) + x0;
    }
}
```

Для других вариантов может потребоваться описать классы круг, обычная («вертикальная») парабола и прямоугольник.

Используя описанные классы мы можем создать объекты, которые будут представлять конкретные фигуры из определенного варианта задачи (создаются в основном классе) и используя созданные объекты реализовать функцию (статический метод) getColor:

```
public class Program {

    // Соответствуют первому варианту задачи
    public static final Line L1 = new Line(-1, 6, 5.0 / 2);
    public static final HorizontalParabola P1 =
            new HorizontalParabola(2, 2, 1);

    public static SimpleColor getColor(double x, double y) {
        // реализовать самостоятельно
        if (P1.isPointRightOfParabola(x, y)) {
            return SimpleColor.ORANGE;
        }
        if (y < -2 && L1.isPointAboveLine(x, y)) {
            return SimpleColor.GREEN;
        }
        if (y < -2 && !L1.isPointAboveLine(x, y)) {
            return SimpleColor.YELLOW;
        }
        return SimpleColor.GRAY;
    }

    ...
```

В методе Main необходимо для несколько заранее выбранных точек на рисунке (обязательно должны быть охвачены все области, на которые разбивается плоскость) распечатать результат, который возвращает getColor (и самостоятельно убедиться, что данная функция работает корректно). Также программа должна запрашивать координаты одной произвольной точки и для нее таким же образом печатать результат.

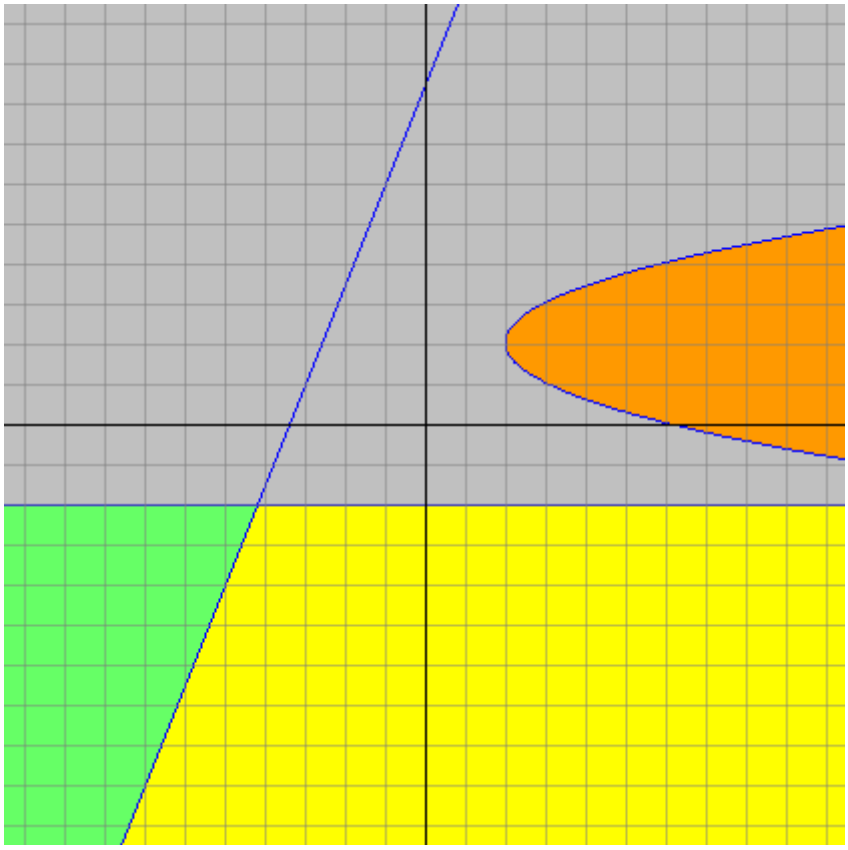
Для печати результата функции для точки (x, y) реализовать функцию:

```
public static void printColorForPoint(double x, double y) {
    ...
}
```

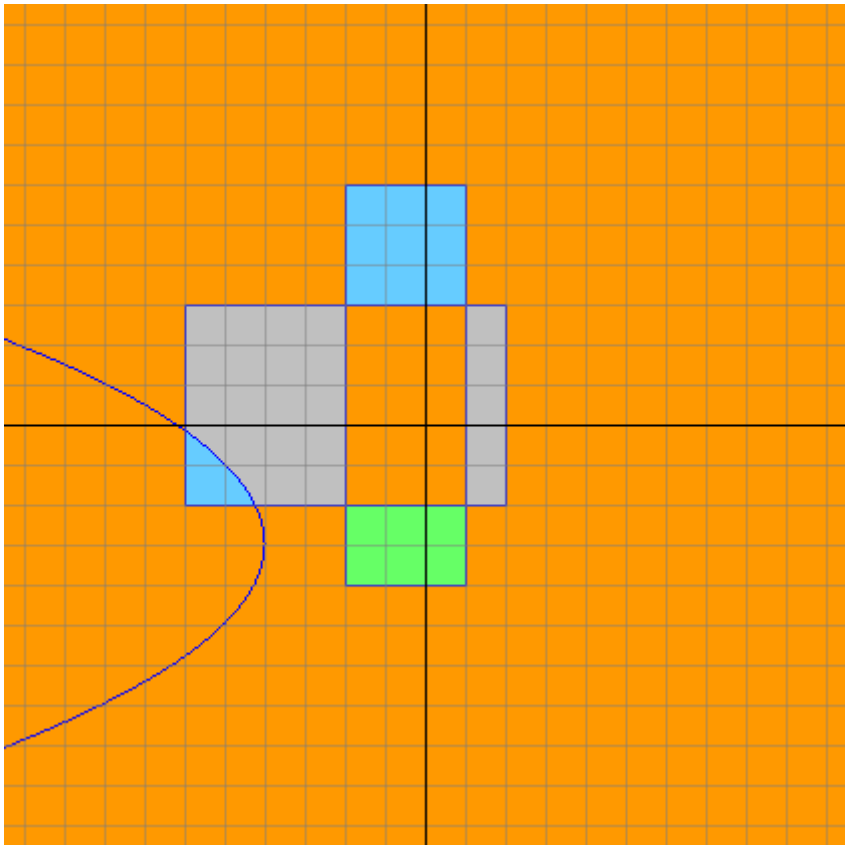
Пример вывода работы программы для варианта 1 приведен ниже:

```
T:\Java\jdk-12.0.2\bin\java.exe "-javaagent:T:\Java\intellij-idea-community-portable-win64\app\lib\idea_rt.jar=49637:T:\Java\intellij-idea-community-portable-win64\app\bin" -Dfile.encoding=UTF-8 -classpath T:\Java\Projects\Task3Sample\out\production\Task3Sample ru.vsu.cs.course1.task3.Main
(1.0, 1.0) -> GRAY
(5.0, 3.0) -> ORANGE
(-6.0, -6.123) -> GREEN
(-5.0, -4.5) -> YELLOW
(3.0, -3.0) -> YELLOW
Input X: 7
Input Y: 0
(7.0, 0.0) -> ORANGE
```

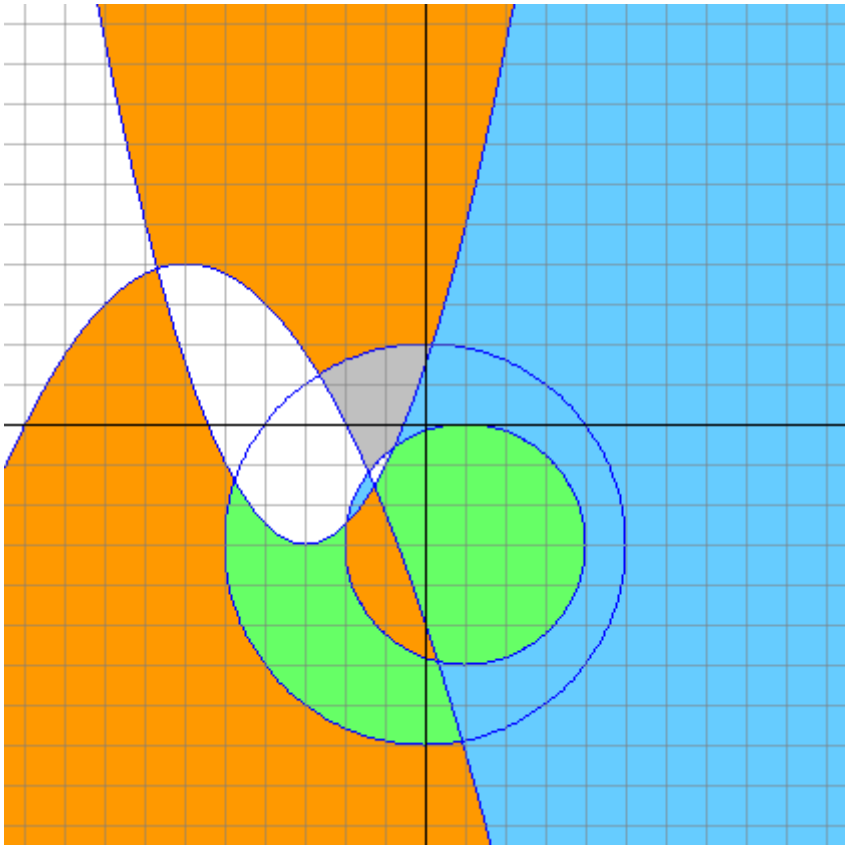
1.



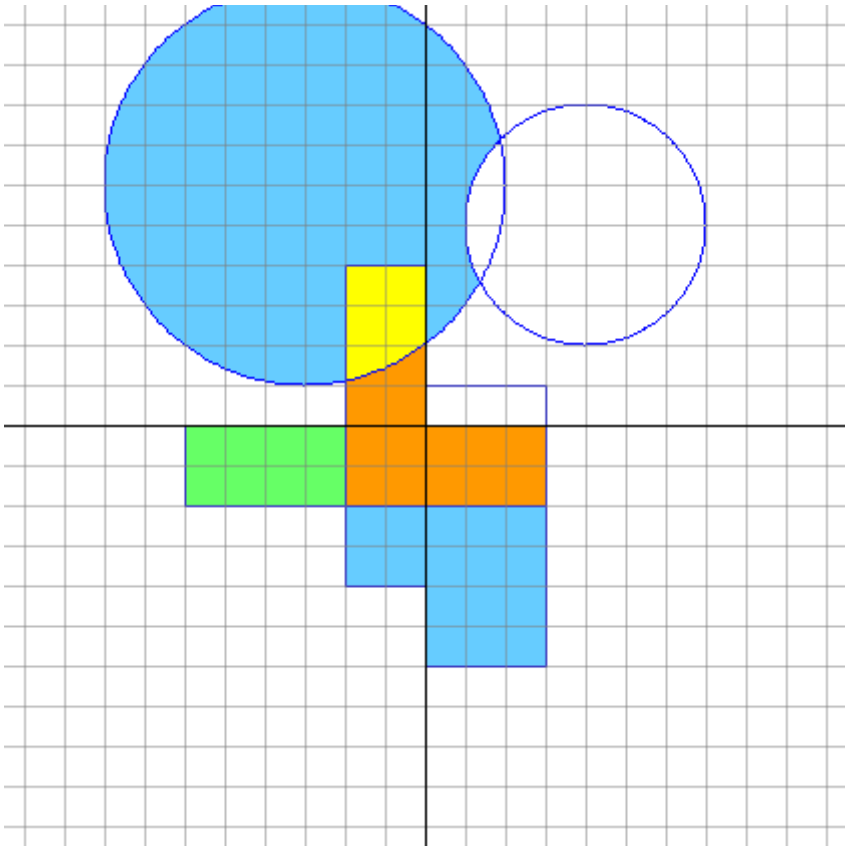
2.



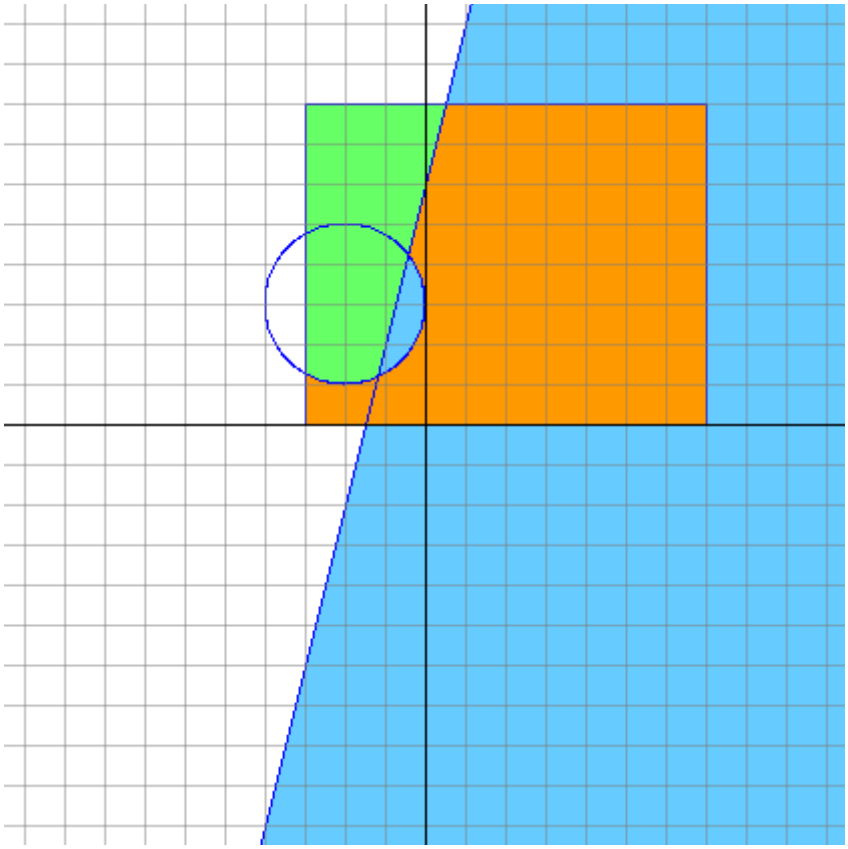
3.



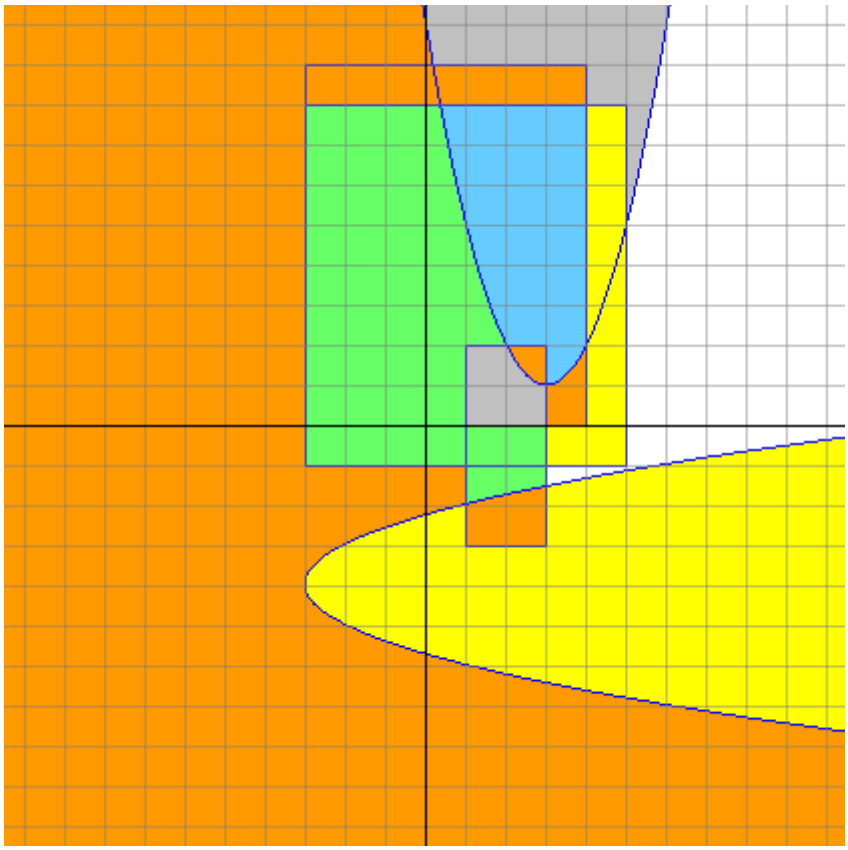
4.



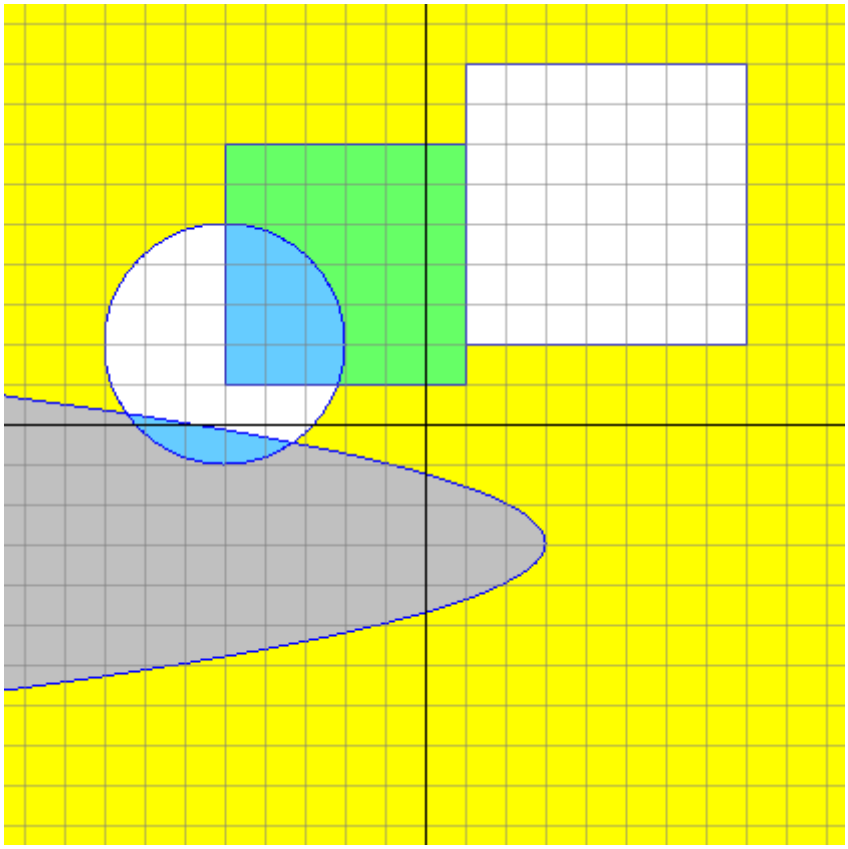
5.



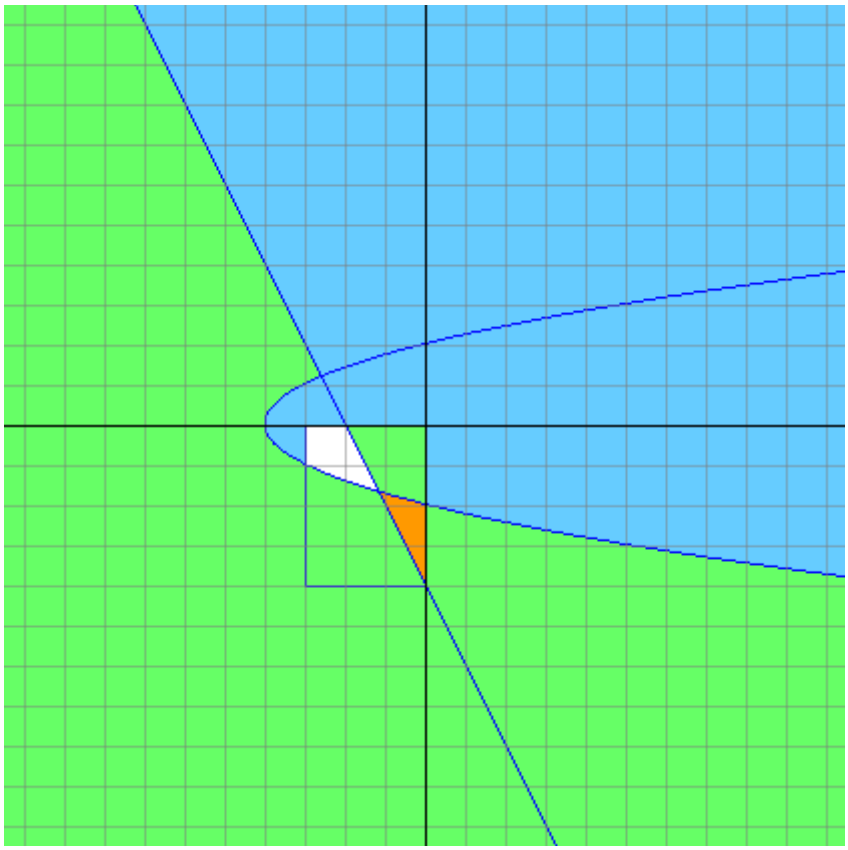
6.



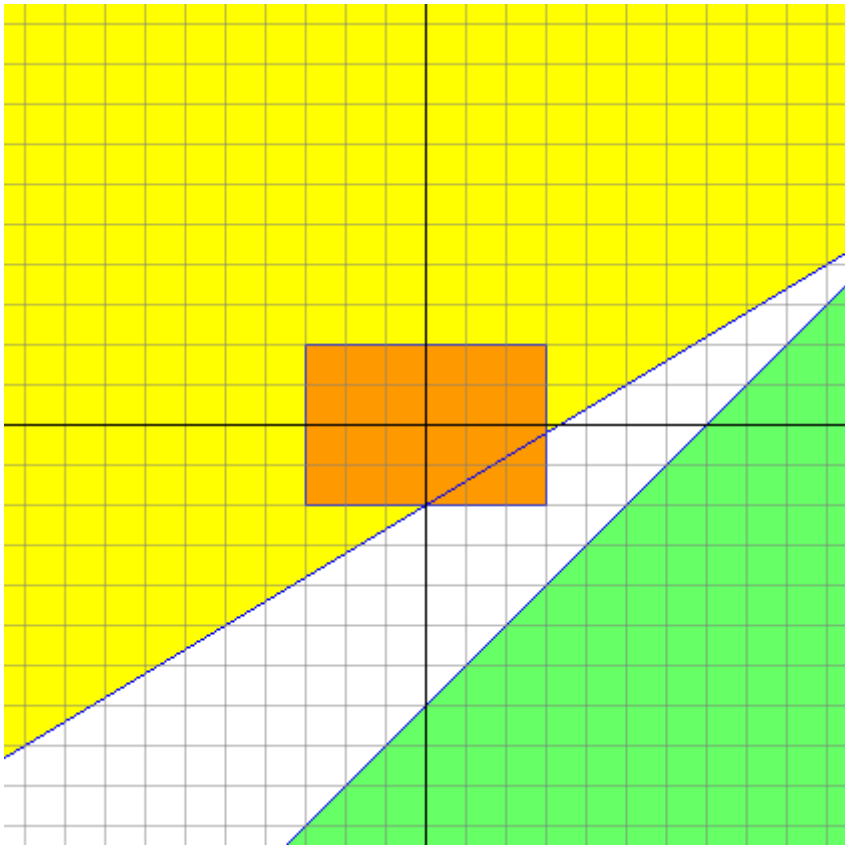
7.



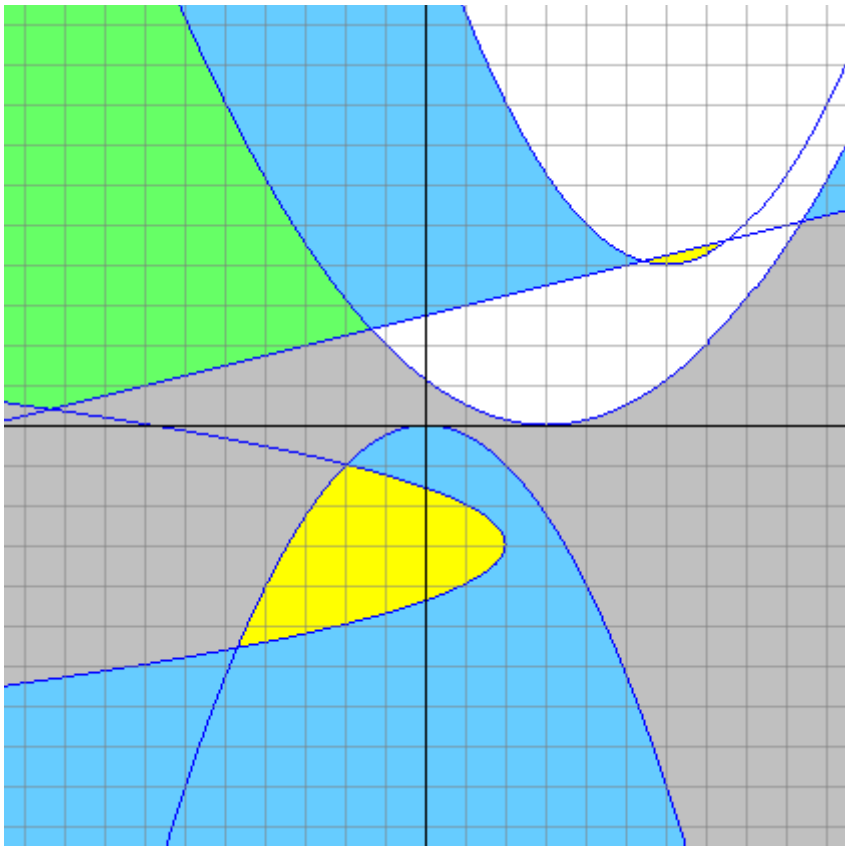
8.



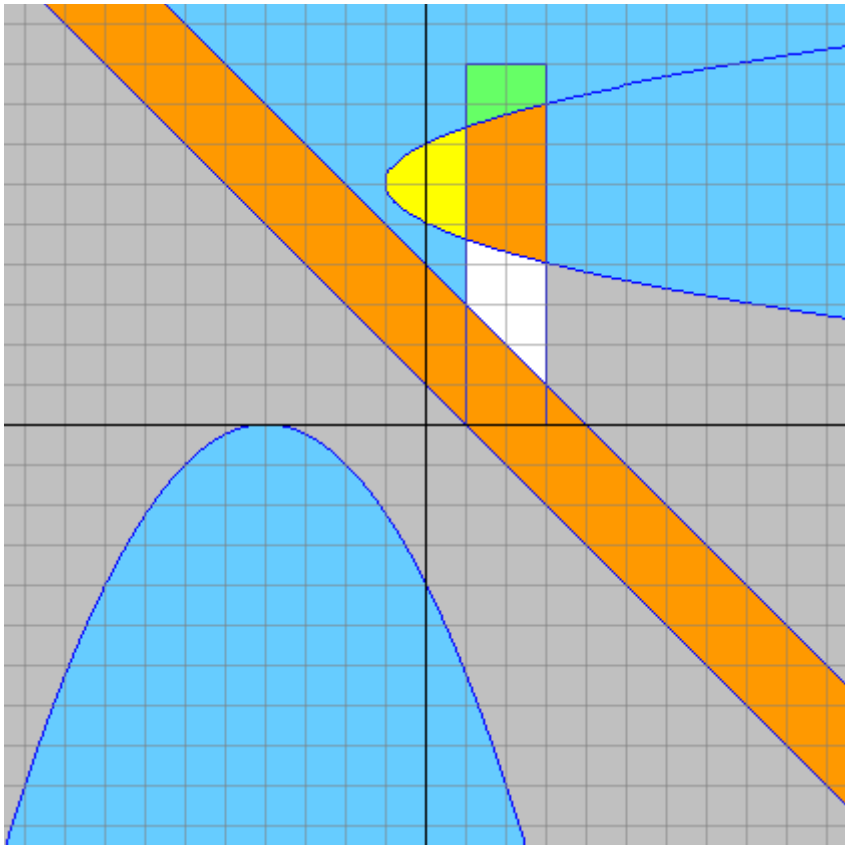
9.



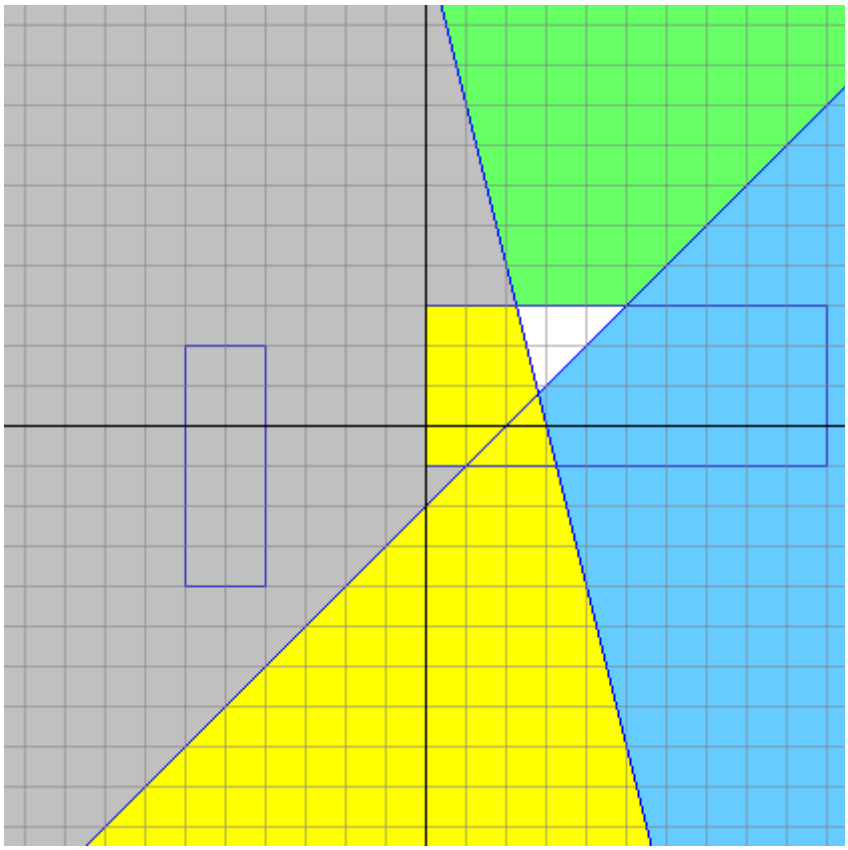
10.



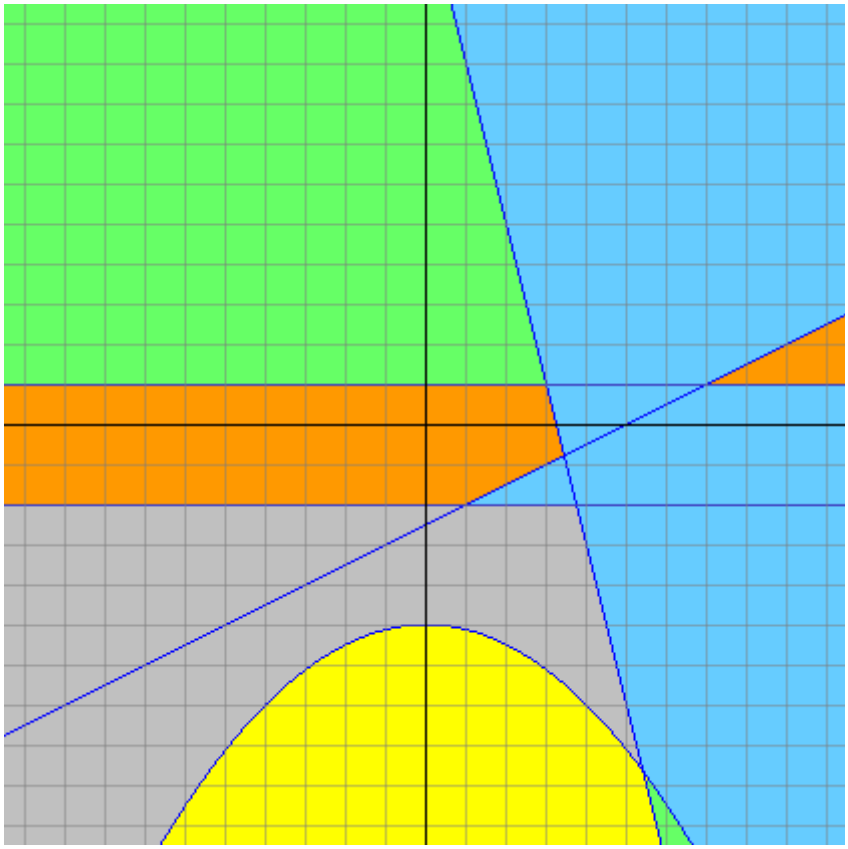
11.



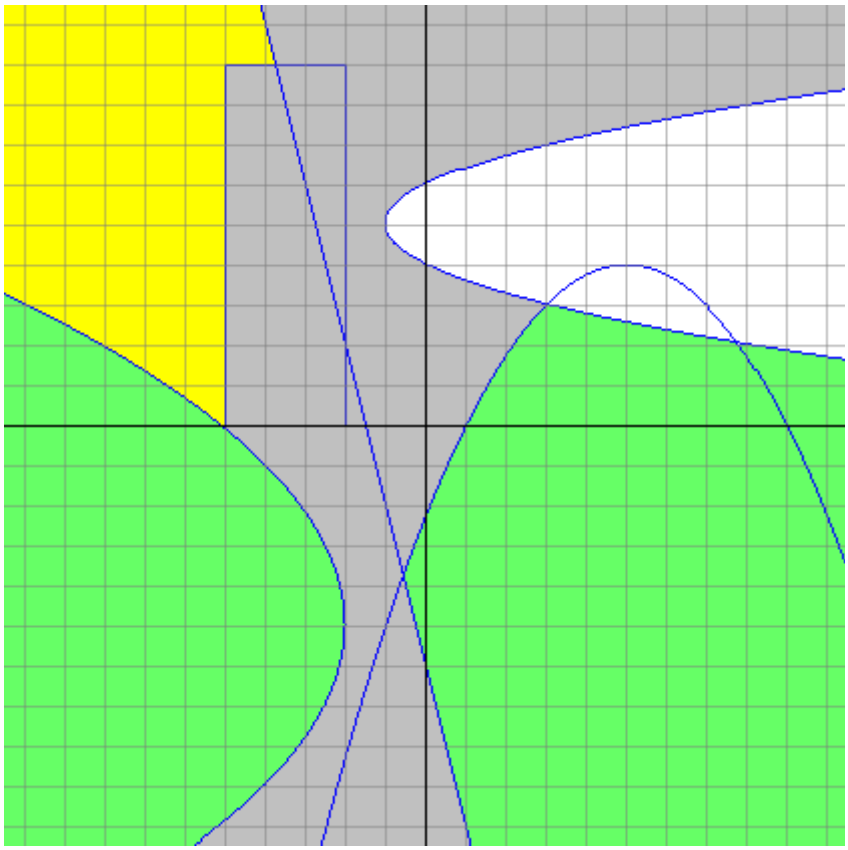
12.



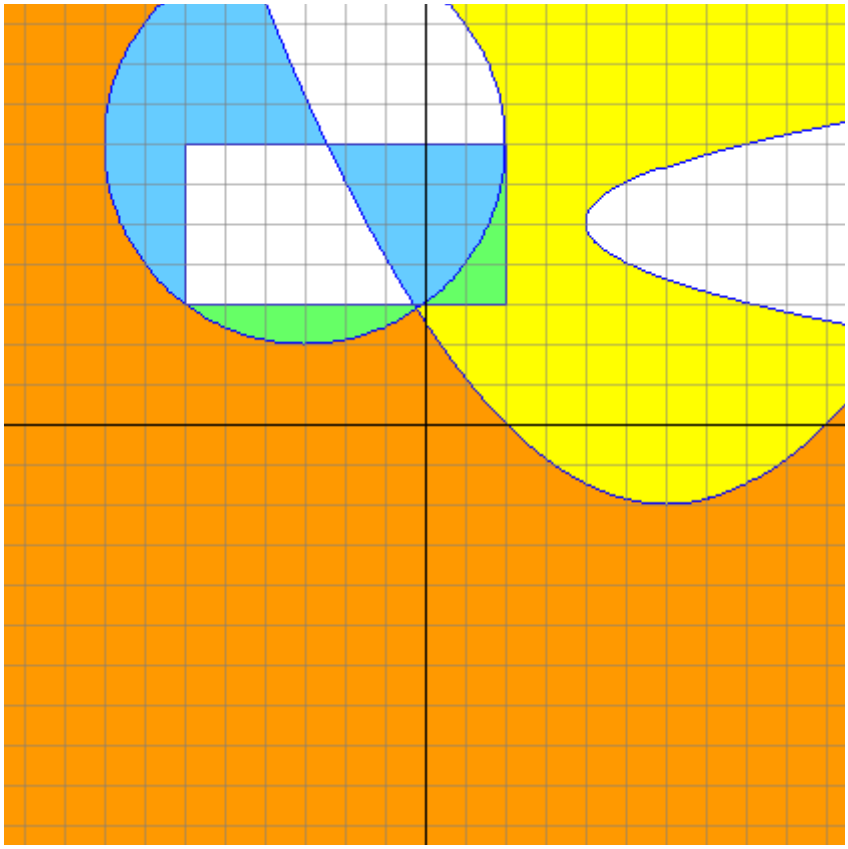
13.



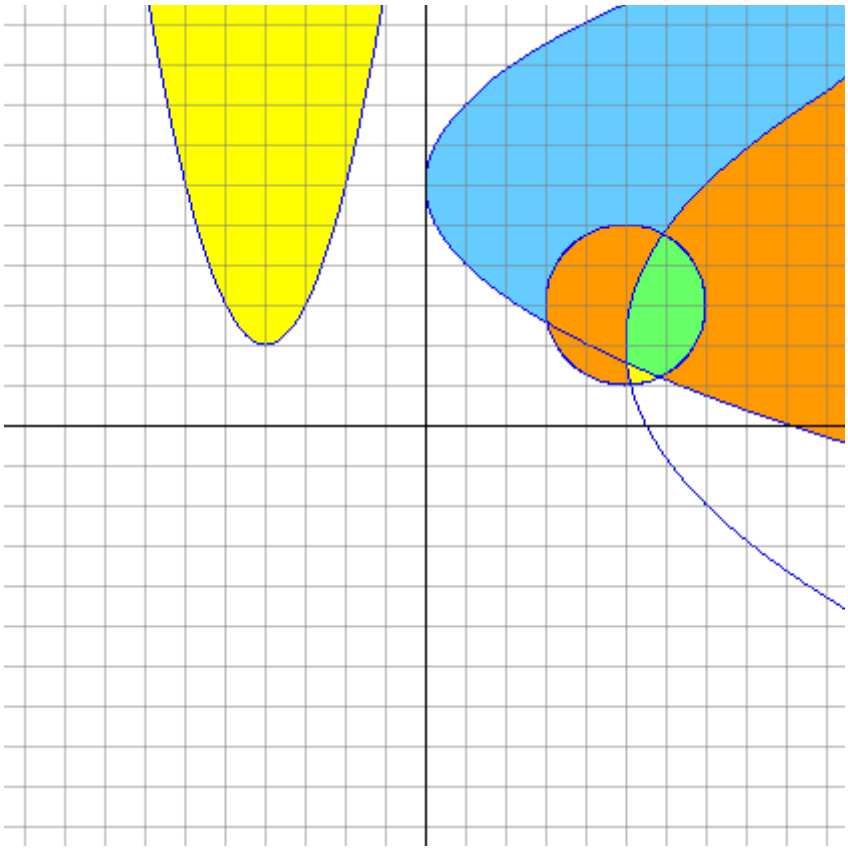
14.



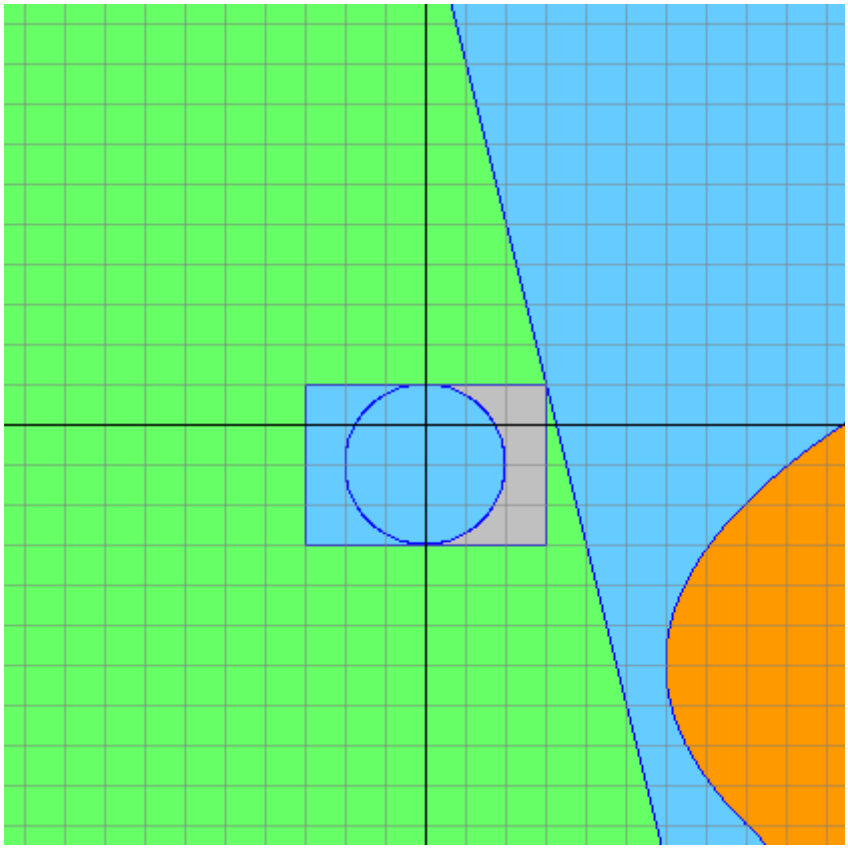
15.



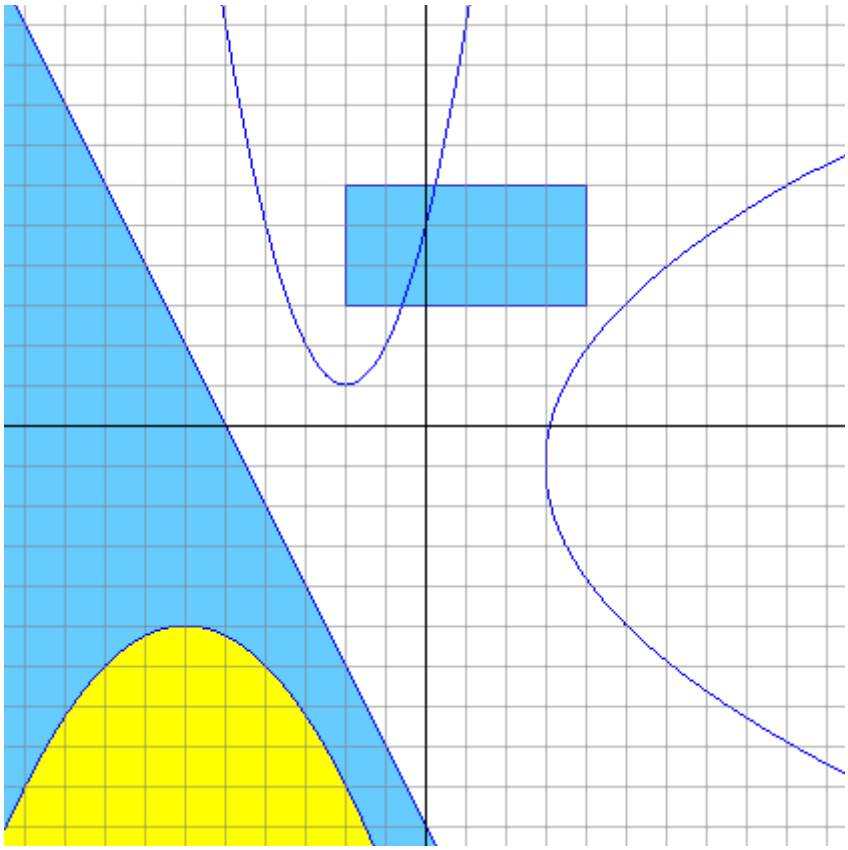
16.



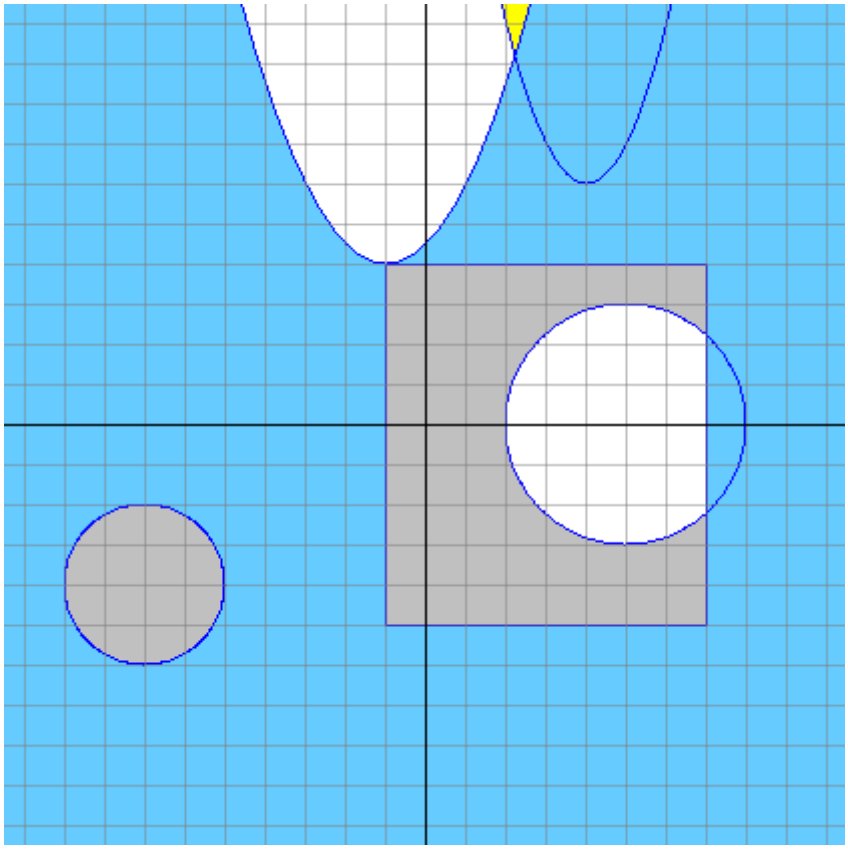
17.



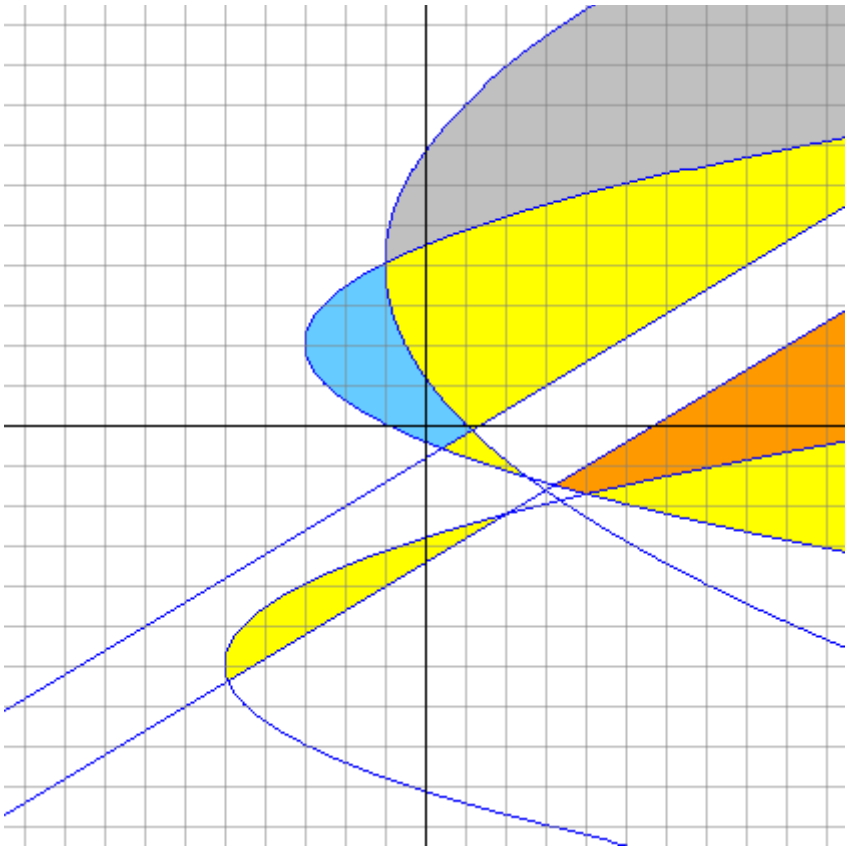
18.



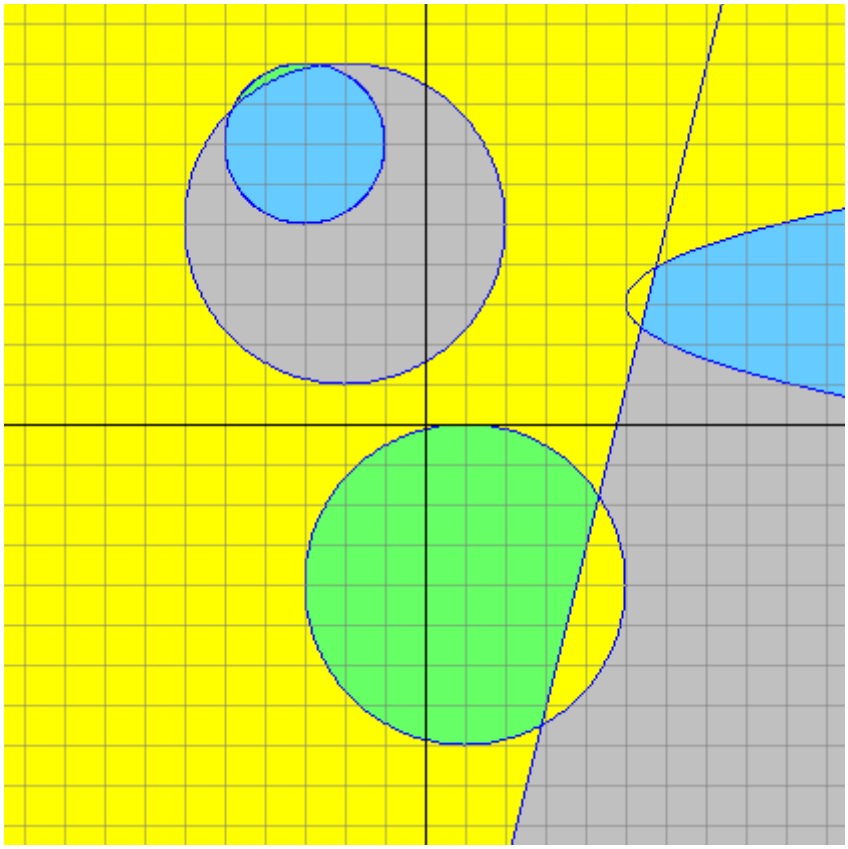
19.



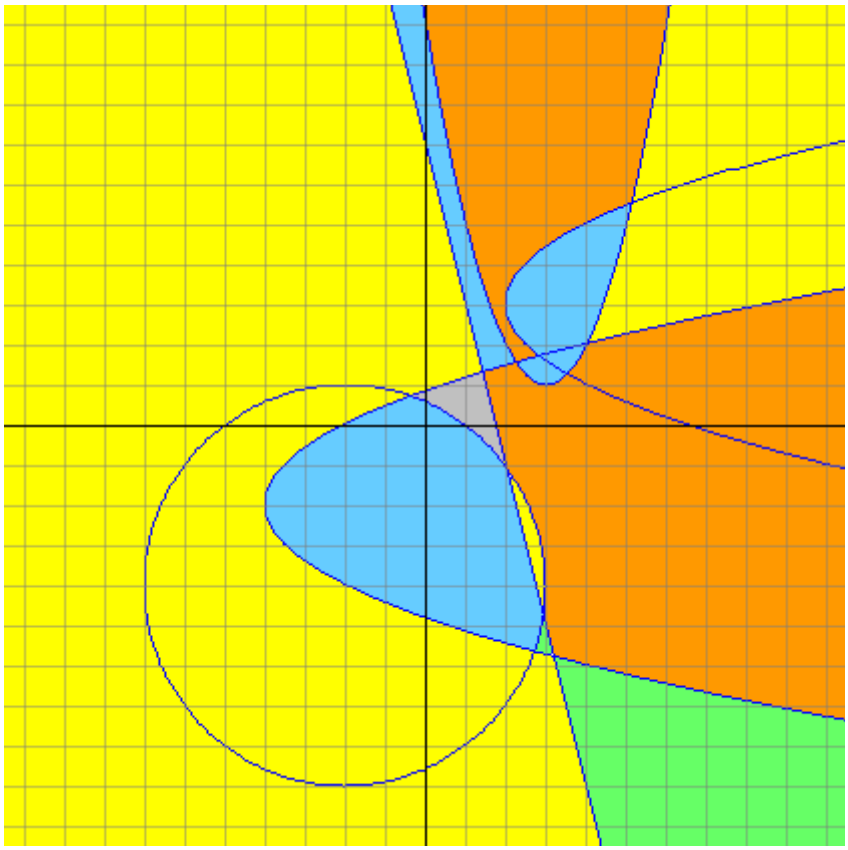
20.



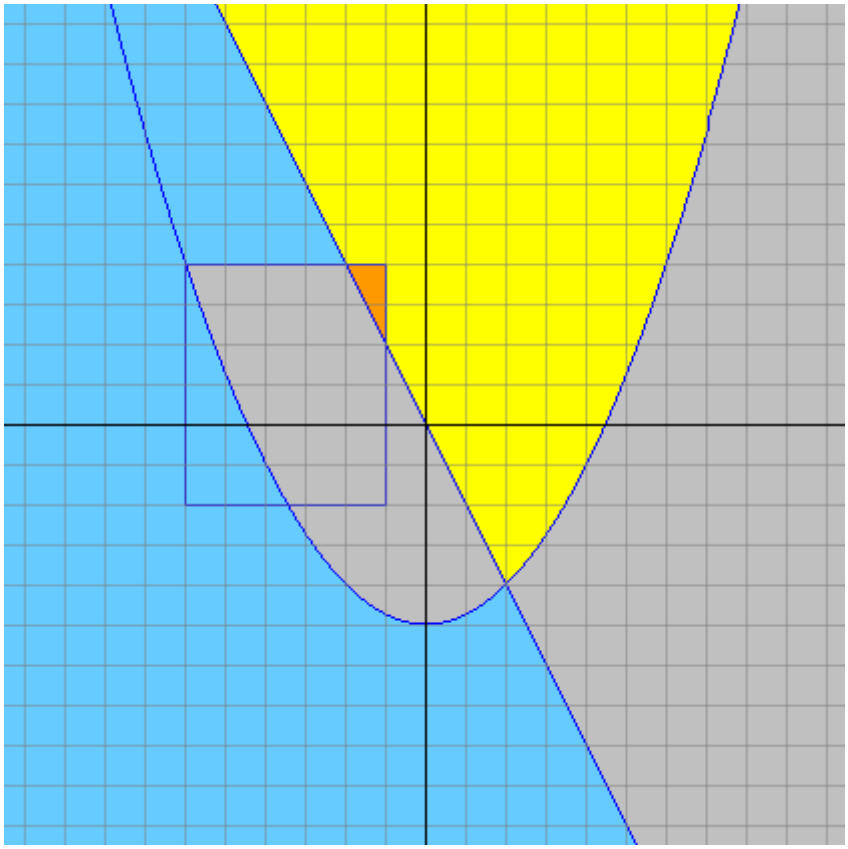
21.



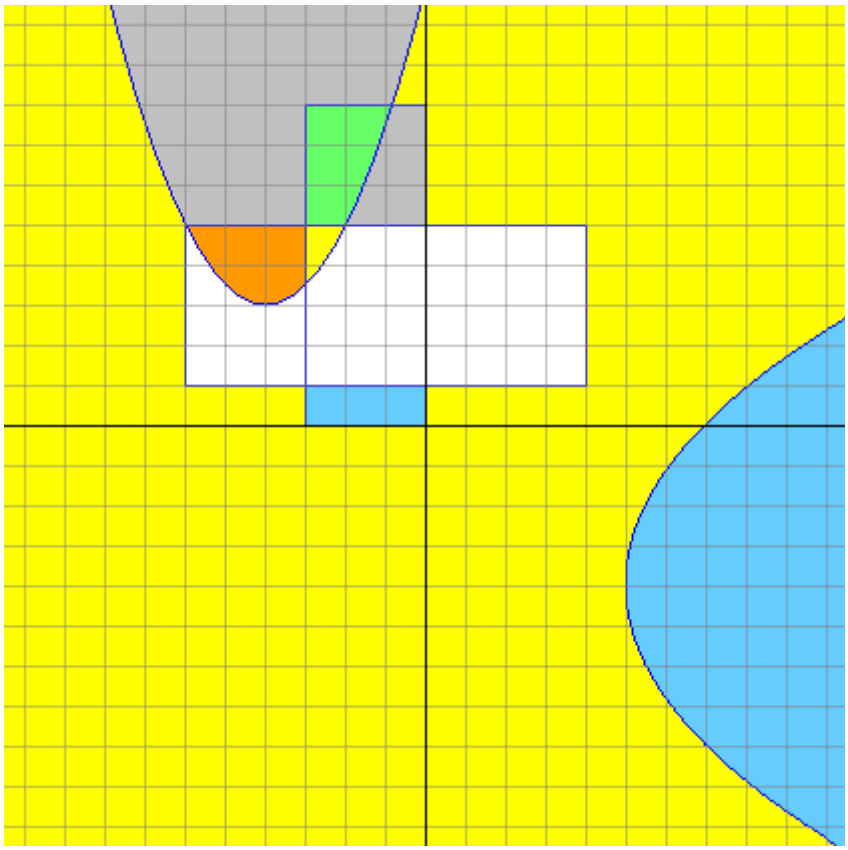
22.



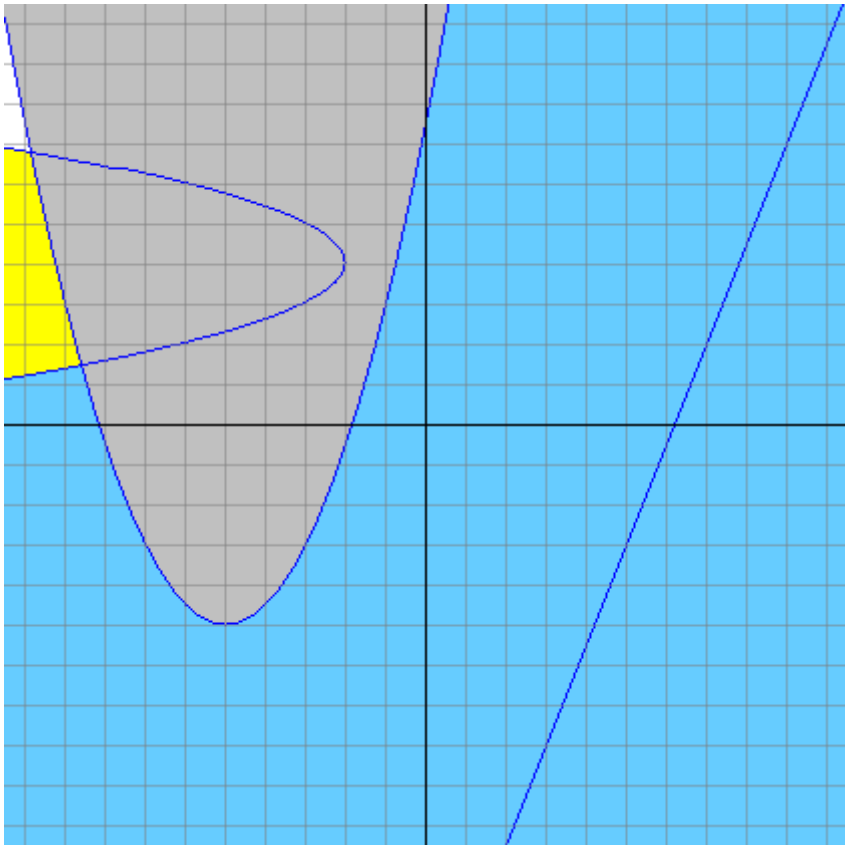
23.



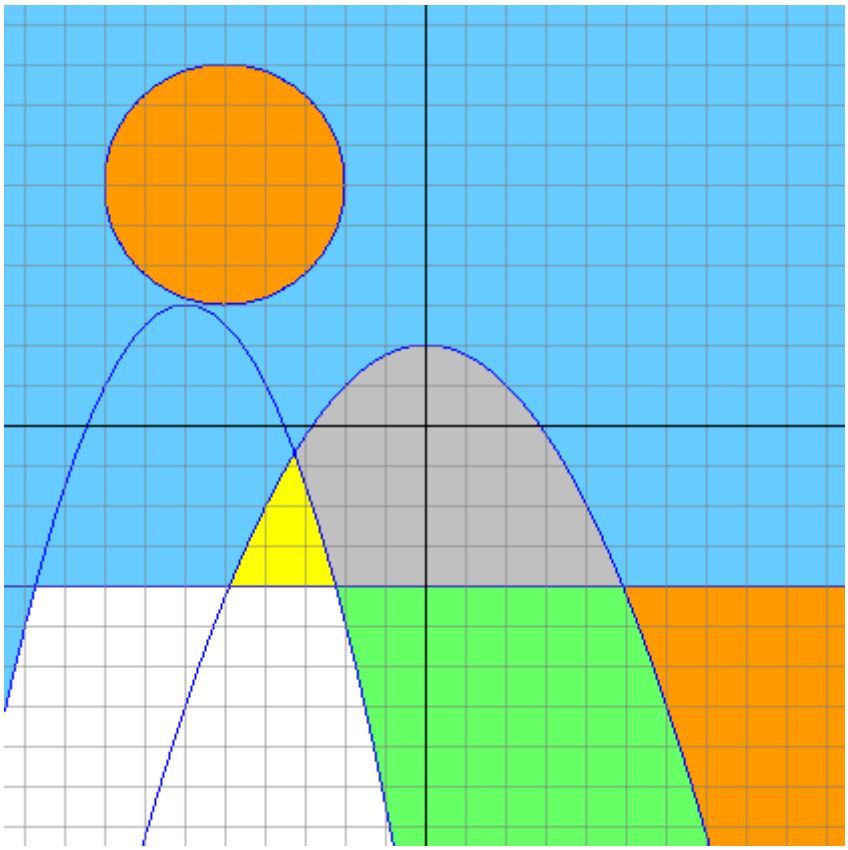
24.



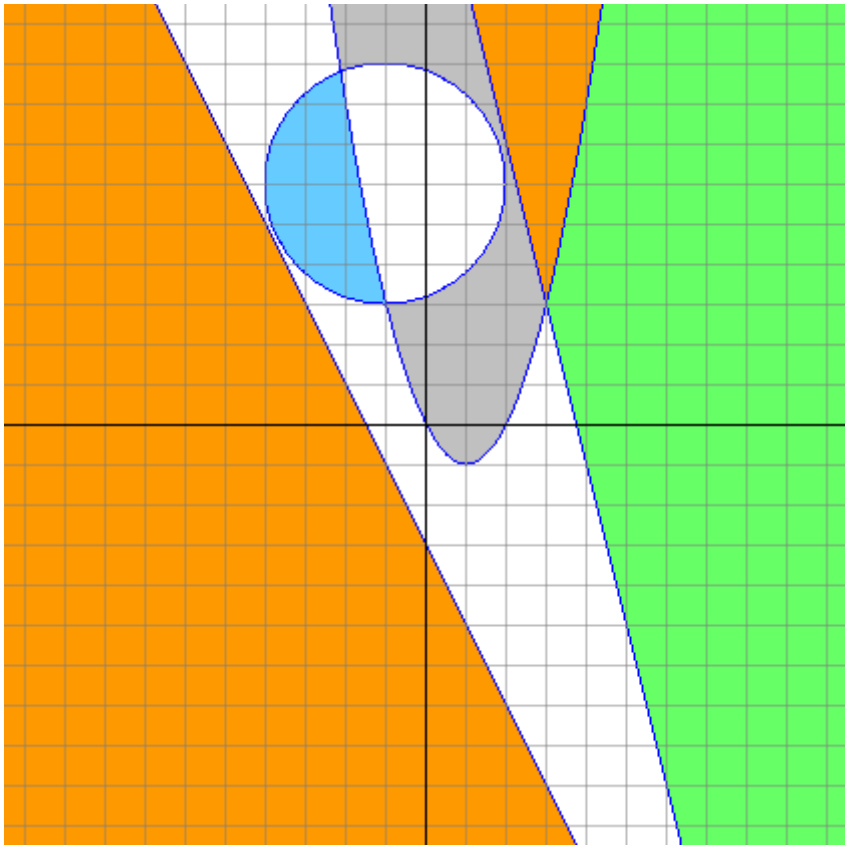
25.



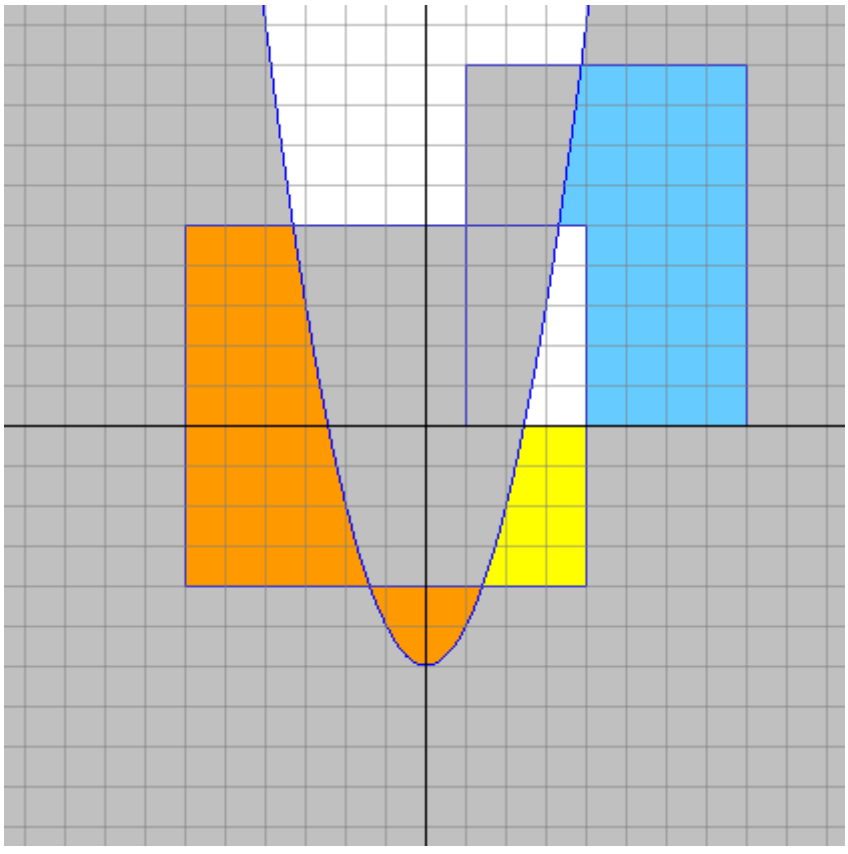
26.



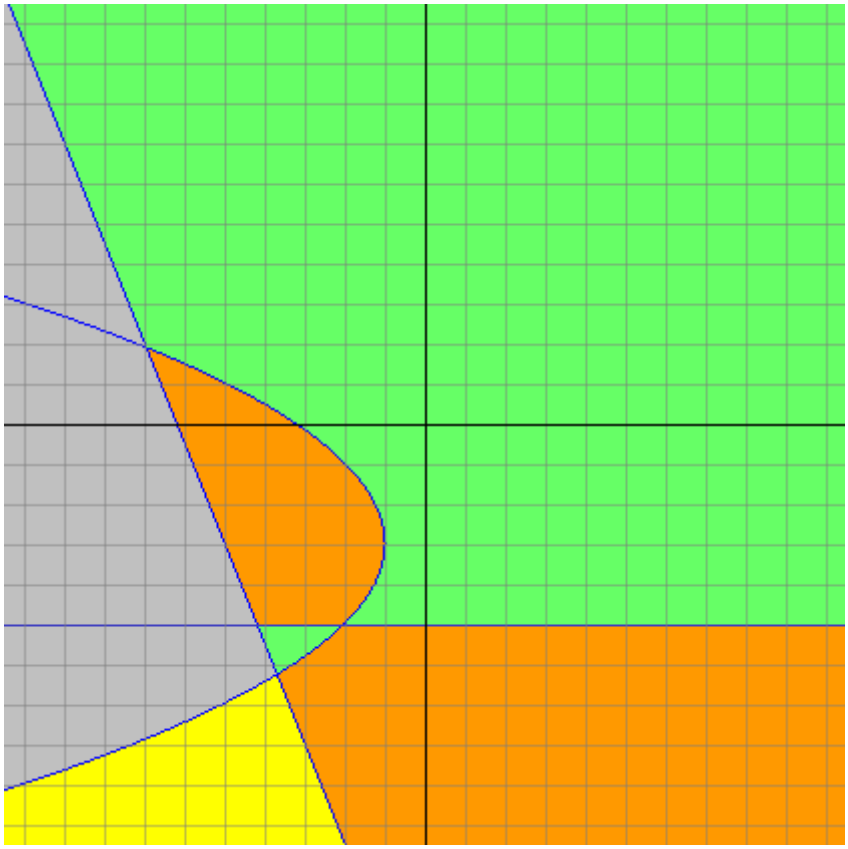
27.



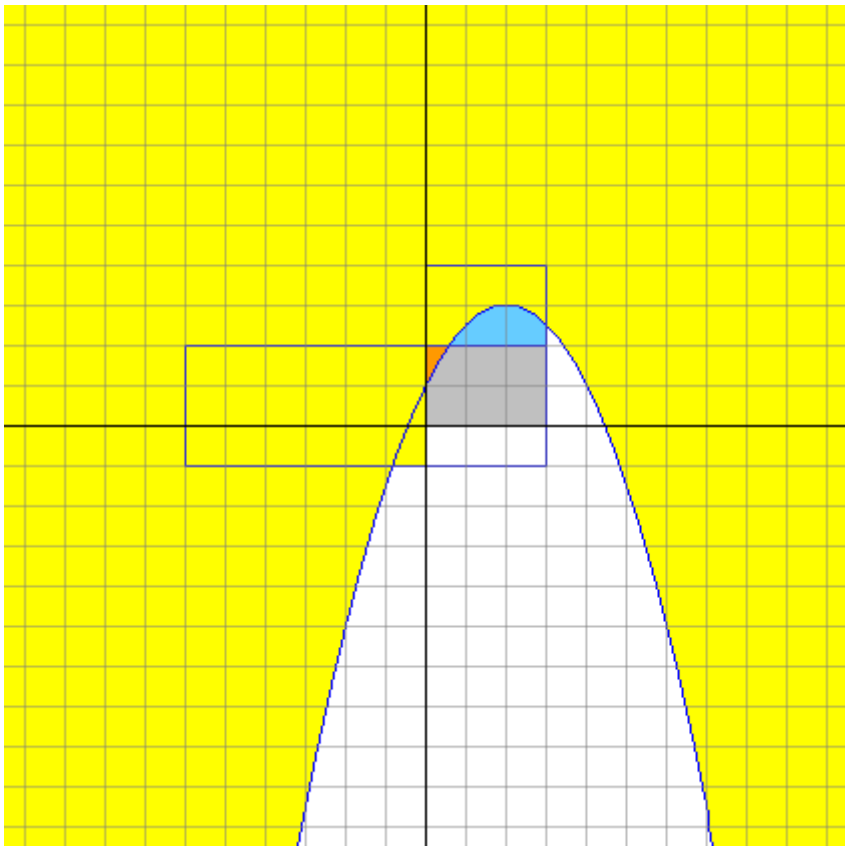
28.



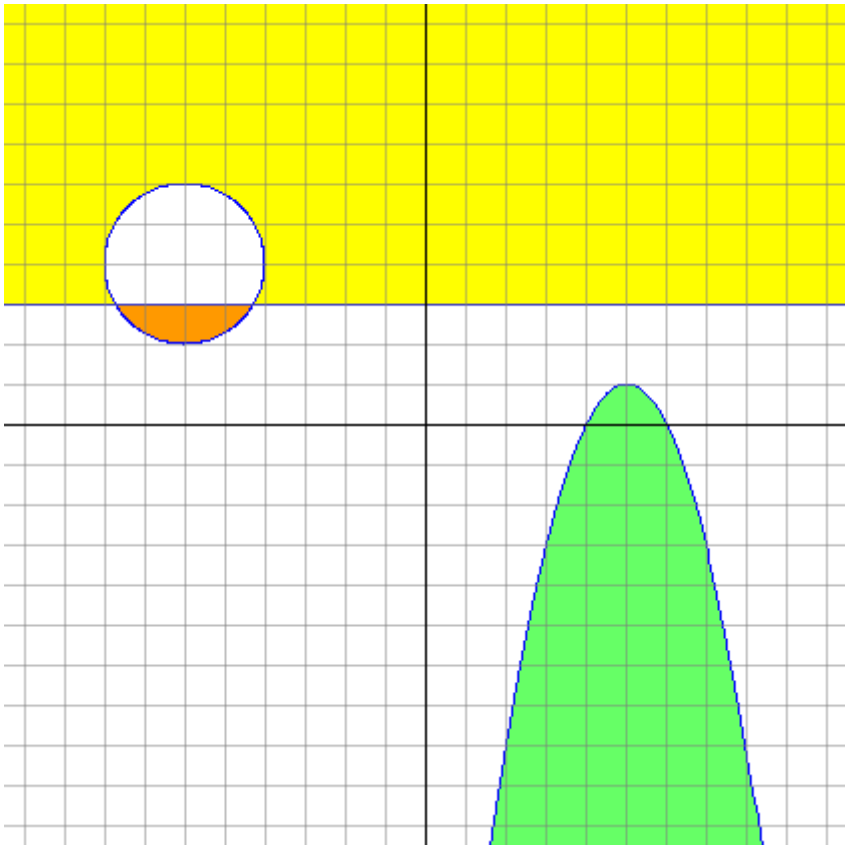
29.



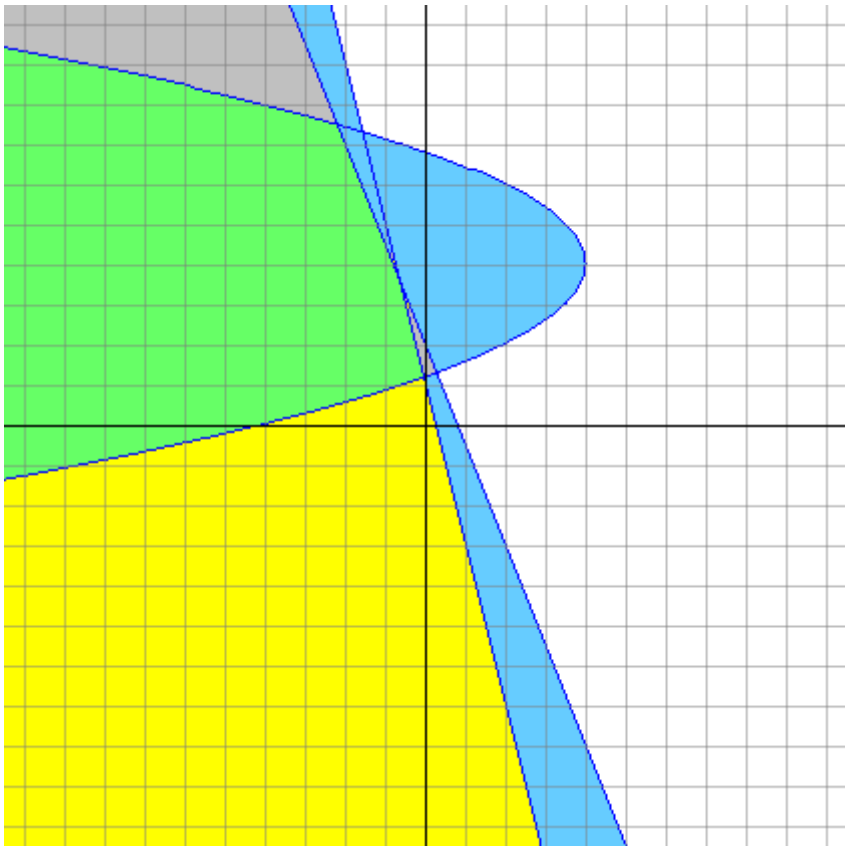
30.



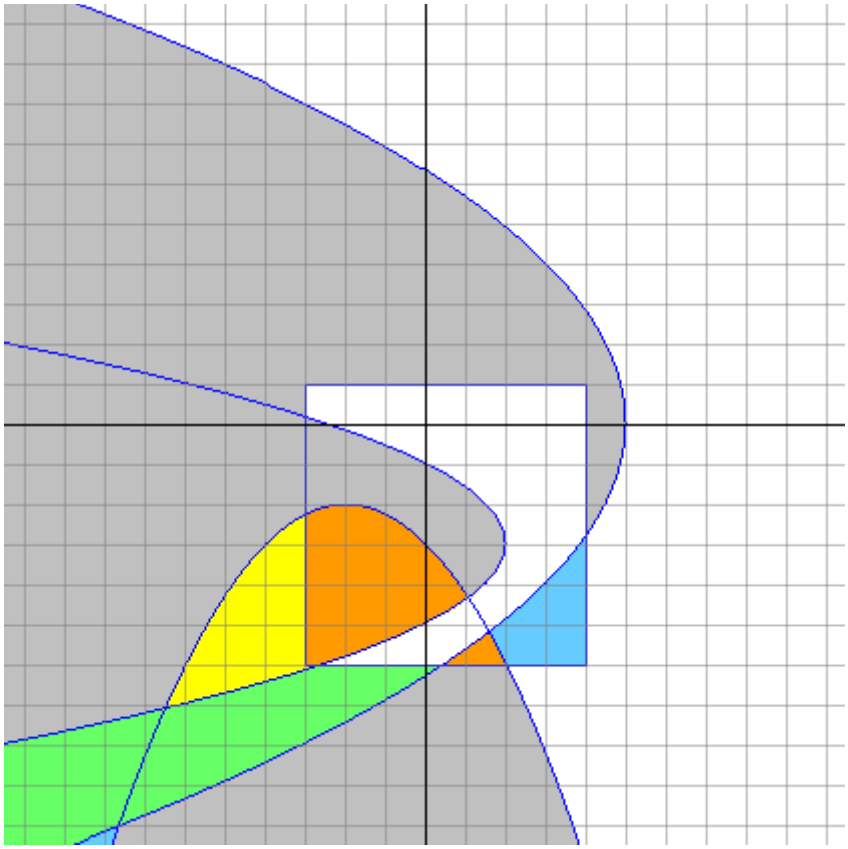
31.



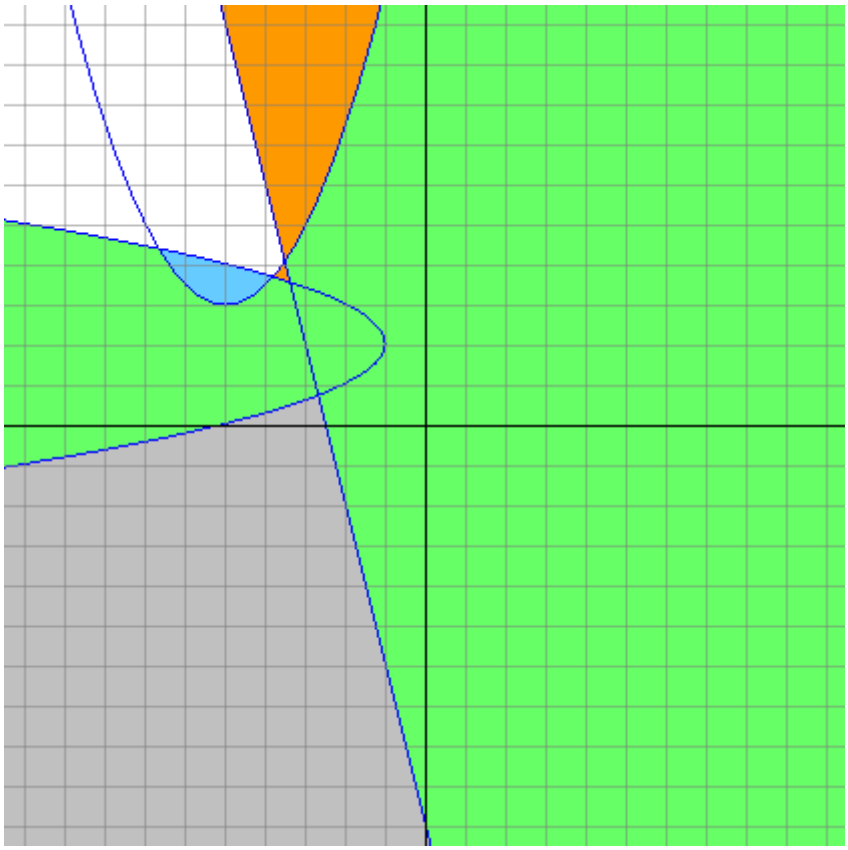
32.



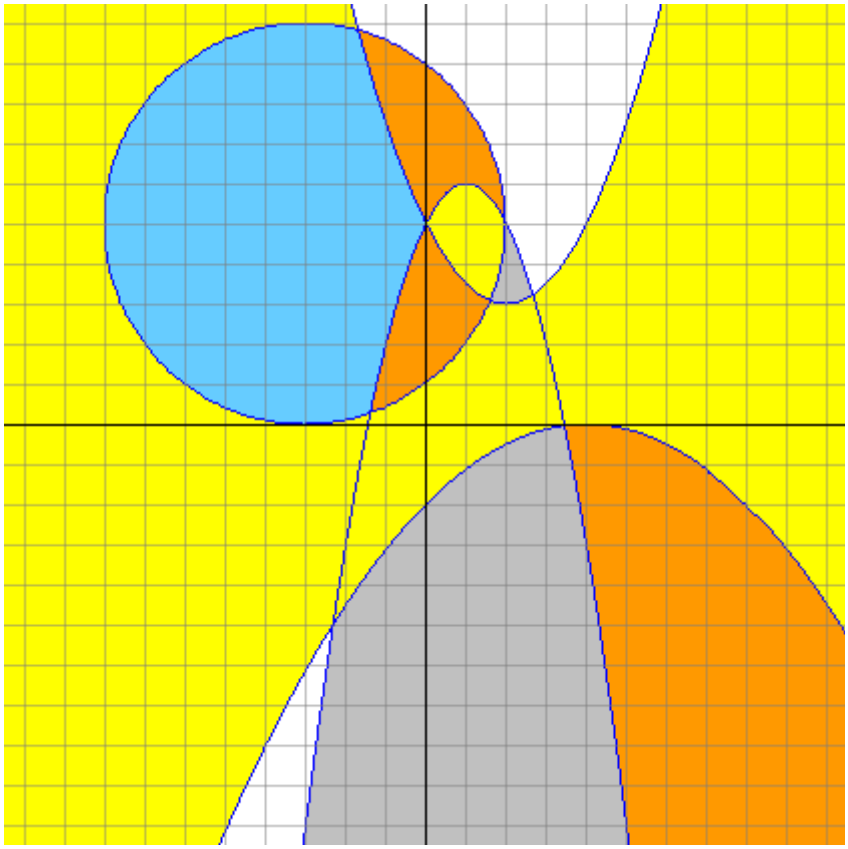
33.



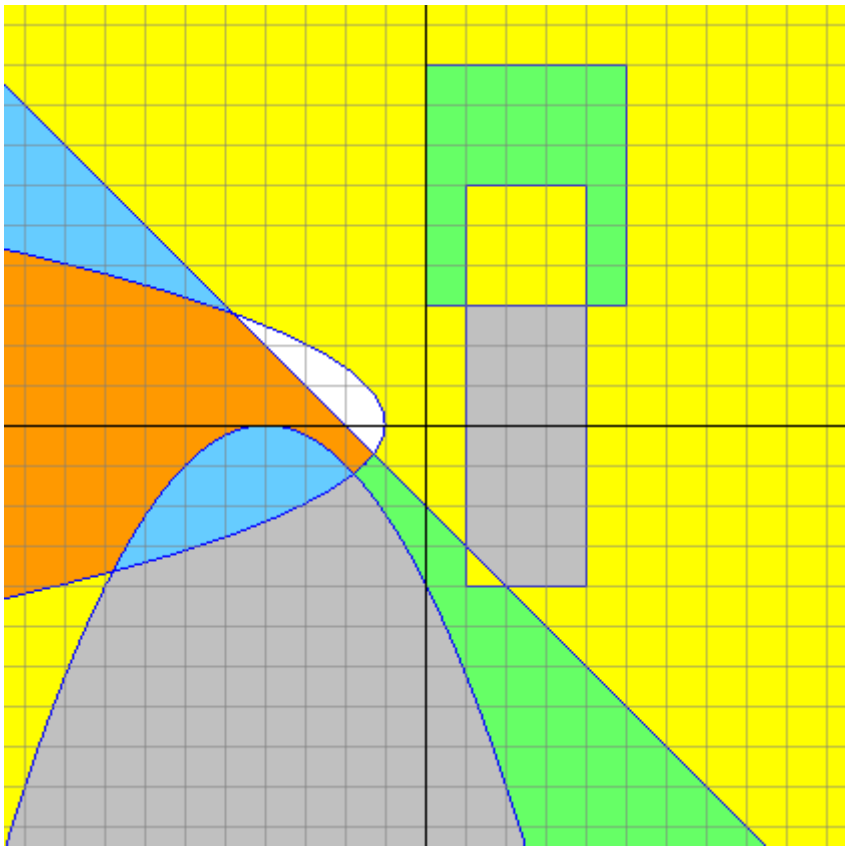
34.



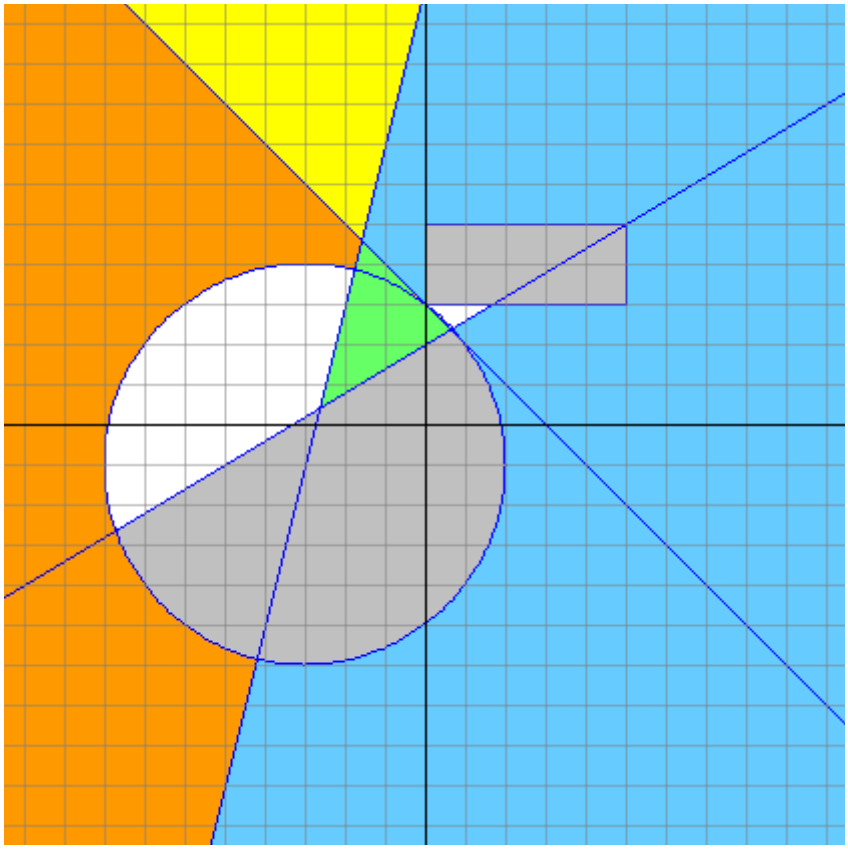
35.



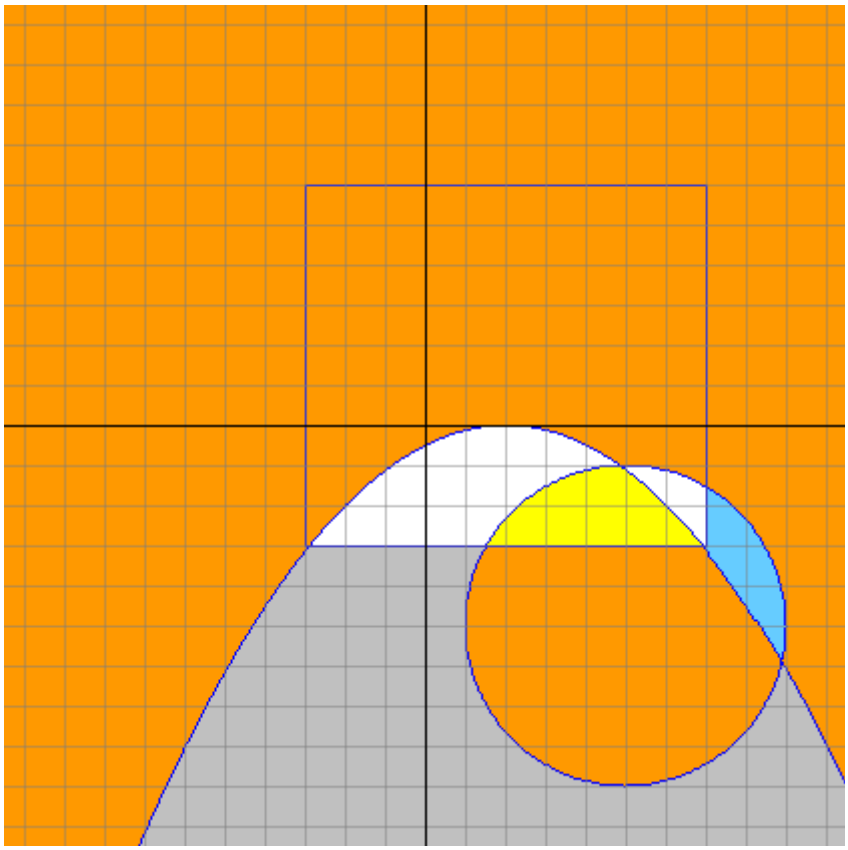
36.



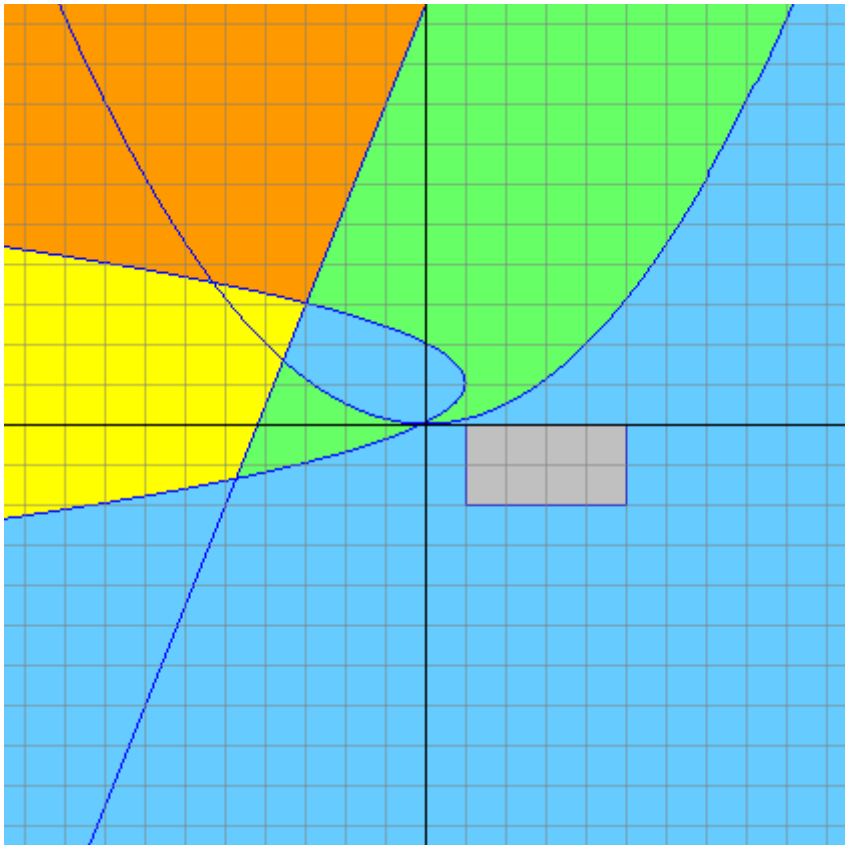
37.



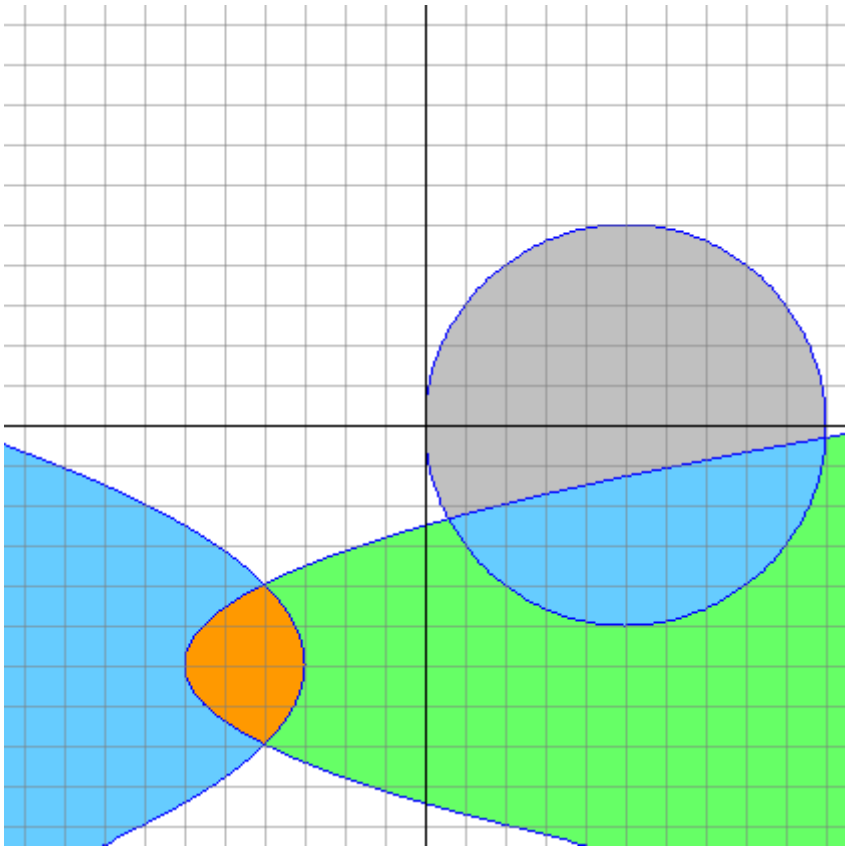
38.



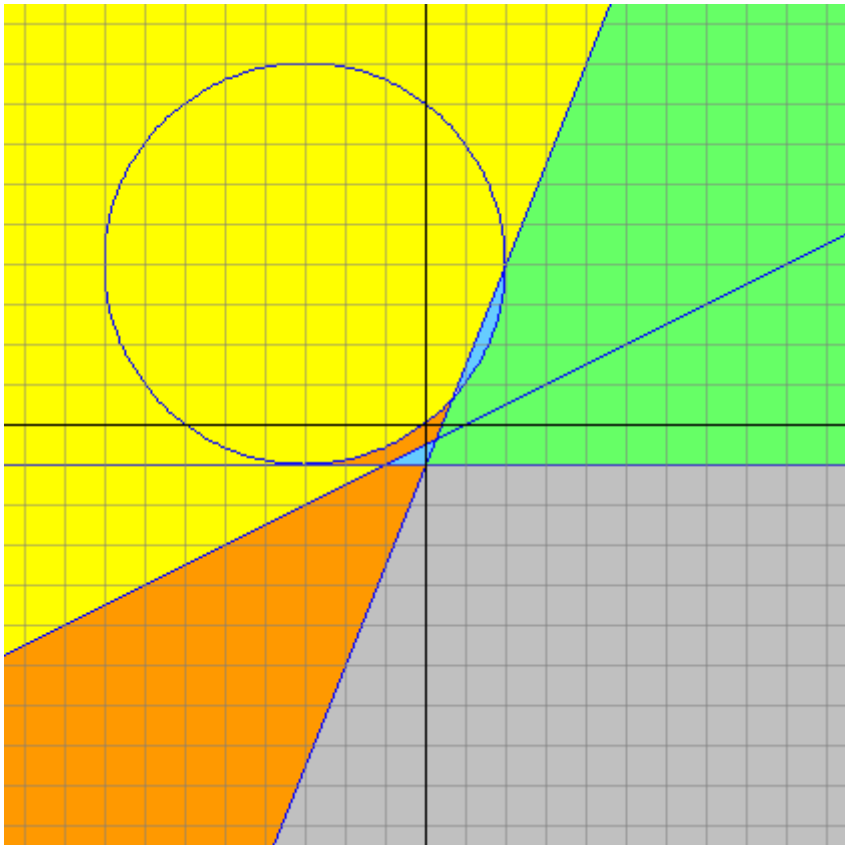
39.



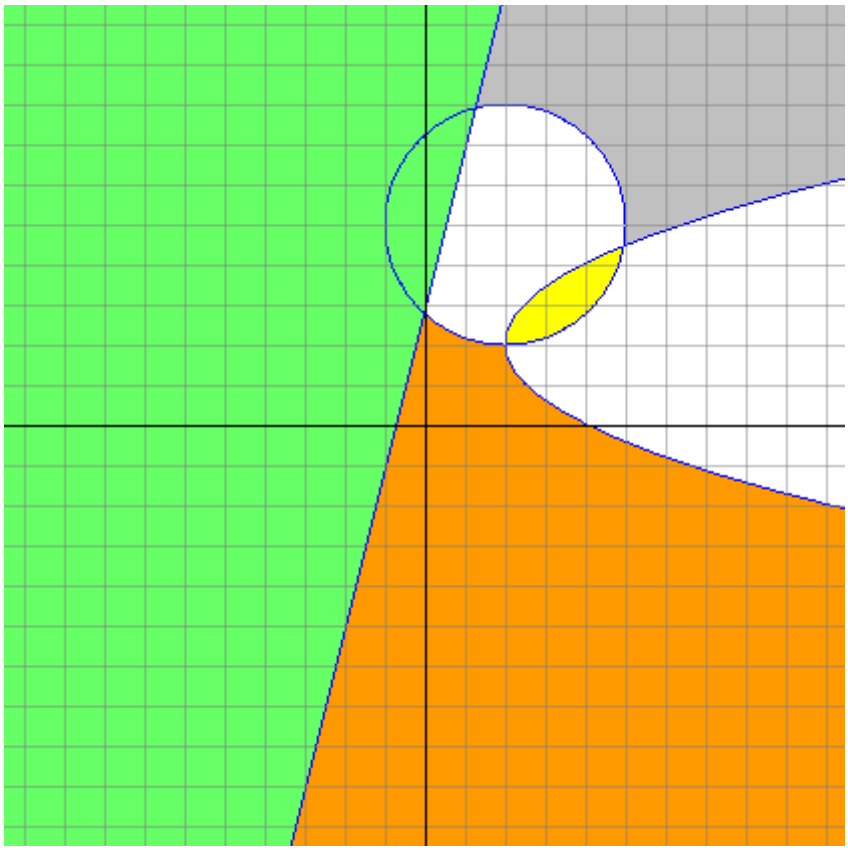
40.



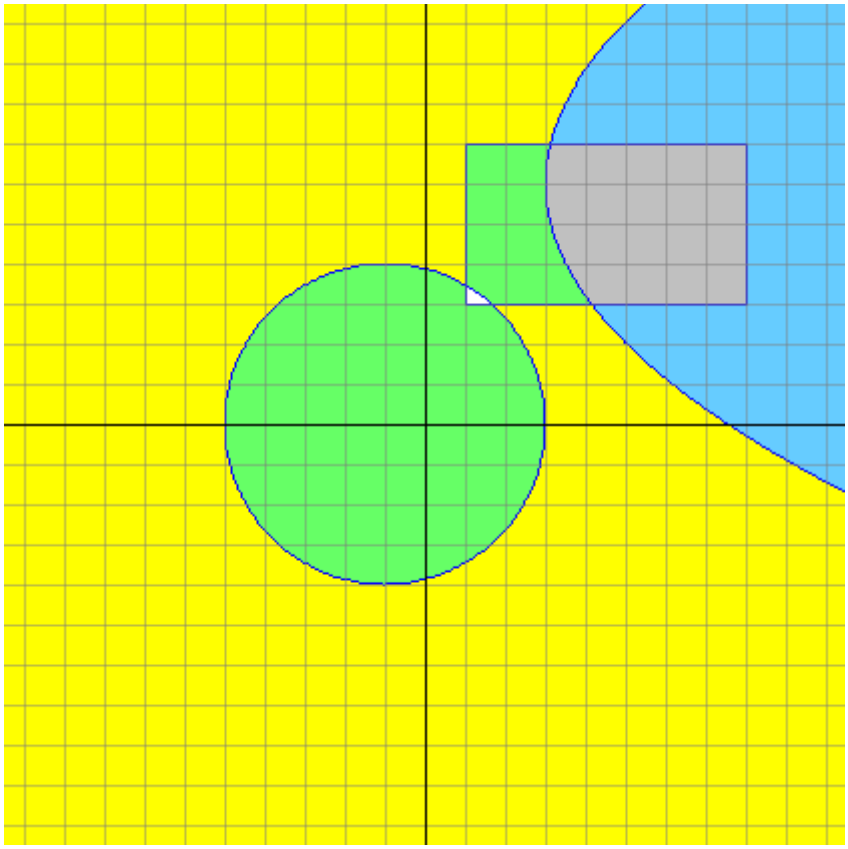
41.



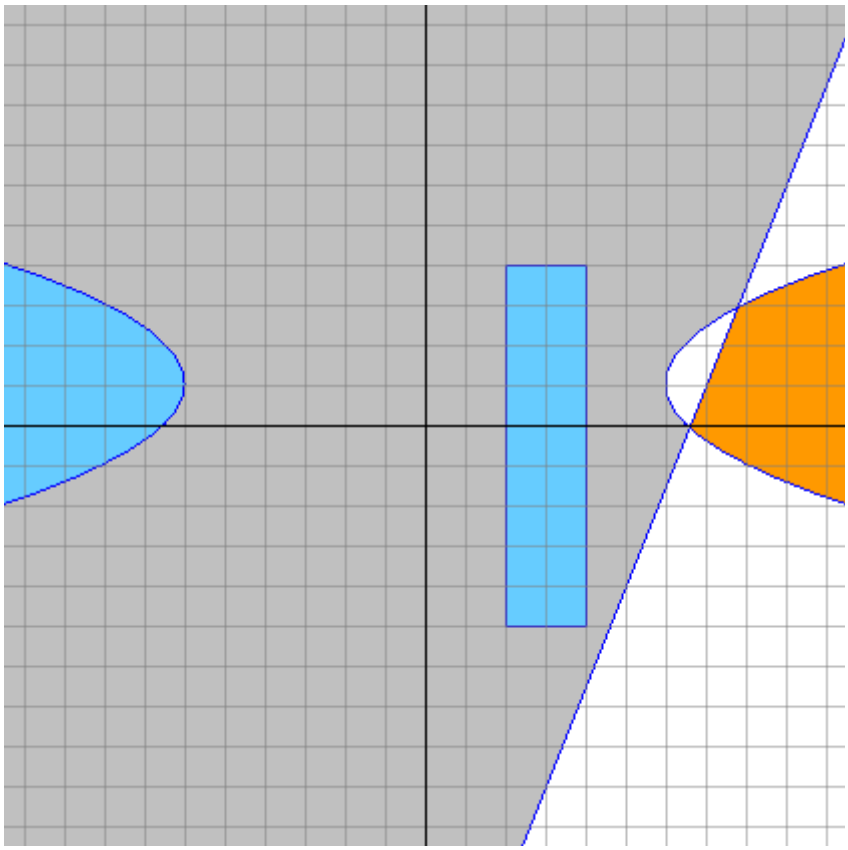
42.



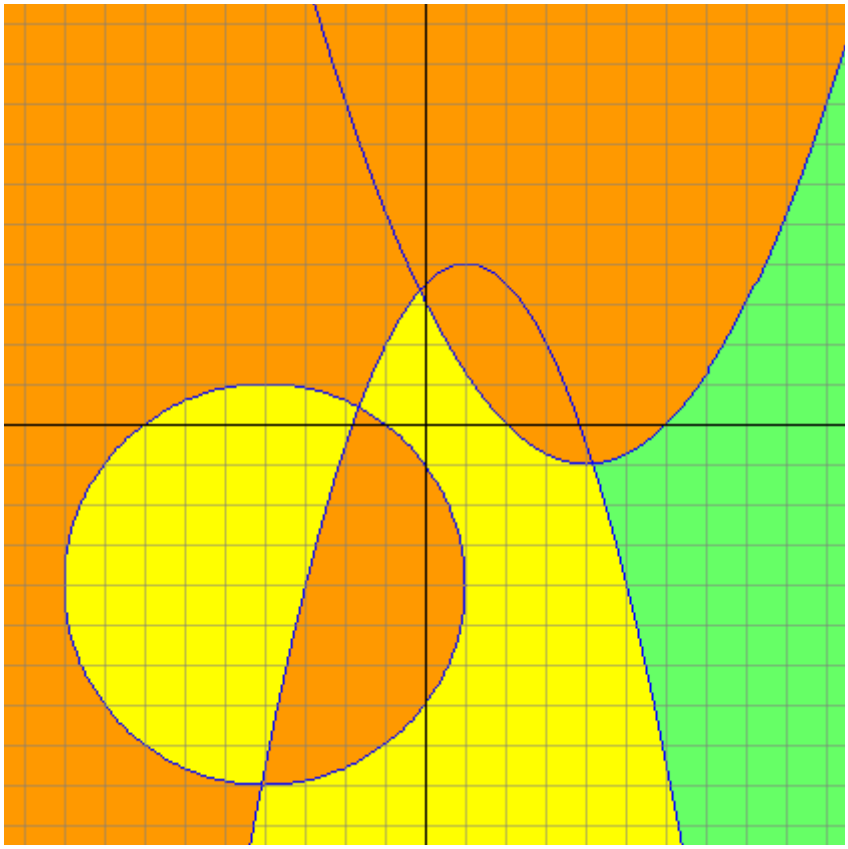
43.



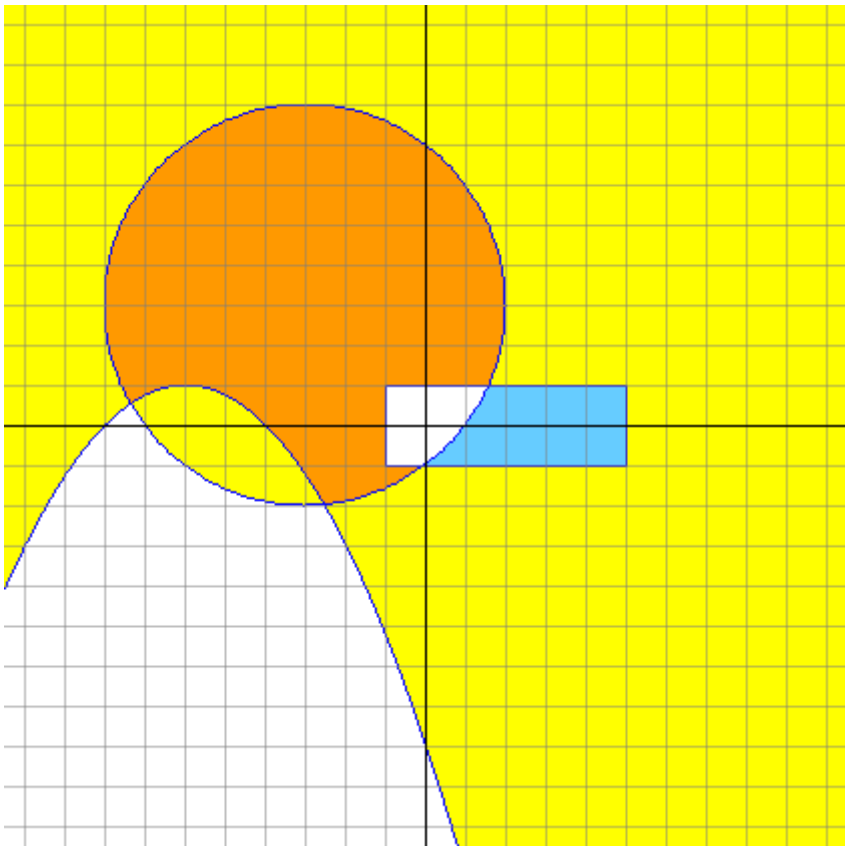
44.



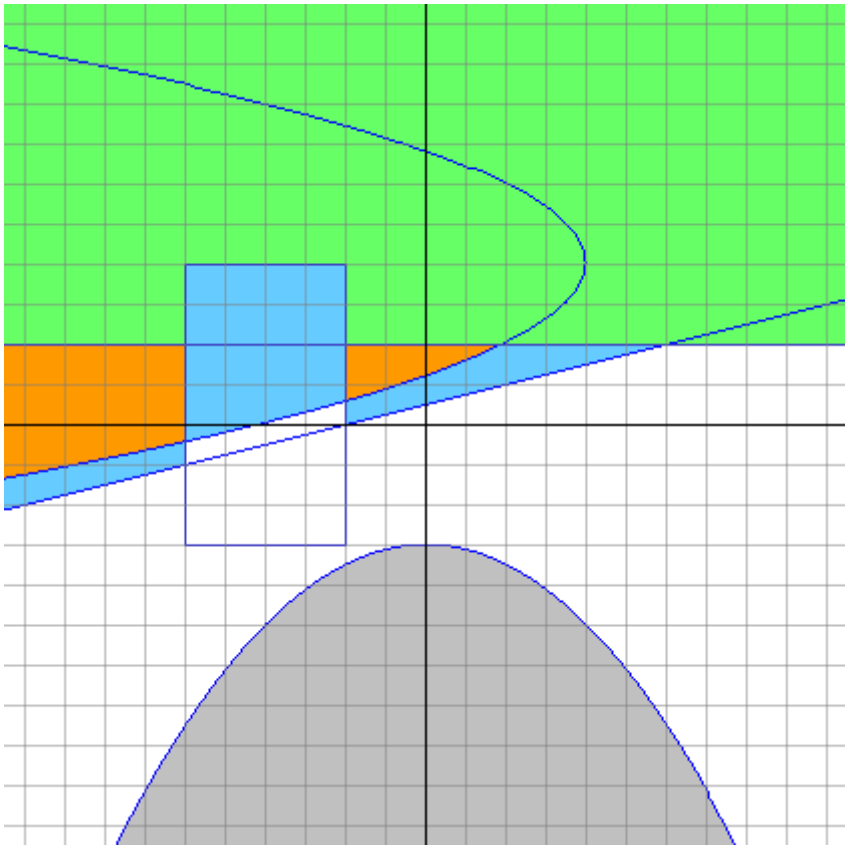
45.



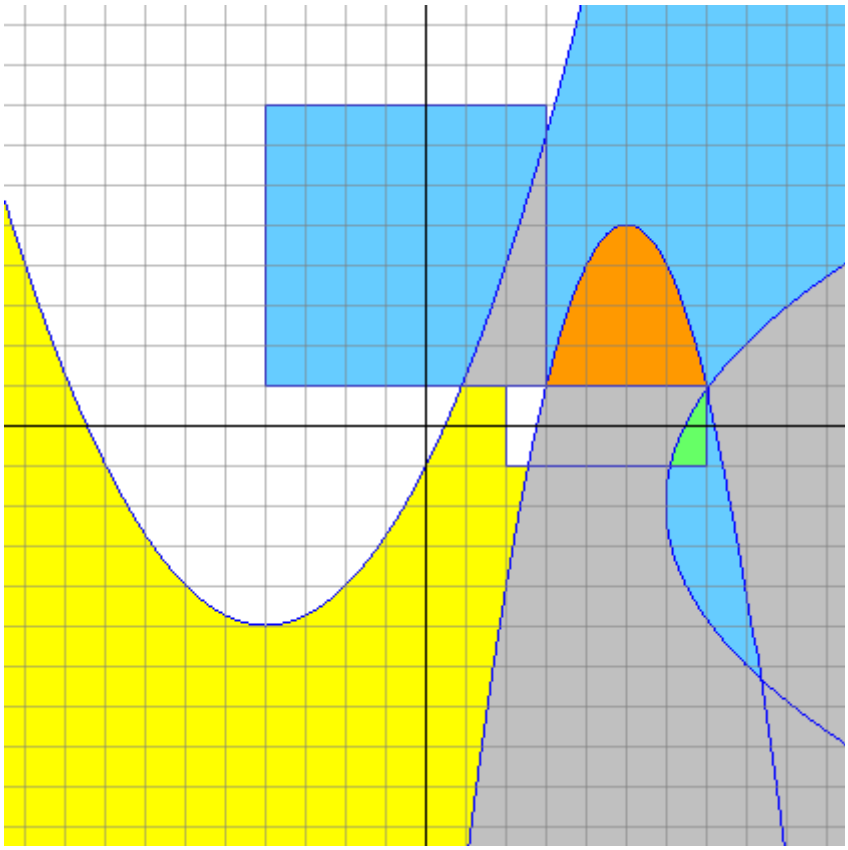
46.



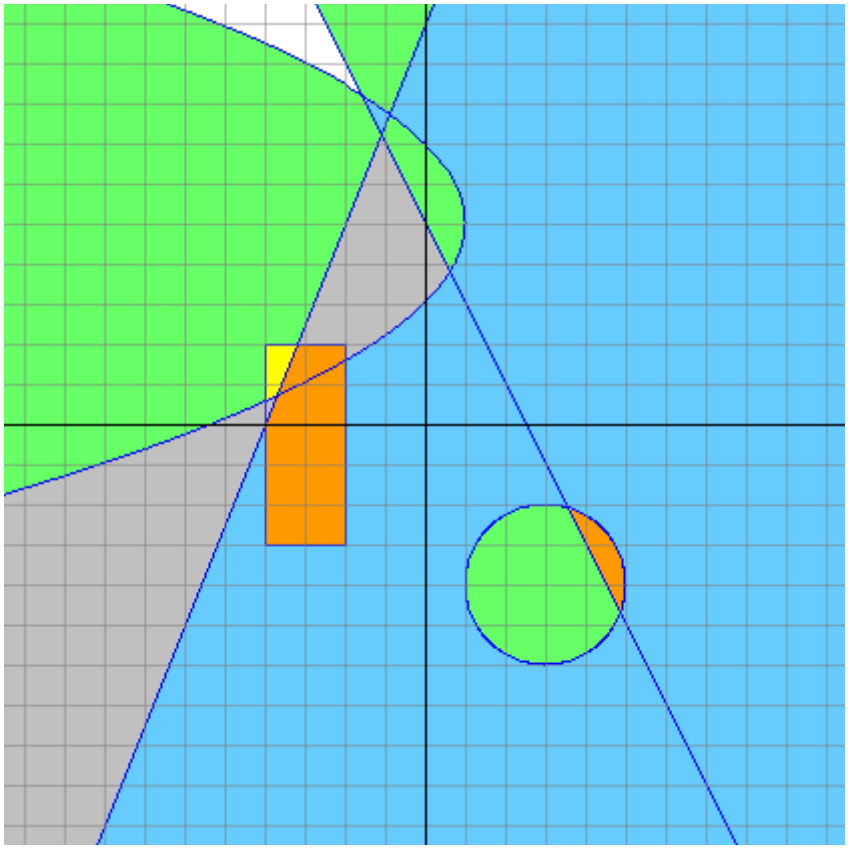
47.



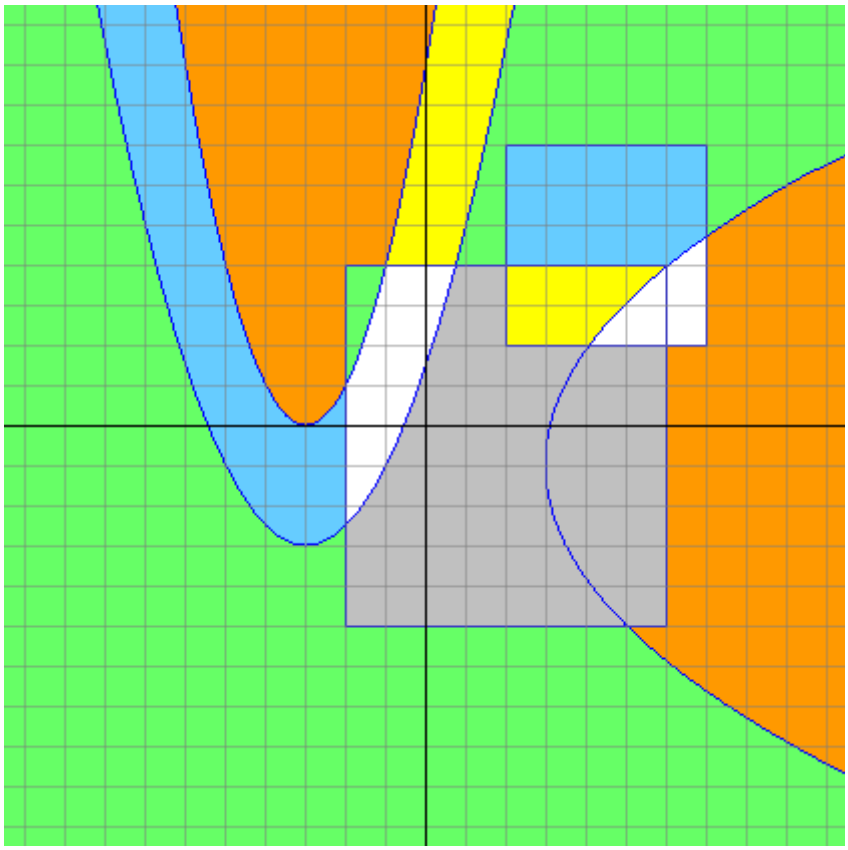
48.



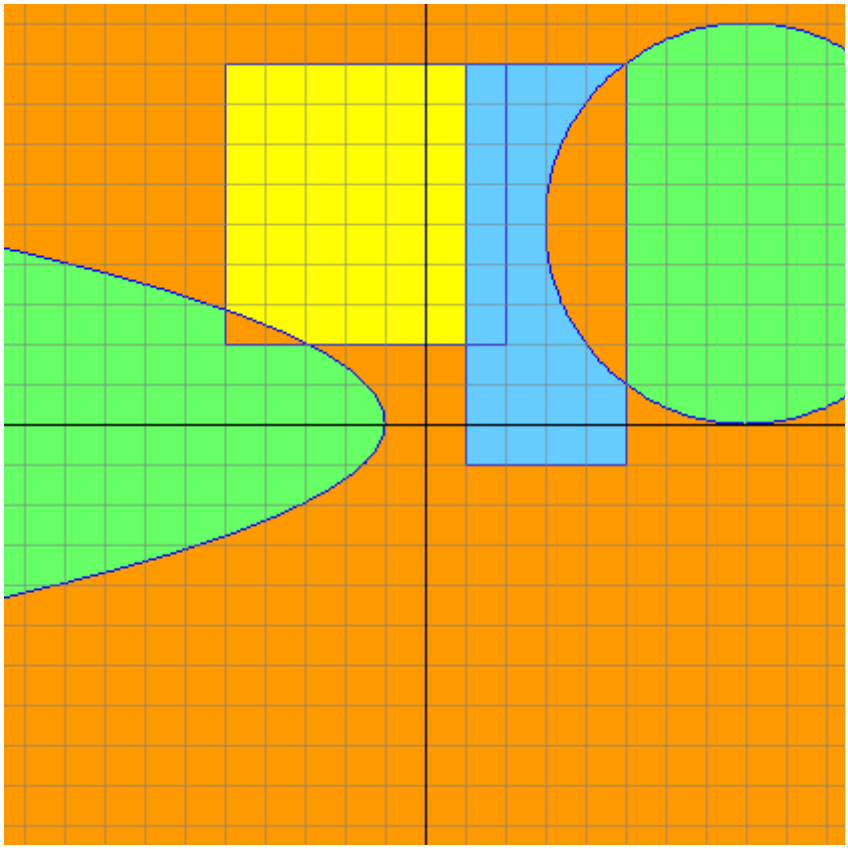
49.



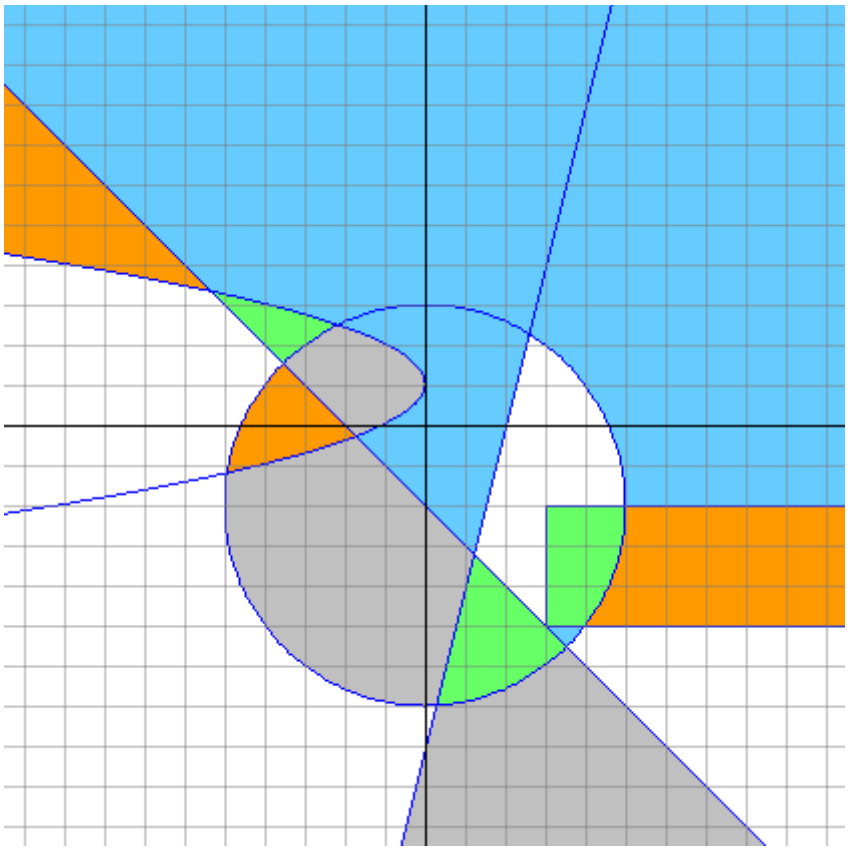
50.



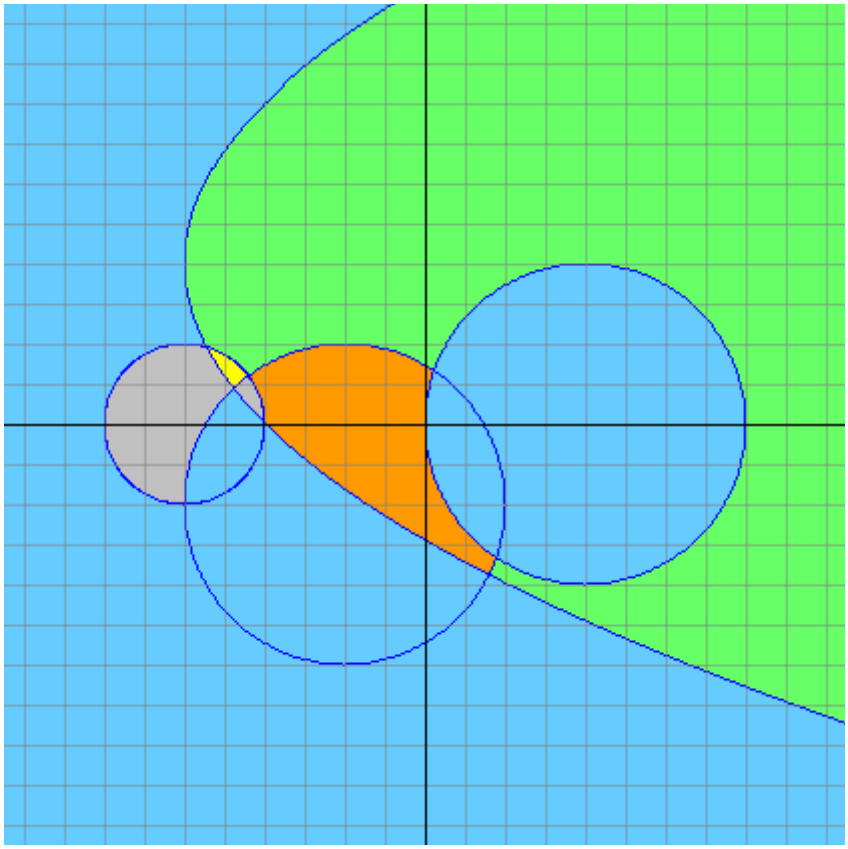
51.



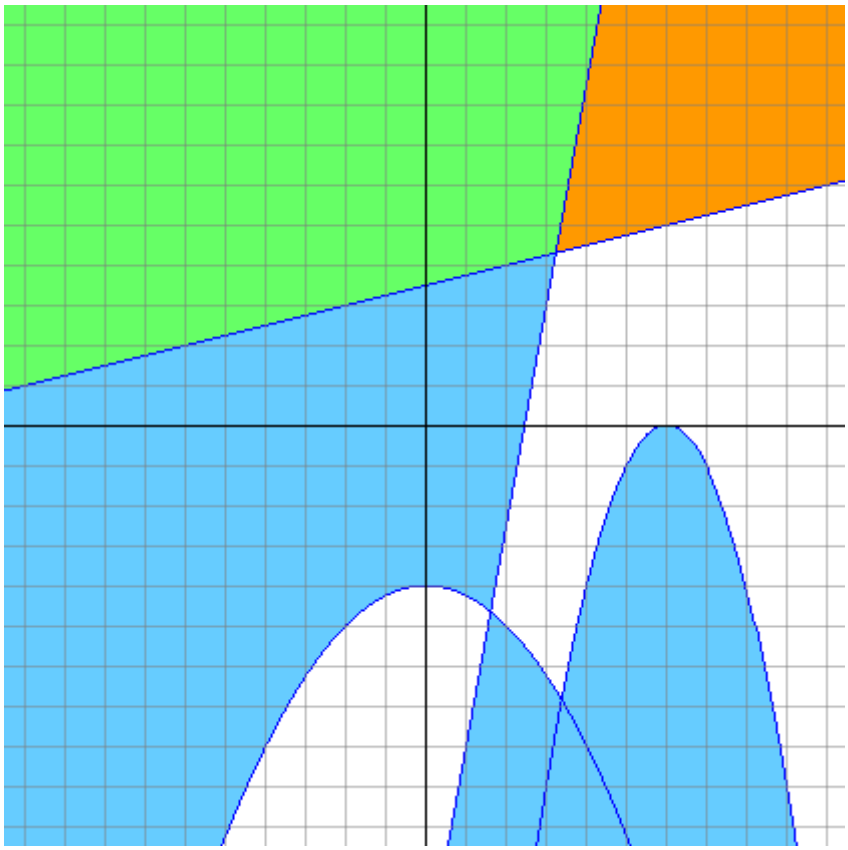
52.



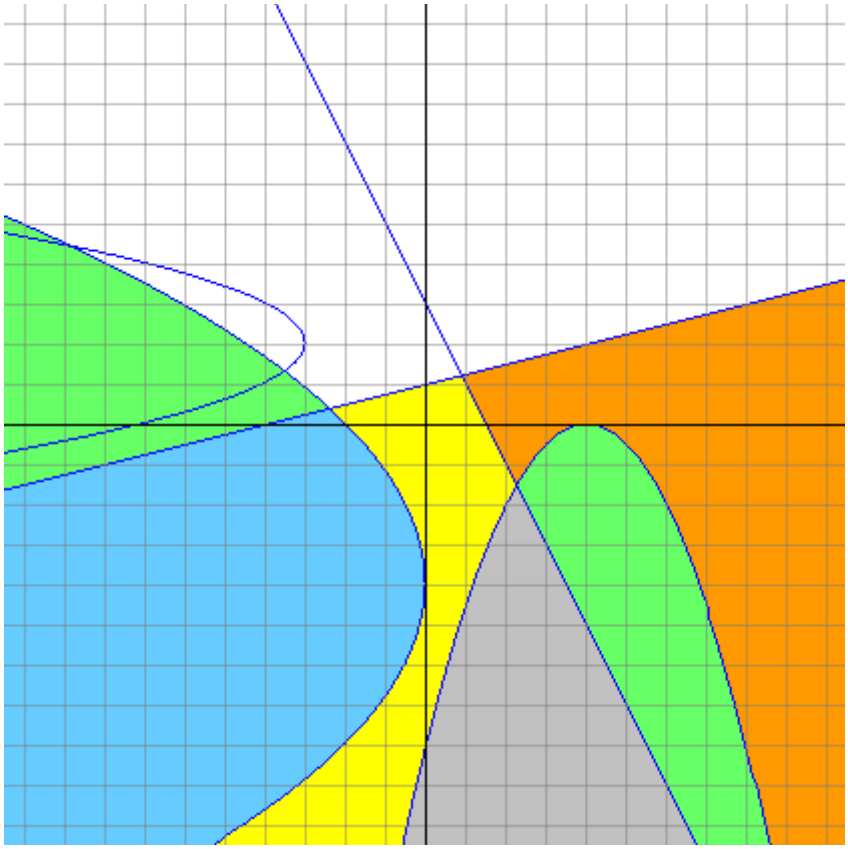
53.



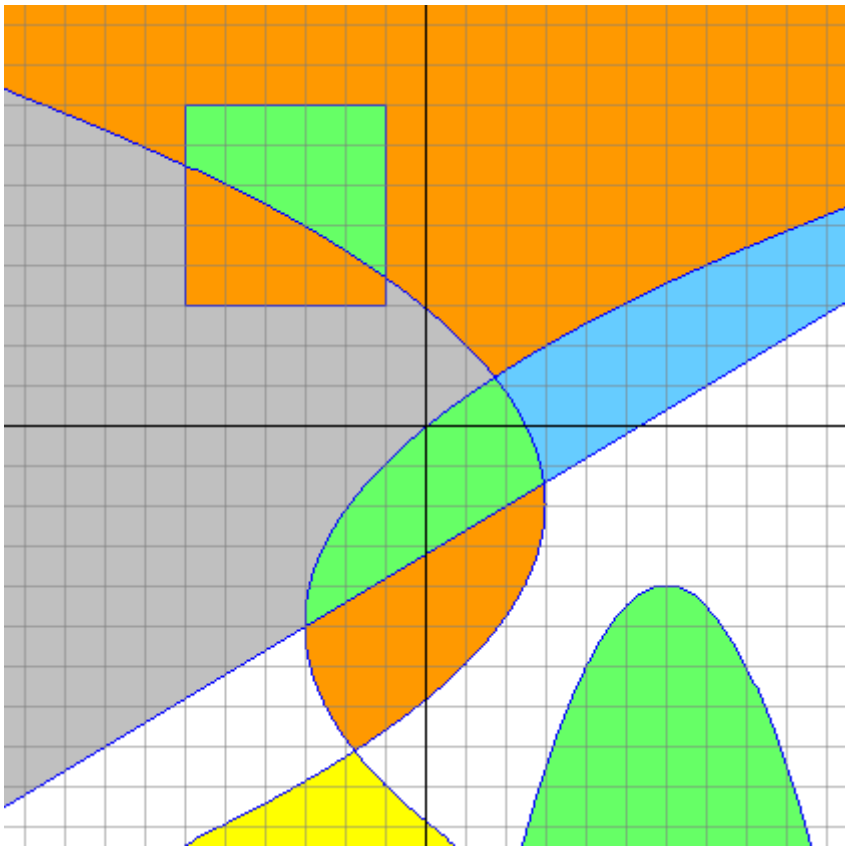
54.



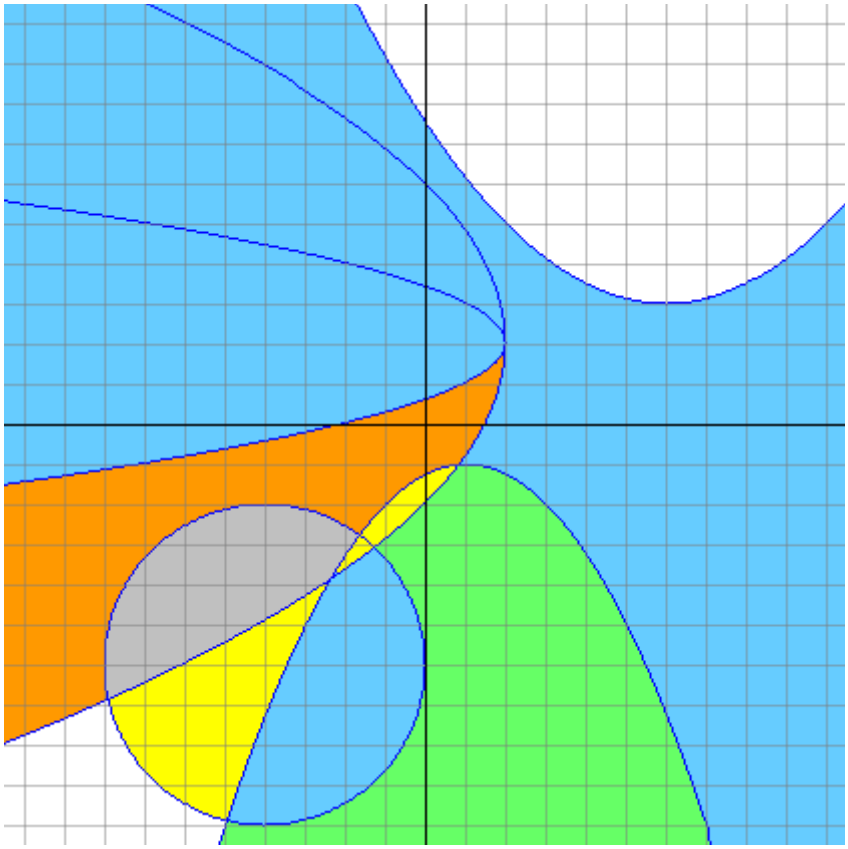
55.



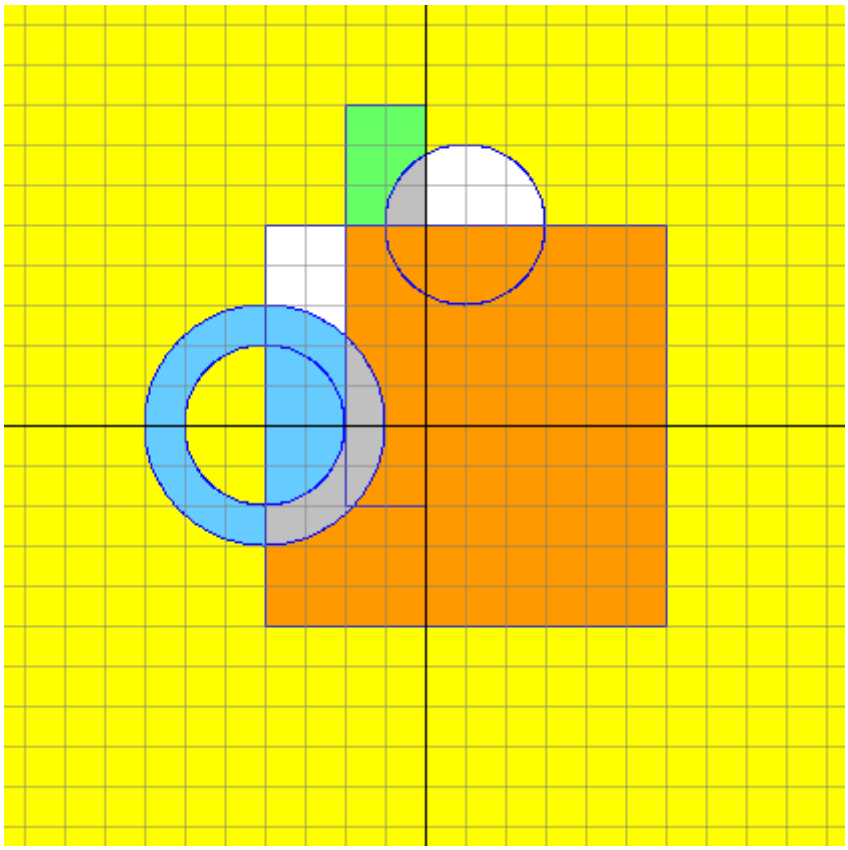
56.



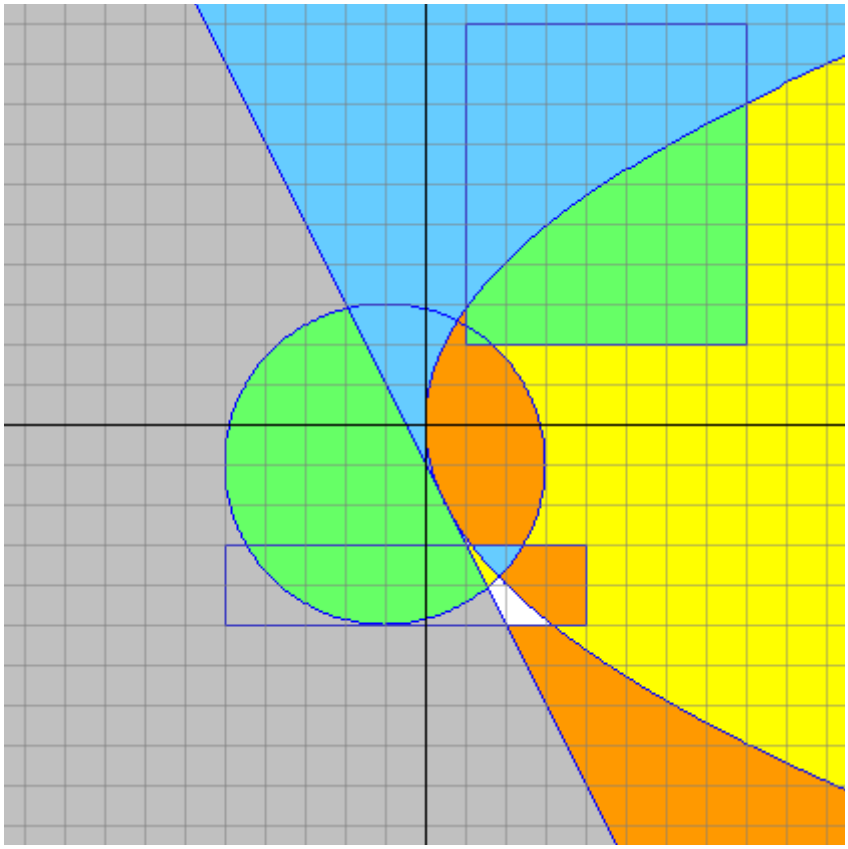
57.



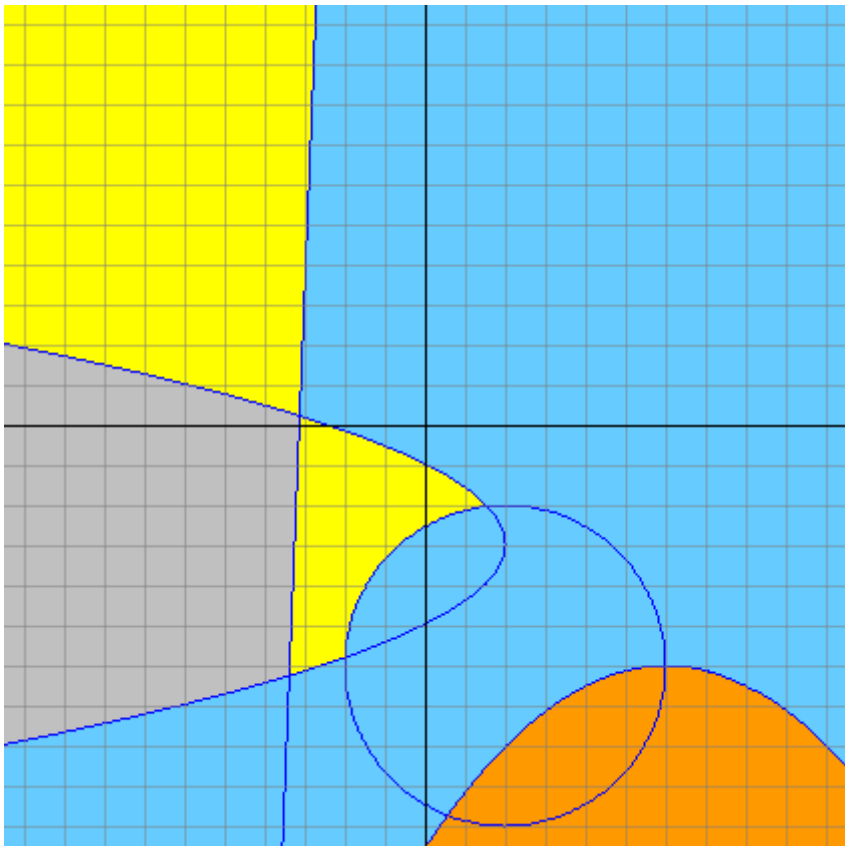
58.



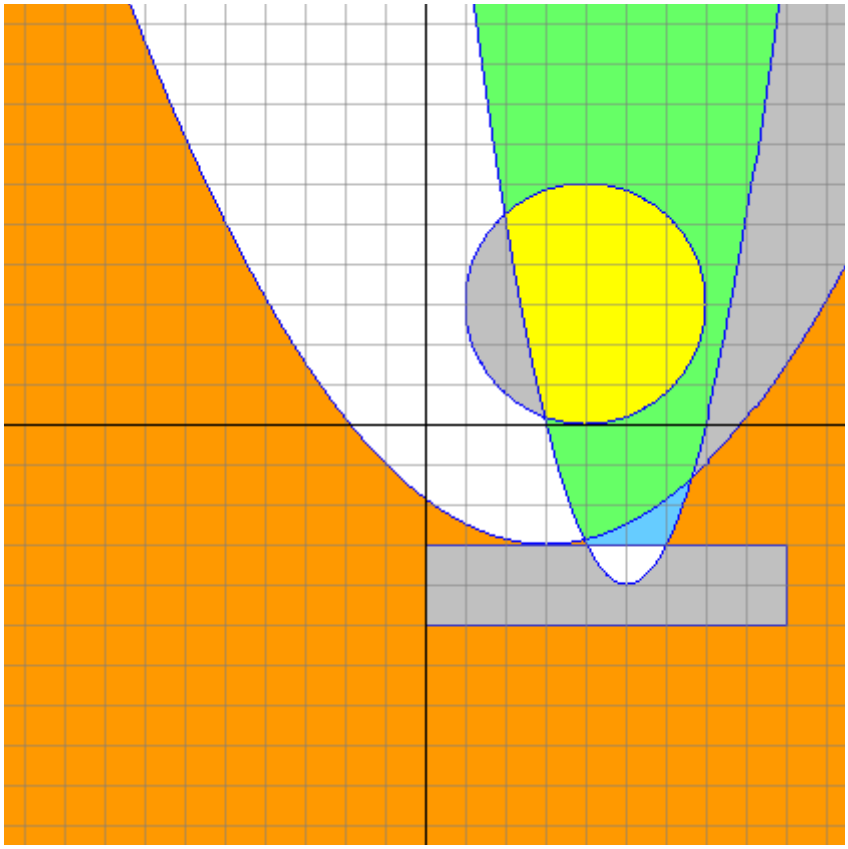
59.



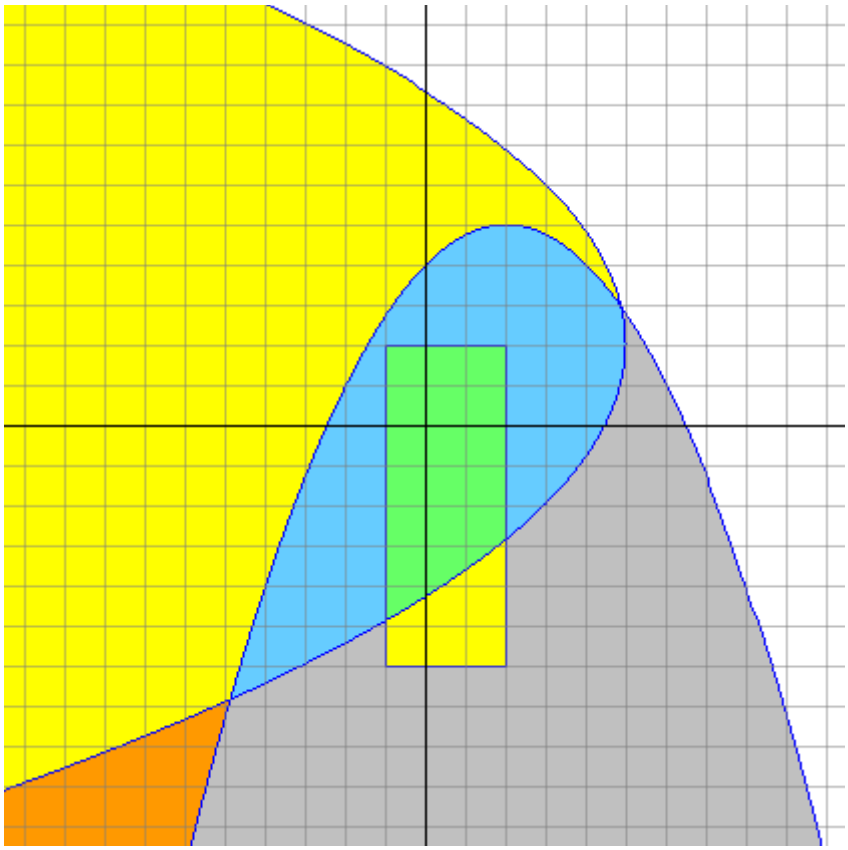
60.



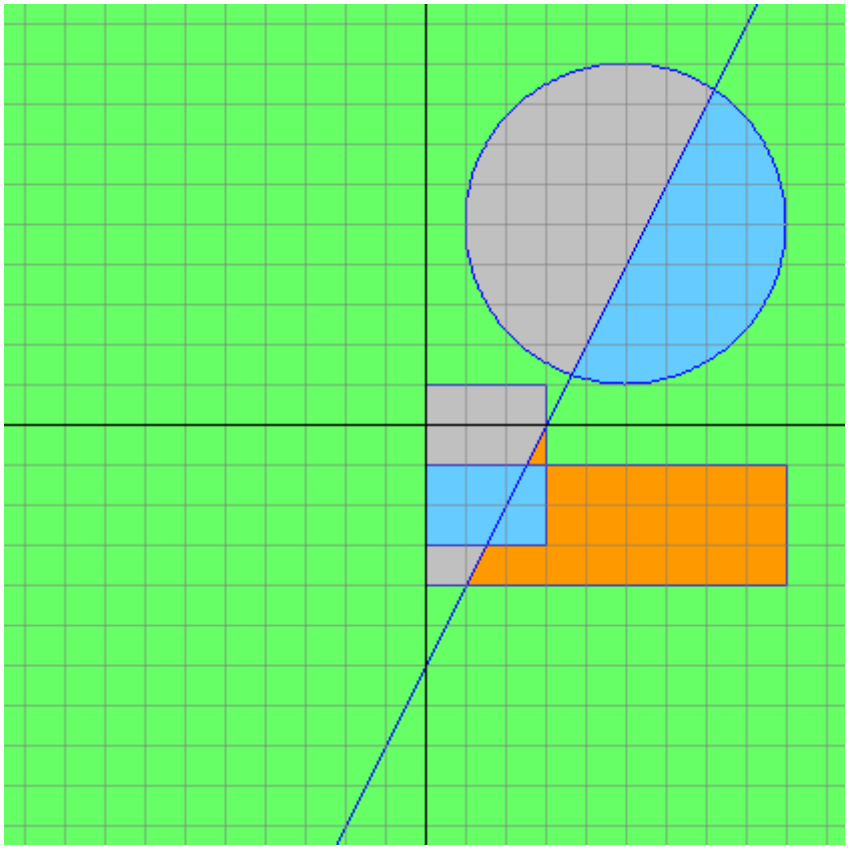
61.



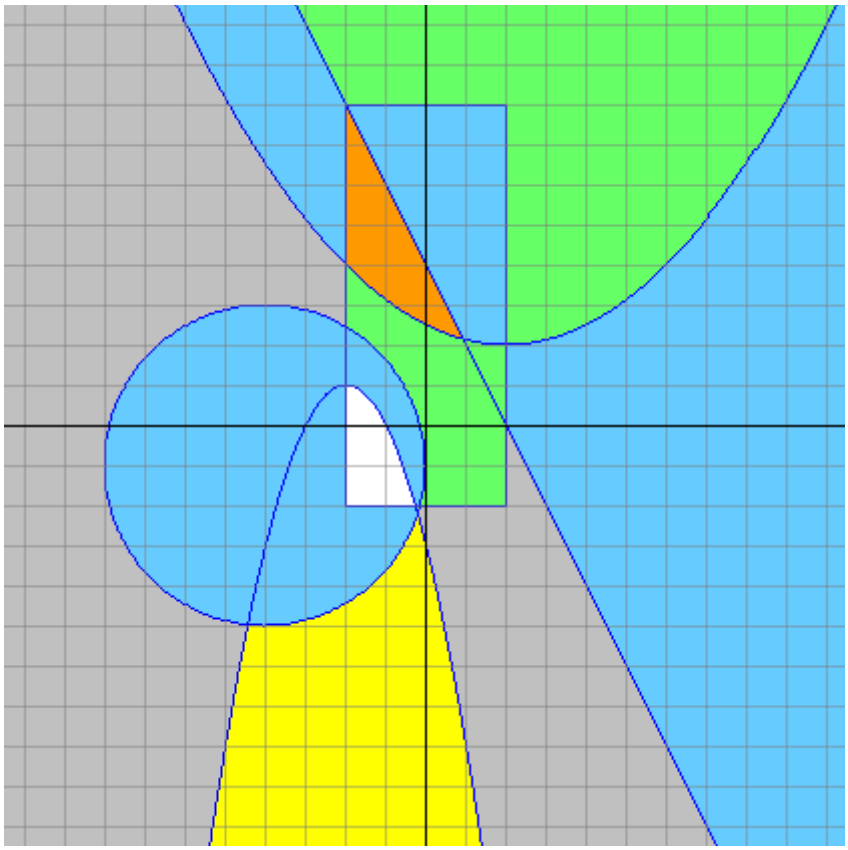
62.



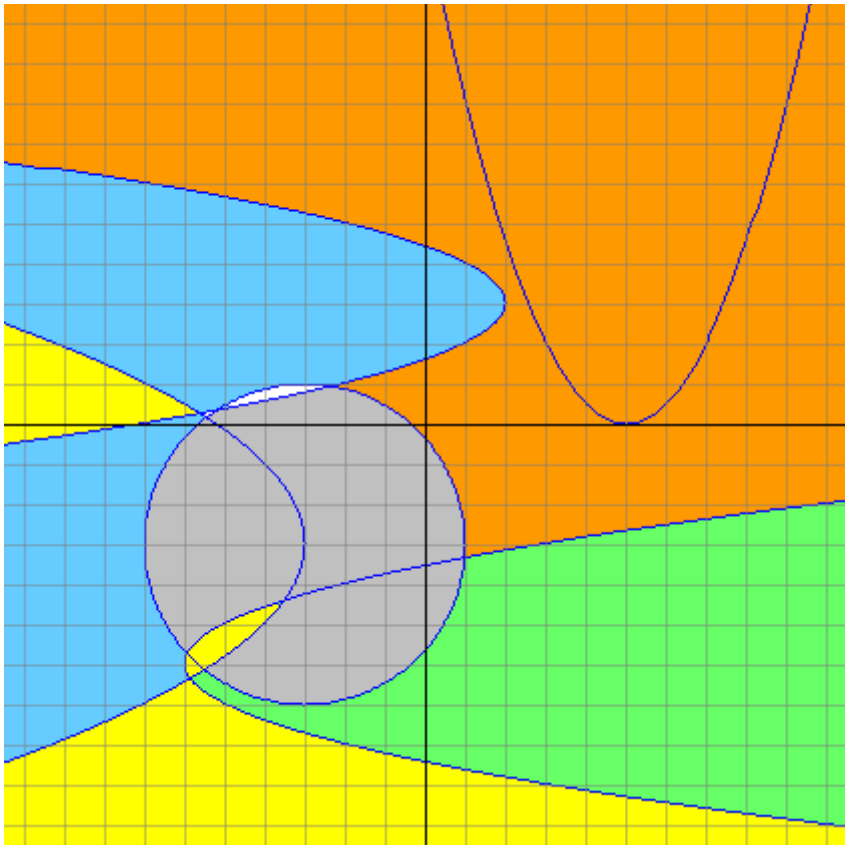
63.



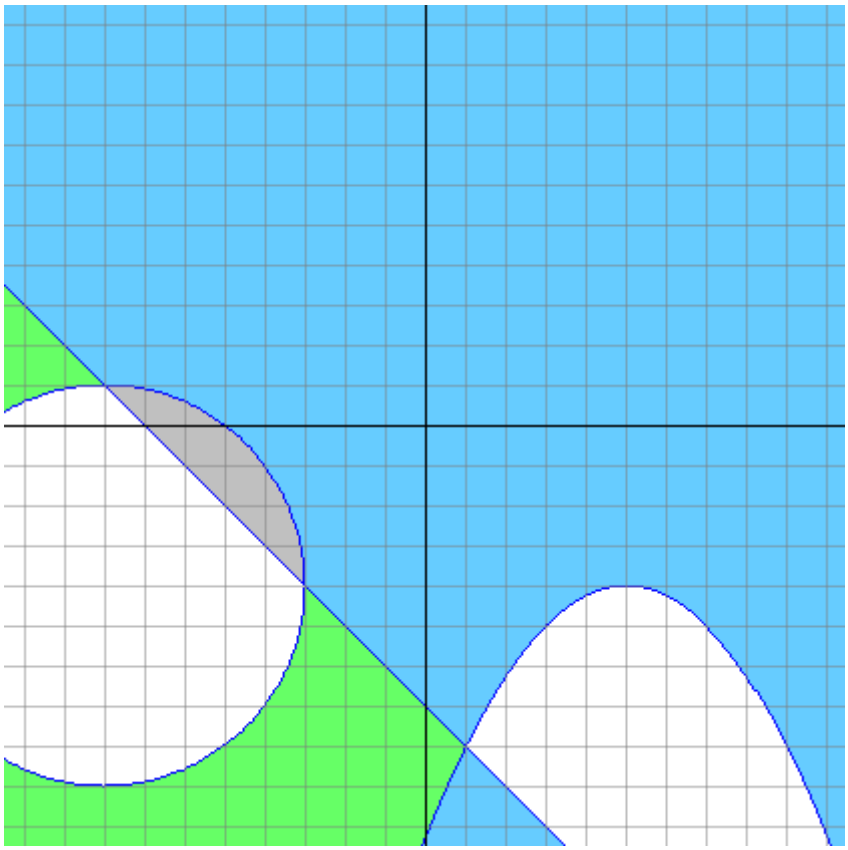
64.



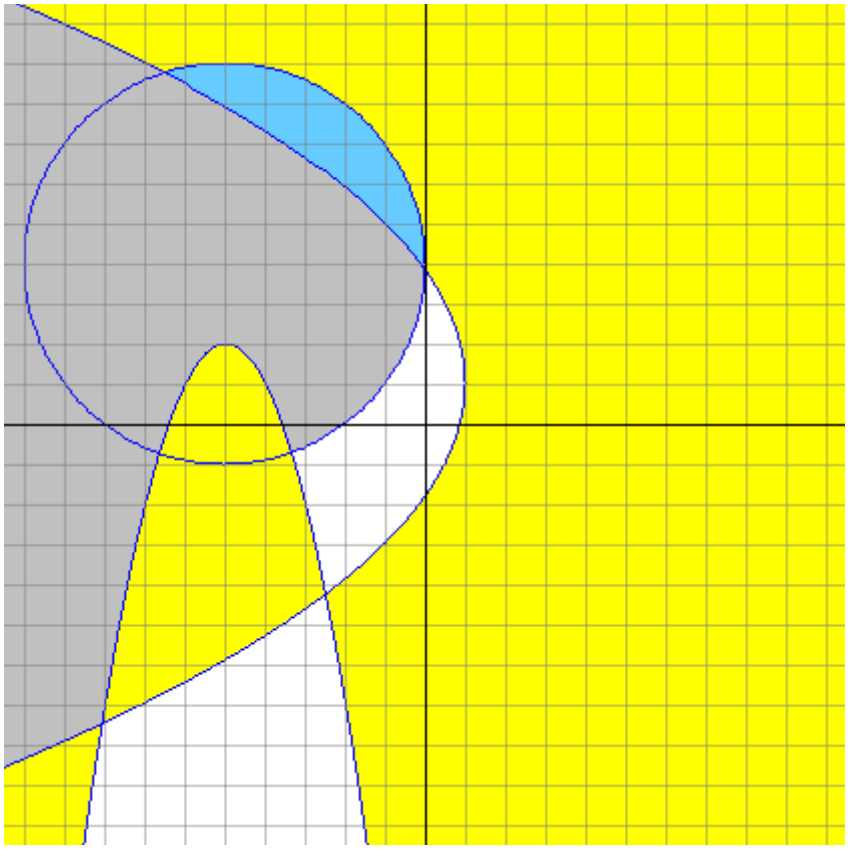
65.



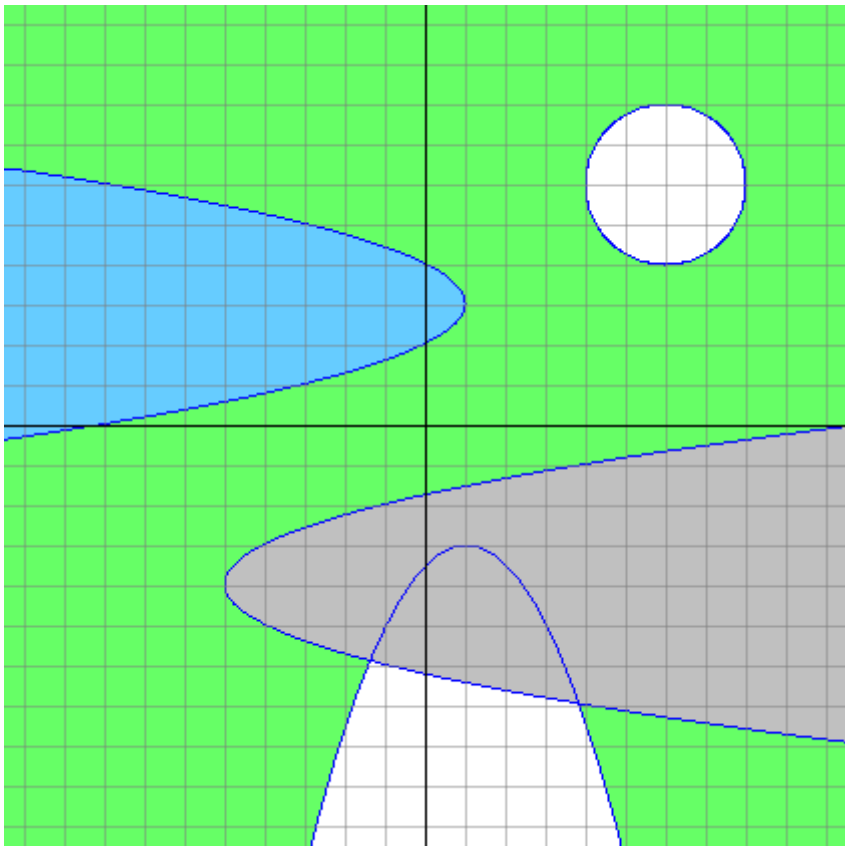
66.



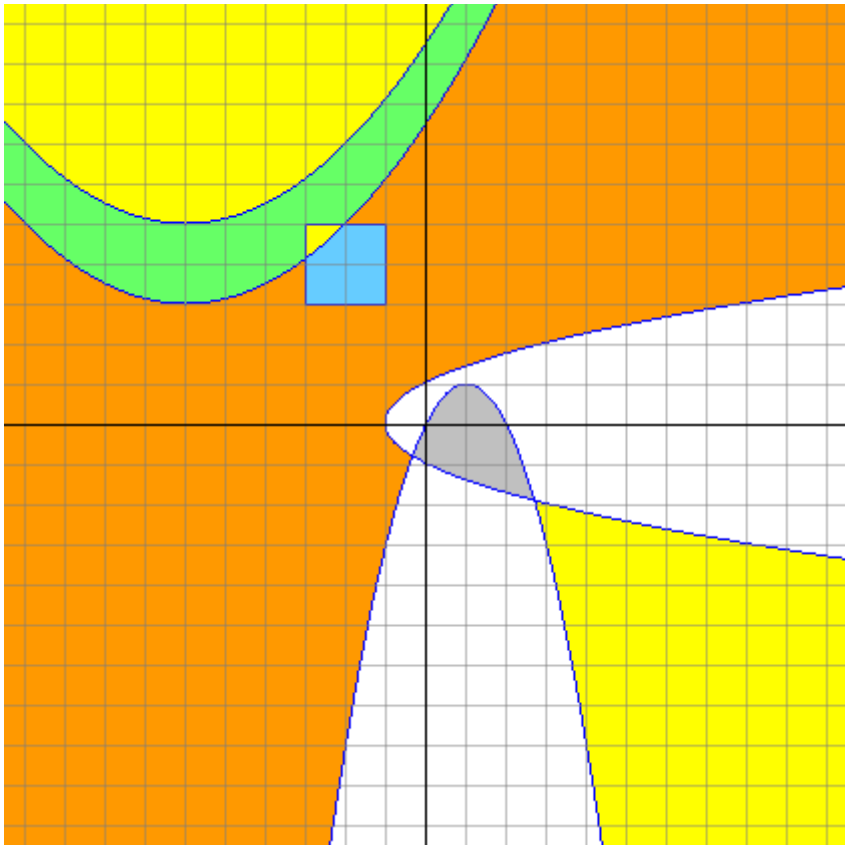
67.



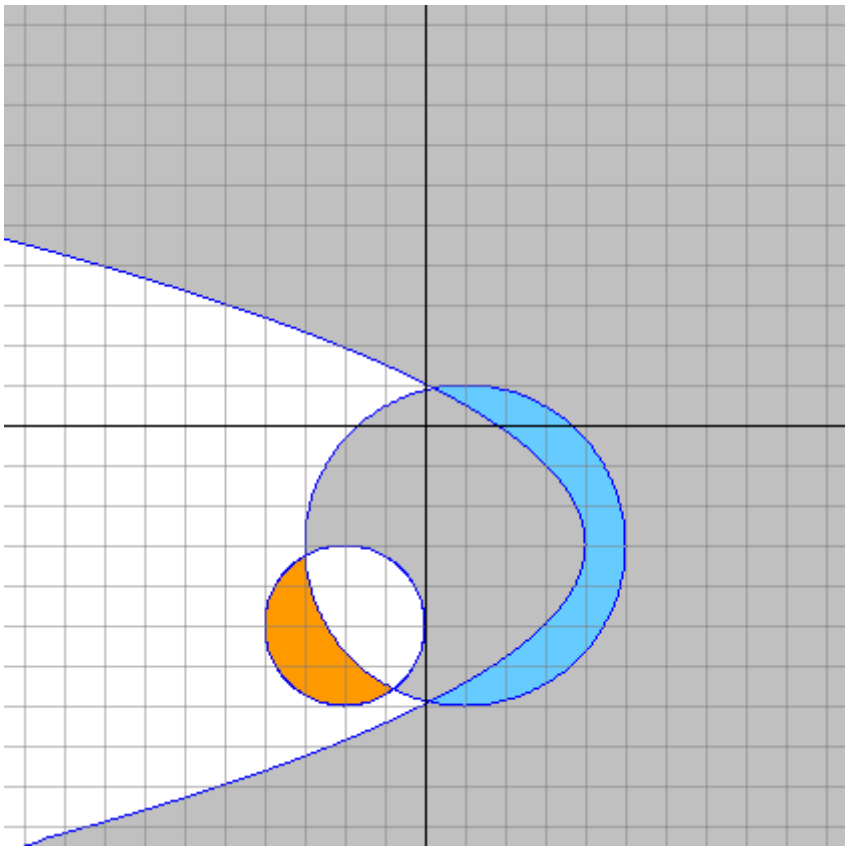
68.



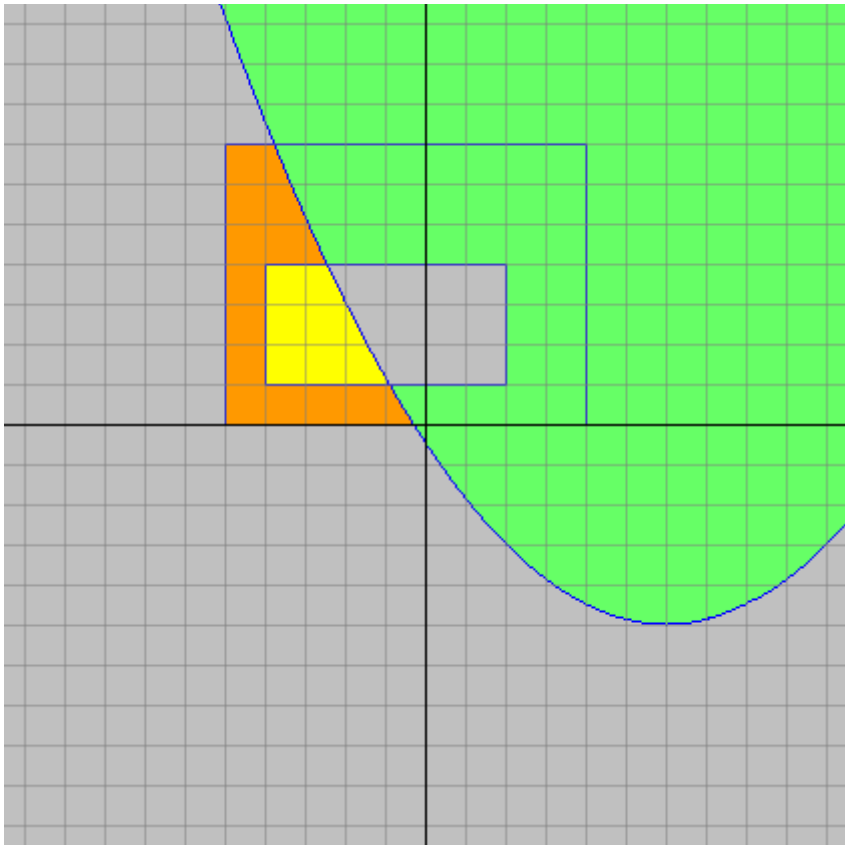
69.



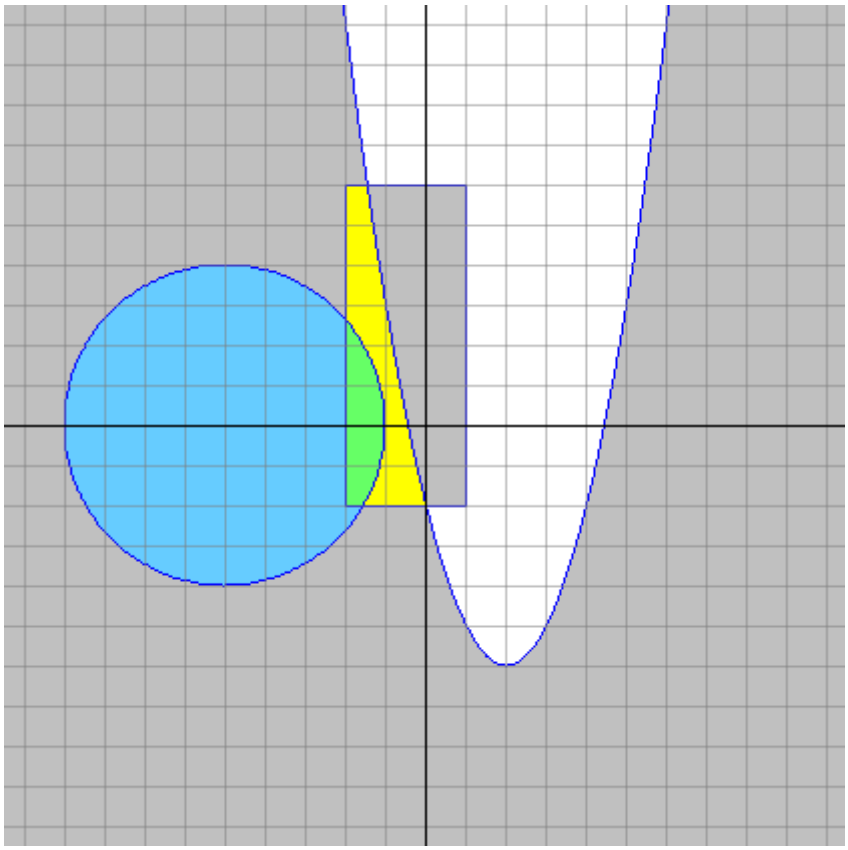
70.



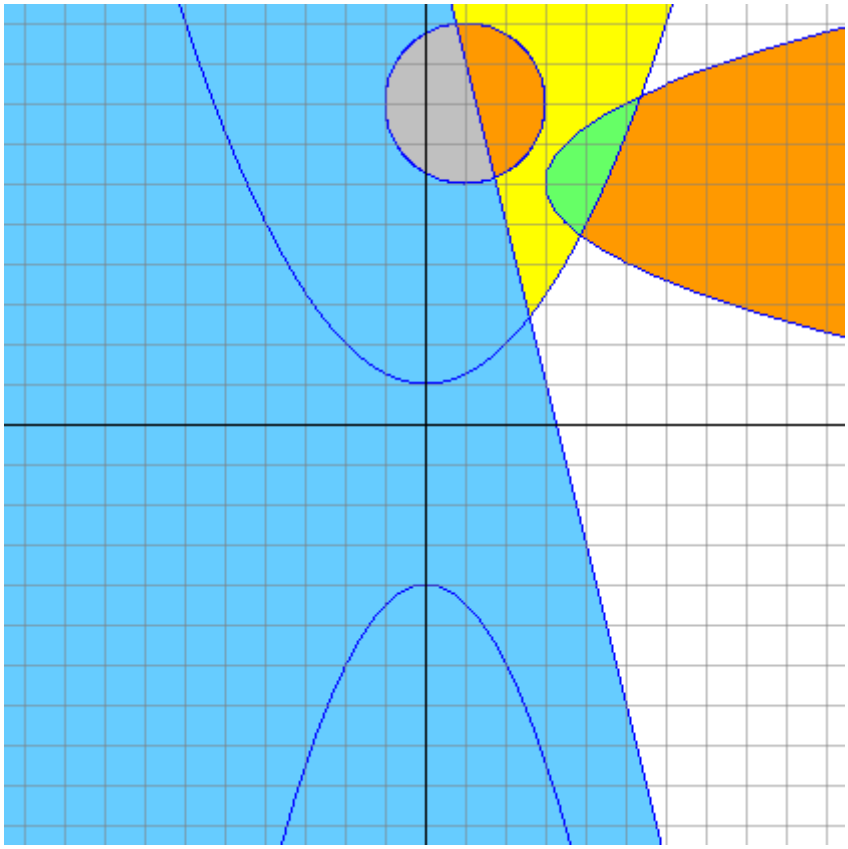
71.



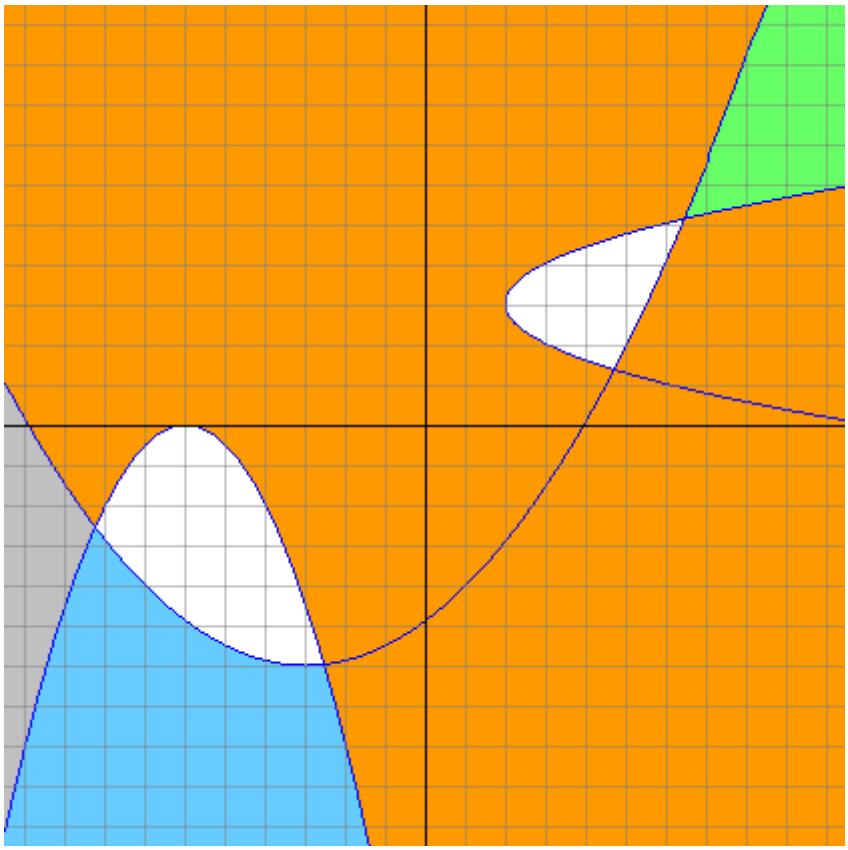
72.



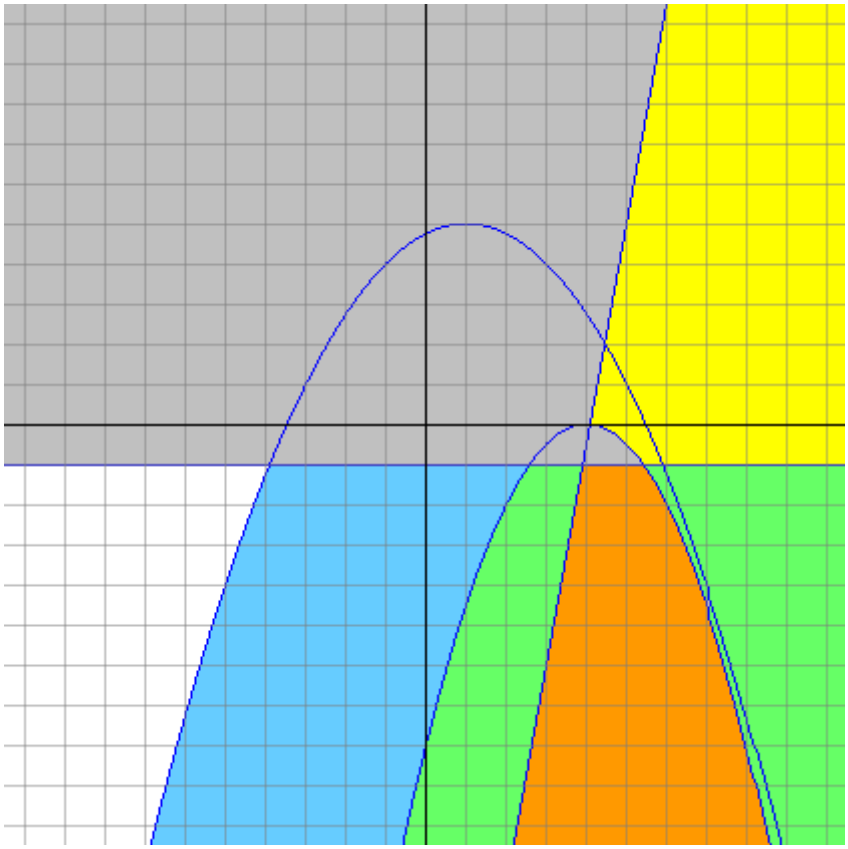
73.



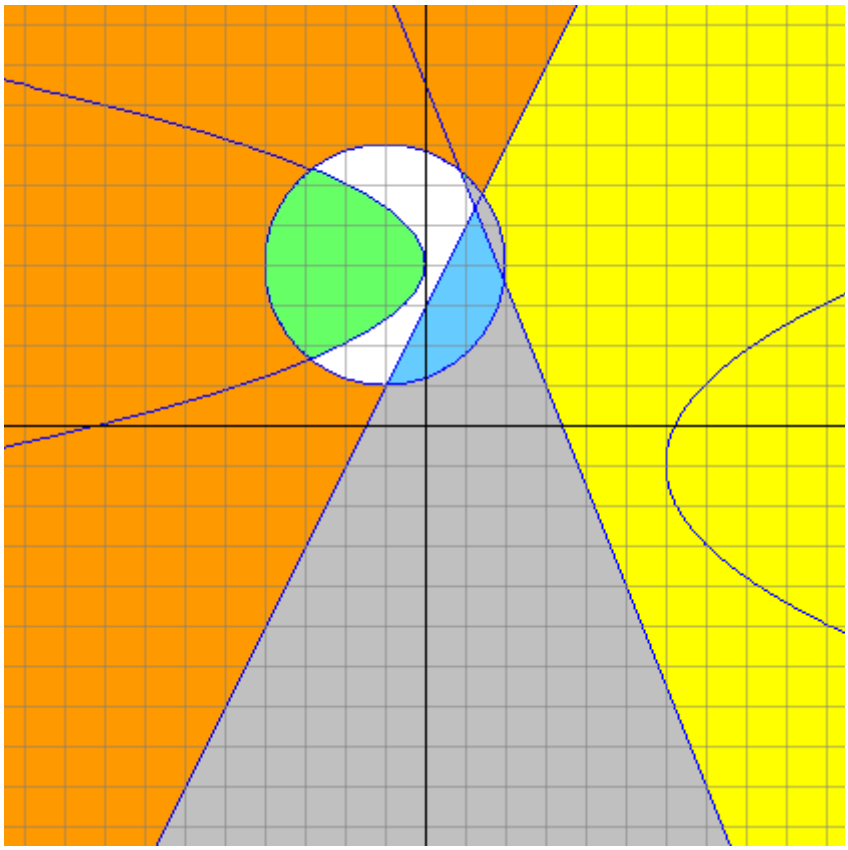
74.



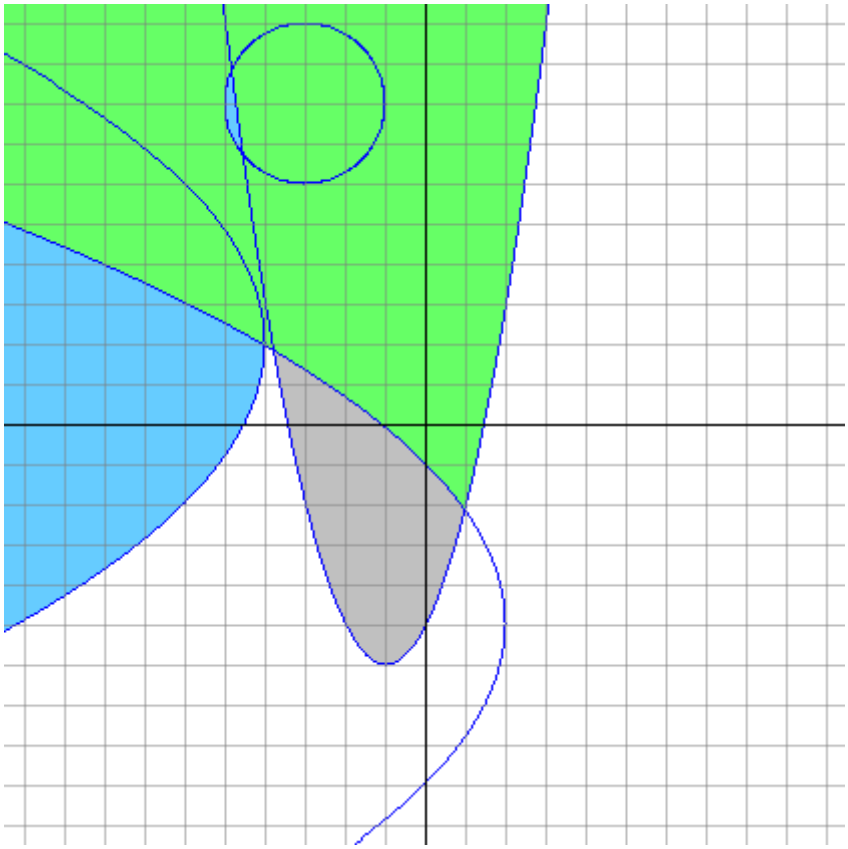
75.



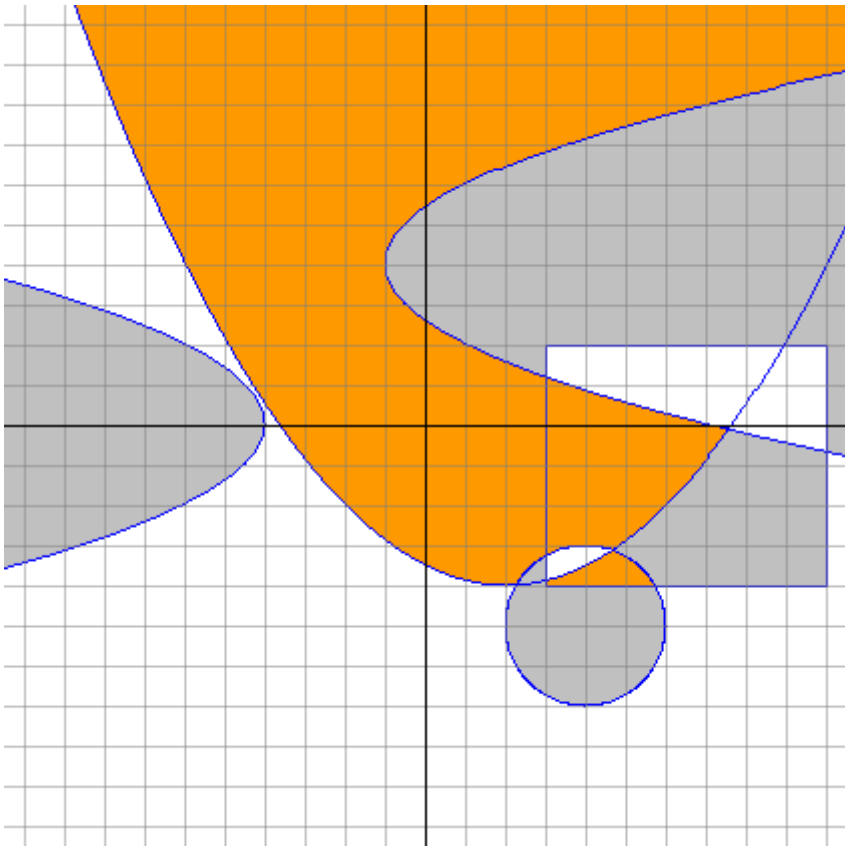
76.



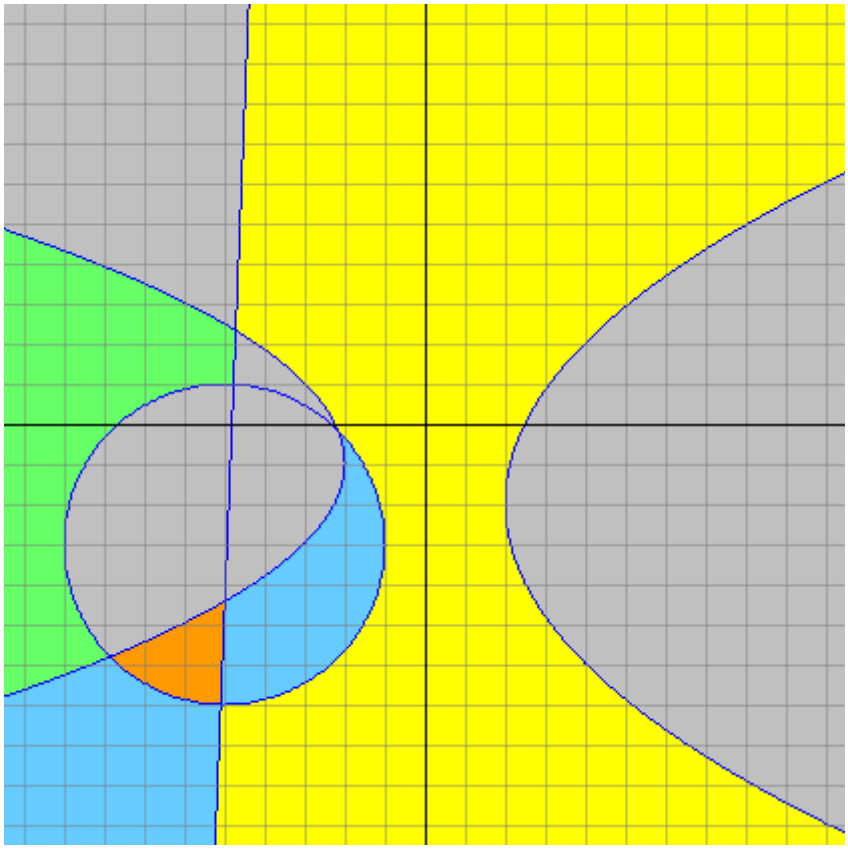
77.



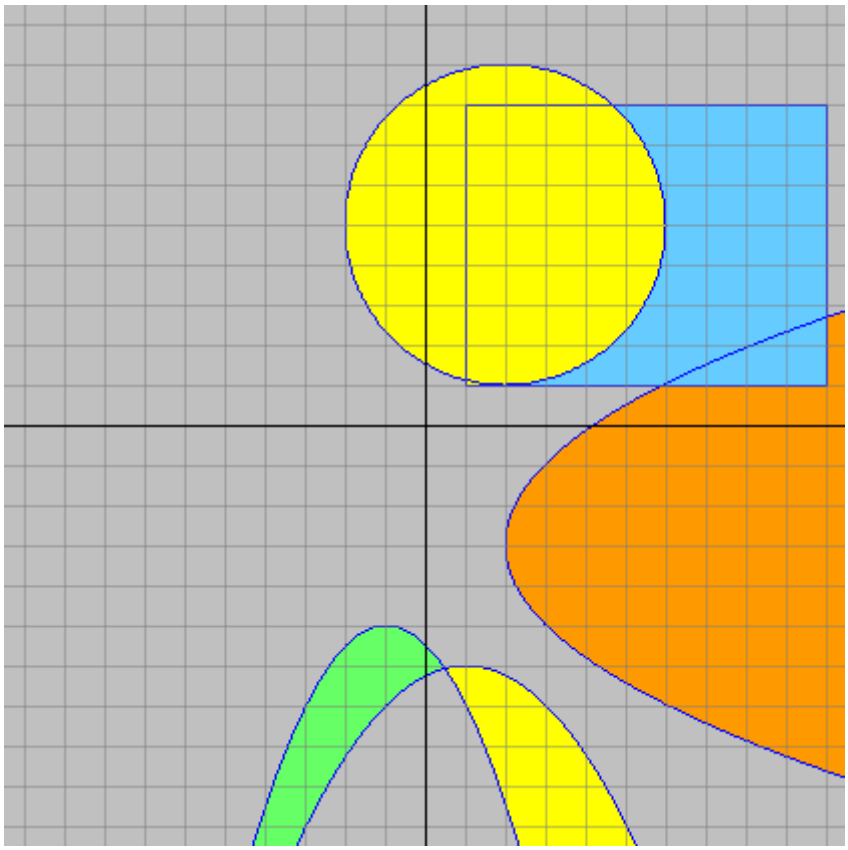
78.



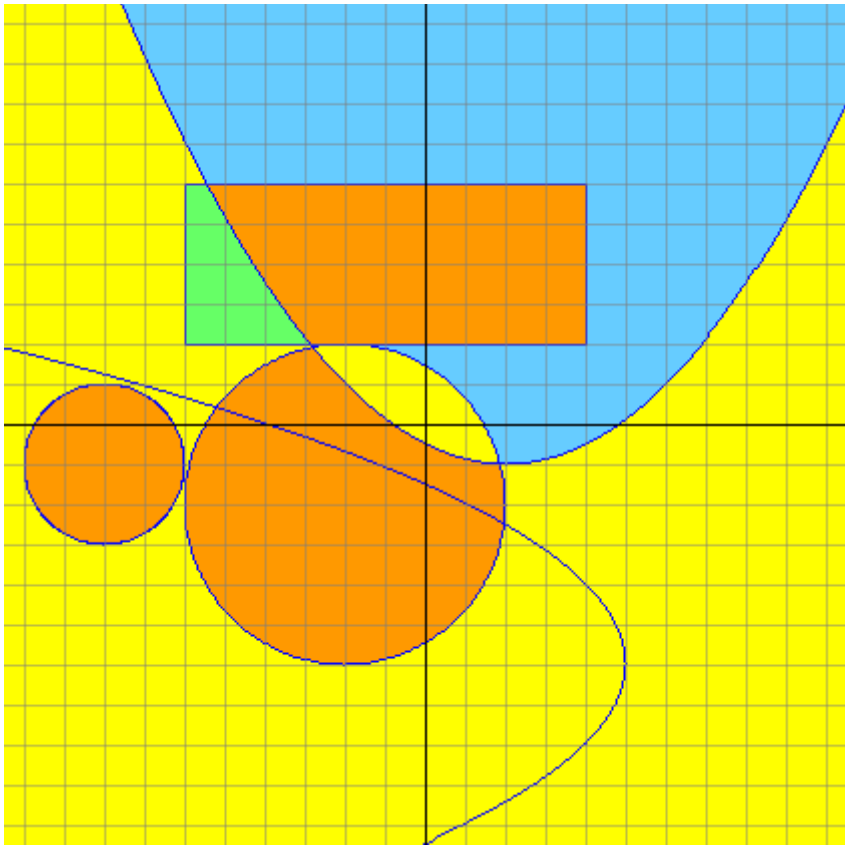
79.



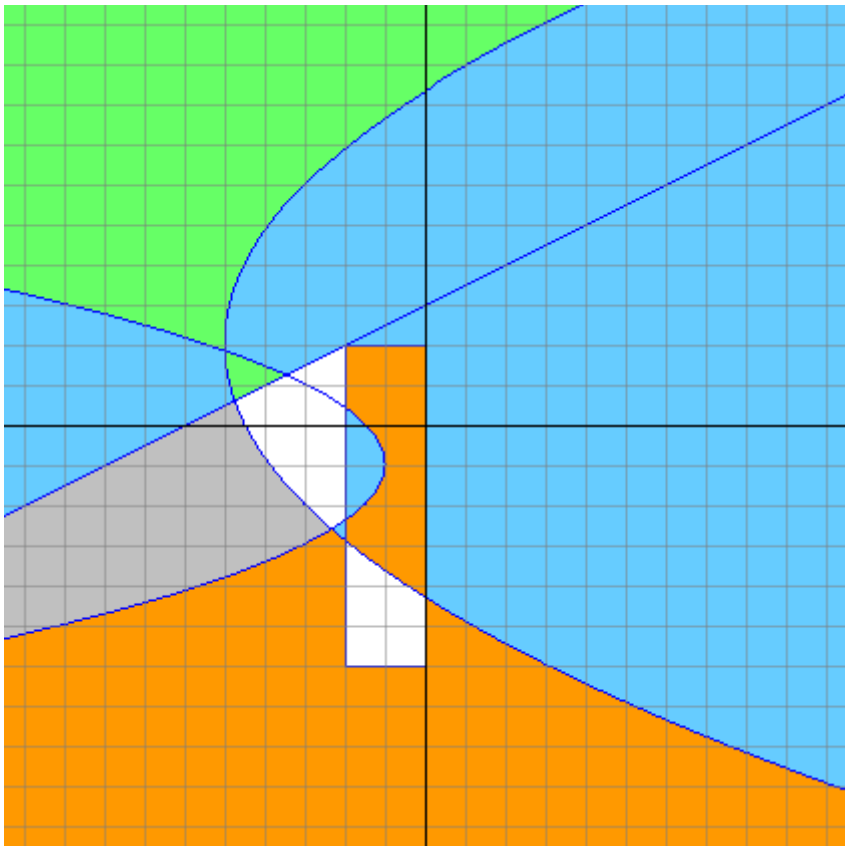
80.



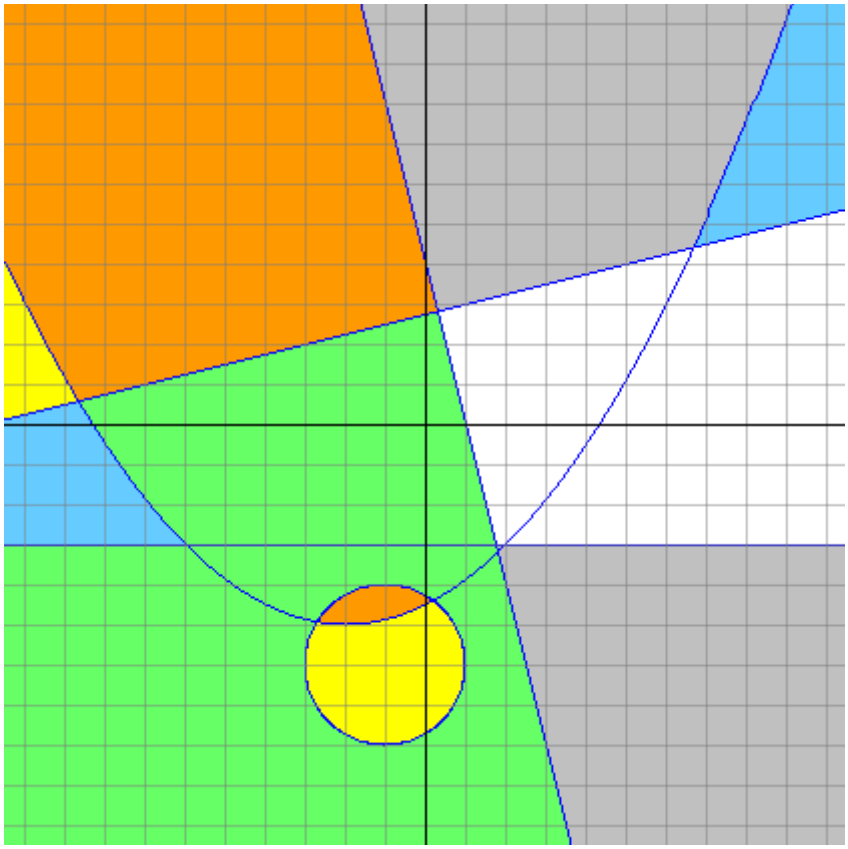
81.



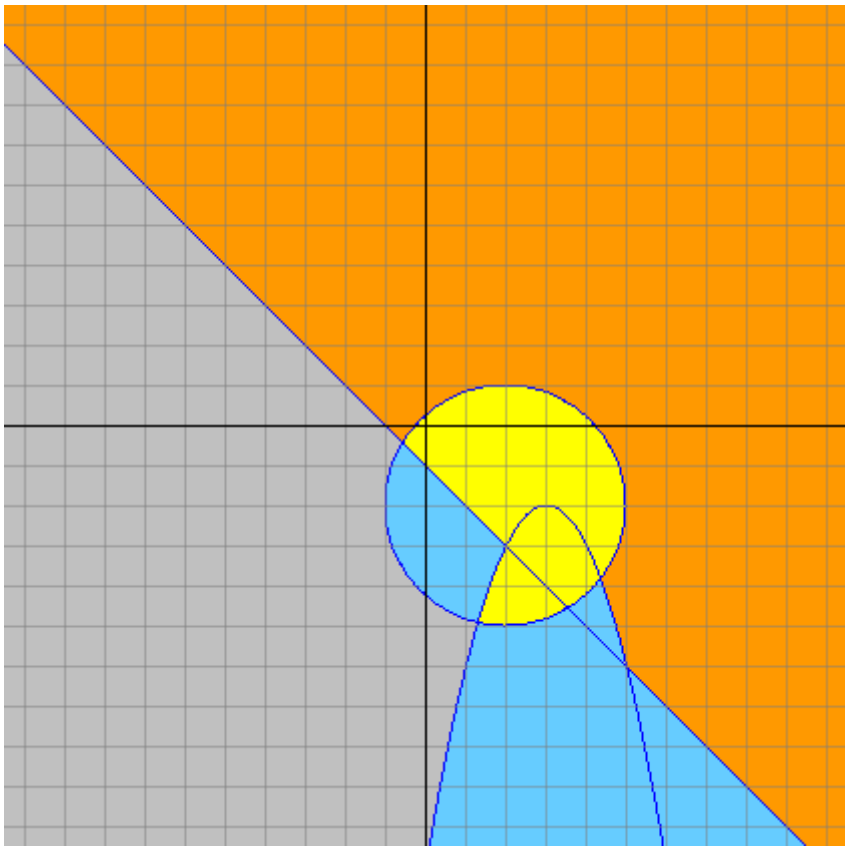
82.



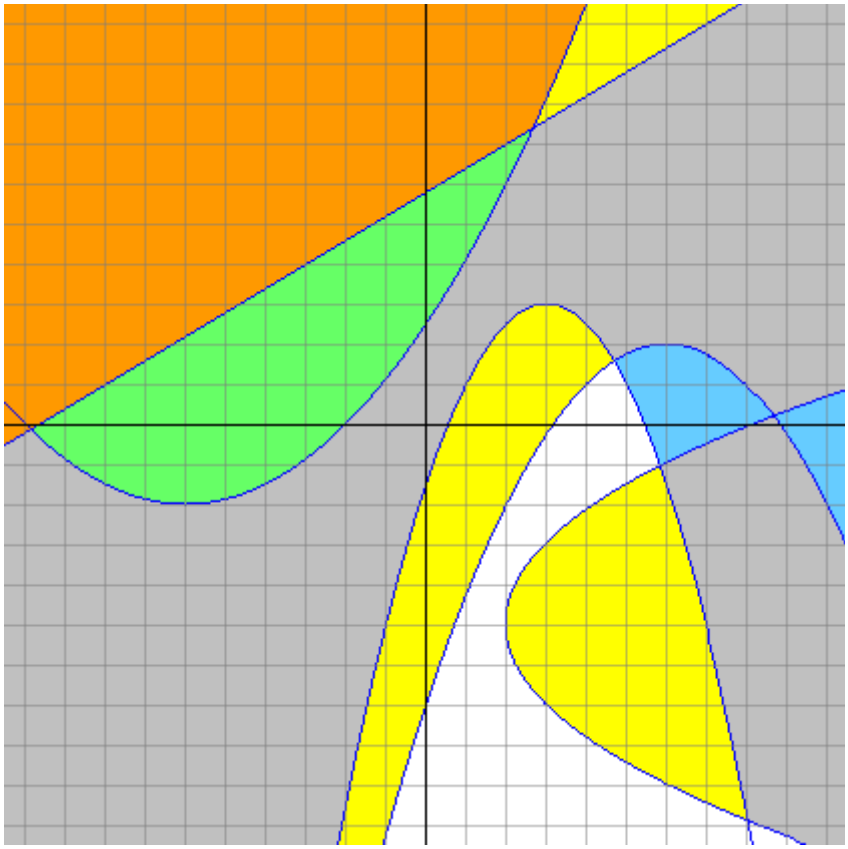
83.



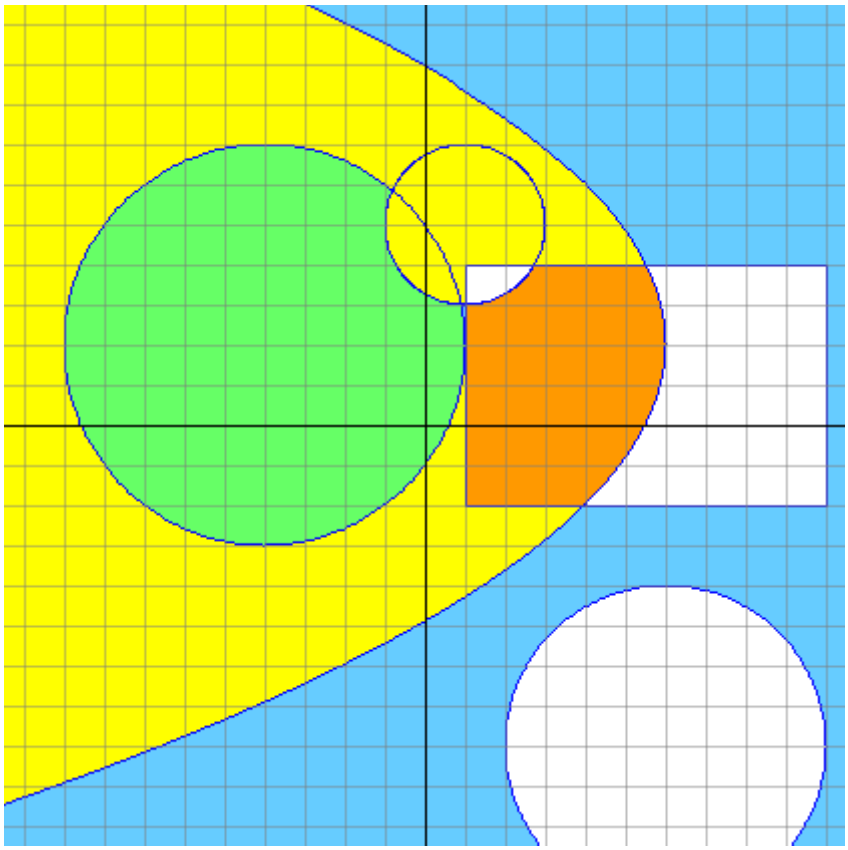
84.



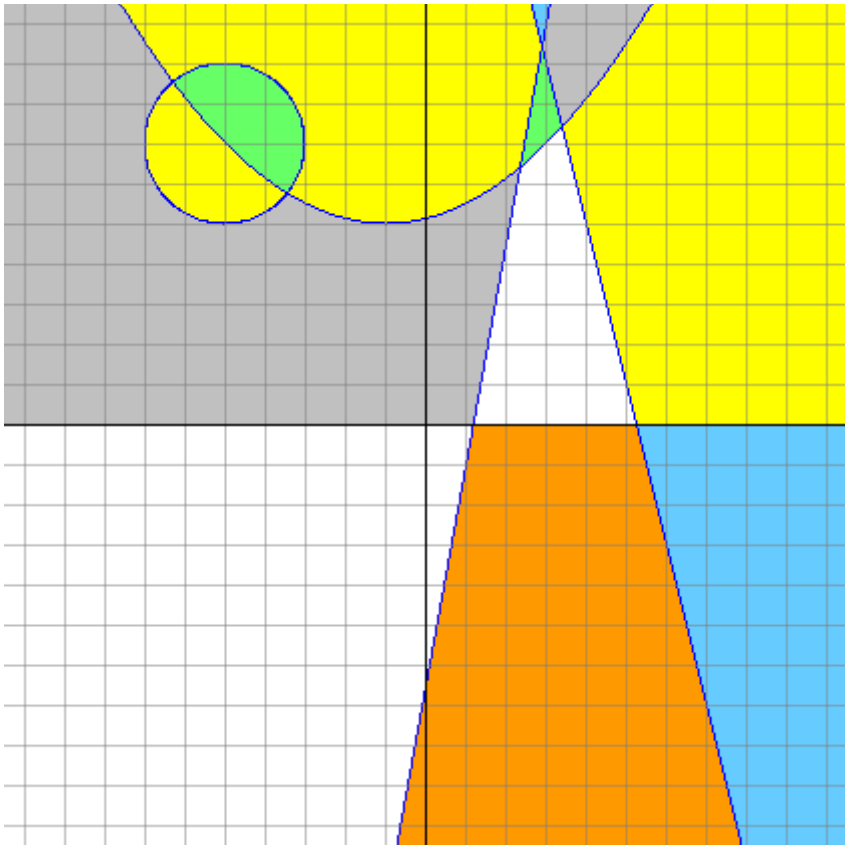
85.



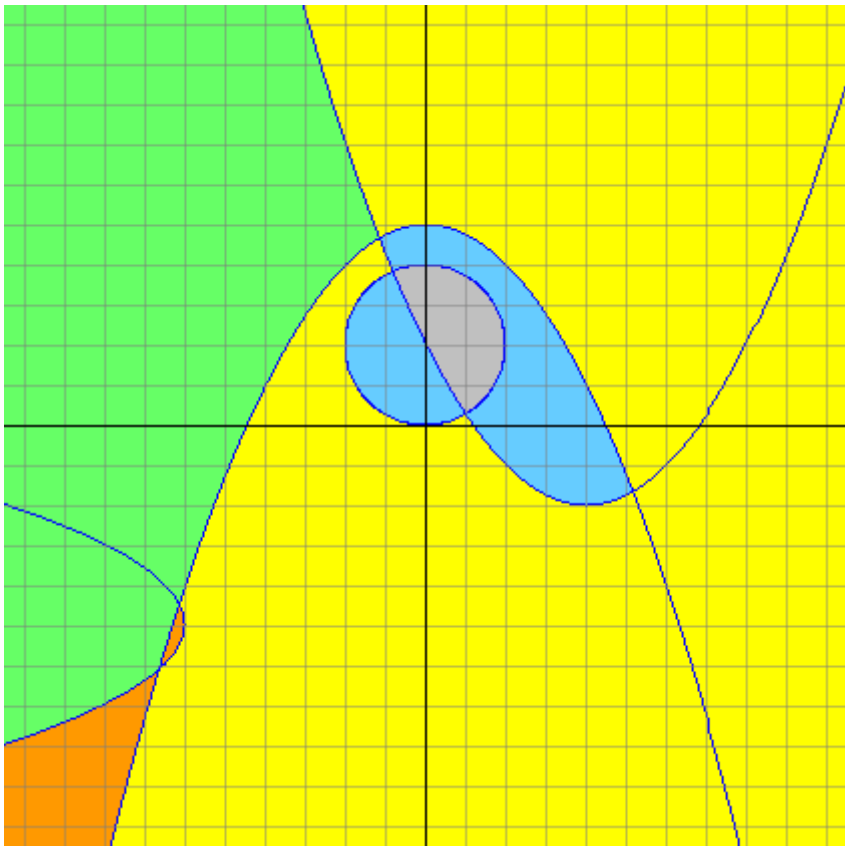
86.



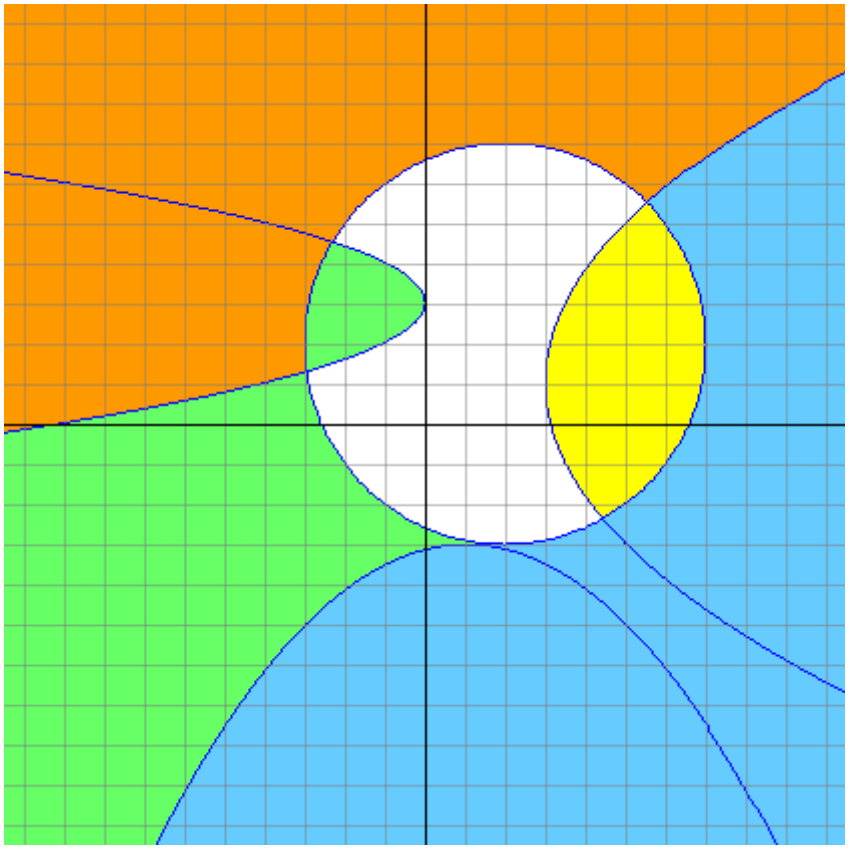
87.



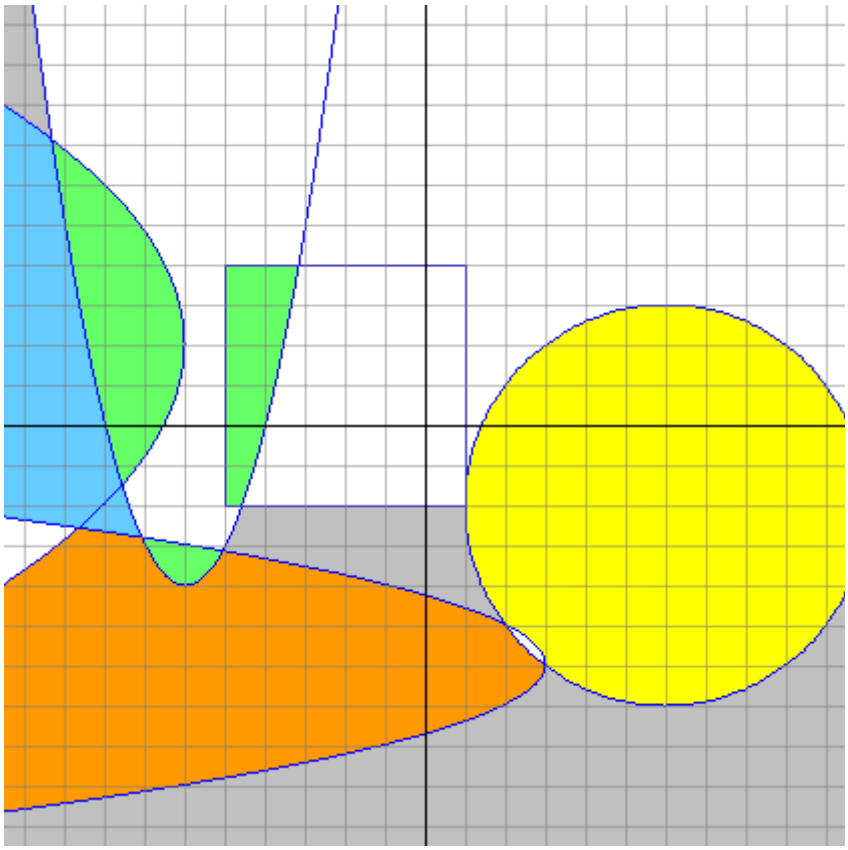
88.



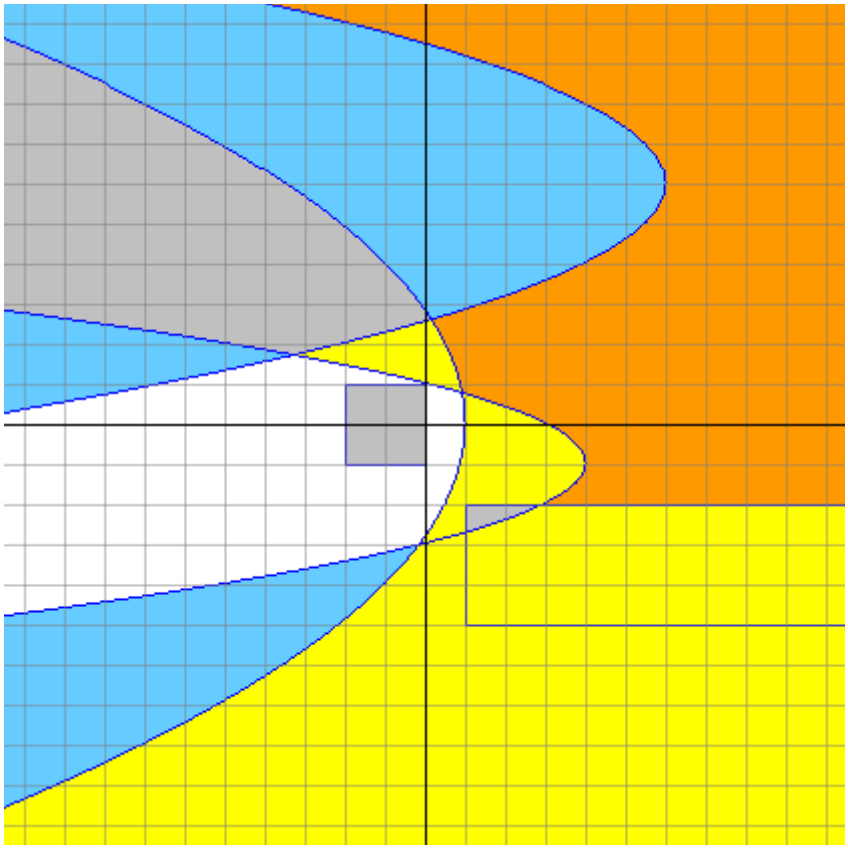
89.



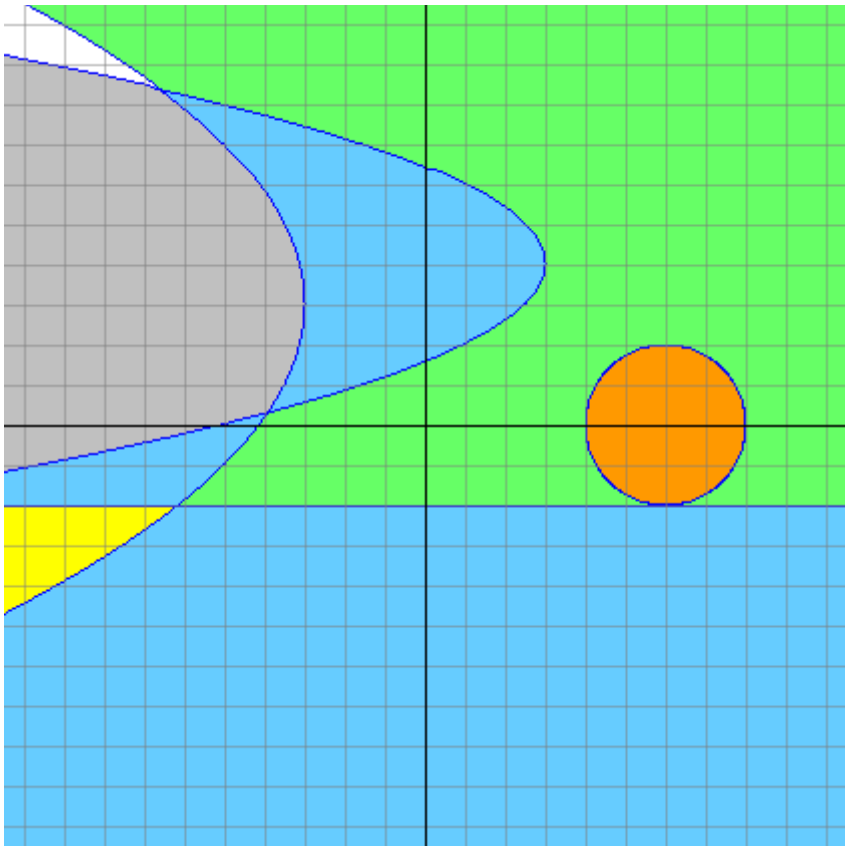
90.



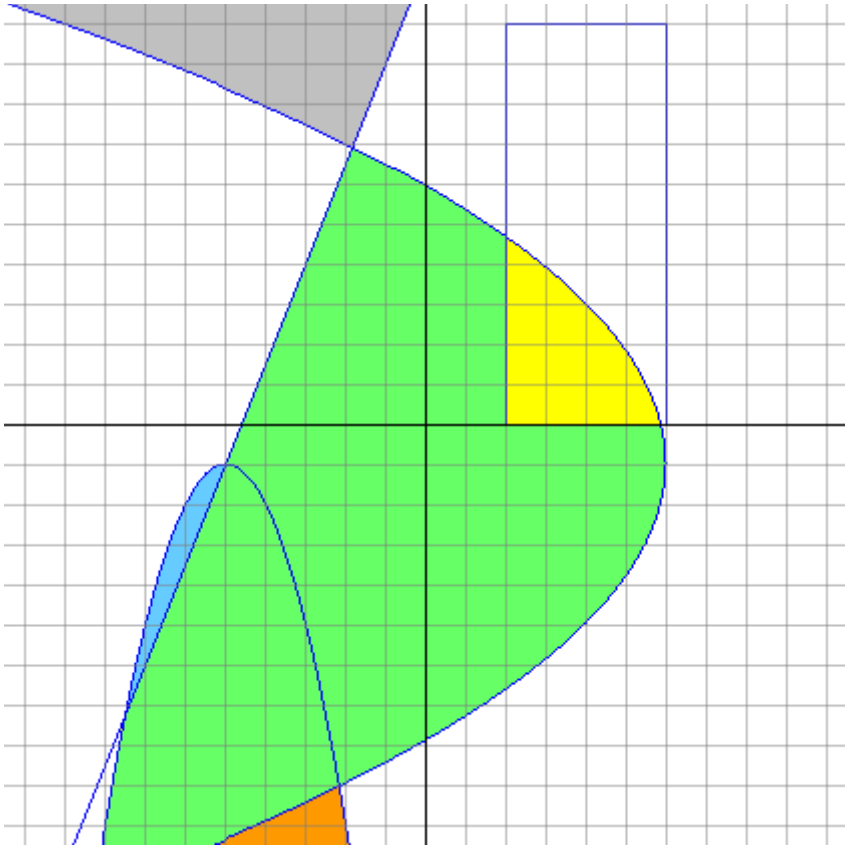
91.



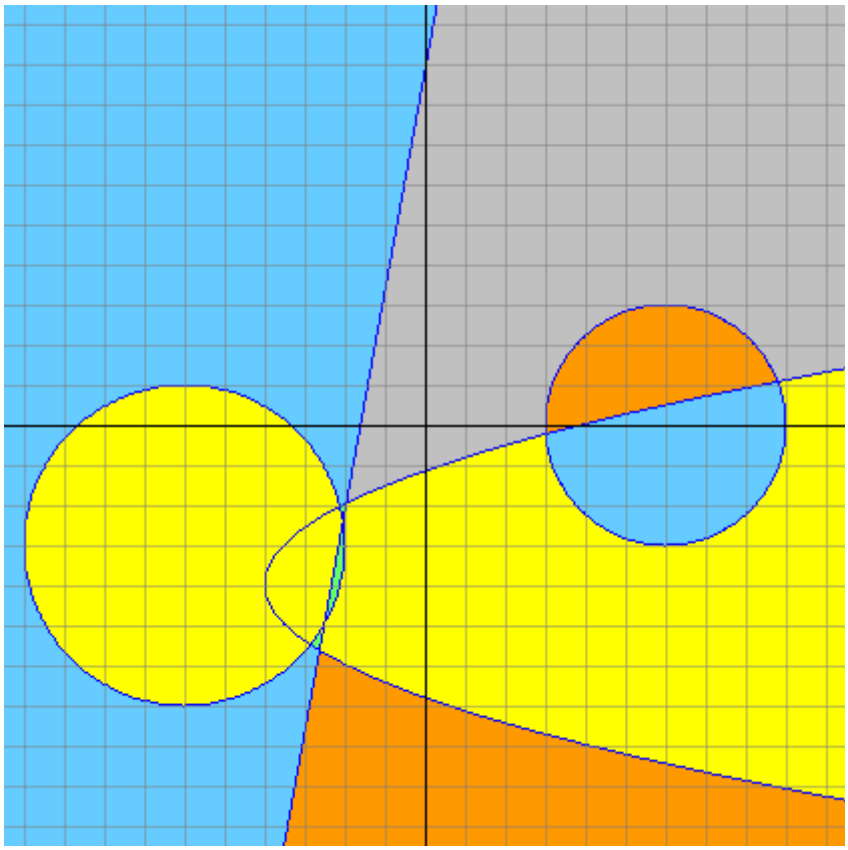
92.



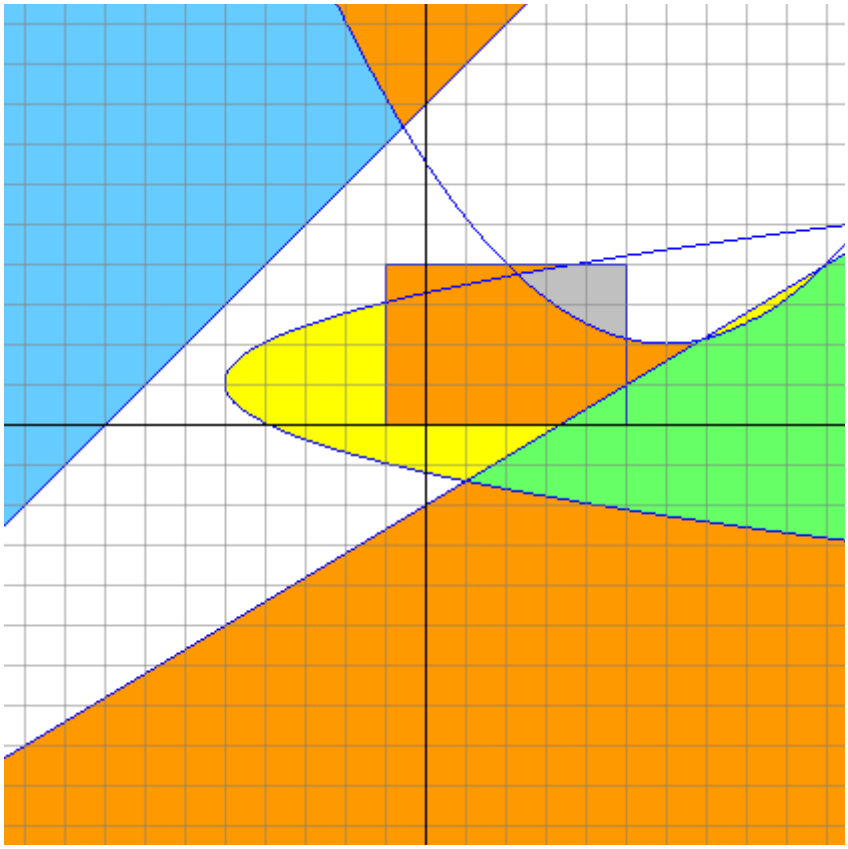
93.



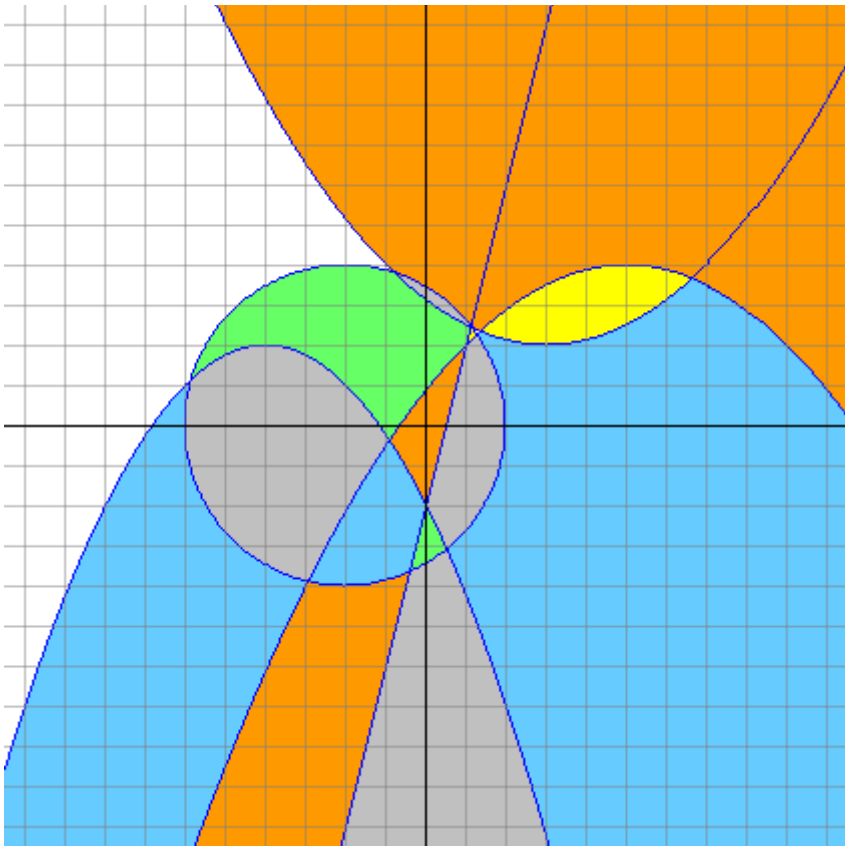
94.



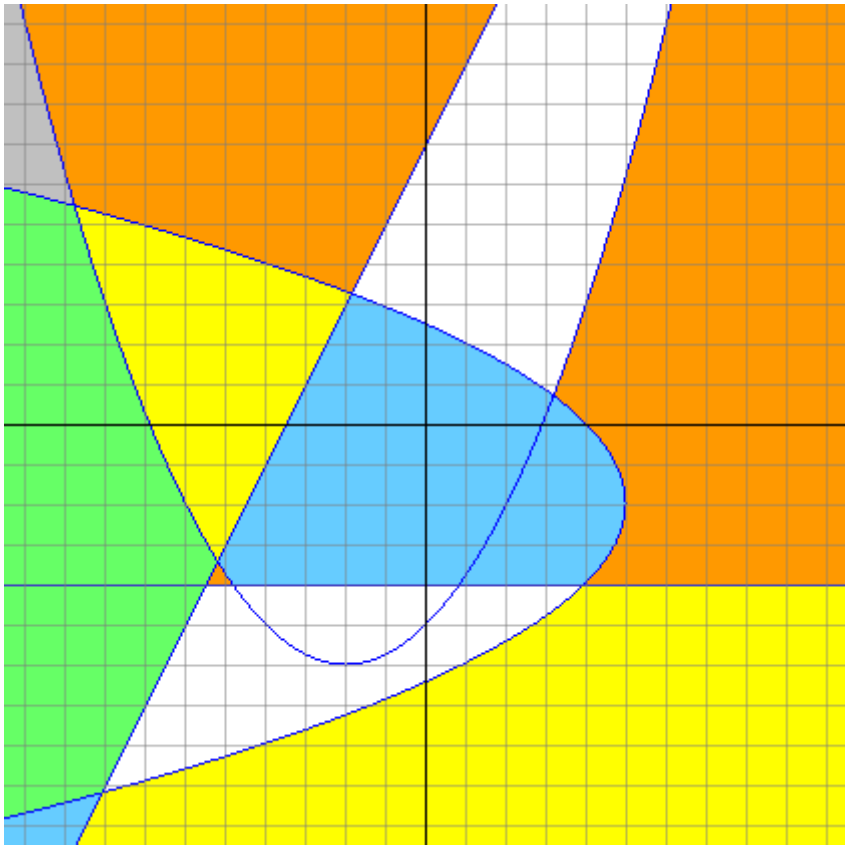
95.



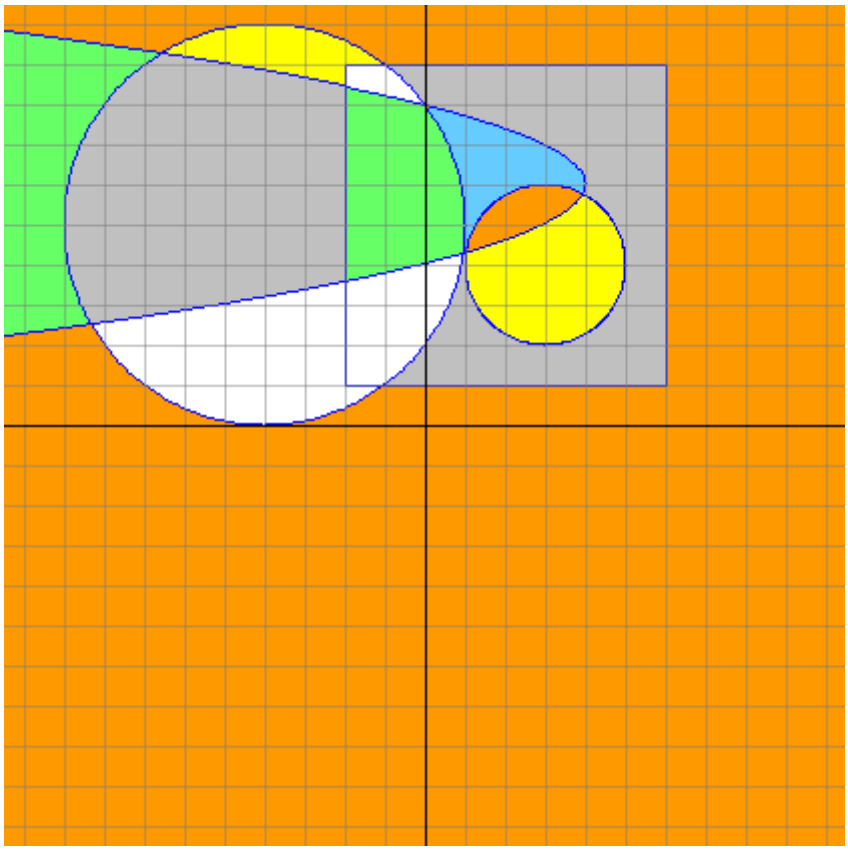
96.



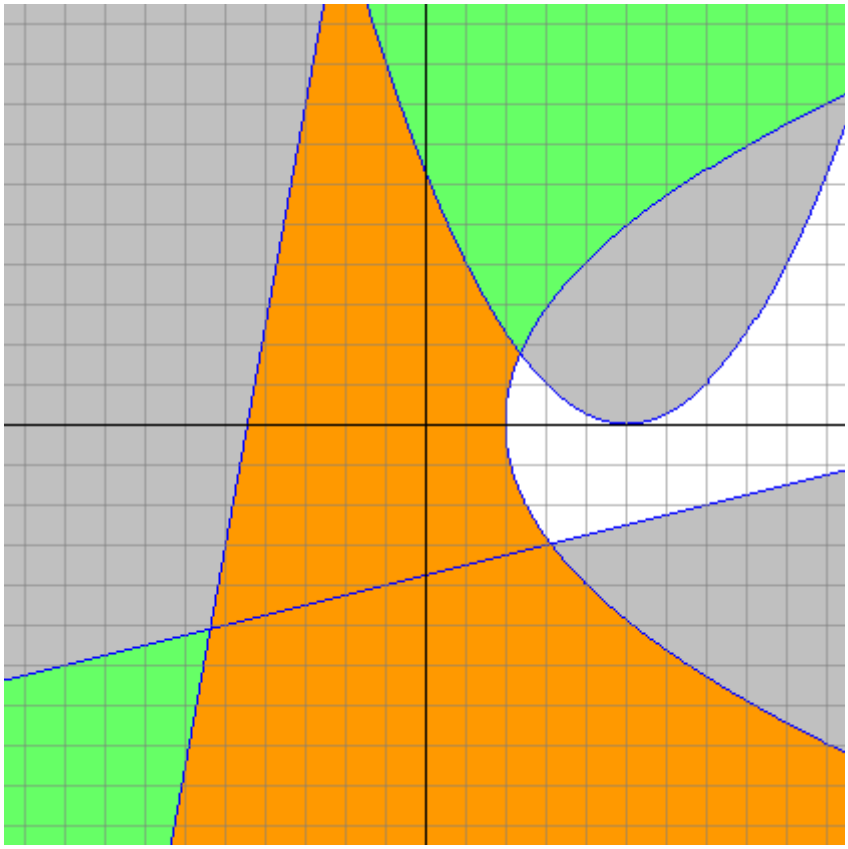
97.



98.



99.



100.