

Министерство образования и науки Российской Федерации

Московский физико-технический институт
(государственный университет)

AeroPython

**Решение задач теоретической
гидромеханики с использованием языка
программирования Python**

Содержание

Ликбез по Python	5
Библиотеки	5
Переменные	6
Отступы в Python	7
Срезы массивов	8
Присвоение и копирование массивов	8
Дополнительные материалы	9
1 Источник и сток	11
1.1 Введение	11
1.2 Математическая постановка	11
1.3 Приступим!	12
1.4 Источник	14
1.5 Сток	16
1.6 Система источник-сток	16
2 Источник и сток в безграничном потоке	18
2.1 Начнем, пожалуй	18
2.2 Источник в равномерном потоке	18
2.3 Система источник-сток в равномерном потоке	23
2.3.1 Уравнение Бернулли и коэффициент давления	24
2.4 Заключительные замечания	25
3 Диполь	26
3.1 Диполь в равномерном потоке	29
3.1.1 Уравнение Бернулли и коэффициент давления	31
Распределение источников по профилю	34
Постановка задачи	34
Вопросы	34
4 Вихрь	35
4.1 Что такое вихрь?	35
4.2 Рассчитаем вихрь	36
4.3 Вихрь и сток	38
4.4 Что такое «безвихревой» вихрь?	40
4.5 Далее...	41
5 Бесконечная цепочка вихрей	42
5.1 Изолированный вихрь (предыдущее занятие)	42
5.2 Суперпозиция нескольких вихрей	42
5.3 Бесконечная цепочка вихрей	44
6 Подъёмная сила цилиндра	47
6.1 Приступим к расчётам!	47
6.2 Коэффициент давления	52
6.3 Подъёмная сила и сопротивление	53
6.4 Эффект Магнуса	54
Преобразование Жуковского	56
Введение	56
Комплексные числа в Python	56
Упражнения	57
Фигуры, созданные при помощи формулы Жуковского	57
Расчётная сетка в плоскости z в полярной системе координат	57
Упражнения	58
Обтекание симметричного профиля Жуковского под нулевым углом атаки	58
Функция и линии тока	58

Векторы скорости и коэффициент давления	59
Упражнения	60
Обтекание симметричного профиля Жуковского под ненулевым углом атаки без циркуляции	61
Упражнения	62
Обтекание симметричного профиля Жуковского под ненулевым углом атаки при наличии циркуляции	63
Упражнения	64
7 Метод зеркального отражения	66
7.1 Источник вблизи плоской поверхности	66
7.2 Вихрь вблизи плоской поверхности	69
7.3 Движение вихревой пары у земли	71
7.4 Диполь в потоке, параллельном плоскости	72
8 Слой источников	76
8.1 Конечное число источников, расположенных на линии	76
8.2 Бесконечная линия источников	79
8.3 Дополнительные материалы	83
9 Обтекание цилиндра, составленного из панелей	84
9.1 Задание геометрии	84
9.2 Разбиение на панели	85
9.3 Граничное условие для потока на поверхности тела	87
9.4 Решение системы линейных уравнений	89
9.5 Коэффициент давления на поверхности	89
10 Панельный метод источников	93
10.1 Разбиение на панели	94
10.2 Параметры набегающего потока	98
10.3 Граничное условие для потока на поверхности тела	99
10.4 Построение системы линейных уравнений	100
10.5 Коэффициент давления на поверхности	101
10.5.1 Теоретическое решение	102
10.5.2 И нарисуем результат!	103
10.6 Линии тока	104
10.7 Заключительные замечания	106
10.8 Литература	107
11 Панельный метод вихрей-источников	108
11.1 Панельный метод для несущего тела	108
11.2 Условие Кутты-Жуковского	108
11.2.1 Разбиение на панели	109
11.2.2 Параметры набегающего потока	112
11.2.3 Граничное условие для потока на поверхности тела	113
11.2.4 Выполнение условия Кутты-Жуковского	114
11.3 Построение системы линейных уравнений	114
11.4 Коэффициент давления на поверхности	118
11.4.1 Проверка точности	120
11.5 Коэффициент подъёмной силы	121
11.5.1 Контрольное задание	121
Упражнение: Вывод панельного метода вихрей-источников	122
Вопрос 1	122
Вопрос 2	123
Вопрос 3	123
Вопрос 4	123
Вопрос 5	124
Вопрос 6	125
Вопрос 7	125

Вопрос 8	125
Двумерный многозвенный профиль 126	
Часть 1: Тест Вильямса	126
Подсказки	127
Вопрос 1	127
Вопрос 2	127
Вопрос 3	128
Вопрос 4	128
Вопрос 5	128
Часть 2: Тест Вензингера	128
Вопрос 1	129
Вопрос 2	129
Вопрос 3	129
Ссылки	129

Материал распространяется по лицензии Creative Commons Attribution license, CC-BY 4.0; code under MIT license.

Оригинальный курс ©2014 Lorena A. Barba. Thanks: Gilbert Forsyth and Olivier Mesnard, and NSF for support via CAREER award #1149784.

Перевод ©2017 И.А. Курсаков

Автор русской версии выражает благодарность А.А. Савельеву за помощь, оказанную при её подготовке

Ликбез по Python

Перед вами короткое и весьма поверхностное введение в Python. Изложенной в нем информации должно хватить для работы с набором блокнотов, составляющих курс *AeroPython*. (Этот документ основан на вводной лекции курса *CFD Python* профессора Лорены А. Барбы (Lorena A. Barba)

Может быть, Python уже установлен на вашем компьютере, вероятность этого особенно высока, если вы используете в качестве операционной системы OSX или какой-нибудь вариант Linux. В любом случае, лучше скачать и установить свободный дистрибутив *Anaconda Scientific Python*. Это сильно упростит знакомство с языком.

Вероятно, вы захотите завести собственную локальную копию этого блокнота, или даже всего набора *AeroPython*. В таком случае советую знакомиться с материалом последовательно, шаг за шагом, экспериментируя с кодом непосредственно в блокноте, или набирая его в отдельном интерактивном окне Python.

Если вы хотите работать на своем компьютере, используя установленную Anacond'у, вам нужно открыть окно терминала в директории, содержащей файлы с расширением .ipynb. Для запуска сервера блокнотов наберите в терминале

```
jupyter notebook
```

Откроется окно браузера со вкладкой, отображающей список блокнотов в рабочей директории. Выбирайте нужный и начинайте работу!

Библиотеки

Python это свободный язык программирования высокого уровня. Экосистема *Python* включает в себя множество пакетов и библиотек, позволяющих, например, выполнять операции с массивами, рисовать графики и многое другое. Библиотеки можно импортировать в вашу программу для расширения возможностей языка.

Давайте начнем с импорта нескольких полезных библиотек. Первая — **NumPy** содержит функции для работы с массивами, похожа на MATLAB. Мы будем очень часто её использовать. Вторая — **Matplotlib**, библиотека для создания двумерной графики, мы будем пользоваться ею для отрисовки результатов.

С этих строчек будет начинаться большинство ваших программ:

In [4]: # <-- комментарии в Python обозначаются знаком решётки

```
import numpy          # импорт библиотеки для работы с массивами
from matplotlib import pyplot    # импорт библиотеки для отрисовки графики
```

Мы импортировали библиотеку `numpy` и модуль `pyplot` из большой библиотеки `matplotlib`.

Чтобы воспользоваться какой-нибудь функцией из этих библиотек, мы должны указать интерпретатору, где именно её искать. Для этого перед именем функции нужно написать имя библиотеки. Имена разделяются точкой.

Например, если нужно использовать функцию `linspace()` для создания массива чисел, равномерно распределённых на каком-либо интервале, это можно сделать при помощи вызова:

In [5]: myarray = numpy.linspace(0, 5, 10)
print(myarray)

```
[ 0.          0.55555556  1.11111111  1.66666667  2.22222222  2.77777778
 3.33333333  3.88888889  4.44444444  5.          ]
```

Если же не добавить к `linspace()` префикс `numpy`, то интерпретатор выдаст ошибку, так как неизвестно, где искать необходимую функцию:

In [6]: myarray = linspace(0, 5, 10)

NameError

Traceback (most recent call last)

```
<ipython-input-6-ed3ba806937a> in <module>()
----> 1 myarray = linspace(0, 5, 10)
```

```
NameError: name 'linspace' is not defined
```

Нужно сказать, что функция `linspace()` очень полезная. Попробуйте изменить входные параметры.

Способы импорта Вам часто будут встречаться фрагменты кода, в которых используются строки такого вида:

```
import numpy as np
import matplotlib.pyplot as plt
```

Для чего нужна конструкция `import-as`? Таким способом можно создать ссылку на импортируемые библиотеки или модули. Это довольно стандартная практика, однако в данном курсе мы будем использовать импорт в *явном виде*. Такой способ делает код более читаемым.

Советы профессионалов Возможно, вам попадутся случаи, когда импортируется все содержимое библиотеки без создания ссылок (например, `from numpy import *`). Такой способ экономит усилия, связанные с набором текста программы, но несколько неряшлив. Лучше заводить хорошие привычки с самого начала!

Для знакомства с доступными функциями, обратитесь к документации [NumPy Reference](#). Тем, кто хорошо знаком с Matlab, может оказаться полезной вики-страница [NumPy for Matlab Users](#)

Переменные

В Python не требуется явно указывать тип декларируемой переменной, в отличие от C, например. Нужно просто присвоить какое-нибудь значение переменной, и интерпретатор все поймет сам:

```
In [7]: a = 5      # a - целое число 5
                  b = 'five' # b - строка 'five'
                  c = 5.0    # c - число с плавающей точкой 5.0
```

Можно узнать, какой тип был приписан переменной:

```
In [8]: type(a)
Out[8]: int
In [9]: type(b)
Out[9]: str
In [10]: type(c)
Out[10]: float
```

В третьей версии языка (Python 3) результатом операции деления целых чисел, чисел с плавающей точкой или их комбинации будет число с плавающей точкой машинной точности. Например, два выражения

```
In [11]: 14/a
Out[11]: 2.8
In [12]: 14/c
Out[12]: 2.8
```

дадут один и тот же результат.

Отступы в Python

Для группировки выражений в Python используются отступы и пробелы. Например, если нам потребуется написать короткий цикл на C, получится что-то такое:

```
for (i = 0, i < 5, i++){
    printf("Hi! \n");
}
```

В Python не используются фигурные скобки, вместо них — отступы. Та же самая программа на Python будет выглядеть так:

```
In [13]: for i in range(5):
          print("Hi \n")
```

Hi

Hi

Hi

Hi

Hi

Заметили функцию `range()`? Это встроенная функция, которая возвращает список, содержащий арифметическую прогрессию.

Если циклы вложенные, то тело внутреннего цикла нужно выделить дополнительным отступом:

```
In [14]: for i in range(3):
          for j in range(3):
              print(i, j)

              print("This statement is within the i-loop, but not the j-loop")

0 0
0 1
0 2
This statement is within the i-loop, but not the j-loop
1 0
1 1
1 2
This statement is within the i-loop, but not the j-loop
2 0
2 1
2 2
This statement is within the i-loop, but not the j-loop
```

Срезы массивов

В NumPy можно оперировать с отдельными частями массивов, примерно теми же способами, что и в Matlab. Возьмём, к примеру, массив значений от 1 до 5:

```
In [15]: myvals = numpy.array([1, 2, 3, 4, 5])
myvals
```

```
Out[15]: array([1, 2, 3, 4, 5])
```

В Python **индексация начинается с 0**, как и в C, и у этого правила есть свои [рациональные обоснования](#). Вооружившись этим знанием, посмотрим на значения первого и последнего элемента созданного массива:

```
In [16]: myvals[0], myvals[4]
```

```
Out[16]: (1, 5)
```

В массиве `myvals` содержится 5 элементов, но если мы попробуем обратиться к элементу с индексом 5 `myvals[5]`, Python расстроится и выдаст ошибку, так как в этом случае фактически вызывается несуществующий 6-й элемент массива.

```
In [17]: myvals[5]
```

```
-----
IndexError                                     Traceback (most recent call last)

<ipython-input-17-6cc4d3ae83cd> in <module>()
----> 1 myvals[5]

IndexError: index 5 is out of bounds for axis 0 with size 5
```

Иногда может потребоваться не весь массив, а его часть — *срез* (по-английски *slice*). Например, первые три элемента,

```
In [18]: myvals[0:3]
```

```
Out[18]: array([1, 2, 3])
```

Обратите внимание, левый конец среза включается, а правый — нет. Так, в предыдущем примере срез состоит из значений `myvals[0]`, `myvals[1]` и `myvals[2]`, но без `myvals[3]`. Это соглашение оказывается удобным, например, потому что позволяет посчитать число элементов в срезе — нужно из правого конца вычесть левый (в нашем случае $3 - 0 = 3$)

Присвоение и копирование массивов

С одной из странностей/особенностей Python, которая часто становится источником ошибок и путаницы, сталкиваешься при присваивании и сравнении массивов. Вот пример. Заведем одномерный массив и назовем его `a`:

```
In [19]: a = numpy.linspace(1,5,5)
```

```
In [20]: a
```

```
Out[20]: array([ 1.,  2.,  3.,  4.,  5.])
```

Супер! У нас есть массив `a` со значениями от 1 до 5. Теперь я хочу сделать копию этого массива и назвать её `b`. Для этого я пишу такой текст:

```
In [21]: b = a
```

```
In [22]: b
```

```
Out[22]: array([ 1.,  2.,  3.,  4.,  5.])
```

Великолепно! В новом массиве `b`, так же как и в `a` теперь содержатся элементы от 1 до 5. Теперь, когда у нас есть бэкап, можно менять значения элементов `a`, не опасаясь потери данных.

```
In [23]: a[2] = 17
In [24]: a
Out[24]: array([ 1.,  2., 17.,  4.,  5.])
```

Вот, третий элемент `a` теперь равен 17. Давайте проверим, что содержится в `b`. Что мы ожидаем там увидеть?

```
In [25]: b
Out[25]: array([ 1.,  2., 17.,  4.,  5.])
```

Неожиданный результат! При выполнении выражения `a = b` не происходит копирования значений массива `a` в новый массив `b`. Вместо этого Python создает ещё одну ссылку, привязанную к тому же массиву, что и `a`. Поэтому изменение значений в `a` влечет за собой точно такие же перемены в массиве `b` (фактически, это *присваивание по ссылке*). Если же нужно создать настоящую копию массива, то нужно указать это явным образом.

```
In [26]: c = a.copy()
```

Теперь можно снова поменять что-то в `a` и посмотреть, приведет ли это к изменениям в `c`.

```
In [27]: a[2] = 3
```

```
In [28]: a
```

```
Out[28]: array([ 1.,  2.,  3.,  4.,  5.])
```

```
In [29]: c
```

```
Out[29]: array([ 1.,  2., 17.,  4.,  5.])
```

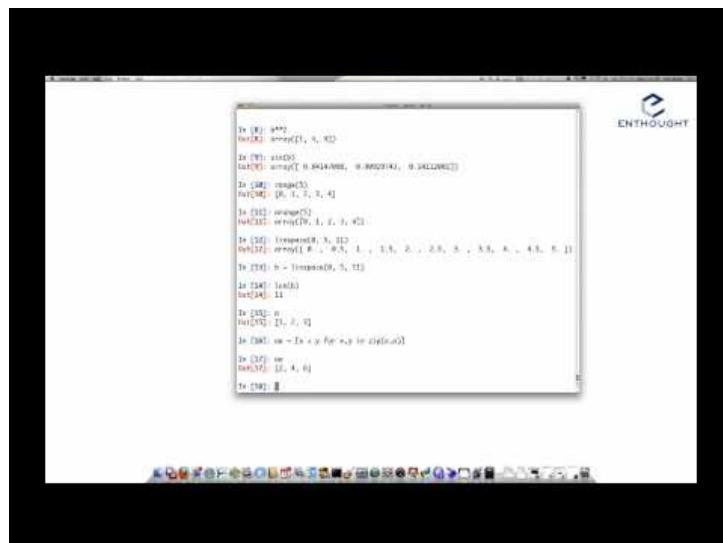
Отлично, всё сработало так, как мы хотели. Если разница между выражениями `a = b` и `a = b.copy()` не ясна, лучше перечитайте вышеизложенное ещё раз. Это сэкономит вам кучу нервов впоследствии.

Дополнительные материалы

Существует огромное число онлайн ресурсов, с информацией об использовании NumPy и других библиотек. Но чтобы продемонстрировать мультимедийные возможности блокнотов Jupyter, вот вам ссылка на короткое YouTube видео на тему использования массивов NumPy.

```
In [30]: from IPython.display import YouTubeVideo
YouTubeVideo('vWkb7VahaXQ')
```

```
Out[30]:
```



1 Источник и сток

Это первый блокнот из серии *AeroPython*, посвященной изучению классической аэродинамики при помощи Python. Если вы не уверены в своих знаниях основ научных библиотек Python, начните с [Ликбеза по Python](#), изложенного во вводном занятии данной серии.

За ходом изложения материала можно следить, просматривая этот блокнот онлайн, при этом набирая исходный код примеров и упражнений в собственном блокноте Jupyter или в отдельном файле в вашей любимой IDE. В любом случае, необходимо, чтобы приведённые здесь примеры были повторены вами самостоятельно.

1.1 Введение

Классическая аэродинамика основывается на *теории потенциального течения*. Математический аппарат, разработанный в рамках этой теории, был движущей силой аэродинамики в эпоху её становления в 20 веке.

В её основе лежат некоторые предположения:

- поток стационарный;
- скорости течения много меньше скорости звука (несжимаемая жидкость);
- в жидкости отсутствует трение (идеальная жидкость);
- завихренность также отсутствует (жидкие частицы не вращаются).

Может показаться, что упрощений многовато, но, оказывается, значительная часть течений может быть рассмотрена в таких предположениях. Роль вязкости велика лишь в очень тонкой области пограничного слоя, кроме того существуют поправки к потенциальной теории, позволяющие учесть это влияние. Течение во многих практических случаях можно считать безвихревым за исключением изолированных точек, линий или поверхностей. И, наконец, большинство интересующих нас случаев относятся к дозвуковым течениям.

У теории потенциального течения есть одно невероятно удобное математическое свойство: *линейность*. Это означает, что применим принцип суперпозиции, и новые решения можно получать как сумму уже известных.

В этом блокноте вы познакомитесь с двумя элементарными потенциальными течениями: **источником** и **стоком**. А затем мы соединим их, и получим еще одно решение.

1.2 Математическая постановка

Давайте запишем пару формул. Из курса гидродинамики вам должно быть известно определение *циркуляции*:

$$\Gamma = \oint \mathbf{v} \cdot d\vec{l}$$

По определению, циркуляция это криволинейный интеграл второго рода скорости по замкнутому контуру. Если напрячься, можно вспомнить также теорему Стокса. Она гласит, что криволинейный интеграл по замкнутому контуру равен *потоку ротора* скорости через поверхность, ограниченную этим контуром. Ротор скорости, в свою очередь, это завихренность, $\omega = \nabla \times \mathbf{v}$:

$$\oint \mathbf{v} \cdot d\vec{l} = \int \int_s \omega \cdot \vec{n} \, ds$$

Если завихренность равна нулю (безвихревое течение), то циркуляция по любому замкнутому контуру равна нулю. Это означает, что криволинейный интеграл по любому контуру, соединяющему точки A и B, равен по абсолютному значению и противоположен по знаку интегралу по контуру, соединяющему точки B и A. Раскрыв скалярное произведение вектора скорости $\mathbf{v} = (u, v, w)$, запишем

$$\int_A^B \mathbf{v} \cdot d\vec{l} = \int_A^B u \, dx + v \, dy + w \, dz$$

В безвихревом течении значение этого интеграла не зависит от выбора контура между A и B. А это, как нам должно быть известно из курса математического анализа, означает, что $u \, dx + v \, dy + w \, dz$ является [полным дифференциалом](#) потенциала ϕ , где

$$u = \frac{\partial \phi}{\partial x}, \quad v = \frac{\partial \phi}{\partial y}, \quad w = \frac{\partial \phi}{\partial z}$$

Или короче: $\mathbf{v} = \nabla \phi$. Подставив сюда уравнение неразрывности для несжимаемой жидкости $\nabla \cdot \mathbf{v} = 0$, мы получим удивительное в своей простоте уравнение потенциального течения:

$$\nabla^2 \phi = 0$$

Уравнение Лапласа! Таким образом любое решение уравнения Лапласа соответствует потенциальному течению.

1.3 Приступим!

Мы хотим получить два решения уравнения Лапласа — потенциалы **источника** и **стока**, чтобы затем визуализировать поля этих потенциальных течений и насладиться результатом.

Начнем с импорта нескольких полезных питоновских библиотек:

- NumPy — научная библиотека для работы с многомерными массивами и матрицами;
- Matplotlib — библиотека для создания графики, при помощи которой мы будем визуализировать наши результаты;
- Модуль `math` делает доступными математические функции из стандарта С.

Если вы не помните зачем и как импортировать библиотеки, вернитесь к *вводному занятию*, Ликбез по Python:

```
In [1]: import math
import numpy
from matplotlib import pyplot
```

Наша цель — нарисовать линии тока для источника и стока. Для этого сначала зададим множество точек, в которых будут рассчитаны компоненты скорости.

Определим декартову сетку, состоящую из равномерно распределённых по расчетной области точек. В качестве расчётной области выберем прямоугольник со сторонами 4 по оси x и 2 по оси y : $x, y \in [-2, 2], [-1, 1]$.

Число точек обозначим переменной `N`, а границы расчетной области — переменными `x_start`, `x_end`, `y_start` и `y_end`.

Для создания одномерных массивов, содержащих равномерно распределённые значения по координатам x и y воспользуемся функцией `linspace()`. В последней строке кода, приведенного в следующем блоке, вызывается функция `meshgrid()`, создающая двумерные массивы с координатами точек, в которых мы будем вычислять скорости. Убедитесь, что вы понимаете, что делает эта функция и какой результат она возвращает. Мы будем часто её использовать.

```
In [2]: N = 50                                     # число точек в каждом направлении
x_start, x_end = -2.0, 2.0                         # границы по оси x
y_start, y_end = -1.0, 1.0                         # границы по оси y
# создаем одномерный массив с координатами x
x = numpy.linspace(x_start, x_end, N)
# создаем одномерный массив с координатами y
y = numpy.linspace(y_start, y_end, N)
print('x = ', x)
print('y = ', y)

X, Y = numpy.meshgrid(x, y)                         # создаем сетку

x = [-2.          -1.91836735 -1.83673469 -1.75510204 -1.67346939 -1.59183673
-1.51020408 -1.42857143 -1.34693878 -1.26530612 -1.18367347 -1.10204082
-1.02040816 -0.93877551 -0.85714286 -0.7755102  -0.69387755 -0.6122449
-0.53061224 -0.44897959 -0.36734694 -0.28571429 -0.20408163 -0.12244898
-0.04081633  0.04081633  0.12244898  0.20408163  0.28571429  0.36734694
 0.44897959  0.53061224  0.6122449   0.69387755  0.7755102   0.85714286]
```

```

0.93877551 1.02040816 1.10204082 1.18367347 1.26530612 1.34693878
1.42857143 1.51020408 1.59183673 1.67346939 1.75510204 1.83673469
1.91836735 2.          ]
y = [-1.          -0.95918367 -0.91836735 -0.87755102 -0.83673469 -0.79591837
-0.75510204 -0.71428571 -0.67346939 -0.63265306 -0.59183673 -0.55102041
-0.51020408 -0.46938776 -0.42857143 -0.3877551  -0.34693878 -0.30612245
-0.26530612 -0.2244898  -0.18367347 -0.14285714 -0.10204082 -0.06122449
-0.02040816  0.02040816  0.06122449  0.10204082  0.14285714  0.18367347
0.2244898  0.26530612  0.30612245  0.34693878  0.3877551  0.42857143
0.46938776 0.51020408  0.55102041  0.59183673  0.63265306  0.67346939
0.71428571 0.75510204  0.79591837  0.83673469  0.87755102  0.91836735
0.95918367 1.          ]

```

Созданную расчётую сетку можно визуализировать при помощи модуля `pyplot` из библиотеки `matplotlib`, используя функцию `scatter()`.

Стоит отметить, что мы довольно детально описываем, как именно должна выглядеть желаемое изображение: мы хотим видеть рисунок определенного размера, с подписями к осям, при этом диапазон на осях тоже задан. Даже цвет маркеров определен. Изучив все тонкости документации Matplotlib, вы научитесь создавать радующие глаз иллюстрации высокого качества.

Для того, чтобы картинки отображались в блокноте, вначале нужно использовать `магическую` команду `%matplotlib inline`. Тогда результаты работы графического модуля будут вставлены в страницу в виде статических изображений в формате PNG:

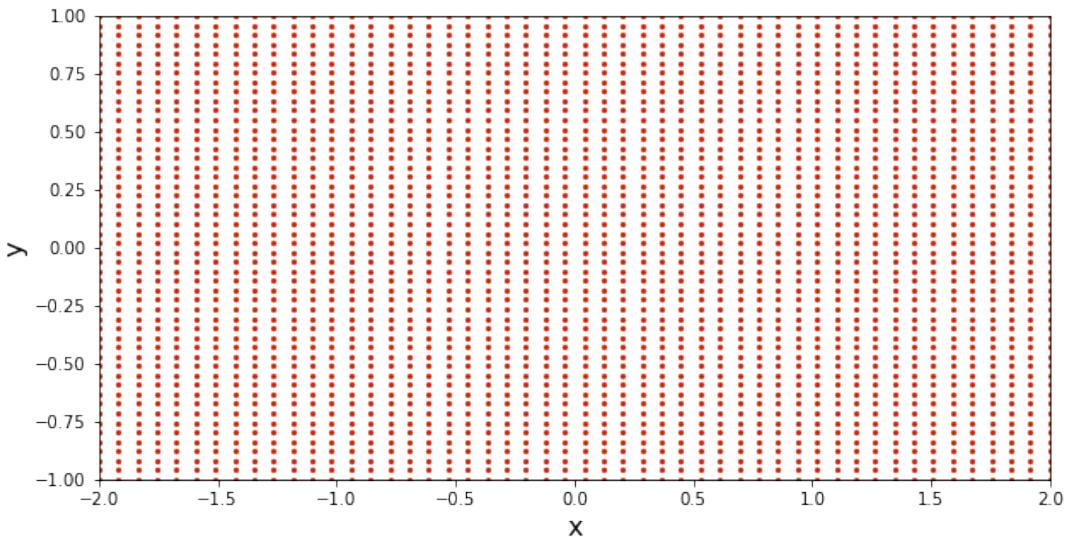
In [3]: `%matplotlib inline`

```

# рисуем сетку из узлов
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.scatter(X, Y, s=10, color='#CD2305', marker='o', linewidth=0)

```

Out[3]: <matplotlib.collections.PathCollection at 0x1e485a1def0>



В каждой из этих выстроенных в ряды точек мы теперь вычислим скорость, соответствующую течению с источником, а затем нарисуем линии тока.

1.4 Источник

Выше уже упоминалось об одном приятном свойстве потенциальных течений: поскольку они описываются линейным уравнением, решения можно получать при помощи суперпозиции. Поэтому очень полезно иметь набор элементарных решений, который можно использовать как кубики. Источники и стоки как раз являются такими элементарными решениями.

Источник это точка, из которой истекает однородный поток жидкости. Из этой точки испускаются линии тока в виде прямых. Рассмотрим двумерную постановку. Задача осесимметричная, поэтому воспользуемся полярными координатами (r, θ) . Угол θ равен $\arctan(y/x)$.

Запишем условие отсутствия завихренности $\omega = 0$ в полярных координатах:

$$\frac{\partial(ru_\theta)}{\partial r} - \frac{\partial u_r}{\partial \theta} = 0$$

Поскольку азимутальная компонента u_θ равна нулю, то $\frac{\partial u_r}{\partial \theta} = 0$. То есть радиальная скорость зависит только от радиуса, $u_r = f(r)$.

Теперь запишем уравнение неразрывности:

$$\frac{\partial(ru_r)}{\partial r} + \frac{\partial u_\theta}{\partial \theta} = 0$$

С учетом равенства нулю азимутальной компоненты и зная характер зависимости радиальной, это уравнение можно преобразовать в обыкновенное дифференциальное уравнение $\frac{d(ru_r)}{dr} = 0$. Раскрыв скобки, получим:

$$u_r + r \frac{du_r}{dr} = 0$$

Решение этого уравнения $u_r = \frac{\text{const}}{r}$. Окончательно радиальная и азимутальная компоненты скорости записываются как

$$u_r(r, \theta) = \frac{\sigma}{2\pi r}, \quad u_\theta(r, \theta) = 0,$$

где σ обозначает *интенсивность* источника. Такой выбор константы станет понятен позже.

Надеюсь, вы помните, что кроме потенциала, существует ещё *функция тока* ψ . В полярных координатах соотношения для компонент скорости и ψ выглядят следующим образом:

$$\frac{1}{r} \frac{\partial \psi}{\partial \theta} = u_r, \quad -\frac{\partial \psi}{\partial r} = u_\theta$$

Интегрирование этих соотношений дает

$$\psi = \frac{\sigma}{2\pi} \theta + \text{constant}$$

На практике интерес представляют как раз компоненты скорости, являющиеся производными функции тока, поэтому константу интегрирования можно смело опустить.

В декартовых координатах поле скорости (u, v) в точке (x, y) , соответствующее источнику с интенсивностью σ , записывается в виде

$$u = \frac{\partial \psi}{\partial y} = \frac{\sigma}{2\pi} \frac{x - x_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

и

$$v = -\frac{\partial \psi}{\partial x} = \frac{\sigma}{2\pi} \frac{y - y_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

Рассчитаем скорости в узлах нашей расчётной сетки. Поместим источник в точку $(-1, 0)$ и зададим интенсивность $\sigma = 5$.

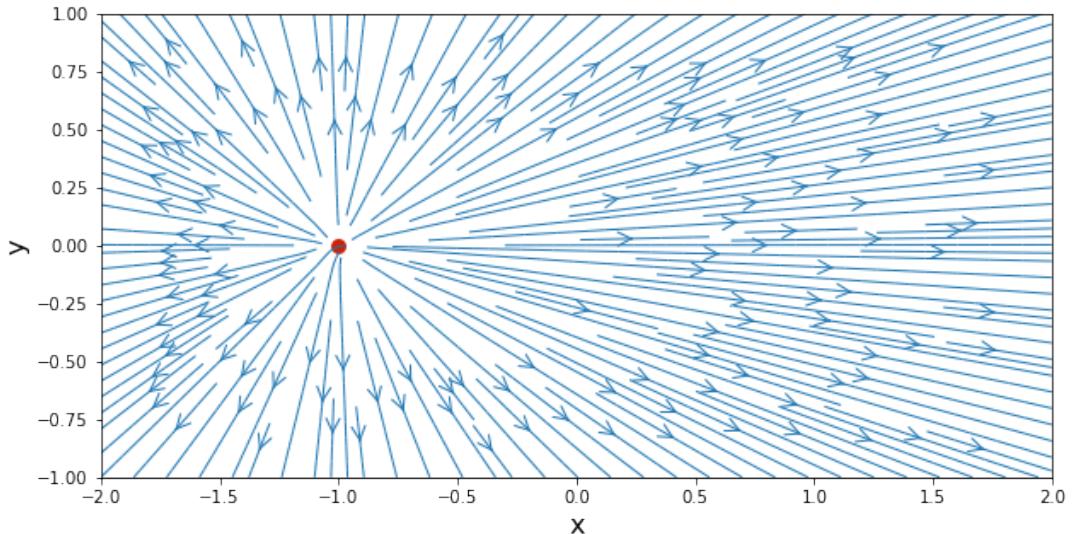
Вместо того чтобы перебирать точки одну за одной в цикле по `[i, j]`, мы сразу вычислим значения массивов (`u_source`, `v_source`), применив к ним арифметические операции. Да, в NumPy арифметические операторы действуют на массив поэлементно, на выходе создаётся новый массив с результатом.

```
In [4]: strength_source = 5.0 # интенсивность источника
x_source, y_source = -1.0, 0.0 # положение источника

# вычисление скорости в узлах расчётной сетки
u_source = strength_source/(2*math.pi) * (X-x_source)/((X-x_source)**2 + \
(Y-y_source)**2)
v_source = strength_source/(2*math.pi) * (Y-y_source)/((X-x_source)**2 + \
(Y-y_source)**2)
```

Пора уже рисовать линии тока! На наше счастье в библиотеке имеется нужная функция `streamplot()`. Ещё воспользуемся знакомой уже функцией `scatter()`, чтобы обозначить красной точкой положение источника.

```
In [5]: # рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u_source, v_source, density=2, \
                   linewidth=1, arrowsize=2, arrowstyle='->')
pyplot.scatter(x_source, y_source, color='#CD2305', s=80, marker='o', \
               linewidth=0);
```



Здорово. Именно так, как мы и представляли себе источник, только чуточку краше. Обратите внимание, что в конце последней строки мы добавили точку с запятой. Это сделано, чтобы убрать вывод стоки типа `<matplotlib ...>`, в которой указан внутренний идентификатор объекта с рисунком. К примеру, в выводе `Out[3]` такая строка присутствовала.

Предполагается, что просмотрев блокнот, вы подготовите собственную версию. При этом *набирайте* код, а не копируя его. Поменяйте параметры, почитайте документацию при необходимости. Дальше мы доберёмся до более сложных вопросов, поэтому хотелось бы, чтобы основы были усвоены довольно крепко.

Вопрос на понимание происходящего Чему равен расход через поверхность (контур), опоясывающую источник?

1.5 Сток

В случае с источником интенсивность была положительной величиной. Если же интенсивность *отрицательная*, получается не источник, а *сток*. Теперь линии тока не испускаются из одной точки, а, наоборот, приходят в неё.

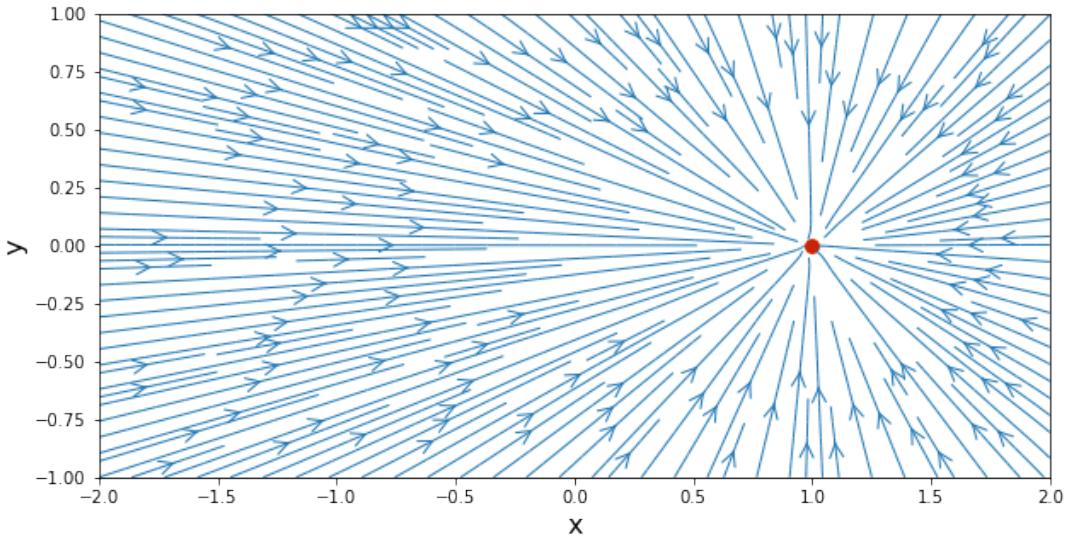
Поле скорости, соответствующее течению со стоком, выглядит аналогично полю с источником, но с противоположным направлением потока. Поэтому код для визуализации картины течения поменяется не сильно.

Поместим сток в точку с координатами $(1, 0)$, интенсивность оставим такой же, как у источника, поменяем только знак.

```
In [6]: strength_sink = -5.0 # интенсивность стока
x_sink, y_sink = 1.0, 0.0 # положение стока

# вычисление скорости в узлах расчётной сетки
u_sink = strength_sink/(2*math.pi) * (X-x_sink)/((X-x_sink)**2 + \
(Y-y_sink)**2)
v_sink = strength_sink/(2*math.pi) * (Y-y_sink)/((X-x_sink)**2 + \
(Y-y_sink)**2)
```

```
In [7]: # рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u_sink, v_sink, density=2, linewidth=1, \
arrowsize=2, arrowstyle='->')
pyplot.scatter(x_sink, y_sink, color='#CD2305', s=80, marker='o', \
linewidth=0);
```

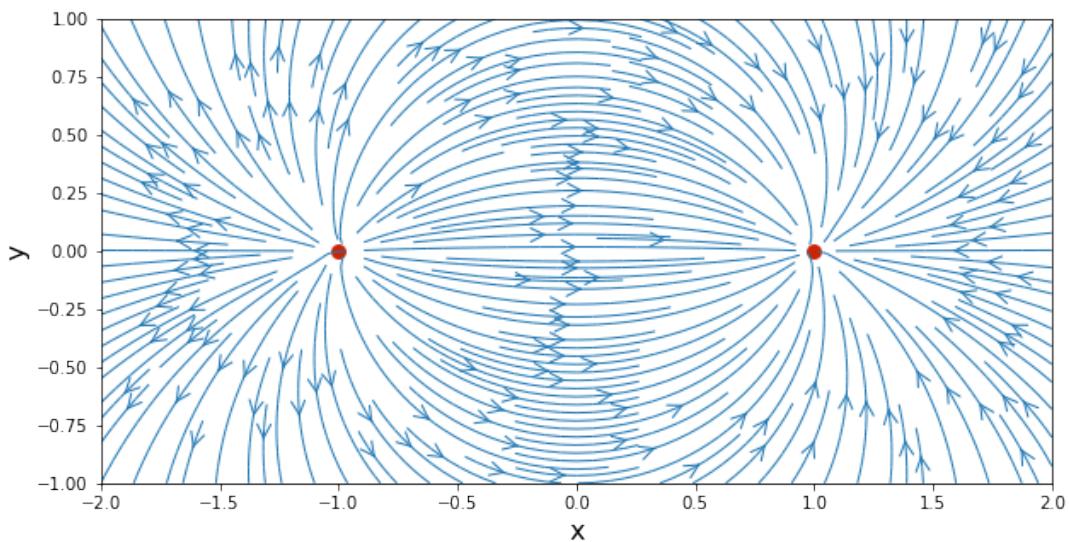


1.6 Система источник-сток

А теперь пришло время, чтобы воспользоваться суперпозицией. У нас есть поля скоростей для источника и стока. Сложив эти поля поточечно, получим новое решение — **систему источник-сток**. Внимательно прочтите приведенный код и убедитесь, что поняли, что там происходит.

```
In [8]: # вычисление скорости для системы источник-сток
# путем сложения элементарных решений
u_pair = u_source + u_sink
v_pair = v_source + v_sink

# рисуем линии тока системы источник-сток
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u_pair, v_pair, density=2.0, linewidth=1, \
                   arrowsize=2, arrowstyle='->')
pyplot.scatter([x_source, x_sink], [y_source, y_sink], \
               color='#CD2305', s=80, marker='o', linewidth=0);
```



Поэкспериментируйте! Попробуйте поменять положение источника и стока, их интенсивности. Как меняется картина течения?

Задание Напишите код для визуализации изолиний потенциала, вместо линий тока. Вероятно, для этого вам потребуется функция `contour()`.

2 Источник и сток в безграничном потоке

Вы добрались до второго блокнота курса *AeroPython*. Готовы? Вы старательно изучили первый блокнот и написали собственную версию питоновского кода для визуализации [Источник и сток](#)?

Блокноты составлены таким образом, чтобы продвигаться шаг за шагом, так что не пропускайте ни одного! Изучив каждый, вы углубите свои знания аэродинамики и Python. В этом блокноте мы получим потенциальное течение, сложив источник, сток и равномерный бесконечный поток. Почему? Потому что можем! Помните *суперпозицию*? Это приятное свойство (линейного) потенциального потока, описываемого уравнением Лапласа: $\nabla^2\phi = 0$.

В этом блокноте мы также научимся придавать коду более модульную структуру путем использования функций. Так кодом будет проще управлять.

2.1 Начнем, пожалуй

Начнем, как и в прошлый раз, с импорта библиотек, которыми мы будем пользоваться в блокноте: **NumPy** для работы с массивами, **Matplotlib** для визуализации и **math** для различных математических функций.

Ещё добавим команду `%matplotlib inline` для встраивания рисунков в блокнот.

```
In [1]: import numpy
        import math
        from matplotlib import pyplot
        # отображение картинок в блокноте
%matplotlib inline
```

Для визуализации линий тока нам нужно создать расчётную сетку из узлов, в которых будет вычисляться скорость. В прошлом блокноте мы уже это делали, поэтому просто скопируем сюда нужный код. И сразу после копирования и вставки кода, вы можете подумать: *А почему бы не написать функцию, которая будет создавать сетку для любого потока? Тогда я могу повторно использовать её!* И вы будете абсолютно правы, подумав так. Но давайте сделаем это по-старому.

```
In [2]: N = 200                      # Число узлов сетки в каждом направлении
        x_start, x_end = -4.0, 4.0      # границы по x
        y_start, y_end = -2.0, 2.0      # границы по y
        x = numpy.linspace(x_start, x_end, N)    # одномерный массив x
        y = numpy.linspace(y_start, y_end, N)    # одномерный массив y
        X, Y = numpy.meshgrid(x, y)          # создаёт сетку
```

Теперь у нас есть набор точек, и два массива **X** и **Y**, содержащих координаты *x* и *y* (соответственно) каждой точки на прямоугольной сетке.

Интересно, насколько хорошо вы понимаете, что делает функция `meshgrid()`? Какой размерности массив **X** и как выглядят его элементы? Вы можете спросить Python о размере массива с помощью функции NumPy `shape()`:

```
In [3]: numpy.shape(X)
```

```
Out[3]: (200, 200)
```

Это то, что вы ожидали? Поразмыслите над этим. Убедитесь, что теперь вы понимаете, что происходит.

2.2 Источник в равномерном потоке

В предыдущем блокноте мы уже вычислили поле скоростей источника и стока. Сначала наложим поле от источника на равномерный поток, и посмотрим, что получится.

Линии тока невозмущенного однородного потока со скоростью U_∞ и углом атаки α задаются выражением:

$$\psi_{freestream}(x, y) = U_\infty (y \cos \alpha - x \sin \alpha)$$

Линии тока представляют собой параллельные прямые, проходящие под углом α к оси x . Если поток горизонтальный, то $\psi = U_\infty y$. Продифференцировав, получим $u = U_\infty$ и $v = 0$.

Напишем код, в котором в каждой точке расчётной сетки вычисляются компоненты скорости u и v и функция тока ($U_\infty, \alpha = 0$). Обратите внимание на весьма полезную функцию `ones()`, которая создаёт новый массив и заполняет его единицами, и на не менее полезную функцию `zeros()`, которая делает ... догадайтесь, что?

```
In [4]: u_inf = 1.0          # скорость на бесконечности

# вычисляем компоненты скорости равномерного потока
u_freestream = u_inf * numpy.ones((N, N), dtype=float)
v_freestream = numpy.zeros((N, N), dtype=float)

# вычисляем функцию тока
psi_freestream = u_inf * Y
```

Заметили, как мы одним махом вычислили все значения `psi_freestream`? Никаких громоздких двойных циклов — и оказывается, когда массивы большие, это ещё и намного быстрее вычисляется! Все благодаря `Numpy`!

Как известно из предыдущего блокнота, функция тока для источника, расположенного в точке $(x_{\text{source}}, y_{\text{source}})$ записывается в виде

$$\psi_{\text{source}}(x, y) = \frac{\sigma}{2\pi} \arctan \left(\frac{y - y_{\text{source}}}{x - x_{\text{source}}} \right)$$

а компоненты скорости —

$$u_{\text{source}}(x, y) = \frac{\sigma}{2\pi} \frac{x - x_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

$$v_{\text{source}}(x, y) = \frac{\sigma}{2\pi} \frac{y - y_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

Вспомним также, что функция тока и поле скорости для источника и стока идентичны, разница лишь в том, что в первом случае интенсивность положительна, а во втором — отрицательна.

Можно написать функцию двойного назначения: при положительной σ она будет давать скорости и функцию тока для источника, а при отрицательной — для стока. Ловко придумано?

Определим функцию `get_velocity()`, вычисляющую компоненты скорости (u, v) на сетке (X, Y) для особенности с заданной интенсивностью `strength`, расположенной в точке (xs, ys) . Функция определяется при помощи ключевого слова `def`, за которым следует имя функции и список параметров в круглых скобках. Страна должна оканчиваться двоеточием:

```
In [5]: def get_velocity(strength, xs, ys, X, Y):
    """
    Returns the velocity field generated by a source/sink.

    Parameters
    -----
    strength: float
        Strength of the source/sink.
    xs: float
        x-coordinate of the source (or sink).
    ys: float
        y-coordinate of the source (or sink).
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.

    Returns
    -----
    
```

```

    u: 2D Numpy array of floats
        x-component of the velocity vector field.
    v: 2D Numpy array of floats
        y-component of the velocity vector field.
    """
    u = strength/(2*numpy.pi)*(X-xs)/((X-xs)**2+(Y-ys)**2)
    v = strength/(2*numpy.pi)*(Y-ys)/((X-xs)**2+(Y-ys)**2)

    return u, v

```

Вы заметили сообщение после определения функции? Оно называется «docstring» и нужно, чтобы пользователь узнал, как работает функция. Любое сообщение в „тройных кавычках“ сразу после определения функции будет отображаться для пользователя при вызове `help()`. Например, так:

In [6]: `help(get_velocity)`

Help on function get_velocity in module __main__:

```

get_velocity(strength, xs, ys, X, Y)
    Returns the velocity field generated by a source/sink.

Parameters
-----
strength: float
    Strength of the source/sink.
xs: float
    x-coordinate of the source (or sink).
ys: float
    y-coordinate of the source (or sink).
X: 2D Numpy array of floats
    x-coordinate of the mesh points.
Y: 2D Numpy array of floats
    y-coordinate of the mesh points.

Returns
-----
u: 2D Numpy array of floats
    x-component of the velocity vector field.
v: 2D Numpy array of floats
    y-component of the velocity vector field.

```

Обратите внимание, что функция возвращает два массива: `u` и `v`. Они вычисляются внутри функции, о чем говорит наличие отступов в строках после двоеточия. Ключевое слово `return` в последней строке означает, что массивы `u` и `v` возвращаются в выражение, в котором вызывалась функция.

Аналогичным образом мы определим другую функцию для вычисления функции тока для особенности (источника или стока) в узлах сетки и назовем её `get_stream_function()`.

In [7]: `def get_stream_function(strength, xs, ys, X, Y):`
 `"""`
 `Returns the stream-function generated by a source/sink.`

 `Parameters`
 `-----`
 `strength: float`
 `Strength of the source/sink.`

```

xs: float
    x-coordinate of the source (or sink).
ys: float
    y-coordinate of the source (or sink).
X: 2D Numpy array of floats
    x-coordinate of the mesh points.
Y: 2D Numpy array of floats
    y-coordinate of the mesh points.

>Returns
-----
psi: 2D Numpy array of floats
    The stream-function.
"""
psi = strength/(2*numpy.pi)*numpy.arctan2((Y-ys), (X-xs))

return psi

```

Теперь, используя эти две функции, можно вычислить поле скорости и функцию тока для источника. Теперь добавить ещё одну особенность в поток, например сток, будет проще простого! Давайте же воспользуемся нашими новенькими функциями:

```

In [8]: strength_source = 5.0          # интенсивность источника
         x_source, y_source = -1.0, 0.0  # положение источника

         # вычисляем поле скоростей
         u_source, v_source = get_velocity(strength_source, x_source, y_source,\n
                                              X, Y)

         # вычисляем функцию тока
         psi_source = get_stream_function(strength_source, x_source, y_source,\n
                                              X, Y)

```

Снова применим принцип суперпозиции. Линии тока для комбинации свободного потока и источника:

$$\psi = \psi_{freestream} + \psi_{source} = U_\infty y + \frac{\sigma}{2\pi} \arctan \left(\frac{y - y_{source}}{x - x_{source}} \right)$$

А поскольку дифференцирование — линейная операция, новое поле скорости является просто суммой соответствующих полей свободного потока и источника:

$$\begin{aligned} u &= u_{freestream} + u_{source} \\ v &= v_{freestream} + v_{source} \end{aligned}$$

В точках торможения скорость потока равна нулю. Чтобы найти их, нужно решить уравнения:

$$u = 0, \quad v = 0$$

Их решение:

$$x_{stagnation} = x_{source} - \frac{\sigma}{2\pi U_\infty}$$

$$y_{stagnation} = y_{source}$$

Линия тока, содержащая точку торможения называется *разделительной линией*. Она отделяет набегающий поток от потока, вытекающего из источника. Обозначим разделительную линию на картине линий тока красным цветом. Это можно сделать при помощи функции [contour\(\)](#).

Обратите особое внимание, на подробные инструкции, которые мы даём нашей библиотеке для визуализации: поскольку мы хотим получить картинку, радующую глаз, то указываем цвет, толщину линий и т. д. Всю необходимую информацию можно почерпнуть из [примеров](#).

Положение точки торможения обозначим зеленой точкой, используя функцию [scatter\(\)](#).

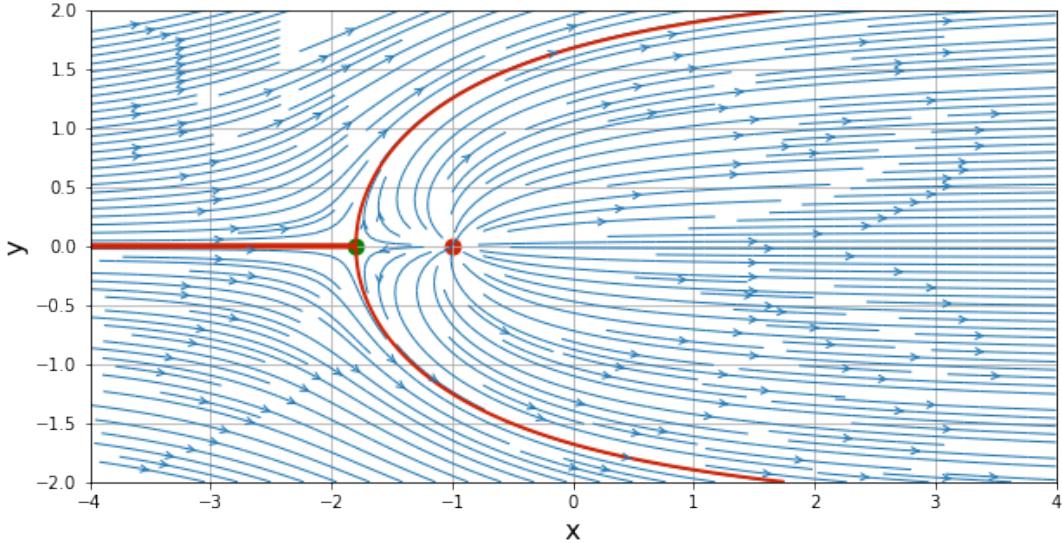
```
In [9]: # наложение источника и свободного потока
u = u_freestream + u_source
v = v_freestream + v_source
psi = psi_freestream + psi_source

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.grid(True)
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1,
                   arrowstyle='->')
pyplot.scatter(x_source, y_source, color='#CD2305', s=80, marker='o')

# определяем точку торможения
x_stagnation = x_source - strength_source/(2*numpy.pi*u_inf)
y_stagnation = y_source

# рисуем точку торможения
pyplot.scatter(x_stagnation, y_stagnation, color='g', s=80, marker='o')

# рисуем разделительную линию тока
pyplot.contour(X, Y, psi,
                levels=[-strength_source/2, +strength_source/2],
                colors='#CD2305', linewidths=2, linestyles='solid');
```



Как по-вашему, на что это похоже? Выделенная таким прямолинейным образом разделительная линия создаёт впечатление, что набегающий слева поток раздваивается, чтобы обтечь препятствие криволинейной формы, похожее, например, на переднюю кромку профиля.

Если не обращать внимание на поток *внутри* области, ограниченной разделительной линией, её можно считать твердым телом. С математической точки зрения, разделительную линию тока можно заменить твердой стенкой, так как на линии тока выполняется условие непротекания. На самом деле, в случае источника в равномерном потоке, у такого тела есть собственное название — *полубесконечное тело Рэнкина*.

Контрольный вопрос Какова максимальная ширина полубесконечного тела Рэнкина?

Подсказка: Используйте закон сохранения массы и рассмотрите случай, когда x стремится к бесконечности.

2.3 Система источник-сток в равномерном потоке

Как вы думаете, что получится, если имеющейся картине добавить сток? Давайте выясним это! Сейчас мы можем в полной мере оценить пользу функций `getVelocity()` и `getStreamfunction()`, позволяющих создать сток без особых усилий.

```
In [10]: strength_sink = -5.0          # интенсивность стока
x_sink, y_sink = 1.0, 0.0      # положение стока

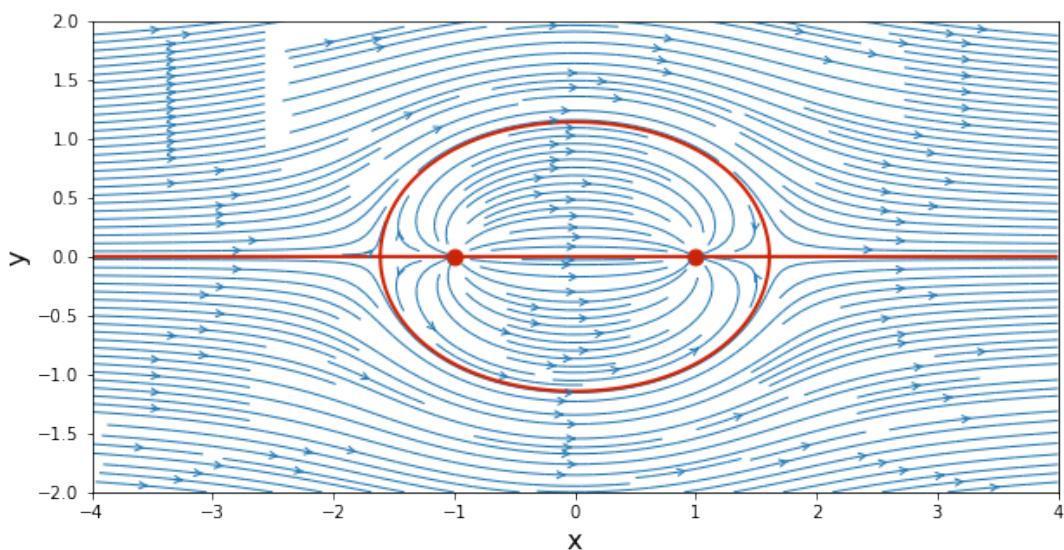
# вычисляем поле скорости в каждой точке сетки
u_sink, v_sink = get_velocity(strength_sink, x_sink, y_sink, X, Y)

# вычисляем функцию тока в каждом узле расчётной сетки
psi_sink = get_stream_function(strength_sink, x_sink, y_sink, X, Y)
```

Суперпозиция свободного потока, источника и стока производится простым сложением. Заметьте, как лихо мы сложили элементы двумерных массивов. Одно удовольствие.

```
In [11]: # суперпозиция свободного потока, источника и стока
u = u_freestream + u_source + u_sink
v = v_freestream + v_source + v_sink
psi = psi_freestream + psi_source + psi_sink

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1,\ 
    arrowstyle='->')
pyplot.scatter([x_source, x_sink], [y_source, y_sink], color='#CD2305',\ 
    s=80, marker='o')
pyplot.contour(X, Y, psi, levels=[0.], colors='red', linewidths=2,\ 
    linestyles='solid');
```



Результат выглядит *весъма* интересно. Поток как будто обтекает яйцо, похоже? На самом деле, замкнутую линию тока, похожую на эллипс, можно рассматривать как твердое тело, и тогда действительно получается, что поток обтекает что-то яйцеобразное. Оно называется *овал Рэнкина* (от лат. *ovum* — яйцо).

Можете поэкспериментировать с интенсивностью источника и стока и посмотреть, как изменится картина течения. Не забудьте также написать собственную версию кода в отдельном питоновском скрипте и запустить его с различными параметрами.

Контрольный вопрос Чему равны длина и ширина овала Рэнкина?

2.3.1 Уравнение Бернулли и коэффициент давления

Крайне важной характеристикой обтекания тела потоком является *коэффициент давления* C_p . Для оценки коэффициента давления применим *уравнение Бернулли* для несжимаемого течения:

$$p_\infty + \frac{1}{2}\rho V_\infty^2 = p + \frac{1}{2}\rho V^2$$

Помните ли вы условия, при которых выводится это уравнение? Если кто-то забыл, напомню, оно справедливо *вдоль линии тока*. Это очень важно.

Коэффициент давления определяется следующим образом:

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho V_\infty^2},$$

то есть

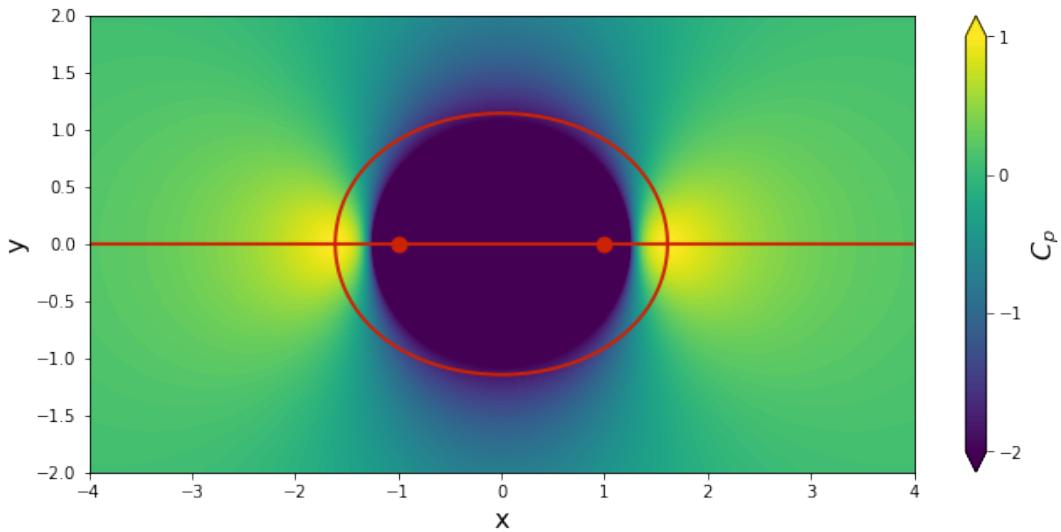
$$C_p = 1 - \left(\frac{V}{V_\infty} \right)^2$$

Обратите внимание, для несжимаемого течения в точке торможения $C_p = 1$.

Теперь нарисуем поле коэффициента давления для рассматриваемого течения. Подумайте и ответьте: Где находится максимум давления? Почему?

```
In [12]: # вычисляем поле коэффициента давления
cp = 1.0 - (u**2+v**2)/u_inf**2

# рисуем поле коэффициента давления
size = 10
pyplot.figure(figsize=(1.1*size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
contf = pyplot.contourf(X, Y, cp, levels=numpy.linspace(-2.0, 1.0, 100),\
                        extend='both')
cbar = pyplot.colorbar(contf)
cbar.set_label('$C_p$', fontsize=16)
cbar.set_ticks([-2.0, -1.0, 0.0, 1.0])
pyplot.scatter([x_source, x_sink], [y_source, y_sink], color='#CD2305',\
               s=80, marker='o')
pyplot.contour(X, Y, psi, levels=[0.], colors='red', linewidths=2, \
                linestyles='solid');
```



2.4 Заключительные замечания

Изучите материал, изложенный в этом блокноте, и попробуйте «подергать за ниточки» прямо в нём. Но в то же время начните писать собственную версию кода в удобном для вас редакторе — и, пожалуйста, *набирайте* код вместо копирования и вставки. При таком способе работы у вас появится время, чтобы подумать о том, что код означает.

На третьем занятии курса *AeroPython* вы познакомитесь с *диполем*. Вперед, к свершениям!

3 Диполь

Добро пожаловать на третье занятие курса *AeroPython!* На прошлых двух занятиях мы рассмотрели несколько интересных потенциальных течений при помощи блокнотов [Источник и сток](#) и [Источник и сток в равномерном потоке](#).

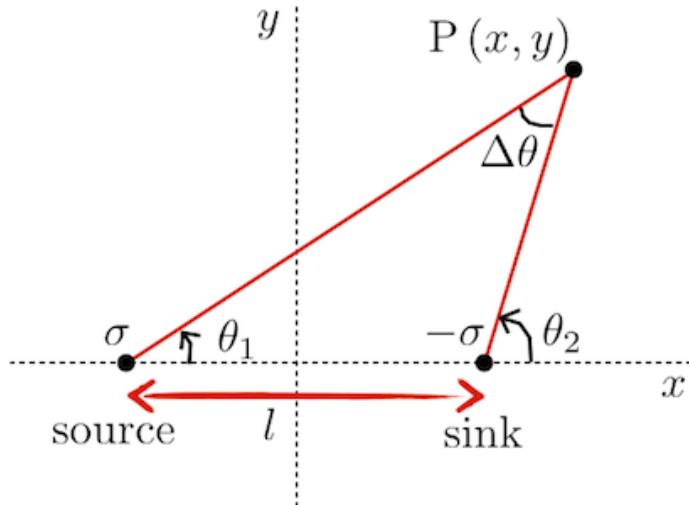
Вспомните систему из источника и стока, а затем представьте, что вы смотрите на картину течения из очень удалённой точки. С такой точки зрения линии между источником и стоком будут очень короткими. А остальные линии станут казаться двумя наборами окружностей, соприкасающихся в начале координат. Если удалиться настолько, что расстояние между источником и стоком приблизится к нулю, то получится конфигурация под названием *диполь*.

Давайте посмотрим, как это выглядит. Начнём с того, что загрузим наши любимые библиотеки.

```
In [1]: import math
import numpy
from matplotlib import pyplot
# помещаем рисунки в блокнот
%matplotlib inline
```

В предыдущем блокноте мы убедились, что пара источник-сток можно использовать для воспроизведения обтекания тела особенной формы, а именно, овала Рэнкина. А в этом — превратим такую пару в диполь.

Во-первых, рассмотрим источник интенсивности σ в точке $(-\frac{l}{2}, 0)$ и сток противоположной по знаку интенсивности, расположенный в $(\frac{l}{2}, 0)$. Вот эскиз, чтобы помочь вам представить картину:



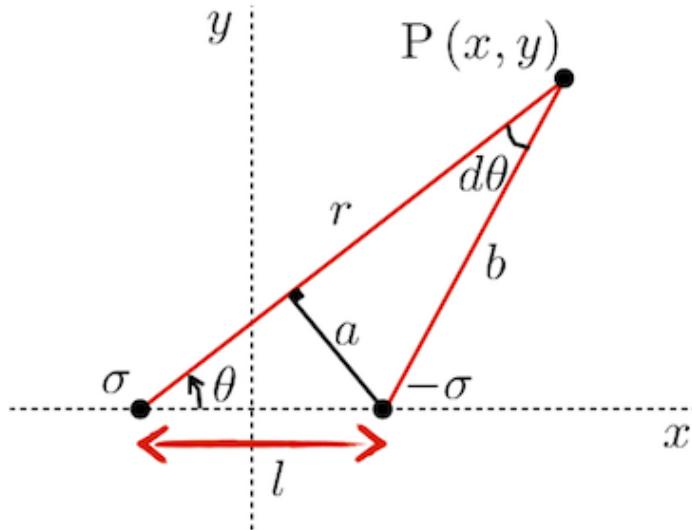
Функция тока пары источник-сток в точке $P(x, y)$ равна

$$\psi(x, y) = \frac{\sigma}{2\pi} (\theta_1 - \theta_2) = -\frac{\sigma}{2\pi} \Delta\theta$$

Пусть расстояние l между особенностями стремится к нулю, в то время как интенсивность возрастает так, чтобы произведение σl оставалось постоянным. В пределе получится *диполь* с интенсивностью $\kappa = \sigma l$.

Функция тока диполя в точке $P(x, y)$ равна

$$\psi(x, y) = \lim_{l \rightarrow 0} \left(-\frac{\sigma}{2\pi} d\theta \right) \quad \text{и} \quad \sigma l = \text{constant}$$



Рассматривая случай бесконечно малого $d\theta$, на основании схемы, приведенной выше, можно вывести:

$$a = l \sin \theta$$

$$b = r - l \cos \theta$$

$$d\theta = \frac{a}{b} = \frac{l \sin \theta}{r - l \cos \theta}$$

И тогда функция тока будет иметь вид

$$\psi(r, \theta) = \lim_{l \rightarrow 0} \left(-\frac{\sigma l}{2\pi} \frac{\sin \theta}{r - l \cos \theta} \right) \quad \text{и} \quad \sigma l = \text{constant}$$

то есть

$$\psi(r, \theta) = -\frac{\kappa}{2\pi} \frac{\sin \theta}{r}$$

В декартовых координатах, диполь, расположенный в начале координат имеет функцию тока

$$\psi(x, y) = -\frac{\kappa}{2\pi} \frac{y}{x^2 + y^2}$$

из которой можно получить выражения для компонент вектора скорости

$$u(x, y) = \frac{\partial \psi}{\partial y} = -\frac{\kappa}{2\pi} \frac{x^2 - y^2}{(x^2 + y^2)^2}$$

$$v(x, y) = -\frac{\partial \psi}{\partial x} = -\frac{\kappa}{2\pi} \frac{2xy}{(x^2 + y^2)^2}$$

С математикой мы разделялись, пора запрограммировать получившиеся соотношения и посмотреть, какие линии тока получаются. Начнём с создания сетки.

```
In [2]: N = 50                                # Число узлов сетки в каждом направлении
x_start, x_end = -2.0, 2.0                  # границы по x
y_start, y_end = -1.0, 1.0                  # границы по y
x = numpy.linspace(x_start, x_end, N)        # создаём одномерный массив x
y = numpy.linspace(y_start, y_end, N)        # создаём одномерный массив y
X, Y = numpy.meshgrid(x, y)                  # создаём сетку
```

Рассмотрим диполь интенсивности $\kappa = 1.0$, расположенныйный в начале координат.

```
In [3]: kappa = 1.0 # интенсивность диполя
x_doublet, y_doublet = 0.0, 0.0 # положение диполя
```

Как показал опыт предыдущего блокнота, мы поступили мудро, определив специальные функции для вычисления компонент скорости и функции тока, которые можно повторно использовать, если потребуется создать в расчётной области более одного диполя.

```
In [4]: def get_velocity_doublet(strength, xd, yd, X, Y):
    """
    Returns the velocity field generated by a doublet.

    Parameters
    -----
    strength: float
        Strength of the doublet.
    xd: float
        x-coordinate of the doublet.
    yd: float
        y-coordinate of the doublet.
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.

    Returns
    -----
    u: 2D Numpy array of floats
        x-component of the velocity vector field.
    v: 2D Numpy array of floats
        y-component of the velocity vector field.
    """
    u = - strength/(2*math.pi)*((X-xd)**2-(Y-yd)**2)/\
        ((X-xd)**2+(Y-yd)**2)**2
    v = - strength/(2*math.pi)*2*(X-xd)*(Y-yd)/\
        ((X-xd)**2+(Y-yd)**2)**2

    return u, v

def get_stream_function_doublet(strength, xd, yd, X, Y):
    """
    Returns the stream-function generated by a doublet.

    Parameters
    -----
    strength: float
        Strength of the doublet.
    xd: float
        x-coordinate of the doublet.
    yd: float
        y-coordinate of the doublet.
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.

    Returns
    -----
    psi: 2D Numpy array of floats
        The stream-function.
```

```

"""
psi = - strength/(2*math.pi)*(Y-yd)/((X-xd)**2+(Y-yd)**2)

return psi

```

Когда функции определены, можно вызвать их, передав в качестве переменных характеристики диполя: его интенсивность `kappa` и положение `x_doublet`, `y_doublet`.

```

In [5]: # рассчитываем поле скорости в узлах расчётной сетки
u_doublet, v_doublet = get_velocity_doublet(kappa, x_doublet, y_doublet,\n                                              X, Y)

# рассчитываем функцию тока в узлах расчётной сетки
psi_doublet = get_stream_function_doublet(kappa, x_doublet, y_doublet,\n                                             X, Y)

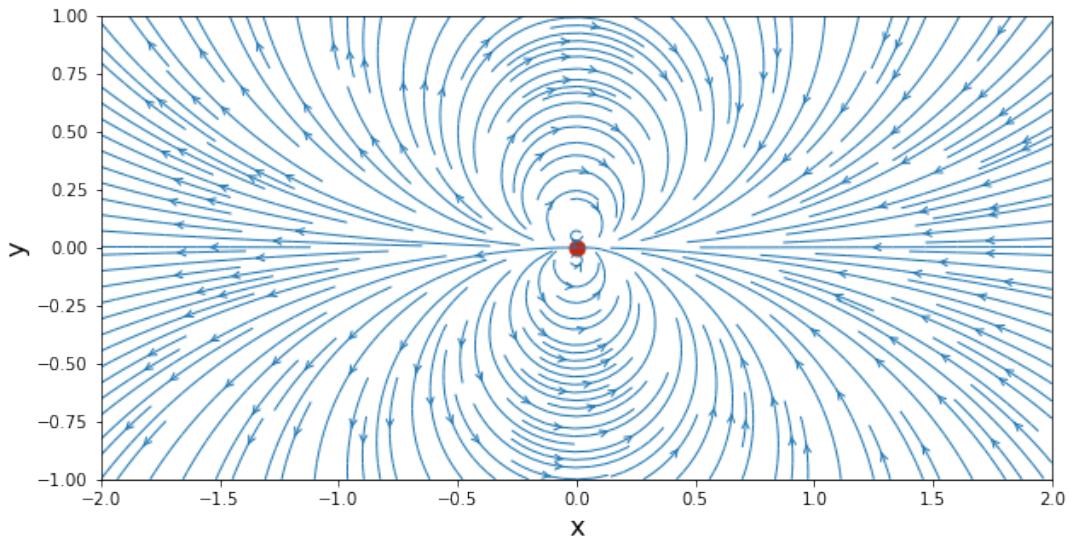
```

Теперь все готово для визуализации течения.

```

In [6]: # рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u_doublet, v_doublet,
                  density=2, linewidth=1, arrowstyle='->')
pyplot.scatter(x_doublet, y_doublet, color='#CD2305', s=80, marker='o');

```



Как мы себе и представляли, похоже на картину линий тока пары источник-сток, если смотреть на неё издали. Вы можете спросить, зачем всё это? Ведь это не похоже ни на одно из течений, важных для практики? Вы ошибаетесь, если действительно так думаете.

3.1 Диполь в равномерном потоке

Сам по себе диполь не представляет особой ценности с точки зрения решения практических задач аэродинамики. Воспользуемся нашим излюбленным приёмом — суперпозицией: диполь в равномерном потоке оказывается весьма полезным классом течений. Давайте для начала зададим равномерный горизонтальный поток.

```
In [7]: u_inf = 1.0          # скорость потока на бесконечности
```

Вспомним предыдущее занятие: в декартовых координатах вектор скорости равномерного потока имеет компоненты $u = U_\infty$ and $v = 0$. Интегрируя, получим функцию тока $\psi = U_\infty y$.

Теперь можно вычислить скорости и функцию тока в каждом узле расчётной сетки. И как мы убедились, сделать это можно, написав всего по одной строчке кода на каждый массив.

```
In [8]: u_freestream = u_inf * numpy.ones((N, N), dtype=float)
v_freestream = numpy.zeros((N, N), dtype=float)

psi_freestream = u_inf * Y
```

Ниже функция тока диполя в равномерном потоке, полученная простым сложением. Как и раньше в блокноте [Источник и сток в свободном потоке](#), красным цветом выделена *разделительная линия тока*.

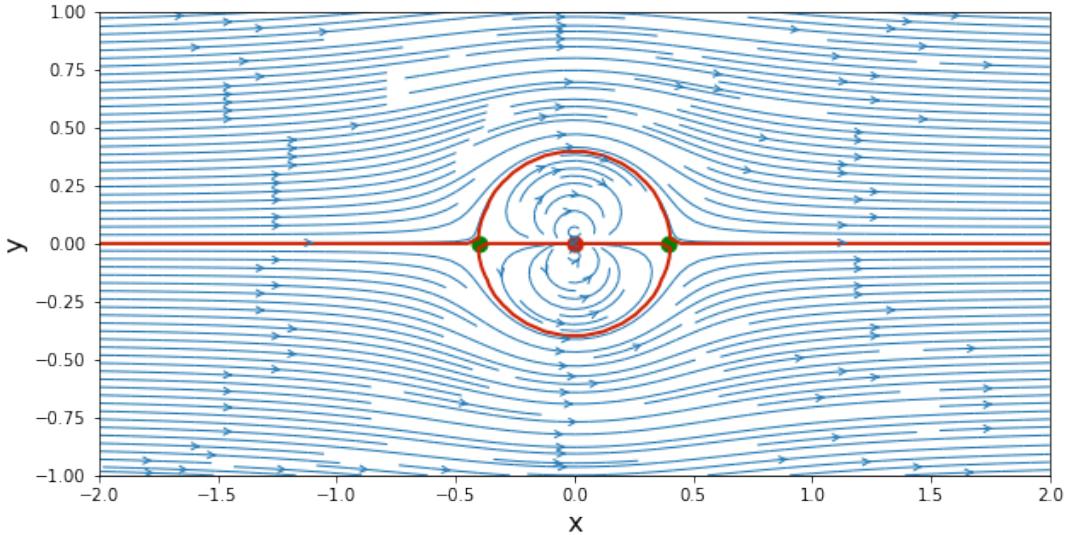
Согласно рисунку, такая картина линий тока соответствует течению вокруг цилиндра с центром в точке расположения диполя. Все линии тока, находящиеся за пределами цилиндра приходят с бесконечности. Линии же тока внутри цилиндра можно проигнорировать, поскольку им соответствует твердое тело. Этот результат важнее, чем кажется на первый взгляд.

```
In [9]: # наложение диполя и равномерного потока
u = u_freestream + u_doublet
v = v_freestream + v_doublet
psi = psi_freestream + psi_doublet

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1,
                  arrowstyle='->')
pyplot.contour(X, Y, psi, levels=[0.], colors='#CD2305', linewidths=2,
                linestyles='solid')
pyplot.scatter(x_doublet, y_doublet, color='#CD2305', s=80, marker='o')

# определяем положение точек торможения
x_stagn1, y_stagn1 = +math.sqrt(kappa/(2*math.pi*u_inf)), 0
x_stagn2, y_stagn2 = -math.sqrt(kappa/(2*math.pi*u_inf)), 0

# рисуем точки торможения
pyplot.scatter([x_stagn1, x_stagn2], [y_stagn1, y_stagn2], color='g',
               s=80, marker='o');
```



Контрольный вопрос Чему равен радиус окружности, возникающей при помещении диполя интенсивности κ в равномерный поток со скоростью U_∞ , движущийся в направлении оси x ?

Контрольное задание Выше выписана функция тока диполя в цилиндрических координатах. Добавьте к ней функцию тока равномерного потока в тех же координатах и проанализируйте результат. Вы обнаружите, что $\psi = 0$ в $r = a$ при любых θ . Линия $\psi = 0$ в цилиндрических координатах соответствует окружности радиуса a . Запишите компоненты скорости в цилиндрических координатах, определите скорость на поверхности цилиндра. О чём говорит этот результат?

3.1.1 Уравнение Бернулли и коэффициент давления

Крайней важной характеристикой обтекания тела потоком является *коэффициент давления* C_p . Для оценки коэффициента давления применим *уравнение Бернулли* для несжимаемого течения, согласно которому для двух точек на линии тока справедливо

$$p_\infty + \frac{1}{2}\rho U_\infty^2 = p + \frac{1}{2}\rho U^2$$

Коэффициент давления определяется как разница местного давления и давления в невозмущённом потоке, отнесённая к скоростному напору (динамическому давлению):

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2},$$

то есть

$$C_p = 1 - \left(\frac{U}{U_\infty} \right)^2$$

Для несжимаемого течения $C_p = 1$ в точке торможения. Изобразим коэффициент давления для всей расчётной области.

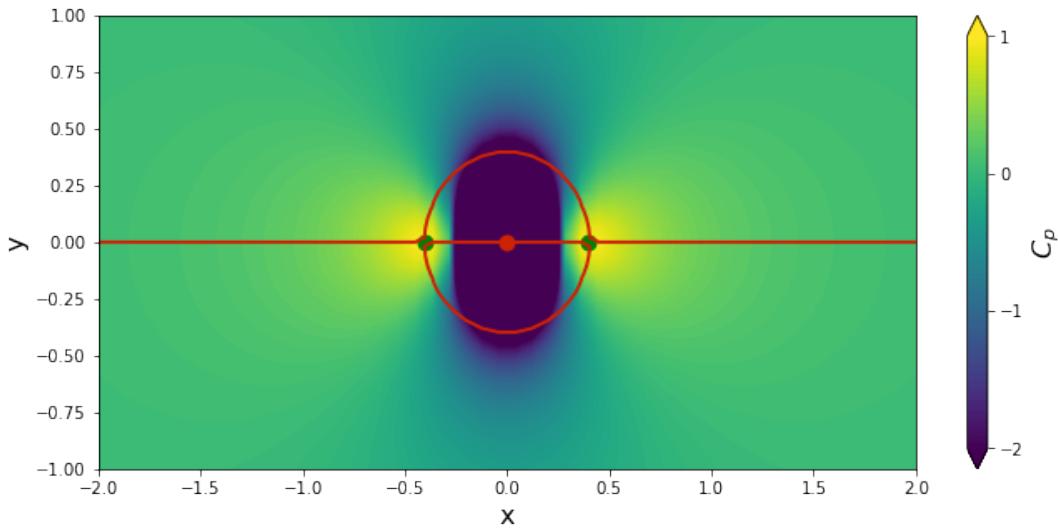
```
In [10]: # вычисляем поле коэффициента давления
cp = 1.0 - (u**2+v**2)/u_inf**2

# рисуем поле коэффициента давления
size = 10
pyplot.figure(figsize=(1.1*size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
```

```

pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
contf = pyplot.contourf(X, Y, cp, levels=numpy.linspace(-2.0, 1.0, 100),\
                        extend='both')
cbar = pyplot.colorbar(contf)
cbar.set_label('$C_p$', fontsize=16)
cbar.set_ticks([-2.0, -1.0, 0.0, 1.0])
pyplot.scatter(x_doublet, y_doublet, color='#CD2305', s=80, marker='o')
pyplot.contour(X,Y,psi, levels=[0.], colors='#CD2305', linewidths=2,\ 
                linestyles='solid')
pyplot.scatter([x_stagn1, x_stagn2], [y_stagn1, y_stagn2], color='g',\ 
                s=80, marker='o');

```



Контрольное задание Покажите, что распределение коэффициента давления на поверхности круглого цилиндра записывается в виде

$$C_p = 1 - 4 \sin^2 \theta$$

и постройте график зависимости коэффициента давления от угла.

Вопрос для размышления Не кажется ли вам странным, что поле коэффициента давления (как и распределение давления по поверхности) симметрично относительно вертикальной оси?

Это означает, что давление в передней части цилиндра совпадает с давлением сзади. Следовательно, продольная компонента силы, действующей на цилиндр, равна нулю.

Однако мы знаем, что даже при низких числах Рейнольдса (ползучее течение), на тело *действует* сила сопротивления. Получается, что теория не в состоянии отразить экспериментально наблюдаемый факт! Это расхождение известно как *парадокс д'Аламбера*.

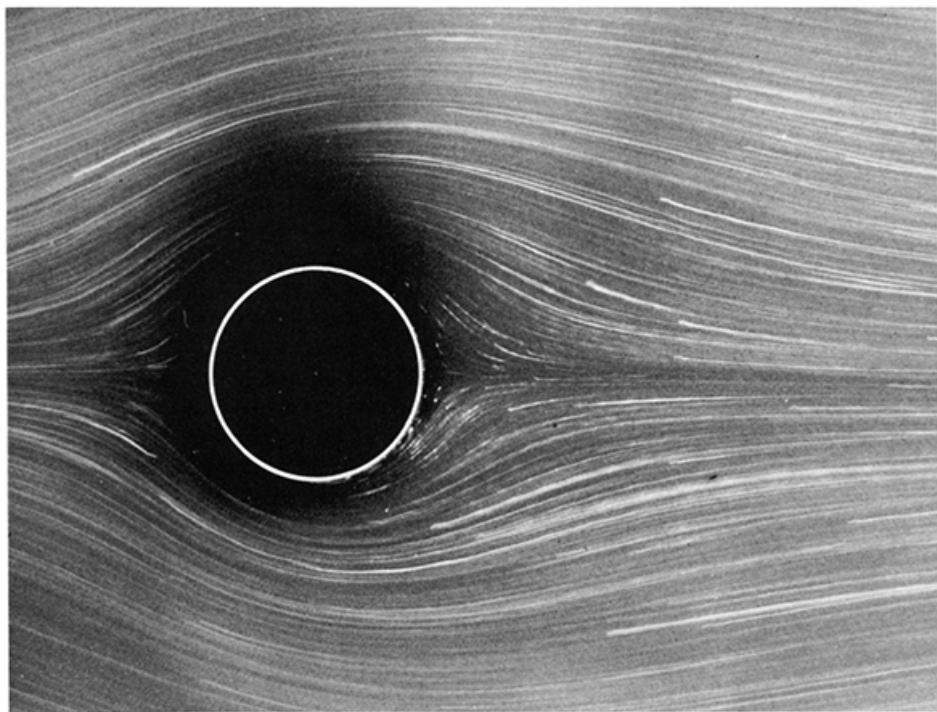
Вот как ползучее течение вокруг цилиндра выглядит *на самом деле*:

```
In [11]: from IPython.display import YouTubeVideo
YouTubeVideo('j_d7_V2gXv0?t=28m59s')
```

```
Out[11]: <IPython.lib.display.YouTubeVideo at 0x1a66b801898>
```

Если вы посмотрите внимательно, то заметите небольшую асимметрию в картине течения. Может ли вы объяснить это? Что из этого следует?

Вот довольно известная визуализация настоящего потока вокруг цилиндра при числе Рейнольдса $Re = 1,54$. Это изображение было получено С. Танедой, оно представлено в «Альбоме течений жидкости и газа», Милтона Ван Дайка. Настоящий шедевр.



24. Circular cylinder at $R=1.54$. At this Reynolds number the streamline pattern has clearly lost the fore-and-aft symmetry of figure 6. However, the flow has not yet separated at the rear. That begins at about $R=5$,

though the value is not known accurately. Streamlines are made visible by aluminum powder in water. *Photograph by Sadatoshi Taneda*

Распределение источников по профилю

На [третьем занятии](#) курса *AeroPython* вы узнали, что потенциальное обтекание круглого цилиндра можно получить, используя суперпозицию диполя и свободного потока. Но потенциальные течения ещё круче: вы можете получить поток вокруг тела *любой* формы. Вы спросите, как такое возможно?

Тела, не создающие подъёмную силу, можно смоделировать при помощи комбинации источников, расположенных на поверхности тела, и свободного потока. В этом задании вам нужно будет смоделировать обтекание профиля NACA0012 методом источников.

Прежде, чем начать, подумайте и ответьте на несколько вопросов. Для случая обтекания симметричного профиля под нулевым углом атаки:

- Где находится точка максимального давления?
- Как называется эта точка?
- Будет ли этот профиль создавать подъёмную силу?

В конце этого задания, вернитесь к этим вопросам, и проверьте ответы.

Постановка задачи

Вам нужно зачитать данные из файлов, содержащих координаты и интенсивности источников, расположенных на поверхности профиля NACA0012.

Этих файлов три: NACA0012_x.txt, NACA0012_y.txt и NACA0012_sigma.txt. Для загрузки данных из файла в массив NumPy вам потребуется функция `numpy.loadtxt()`. Необходимые файлы находятся в папке `resources`.

Используя расчётную область $[-1, 2] \times [-0.5, 0.5]$ и сетку, размерности 51 узел в каждом направлении, вычислите скорость, обусловленную набором источников в набегающем потоке, направленном вдоль оси x , со скоростью $U_\infty = 1$. Определите также коэффициент давления в каждом узле сетки.

Вопросы

1. Каково максимальное значение коэффициента давления C_p ?
2. Какой индекс у элемента массива, содержащего максимальное значение C_p ?

Сделайте следующие визуализации и проанализируйте получившиеся картины течений:

- Линии тока в расчётной области и профиль NACA0012 на одном рисунке
- Распределение коэффициента давления, маркером отметьте положение максимума давления

Подсказка: Возможно, вам потребуются некоторые новые функции NumPy: `numpy.unravel_index()` и `numpy.argmax()`

Подумайте

1. Линии тока выглядят так, как вы ожидали?
2. Что говорит распределение давления о подъёмной силе, создаваемой профилем?
3. Кажется ли вам правильным расположение точки максимального давления?

4 Вихрь

Давайте освежим пройденное. На первых трех занятиях курса *AeroPython* мы рассмотрели:

1. **пару источник-сток** — вы научились создавать картины линий тока (`streamplot()`) и отмечать на них особые точки при помощи функции `scatter()`. Если вы выполнили домашнее задание, то убедились, что эквипотенциальные линии перпендикулярны линиям тока.
2. **пару источник-сток в набегающем потоке** — получили *овал Рэнжина*. Мы впервые увидели, что с помощью потенциальных течений можно конструировать обтекание геометрических объектов. Вы также научились определять пользовательские функции. Они заставляют Python работать на вас, и делают его ещё больше похожим на обычный английский.
3. **диполь** — картина линий тока соответствует двумерному обтеканию кругового цилиндра. Вы столкнулись с парадоксом д'Аламбера и, надеюсь, это заставило вас задуматься. Если нет, вернитесь к этому уроку и *задумайтесь!*

Вы выполнили [задание](#)? Оно показывает, как с помощью *распределённых источников* смоделировать потенциальное обтекание профиля. Это уже похоже на аэродинамику!

Какой же главный результат мы желаем получить от прикладной аэродинамики? Конечно же, полететь, опираясь не на силу мускулов, а на силу разума! А для этого нужна *подъёмная сила*, превосходящая вес тела, желающего взлететь.

В этом разделе нашего курса речь пойдет о подъёмной силе. Вначале мы смоделируем поток в окрестности потенциального вихря. Оказывается, циркуляция вихря и подъёмная сила тесно связаны.

4.1 Что такое вихрь?

Этот вопрос сложнее, чем кажется! Простой ответ заключается в том, что вихрь это движение по круговым линиям тока. Представьте себе линии тока в виде концентрических окружностей с центром в заданной точке — при этом *совсем не обязательно*, чтобы жидкие частицы врашались вокруг своих осей.

При безвихревом циркуляционном движении азимутальная компонента скорости постоянна вдоль круговых линий тока и обратно пропорциональна радиусу, а радиальная компонента равна нулю. В полярных координатах:

$$u_\theta(r, \theta) = \frac{\text{constant}}{r} \quad , \quad u_r(r, \theta) = 0 \quad (1)$$

Завихренность равна нулю во всей области за исключением точки расположения вихря, в которой производная u_θ обращается в бесконечность.

Понятие циркуляции было введено на первом занятии ([Источник и сток](#)). Воспользуемся этим определением. Согласно правилу знаков, вихрь, закрученный против часовой стрелки, имеет отрицательный знак. Тогда вдоль любой линии тока, окружающей вихрь, справедливо:

$$\Gamma = - \oint \mathbf{v} \cdot d\vec{l} = -u_\theta 2\pi r \quad (2)$$

Таким образом, константа в выражении для u_θ в (1) равна $\Gamma/2\pi$, и можно записать:

$$u_\theta(r, \theta) = \frac{\Gamma}{2\pi r} \quad (3)$$

Функцию тока можно получить, интегрируя компоненты скорости:

$$\psi(r, \theta) = \frac{\Gamma}{2\pi} \ln r \quad (4)$$

В декартовых координатах функция тока записывается как

$$\psi(x, y) = \frac{\Gamma}{4\pi} \ln(x^2 + y^2) \quad (5)$$

а компоненты скорости:

$$u(x, y) = \frac{\Gamma}{2\pi} \frac{y}{x^2 + y^2} \quad v(x, y) = -\frac{\Gamma}{2\pi} \frac{x}{x^2 + y^2} \quad (6)$$

Такое циркуляционное течение является безвихревым везде, кроме центра вихря. В нем завихренность бесконечная. Интенсивность точечного вихря равна циркуляции Γ .

4.2 Рассчитаем вихрь

Подготовительные шаги остаются прежними: импортируем любимые библиотеки и создаём сетку, в узлах которой будет рассчитываться поле скорости.

```
In [1]: import numpy
import math
from matplotlib import pyplot
# помещаем рисунки в блокнот
%matplotlib inline

In [2]: N = 50                      # Число узлов сетки в каждом направлении
        x_start, x_end = -2.0, 2.0      # границы по x
        y_start, y_end = -1.0, 1.0      # границы по y
        x = numpy.linspace(x_start, x_end, N)    # создаем одномерный массив x
        y = numpy.linspace(y_start, y_end, N)    # создаем одномерный массив y
        X, Y = numpy.meshgrid(x, y)          # создаем сетку
```

Зададим вихрь с интенсивностью $\Gamma = 5$ и поместим его в центр расчётной области:

```
In [3]: gamma = 5.0                  # интенсивность вихря
        x_vortex, y_vortex = 0.0, 0.0  # положение вихря
```

Определим две функции,

- `get_velocity_vortex()` и
- `get_stream_function_vortex()`,

для расчёта компонент скорости и функции тока на декартовой сетке по заданным интенсивности и положению вихря. В последствии мы воспользуемся этими функциями и все посчитаем. Но сначала, давайте их напишем.

```
In [4]: def get_velocity_vortex(strength, xv, yv, X, Y):
        """
        Returns the velocity field generated by a vortex.

        Parameters
        -----
        strength: float
            Strength of the vortex.
        xv: float
            x-coordinate of the vortex.
        yv: float
            y-coordinate of the vortex.
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.

        Returns
        -----
        u: 2D Numpy array of floats
            x-component of the velocity vector field.
        v: 2D Numpy array of floats
            y-component of the velocity vector field.
        """

        r_sq = (X - xv)**2 + (Y - yv)**2
        r_sq[r_sq == 0] = 1e-06
        u = strength * yv / r_sq
        v = -strength * xv / r_sq
        return u, v
```

```

    y-component of the velocity vector field.
"""
u = + strength/(2*math.pi)*(Y-yv)/((X-xv)**2+(Y-yv)**2)
v = - strength/(2*math.pi)*(X-xv)/((X-xv)**2+(Y-yv)**2)

return u, v

In [5]: def get_stream_function_vortex(strength, xv, yv, X, Y):
"""
    Returns the stream-function generated by a vortex.

Parameters
-----
strength: float
    Strength of the vortex.
xv: float
    x-coordinate of the vortex.
yv: float
    y-coordinate of the vortex.
X: 2D Numpy array of floats
    x-coordinate of the mesh points.
Y: 2D Numpy array of floats
    y-coordinate of the mesh points.

Returns
-----
psi: 2D Numpy array of floats
    The stream-function.
"""

psi = strength/(4*math.pi)*numpy.log((X-xv)**2+(Y-yv)**2)

return psi

```

А теперь вызовем их, используя интенсивность и положение вихря, а также координаты узлов расчётной сетки в качестве параметров.

```

In [6]: # рассчитываем поле скорости в узлах расчётной сетки
u_vortex, v_vortex = get_velocity_vortex(gamma, x_vortex, y_vortex, X, Y)

# рассчитываем функцию тока в узлах расчётной сетки
psi_vortex = get_stream_function_vortex(gamma, x_vortex, y_vortex, X, Y)

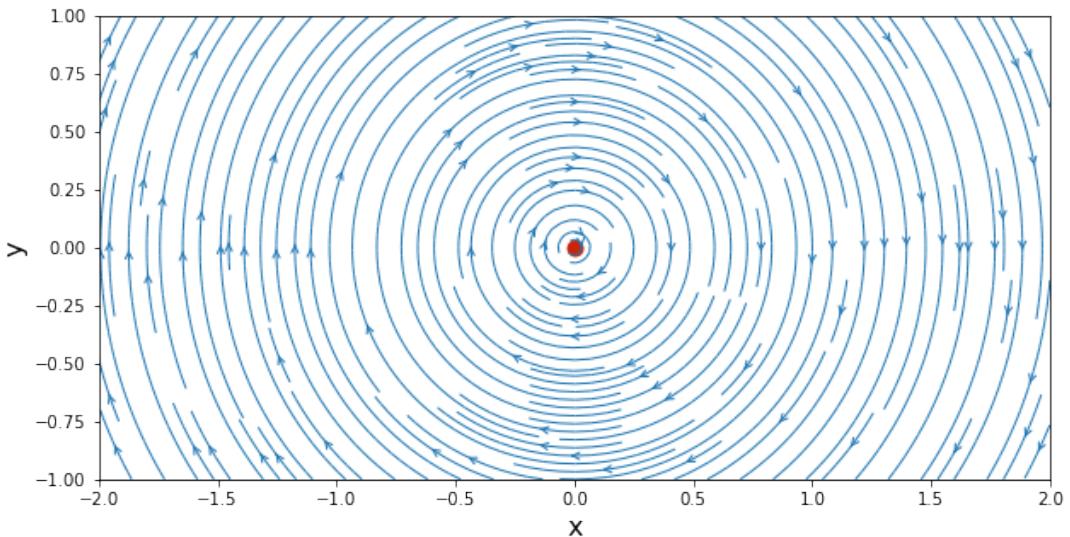
```

Теперь мы можем визуализировать линии тока для потенциального вихря и убедиться, что они выглядят как концентрические окружности с центром в месте расположения вихря, как, впрочем, и ожидалось.

```

In [7]: # рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u_vortex, v_vortex, density=2, linewidth=1,
                  arrowsize=1, arrowstyle='->')
pyplot.scatter(x_vortex, y_vortex, color='#CD2305', s=80, marker='o');

```



4.3 Вихрь и сток

Забавы ради, применим нашу суперспособность — суперпозицию. Добавим к вихрю сток. Для этого воспользуемся функциями, возвращающими компоненты скорости и функцию тока для стока, и добавим полученные значения к соответствующим величинам для вихря. Получим *вихресток*.

```
In [8]: strength_sink = -1.0          # интенсивность стока
      x_sink, y_sink = 0.0, 0.0        # положение стока

In [9]: def get_velocity_sink(strength, xs, ys, X, Y):
        """
        Returns the velocity field generated by a sink.

        Parameters
        -----
        strength: float
            Strength of the sink.
        xs: float
            x-coordinate of the sink.
        ys: float
            y-coordinate of the sink.
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.

        Returns
        -----
        u: 2D Numpy array of floats
            x-component of the velocity vector field.
        v: 2D Numpy array of floats
            y-component of the velocity vector field.
        """
        u = strength/(2*math.pi)*(X-xs)/((X-xs)**2+(Y-ys)**2)
        v = strength/(2*math.pi)*(Y-ys)/((X-xs)**2+(Y-ys)**2)

        return u, v
```

```
In [10]: def get_stream_function_sink(strength, xs, ys, X, Y):
    """
    Returns the stream-function generated by a sink.

    Parameters
    -----
    strength: float
        Strength of the sink.
    xs: float
        x-coordinate of the sink.
    ys: float
        y-coordinate of the sink.
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.

    Returns
    -----
    psi: 2D Numpy array of floats
        The stream-function.
    """
    psi = strength/(2*math.pi)*numpy.arctan2((Y-ys), (X-xs))

    return psi
```

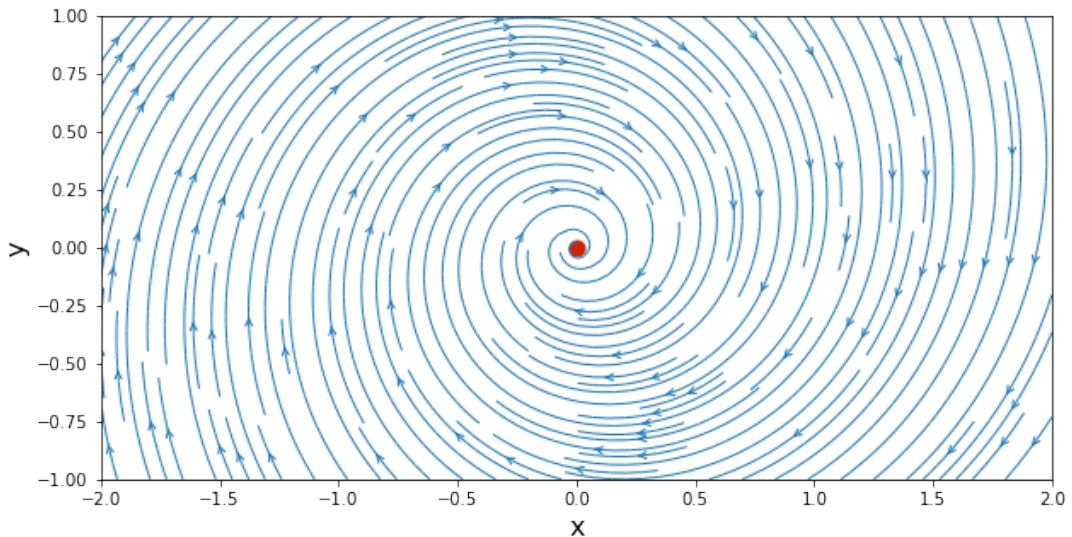
```
In [11]: # рассчитываем поле скорости в узлах расчётной сетки
u_sink, v_sink = get_velocity_sink(strength_sink, x_sink, y_sink, X, Y)

# рассчитываем функцию тока в узлах расчётной сетки
psi_sink = get_stream_function_sink(strength_sink, x_sink, y_sink, X, Y)
```

Теперь, давайте визуализируем линии тока для вихревого, и восхитимся нашим художественным творением:

```
In [12]: # суперпозиция стока и вихря
u = u_vortex + u_sink
v = v_vortex + v_sink
psi = psi_vortex + psi_sink

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowstyle='->')
pyplot.scatter(x_vortex, y_vortex, color='#CD2305', s=80, marker='o');
```



Получилась отличная иллюстрация в стиле оп-арт. Кроме того, это неплохая модель слива воды в раковине. Но есть ли от этого какая-нибудь польза? Есть!

Сперва вам придётся отбросить ходунки и — самостоятельно — смоделировать течение от бесконечной цепочки вихрей. Ваше задание в следующем блокноте.

Затем мы узнаем, как связаны вихрь и подъёмная сила. Вот увидите, вихрь — очень важное понятие в аэродинамике.

4.4 Что такое «безвихревой» вихрь?

Остался непроясненным ещё один важный вопрос.

Мы рассматриваем вихрь как потенциальное, а значит безвихревое течение. Это как? Конечно, если есть вихрь, есть вращение!

Глаза нас не обманывают. Такой ход мыслей довольно естественен. Но в случае с потенциальным вихревым течением, поток хоть и движется по круговым траекториям, элементарные объемы жидкости не вращаются вокруг собственных осей.

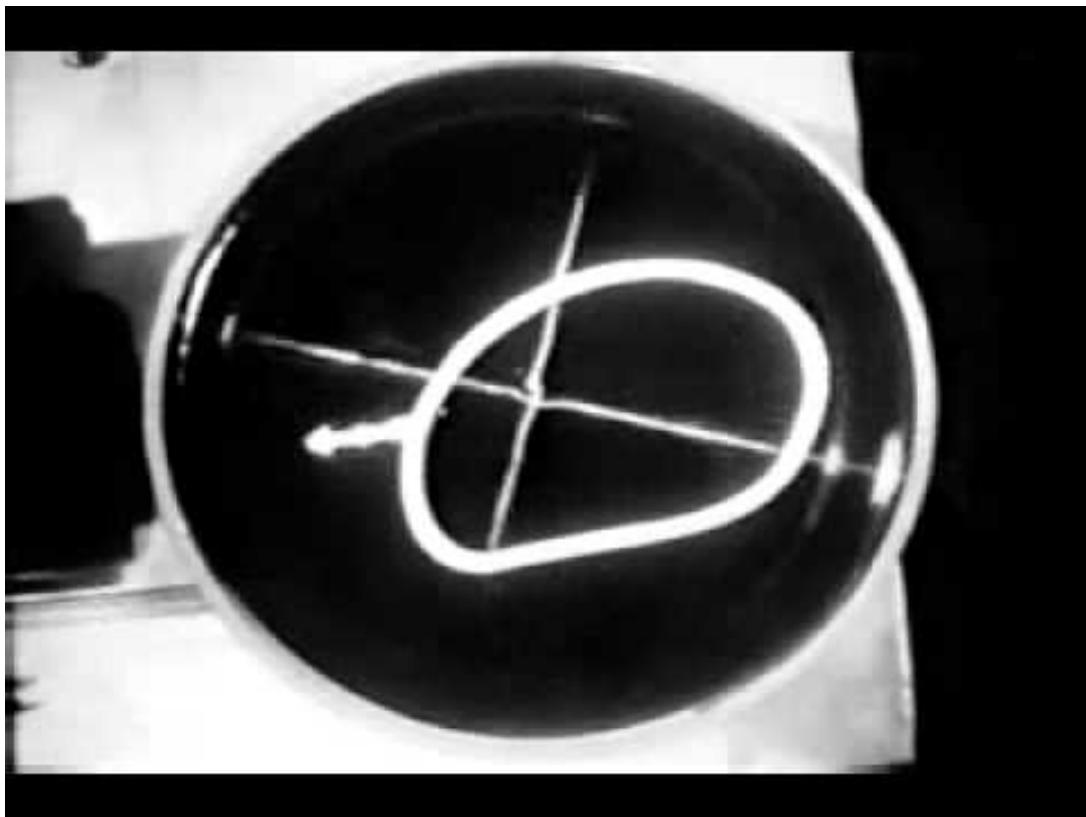
Классический эксперимент, демонстрируемый в этом видеоролике, поможет в понимании. Помимо 25 секунд после отметки 4:25 и увидите, что «измеритель завихренности» перемещается вокруг изолированного вихря, при этом не вращаясь.

```
In [14]: from IPython.display import YouTubeVideo
        from datetime import timedelta

        start=int(timedelta(hours=0, minutes=4, seconds=25).total_seconds())

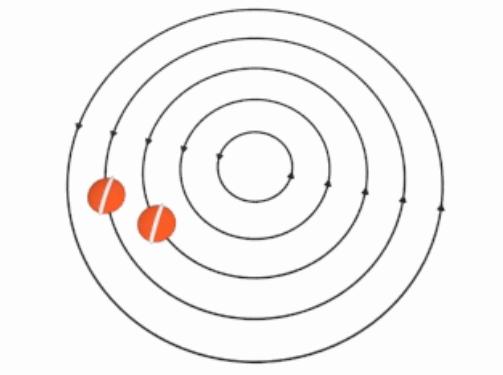
        YouTubeVideo("loCLkcYEWD4", start=start)
```

Out[14]:



Помните: завихренность является мерой локальной угловой скорости каждого жидкого элемента. Если жидкые элементы движутся по кругу, но не вращаются сами по себе, то нет завихренности!

Иллюстрация из [Википедии](#) поможет лучше понять, что происходит в изолированном вихре. Оранжевые маркеры движутся по окружностям, не меняя ориентации в пространстве.



4.5 Далее...

В следующем блокноте курса *AeroPython* содержится задание для самостоятельной работы "Бесконечная цепочка вихрей".

5 Бесконечная цепочка вихрей

Цель этого задания — визуализация обтекания бесконечной цепочки вихрей. Сначала рассмотрим случай конечной цепочки вихрей, полученную простой суперпозицией изолированных вихрей. Добавляя вихри, можно увидеть как будет меняться картина течения, по мере приближения к предельному случаю бесконечной цепочки. Но при этом некоторые отличия будут присутствовать.

Можно вывести аналитическое выражение для бесконечного случая, такой вывод приводится ниже. С помощью этого аналитического выражения можно визуализировать линии тока для бесконечного случая. Сравните две картины течений и подумайте, чем они отличаются друг от друга?

В этом блокноте нет ячеек с кодом. Ваша задача — изучить теорию (проделав все математические выкладки самостоятельно на листе бумаги); подумать, как наиболее эффективно реализовать полученные соотношения; и, наконец, написать код для визуализации результатов.

5.1 Изолированный вихрь (предыдущее занятие)

Вы можете не подозревать этого, но вихрь играет очень важную роль в классической аэродинамической теории. В этом задании вы откроете для себя некоторые особенности.

Для начала освежим пройденное. Как мы узнали на прошлом занятии, вихрь с интенсивностью Γ описывается функцией тока:

$$\psi(r, \theta) = \frac{\Gamma}{2\pi} \ln r$$

и потенциалом скорости

$$\phi(r, \theta) = -\frac{\Gamma}{2\pi} \theta$$

Компоненты скорости в полярной системе координат записываются в виде

$$u_r(r, \theta) = 0$$

$$u_\theta(r, \theta) = -\frac{\Gamma}{2\pi r}$$

В декартовых координатах (x, y) компоненты скорости для вихря с интенсивности Γ , расположенного в $(x_{\text{vortex}}, y_{\text{vortex}})$, задаются соотношениями

$$u(x, y) = +\frac{\Gamma}{2\pi} \frac{y - y_{\text{vortex}}}{(x - x_{\text{vortex}})^2 + (y - y_{\text{vortex}})^2}$$

$$v(x, y) = -\frac{\Gamma}{2\pi} \frac{x - x_{\text{vortex}}}{(x - x_{\text{vortex}})^2 + (y - y_{\text{vortex}})^2}$$

а функция тока записывается в виде

$$\psi(x, y) = \frac{\Gamma}{4\pi} \ln((x - x_{\text{vortex}})^2 + (y - y_{\text{vortex}})^2)$$

5.2 Суперпозиция нескольких вихрей

В этом задании мы рассмотрим важный пример для иллюстрации понятие *вихревая пелена*: бесконечную цепочку вихрей одинаковой интенсивности Γ , равномерно распределённых по оси x с интервалом a . Но начнем мы с конечного ряда вихрей.

Функция тока i — вихря ψ_i на расстоянии r_i равна

$$\psi_i = \frac{\Gamma}{2\pi} \ln r_i$$

Согласно принципу суперпозиции, функция тока для N вихрей запишется в виде:

$$\psi = \frac{\Gamma}{2\pi} \sum_{i=1}^N \ln r_i$$

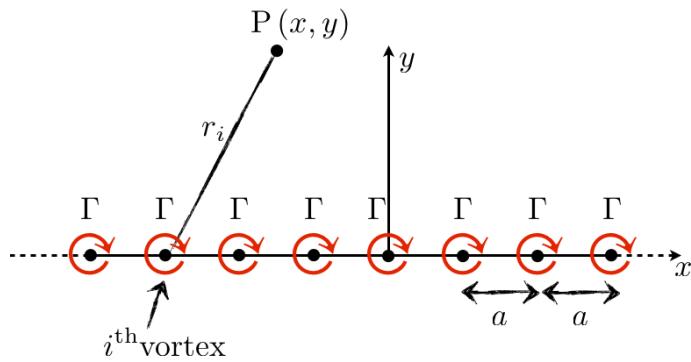
А поле скорости в декартовых координатах для цепочки вихрей задается выражениями:

$$u(x, y) = +\frac{\Gamma}{2\pi} \sum_{i=1}^N \frac{y - y_i}{(x - x_i)^2 + (y - y_i)^2}$$

$$v(x, y) = -\frac{\Gamma}{2\pi} \sum_{i=1}^N \frac{x - x_i}{(x - x_i)^2 + (y - y_i)^2}$$

где (x_i, y_i) — декартовы координаты i -го вихря.

Вот схема, поясняющая, что где находится:



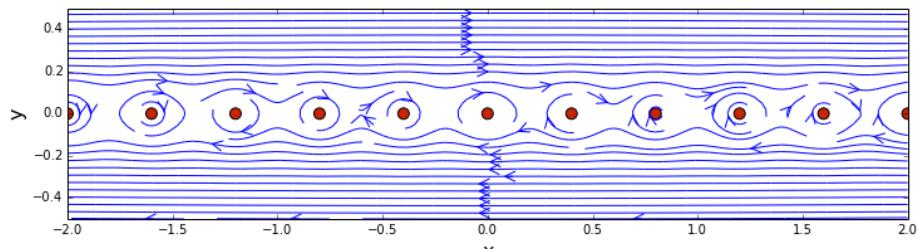
В следующем разделе мы покажем, как вывести аналитическое выражение для бесконечной суммы вихрей. Потерпите.

Задача

Вычислить поле скорости и нарисовать линии тока для цепочки вихрей Поместите N вихрей на горизонтальную ось и визуализируйте картину течения. Выполните следующие действия:

- С помощью уравнений, полученных выше, вычислите компоненты скорости каждого вихря в узлах расчётной сетки.
- Помните, что конечное количество вихрей можно представить с помощью списка *list* или массива NumPy *array*. Подумайте и решите, какой способ использовать.
- Определите функции, чтобы избежать повторения кода (можете также использовать классы, если вам так удобнее).
- После того, как вы получите скорости, примените принцип суперпозиции и нарисуйте получившееся поле.
- Поэкспериментируйте с размерами цепочки вихрей и диапазоном рисунка. Сделайте свою иллюстрацию публикуемого качества!

В конце концов вы должны получить нечто похожее на это:



5.3 Бесконечная цепочка вихрей

Для вывода аналитических зависимостей воспользуемся комплексным представлением:

$$z = x + jy$$

где $j^2 = -1$. Для мнимой единицы обычно используется обозначение i , но его можно спутать с обозначением индекса.

Комплексный потенциал определяется как $w = \phi + j\psi$, где ϕ — потенциал скоростей, а ψ — функция тока. Если продифференцировать комплексный потенциал w по переменной z , получим комплексную скорость

$$\frac{dw}{dz} = u - jv$$

где u и v — декартовы компоненты вектора скорости.

Комплексный потенциал описывает вихрь интенсивности Γ , расположенный в начале координат

$$w = \frac{j\Gamma}{2\pi} \ln z$$

Почему?

Так как $z = re^{j\theta}$ и w преобразуется в

$$w = -\frac{\Gamma}{2\pi}\theta + j\frac{\Gamma}{2\pi} \ln r = \phi + j\psi$$

Рассмотрим второй вихрь, расположенный в точке $(a, 0)$ с той же интенсивностью Γ . Его комплексный потенциал:

$$w = \frac{j\Gamma}{2\pi} \ln(z - a)$$

Следующий вихрь, расположенный ещё на a дальше, будет иметь потенциал

$$w = \frac{j\Gamma}{2\pi} \ln(z - 2a)$$

и так далее...

Таким образом, комплексный потенциал, соответствующий бесконечной цепочке вихрей (на линии $y = 0$), задается

$$w = \frac{j\Gamma}{2\pi} \sum_{m=-\infty}^{+\infty} \ln(z - ma)$$

Когда в предыдущих блокнотах мы интегрировали компоненты скорости, чтобы найти функцию тока и потенциал, то отбрасывали константу интегрирования. А теперь, наоборот, добавим константу, но не произвольную! Почему бы и нет, ведь обратное дифференцирование даст те же результаты.

$$w = \frac{j\Gamma}{2\pi} \sum_{m=-\infty}^{+\infty} \ln(z - ma) + \text{constant}$$

где

$$\text{constant} = -\frac{j\Gamma}{2\pi} \sum_{m=-\infty, m \neq 0}^{+\infty} \ln(-ma)$$

так что, комплексный потенциал можно представить в следующем виде

$$w = \frac{j\Gamma}{2\pi} \sum_{m=-\infty, m \neq 0}^{+\infty} \ln\left(\frac{z - ma}{-ma}\right) + \frac{j\Gamma}{2\pi} \ln z$$

Теперь пришло время поупражняться в математике.

$$\begin{aligned}
w &= \frac{j\Gamma}{2\pi} \sum_{m=-\infty, m \neq 0}^{+\infty} \ln \left(1 - \frac{z}{ma}\right) + \frac{j\Gamma}{2\pi} \ln z \\
w &= \frac{j\Gamma}{2\pi} \sum_{m=1}^{+\infty} \left\{ \ln \left(1 - \frac{z}{ma}\right) + \ln \left(1 + \frac{z}{ma}\right) \right\} + \frac{j\Gamma}{2\pi} \ln z \\
w &= \frac{j\Gamma}{2\pi} \sum_{m=1}^{+\infty} \ln \left(1 - \frac{z^2}{m^2 a^2}\right) + \frac{j\Gamma}{2\pi} \ln z \\
w &= \frac{j\Gamma}{2\pi} \ln \left(\prod_{m=1}^{+\infty} \left(1 - \frac{z^2}{m^2 a^2}\right) \right) + \frac{j\Gamma}{2\pi} \ln z \\
w &= \frac{j\Gamma}{2\pi} \ln \left(z \prod_{m=1}^{+\infty} \left(1 - \frac{z^2}{m^2 a^2}\right) \right) \\
w &= \frac{j\Gamma}{2\pi} \ln \left(z \prod_{m=1}^{+\infty} \left(1 - \frac{\left(\frac{z\pi}{a}\right)^2}{m^2 \pi^2}\right) \right)
\end{aligned}$$

Это произведение есть не что иное, как представление синуса. Тогда комплексный потенциал превращается в

$$w = \frac{j\Gamma}{2\pi} \ln \left(\sin \left(\frac{z\pi}{a} \right) \right)$$

Теперь дифференцируем его по комплексной переменной и получим скорости:

$$\begin{aligned}
\frac{dw}{dz} &= u - iv = \frac{j\Gamma}{2a} \cot \left(\frac{z\pi}{a} \right) \\
u - jv &= \frac{j\Gamma}{2a} \frac{\cos \left(\frac{\pi x}{a} + j \frac{\pi y}{a} \right)}{\sin \left(\frac{\pi x}{a} + j \frac{\pi y}{a} \right)}
\end{aligned}$$

Применяя тригонометрические тождества, найдем следующее выражение

$$u - jv = \frac{j\Gamma}{2a} \frac{\cos \left(\frac{\pi x}{a} \right) \cosh \left(\frac{\pi y}{a} \right) - j \sin \left(\frac{\pi x}{a} \right) \sinh \left(\frac{\pi y}{a} \right)}{\sin \left(\frac{\pi x}{a} \right) \cosh \left(\frac{\pi y}{a} \right) + j \cos \left(\frac{\pi x}{a} \right) \sinh \left(\frac{\pi y}{a} \right)},$$

которое может быть представлено в виде

$$u - jv = \frac{\Gamma}{2a} \frac{\sinh \left(\frac{2\pi y}{a} \right)}{\cosh \left(\frac{2\pi y}{a} \right) - \cos \left(\frac{2\pi x}{a} \right)} + j \frac{\Gamma}{2a} \frac{\sin \left(\frac{2\pi x}{a} \right)}{\cosh \left(\frac{2\pi y}{a} \right) - \cos \left(\frac{2\pi x}{a} \right)}$$

Таким образом декартовы компоненты скорости бесконечной цепочки вихрей задаются выражениями

$$\begin{aligned}
u(x, y) &= + \frac{\Gamma}{2a} \frac{\sinh \left(\frac{2\pi y}{a} \right)}{\cosh \left(\frac{2\pi y}{a} \right) - \cos \left(\frac{2\pi x}{a} \right)} \\
v(x, y) &= - \frac{\Gamma}{2a} \frac{\sin \left(\frac{2\pi x}{a} \right)}{\cosh \left(\frac{2\pi y}{a} \right) - \cos \left(\frac{2\pi x}{a} \right)}
\end{aligned}$$

Задача

Вычислить поле скорости и нарисовать линии тока для бесконечной цепочки вихрей
Теперь, когда мы вывели соотношения для компонент скорости, запрограммируйте их и нарисуйте линии тока.

Можете ли вы заметить разницу с предыдущим случаем, когда количество вихрей было конечным?

Попробуйте поменять настройки для картины течения вокруг конечного числа вихрей так, чтобы получилась картинка, похожая на бесконечный случай. Когда можно сказать, что конечный случай является хорошим приближением бесконечного?

Подумайте Обратите внимание, что линии тока параллельны *вихревому слою*: отсутствует течение по нормали. Значит, с помощью такого слоя можно моделировать твердую поверхность в потенциальном потоке. Мы получили скорость скольжения на поверхности: как это согласуется с потенциальным течением?

6 Подъёмная сила цилиндра

Помните, мы моделировали обтекание [диполя](#)? И у нас получилась картина линий тока, как при обтекании цилиндра. Проанализировав распределение коэффициента давления, мы увидели, что сопротивление цилиндра равно нулю, что привело нас к парадоксу д'Аламбера.

А что с подъёмной силой? Действует ли она на цилиндр круглого сечения? А если такой цилиндр вращается? Вы слышали об эффекте Магнуса?

Возможно вы удивитесь, узнав, что нам всего навсего нужно добавить [вихрь](#) в центр цилиндра. Давайте посмотрим, как это выглядит.

Для начала вспомним уравнения для течения с диполем. В декартовых координатах, диполь, расположенный в начале координат имеет следующие функцию тока и компоненты скорости:

$$\begin{aligned}\psi(x, y) &= -\frac{\kappa}{2\pi} \frac{y}{x^2 + y^2} \\ u(x, y) &= \frac{\partial \psi}{\partial y} = -\frac{\kappa}{2\pi} \frac{x^2 - y^2}{(x^2 + y^2)^2} \\ v(x, y) &= -\frac{\partial \psi}{\partial x} = -\frac{\kappa}{2\pi} \frac{2xy}{(x^2 + y^2)^2}\end{aligned}$$

6.1 Приступим к расчётом!

Поместим диполь интенсивностью $\kappa = 1$ в начало координат и добавим набегающий поток со скоростью $U_\infty = 1$ (да, единицы — это круто). Можно повторно использовать код, написанный нами ранее — это удачная мысль.

```
In [1]: import math
import numpy
from matplotlib import pyplot
# помещаем графику внутрь блокнота
%matplotlib inline

In [2]: N = 50
         # Число точек в каждом направлении
x_start, x_end = -2.0, 2.0
         # граница по x
y_start, y_end = -1.0, 1.0
         # граница по y
x = numpy.linspace(x_start, x_end, N)
         # вычисляем одномерный массив x
y = numpy.linspace(y_start, y_end, N)
         # вычисляем одномерный массив y
X, Y = numpy.meshgrid(x, y)
         # создаём расчётную сетку

In [3]: kappa = 1.0
         # интенсивность диполя
x_doublet, y_doublet = 0.0, 0.0
         # положение диполя

u_inf = 1.0
         # скорость набегающего потока
```

Вот, как мы определяли функции для диполя:

```
In [4]: def get_velocity_doublet(strength, xd, yd, X, Y):
        """
        Returns the velocity field generated by a doublet.

        Parameters
        -----
        strength: float
            Strength of the doublet.
        xd: float
            x-coordinate of the doublet.
        yd: float
            y-coordinate of the doublet.
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
```

```

Y: 2D Numpy array of floats
    y-coordinate of the mesh points.

>Returns
-----
u: 2D Numpy array of floats
    x-component of the velocity vector field.
v: 2D Numpy array of floats
    y-component of the velocity vector field.
"""
u = - strength/(2*math.pi)*((X-xd)**2-(Y-yd)**2)/\
((X-xd)**2+(Y-yd)**2)**2
v = - strength/(2*math.pi)*2*(X-xd)*(Y-yd)/\
((X-xd)**2+(Y-yd)**2)**2

return u, v

def get_stream_function_doublet(strength, xd, yd, X, Y):
    """
>Returns the stream-function generated by a doublet.

>Parameters
-----
strength: float
    Strength of the doublet.
xd: float
    x-coordinate of the doublet.
yd: float
    y-coordinate of the doublet.
X: 2D Numpy array of floats
    x-coordinate of the mesh points.
Y: 2D Numpy array of floats
    y-coordinate of the mesh points.

>Returns
-----
psi: 2D Numpy array of floats
    The stream-function.
"""
psi = - strength/(2*math.pi)*(Y-yd)/((X-xd)**2+(Y-yd)**2)

return psi

```

А теперь добавим набегающий поток к диполю и вычислим всё, что нужно, чтобы получить обтекание цилиндра:

```

In [5]: # рассчитываем поле скоростей в узлах расчётной сетки
        u_doublet, v_doublet = get_velocity_doublet(kappa, x_doublet, y_doublet,\n
                                                       X, Y)

        # рассчитываем функцию тока в узлах расчётной сетки
        psi_doublet = get_stream_function_doublet(kappa, x_doublet, y_doublet,\n
                                                       X, Y)

        # компоненты скорости невозмущённого потока
        u_freestream = u_inf * numpy.ones((N, N), dtype=float)
        v_freestream = numpy.zeros((N, N), dtype=float)

        # функция тока равномерного течения

```

```

psi_freestream = u_inf * Y

# суперпозиция диполя и равномерного потока
u = u_freestream + u_doublet
v = v_freestream + v_doublet
psi = psi_freestream + psi_doublet

```

Теперь всё готово для визуализации течения.

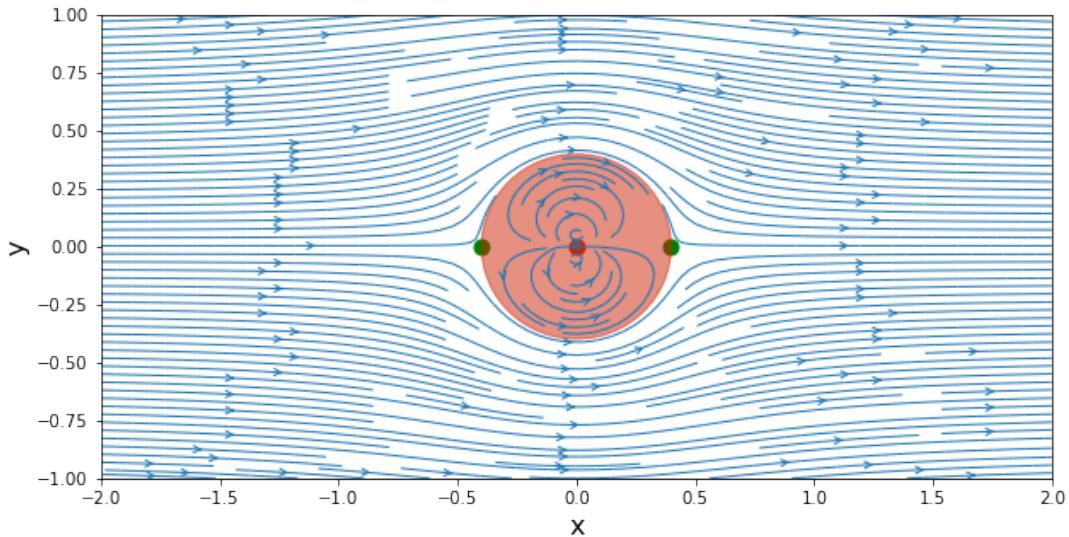
```

In [6]: # рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowstyle='->')
pyplot.scatter(x_doublet, y_doublet, color='#CD2305', s=80, marker='o')

# вычисляем радиус цилиндра и рисуем окружность с такими радиусом
R = math.sqrt(kappa/(2*math.pi*u_inf))
circle = pyplot.Circle((0, 0), radius=R, color='#CD2305', alpha=0.5)
pyplot.gca().add_patch(circle)

# вычисляем положение точек торможения и добавляем их на рисунок
x_stagn1, y_stagn1 = +math.sqrt(kappa/(2*math.pi*u_inf)), 0
x_stagn2, y_stagn2 = -math.sqrt(kappa/(2*math.pi*u_inf)), 0
pyplot.scatter([x_stagn1, x_stagn2], [y_stagn1, y_stagn2], color='g', \
s=80, marker='o');

```



Прекрасно! Теперь у нас есть течение вокруг цилиндра.

Теперь добавим вихрь с положительной интенсивностью Γ , расположенный в начале координат. В декартовых координатах функция тока и компоненты скорости задаются:

$$\psi(x, y) = \frac{\Gamma}{4\pi} \ln(x^2 + y^2)$$

$$u(x, y) = \frac{\Gamma}{2\pi} \frac{y}{x^2 + y^2} \quad v(x, y) = -\frac{\Gamma}{2\pi} \frac{x}{x^2 + y^2}$$

Исходя из этих уравнений определим функции `get_velocity_vortex()` и `get_stream_function_vortex()`, которые ... ну понятно из их названий, что они делают. Вообще, это хорошая идея — давать имена переменным, классам и функциям так, чтобы было понятно, для чего они нужны. Самодокументируемый код. Поэкспериментируйте со значением Γ и перестройте картину течения. Посмотрите, что происходит.

```
In [7]: gamma = 4.0                      # интенсивность вихря
        xv_vortex, yv_vortex = 0.0, 0.0      # положение вихря

In [8]: def get_velocity_vortex(strength, xv, yv, X, Y):
        """
        Returns the velocity field generated by a vortex.

        Parameters
        -----
        strength: float
            Strength of the vortex.
        xv: float
            x-coordinate of the vortex.
        yv: float
            y-coordinate of the vortex.
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.

        Returns
        -----
        u: 2D Numpy array of floats
            x-component of the velocity vector field.
        v: 2D Numpy array of floats
            y-component of the velocity vector field.
        """
        u = + strength/(2*math.pi)*(Y-yv)/((X-xv)**2+(Y-yv)**2)
        v = - strength/(2*math.pi)*(X-xv)/((X-xv)**2+(Y-yv)**2)
        return u, v

def get_stream_function_vortex(strength, xv, yv, X, Y):
        """
        Returns the stream-function generated by a vortex.

        Parameters
        -----
        strength: float
            Strength of the vortex.
        xv: float
            x-coordinate of the vortex.
        yv: float
            y-coordinate of the vortex.
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.

        Returns
        -----
        psi: 2D Numpy array of floats
            The stream-function.
        """

```

```

psi = strength/(4*math.pi)*numpy.log((X-xv)**2+(Y-yv)**2)

return psi

```

```

In [9]: # рассчитываем поле скоростей в узлах расчётной сетки
u_vortex, v_vortex = get_velocity_vortex(gamma, x_vortex, y_vortex, X, Y)

# рассчитываем функцию тока в узлах расчётной сетки
psi_vortex = get_stream_function_vortex(gamma, x_vortex, y_vortex, X, Y)

```

Теперь, когда у нас есть все необходимые ингредиенты (равномерный поток, диполь и вихрь), мы применим принцип суперпозиции, а затем сделаем красивую картинку.

```

In [24]: # суперпозиция диполя, вихря и набегающего потока
u = u_freestream + u_doublet + u_vortex
v = v_freestream + v_doublet + v_vortex
psi = psi_freestream + psi_doublet + psi_vortex

```

```

In [10]: # вычисляем радиус цилиндра
R = math.sqrt(kappa/(2*math.pi*u_inf))

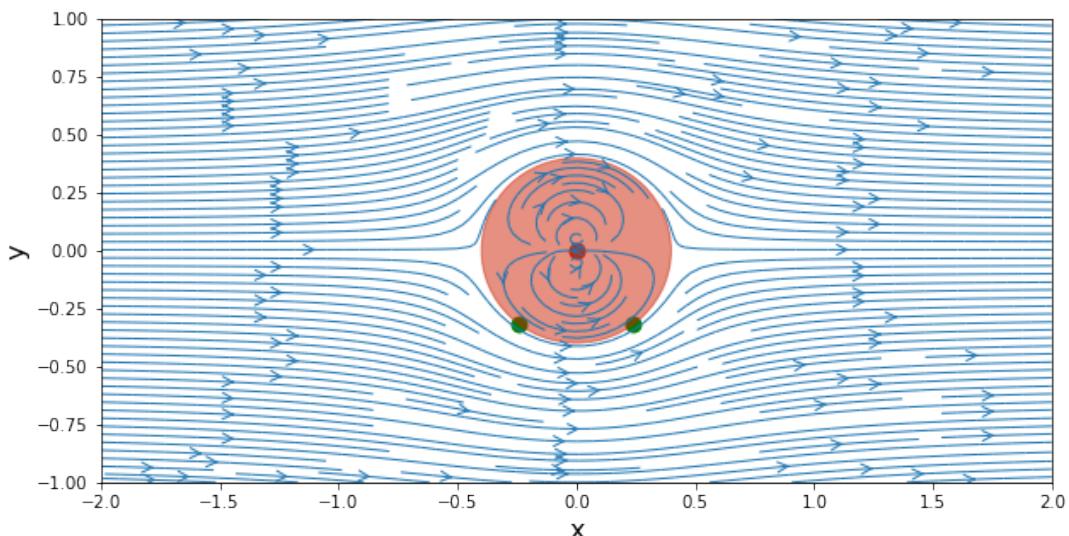
# определяем координаты точек торможения
x_stagn1, y_stagn1 = +math.sqrt(R**2-(gamma/(4*math.pi*u_inf))**2), \
-gamma/(4*math.pi*u_inf)
x_stagn2, y_stagn2 = -math.sqrt(R**2-(gamma/(4*math.pi*u_inf))**2), \
-gamma/(4*math.pi*u_inf)

```

```

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1.5,\n
arrowstyle='->')
circle = pyplot.Circle((0, 0), radius=R, color='#CD2305', alpha=0.5)
pyplot.gca().add_patch(circle)
pyplot.scatter(x_vortex, y_vortex, color='#CD2305', s=80, marker='o')
pyplot.scatter([x_stagn1, x_stagn2], [y_stagn1, y_stagn2], color='g',\n
s=80, marker='o');

```



Контрольная задача В блокноте третьего занятия, посвященного [диполю](#), вам нужно было вычислить радиус цилиндра, создаваемого диполем в равномерном потоке. У вас должно было получиться

$$R = \sqrt{\frac{\kappa}{2\pi U_\infty}}$$

Новое задание — определите, где на поверхности цилиндра находятся точки торможения потока при наличии вихря.

Что произойдет, если $\frac{\Gamma}{4\pi U_\infty R} > 1$?

Вернитесь назад и поэкспериментируйте со значениями Γ , при которых реализуется такой случай.

6.2 Коэффициент давления

Давайте получим коэффициент давления на поверхности цилиндра и сравним со случаем, когда не было вихря. Компоненты скорости в полярных координатах для комбинации набегающий поток + диполь + вихрь:

$$u_r(r, \theta) = U_\infty \cos \theta \left(1 - \frac{R^2}{r^2}\right)$$

$$u_\theta(r, \theta) = -U_\infty \sin \theta \left(1 + \frac{R^2}{r^2}\right) - \frac{\Gamma}{2\pi r}$$

где R — радиус цилиндра.

Видно, что радиальная компонента на поверхности цилиндра исчезает, тогда как азимутальная скорость равна

$$u_\theta(R, \theta) = -2U_\infty \sin \theta - \frac{\Gamma}{2\pi R}.$$

Заметим, что без вихря азимутальная скорость на поверхности цилиндра получается

$$u_\theta(R, \theta) = -2U_\infty \sin \theta.$$

Из блокнота про диполь мы знаем, что коэффициент давления определяется как

$$C_p = 1 - \frac{U^2}{U_\infty^2}$$

где $U^2 = u^2 + v^2 = u_r^2 + u_\theta^2$.

Давайте изобразим это на графике!

```
In [11]: # вычисляем касательные компоненты скорости на поверхности цилиндра
theta = numpy.linspace(0, 2*math.pi, 100)
u_theta = -2*u_inf*numpy.sin(theta) - gamma/(2*math.pi*R)

# вычисляем коэффициент давления на поверхности цилиндра
cp = 1.0 - (u_theta/u_inf)**2

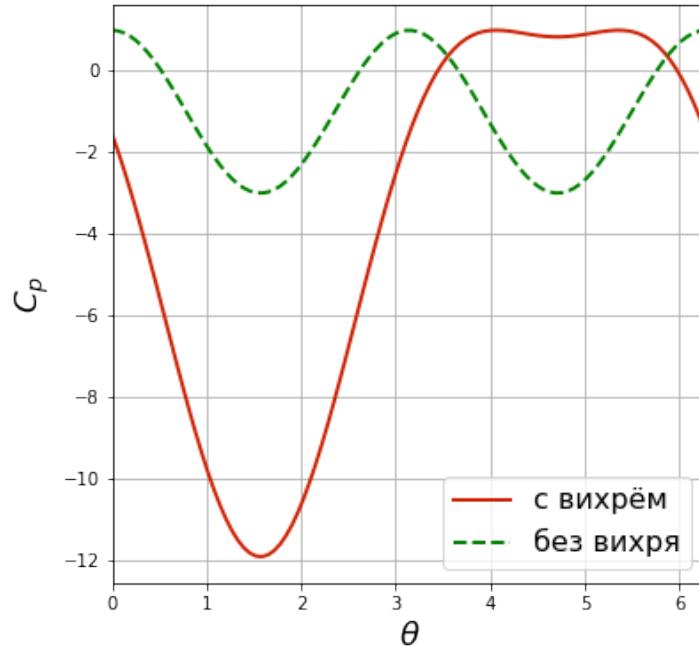
# без вихря
u_theta_no_vortex = -2*u_inf*numpy.sin(theta)
cp_no_vortex = 1.0 - (u_theta_no_vortex/u_inf)**2

# рисуем распределение коэффициента давления
size = 6
pyplot.figure(figsize=(size, size))
```

```

pyplot.grid(True)
pyplot.xlabel(r'$\theta$', fontsize=18)
pyplot.ylabel('$C_p$', fontsize=18)
pyplot.xlim(theta.min(), theta.max())
pyplot.plot(theta, cp, label='с вихрём', color="#CD2305", \
            linewidth=2, linestyle='solid')
pyplot.plot(theta, cp_no_vortex, label='без вихря', color='g', \
            linewidth=2, linestyle='dashed')
pyplot.legend(loc='best', prop={'size':16});

```



6.3 Подъёмная сила и сопротивление

Подъёмная сила — это компонента силы, действующей на тело со стороны потока, перпендикулярная скорости U_∞ , а сопротивление — компонента, параллельная скорости. Как вычислить их, исходя из полученных данных?

Сила, действующая на цилиндр — интеграл давления, действующего на его поверхность (должна быть еще вязкость, но её нет, так как мы рассматриваем идеальную жидкость). Если нарисовать вектора силы, действующей на цилиндр, и скорости, то станет ясно, что

$$D = - \int_0^{2\pi} p \cos \theta R d\theta$$

$$L = - \int_0^{2\pi} p \sin \theta R d\theta$$

Контрольная задача Используя уравнение Бернуlli, замените p в уравнениях выше, чтобы получить подъёмную силу и сопротивление.

Что это значит?

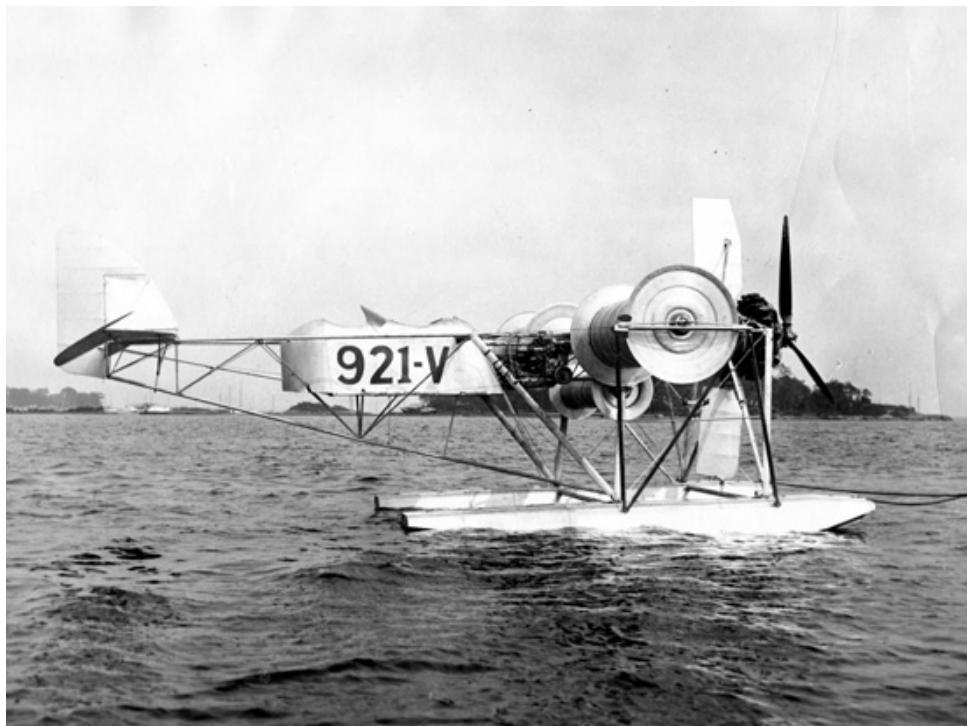
6.4 Эффект Магнуса

Возникновение силы при обтекании вращающегося цилиндра (а также сферы, или любого другого объекта) известно как *эффект Магнуса*.

Хотите верьте, хотите нет, но на основе этого эффекта предпринимались попытки создания самолетов с вращающимся цилиндром вместо крыла. Согласно статье в [PilotFriend](#), в 1930 был создан летательный аппарат 921-V, который «взлетел как минимум один раз».

```
In [27]: from IPython.display import Image  
Image(url='http://upload.wikimedia.org/wikipedia/commons/7/78/Flettner_Rotor_Aircraft.jpg')
```

Out[27] :



Да и сейчас любители радиоуправляемых моделей собирают просмотры на Ютубе, развлекаясь с моделями «rotorwings» на основе эффекта Магнуса.

```
In [28]: from IPython.display import YouTubeVideo  
YouTubeVideo('POHre1P_E1k')
```

Out[28] :



Существуют и более сложные технические решения, основанные на этом эффекте. В первую очередь это система Турбопарус, разработанная специалистами фонда Кусто. Такая силовая установка используется на судне «Алсион». Утверждается, что использование этой системы позволяет сэкономить до 30% топлива!



Преобразование Жуковского

Это задание ко второму модулю «**Потенциальные вихри и подъёмная сила**» курса [AeroPython](#). Первый модуль курса, «Строительные элементы потенциальных течений», который состоял из трех блокнотов, закончился рассмотрением потенциального обтекания двумерного цилиндра, полученного путем суперпозиции диполя и равномерного потока. А во втором модуле вы узнали, что добавление еще одной особенности – вихря, позволяет получить цилиндр, на который действует подъемная сила. Может возникнуть вопрос: *есть ли польза от этих знаний?*

Да, есть. И вот как они станут полезными. Используя простые методы ТФКП, мы получим из обтекания цилиндра течение вокруг профиля. Фокус в том, чтобы использовать конформное *отображение* (комплексную функцию, сохраняющую углы) для перехода из плоскости цилиндра в плоскость профиля.

Давайте исследуем этот классический подход!

Введение

Вы узнали, как получить потенциальное обтекание цилиндра путем суперпозиции равномерного течения и [диполя](#). Еще вы узнали, что при добавлении вихря возникает [подъемная сила](#). Ну и что с того? Почему это так важно? Что нам делать с потенциальным течением вокруг цилиндра?

В те времена, когда еще не было компьютеров, учёные-аэродинамики и математики использовали мощный инструмент — ТФКП — для изучения потенциальных течений, не решая напрямую системы уравнений в частных производных. Пользуясь им и зная решение для потенциального обтекания цилиндра, они с легкостью получали решения для множества различных внешних потенциальных течений, включая обтекание нескольких видов профилей.

Однако сегодня мы больше не пользуемся этим волшебным инструментом. Тем не менее, по-прежнему полезно знать основной принцип, стоящий за этим волшеством — конформные отображения. В этом задании мы пройдем все шаги в процедуре получения потенциального обтекания профиля из течения вокруг цилиндра при помощи знаменитого конформного преобразования — *функции Жуковского*. И вам станет понятно, какое важное значение в истории аэродинамики у потенциального обтекания цилиндра!

Не волнуйтесь. Математики будет немного. И вам, в отличие от пионеров аэродинамики, не придется ничего вычислять вручную. Просто следуйте инструкциям, а Python сделает всю тяжелую работу за вас.

Комплексные числа в Python

Начнем с задания двух комплексных плоскостей: в одной заданы точки $z = x + iy$, в другой — точки $\xi = \xi_x + i\xi_y$. Функция Жуковского переводит точку из плоскости z в точку на плоскости ξ :

$$\xi = z + \frac{c^2}{z} \quad (7)$$

где c — константа. Прежде чем перейти к обсуждению формулы Жуковского, попрактикуемся немного с комплексными числами в Python.

Если использовать комплексные числа, то формула Жуковского приобретает простой вид, при этом отпадает необходимость вычислять действительную и мнимую части по отдельности.

Питон, а следовательно и NumPy, умеет работать с комплексными числами, что называется, из коробки. Мнимая единица $i = \sqrt{-1}$ обозначается символом `j`, а не `i`, чтобы не возникало путаницы с итерационной переменной `i`.

Если вы впервые сталкиваетесь с комплексными переменными, попробуйте выполнить несколько простых операций. Например, введите в следующую ячейку код:

```
3 + 2j
```

А теперь попробуйте так:

```
a = 3
b = 3
z = a + b * 1j
print('z = ', z)
print('The type of the variable is ', type(z))
```

Упражнения

Познакомьтесь с комплексными операциями в Python и ответьте на следующие вопросы:

1. $(2.75 + 3.69i) \times (8.55 - 6.13i) =$
2. $1.4 \times e^{i5.32} =$
3. $\frac{7.51-9.15i}{4.43+9.64i} =$

Фигуры, созданные при помощи формулы Жуковского

Начните с написания функции, которая принимает z и c в качестве параметров и возвращает преобразованное значение ξ .

При помощи такого преобразования можно получить несколько типов кривых. Используйте только что созданную функцию, чтобы проделать вычисления, описанные ниже, и ответьте на вопросы.

Для простоты предположим, что $c = 1$.

1. На плоскости z расположите окружность радиусом R больше $c = 1$, скажем $R = 1.5$, с центром в начале координат. Какой формы получится отображение этой окружности на плоскость ξ ?
 1. окружность
 2. эллипс
 3. симметричный профиль
 4. несимметричный профиль
2. Теперь поместите на плоскости z окружность с центром в точке $(x_c, y_c) = (c - R, 0)$, радиус которой $c < R < 2c$ (например, $c = 1; R = 1.2$) Какой формы теперь получится отображение на плоскость ξ ?
 1. окружность
 2. эллипс
 3. симметричный профиль
 4. несимметричный профиль
3. Поместите центр окружности в точку $(x_c, y_c) = (-\Delta x, \Delta y)$, где Δx и Δy — небольшие положительные значения, например, $\Delta x = 0.1$ и $\Delta y = 0.1$. Радиус окружности задайте как $R = \sqrt{(c - x_c)^2 + y_c^2}$. Что получилось на плоскости ξ ?
 1. окружность
 2. эллипс
 3. симметричный профиль
 4. несимметричный профиль
4. Рассмотрим случай симметричного профиля. В полярных координатах $(\theta, r = R)$, какая точка на окружности соответствует задней кромке профиля?
 - $\theta = ?$

Расчётная сетка в плоскости z в полярной системе координат

Формула Жуковского ставит в соответствие точке на плоскости z точку на плоскости ξ . Как вы видели в предыдущем разделе, такое преобразование иногда дает фигуры, сильно похожие на аэродинамические профили. *Ну и что?*

Оказывается, согласно ТФКП, если применить [конформное отображение](#) к решению уравнения Лапласа, то получившаяся функция тоже будет решением уравнения Лапласа.

Значит, можно отобразить потенциал и функцию тока для обтекания цилиндра из плоскости z в ξ и получить течение вокруг профиля. Функция тока для обтекания профиля задается уравнением

$$\psi(\xi_x, \xi_y) = \psi(\xi_x(x, y), \xi_y(x, y))$$

в котором комплексные координаты ξ , ξ_x и ξ_y получены путем преобразования по формуле Жуковского из $z = x + iy$.

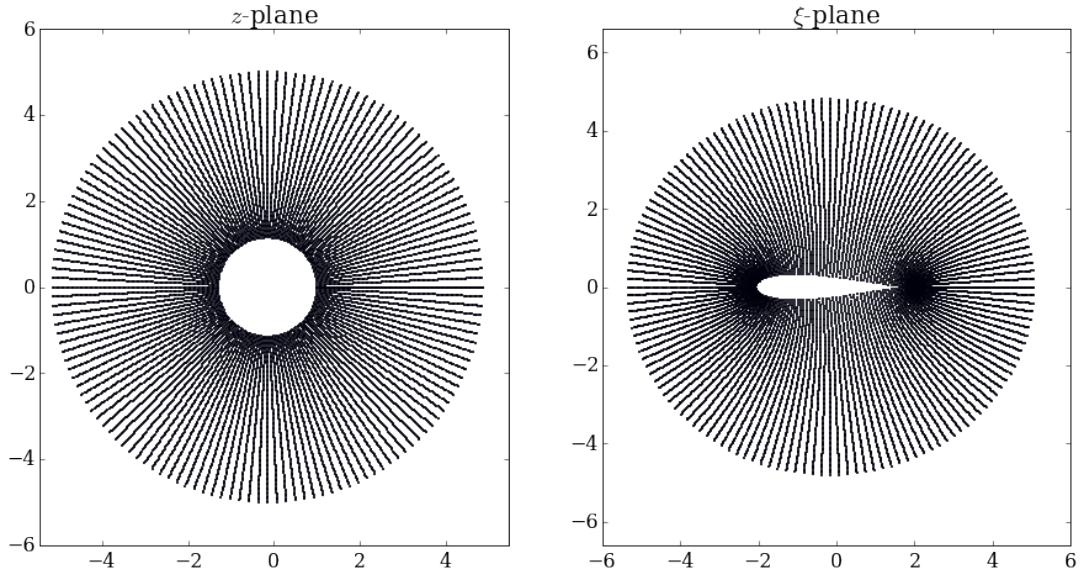
Выполнив это задание, вы получите обтекание симметричного профиля Жуковского под нулевым и ненулевым углами атаки. Форму профиля можно получить, задав в плоскости z окружность с центром в точке $(x_c, y_c) = (-0.15, 0)$ и радиусом $R = 1.15$ (параметр $c = 1$). Для того, чтобы достичь поставленной цели, нужно последовательно решить задачи, описанные в этом и последующих разделах.

Сначала построим расчётную сетку в плоскости z и посмотрим, как её узлы будут выглядеть на плоскости ξ после преобразования. Используйте в плоскости z полярные координаты. Если поместить узлы расчётной сетки внутрь окружности, то после отображения они окажутся снаружи профиля (убедитесь в этом сами!) Это, вроде бы, проблема. С другой стороны, на окружности ставится граничное условие непротекания, поэтому линии тока внутри окружности нет, и эту область можно просто игнорировать.

Упражнения

Постройте расчётную сетку в плоскости z в полярных координатах и при помощи формулы Жуковского отразите её на плоскость ξ .

Поместите $N_r = 100$ узлов сетки в радиальном направлении при $R \leq r \leq 5$ и $N_\theta = 145$ узлов в трансверсальном. В этом направлении θ меняется от 0 до 2π . Теперь, если нарисовать результат при помощи функции `pyplot.scatter()`, должно получиться что-то такое:



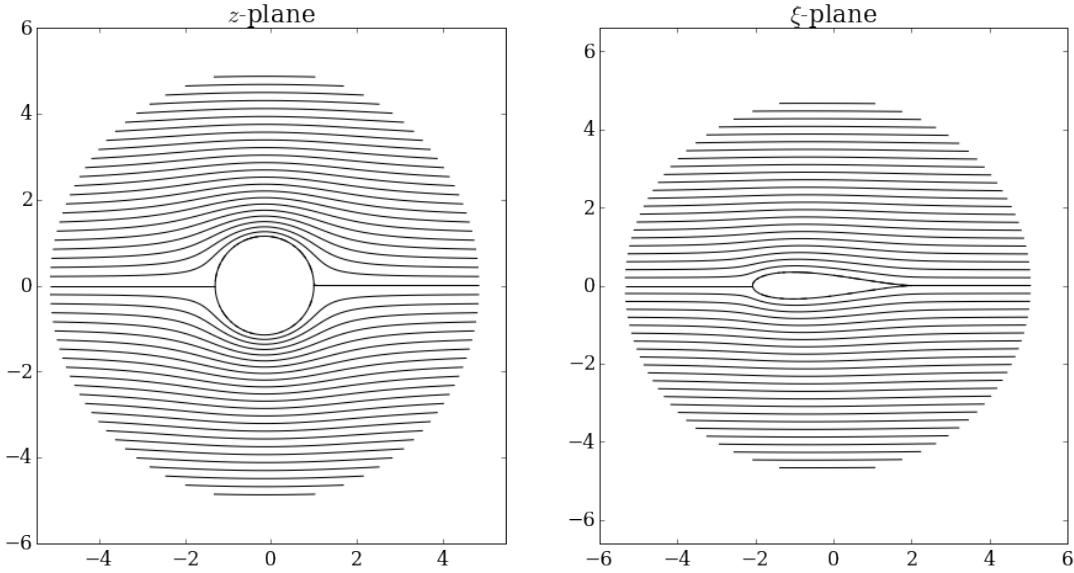
Обтекание симметричного профиля Жуковского под нулевым углом атаки

Функция и линии тока

Рассмотрим потенциальное обтекание цилиндра в плоскости z . Как отмечалось выше, $\psi(\xi) = \psi(\xi(z))$. Значит, значение функции тока в точке на плоскости z будет таким же, как и в соответствующей ей точке плоскости ξ . Мы можем построить линии тока в обеих плоскостях с помощью функции `pyplot.contour()`.

В качестве скорости на бесконечности используйте 1, т. е. $U_\infty = 1$. Для того чтобы получить цилиндр с радиусом $R = 1.15$, сначала нужно вычислить интенсивность диполя.

У вас должны получиться картины линий тока, похожие на те, что представлены на рисунках ниже.



Векторы скорости и коэффициент давления

Чтобы вычислить коэффициент давления, нам потребуется поле скорости. Поле скорости в плоскости z можно легко получить, используя координаты узлов расчётной сетки. Но можно ли утверждать, что скорости в соответствующих точках плоскости ξ будут такими же, как на плоскости z , по аналогии с функцией тока? *Правильный ответ: нет.*

Значения функции тока остаются неизменными в исходных и отображенных точках, поскольку функция потока является решением скалярного уравнения Лапласа!

Однако, скорость — это вектор и он не является решением уравнения Лапласа. Координаты вектора изменяются при смене системы координат путем конформного преобразования.

В нашем случае плоскости z и ξ будут двумя различными системами координат. И скорость в заданной точке плоскости z будет отличаться от скорости в соответствующей точке на плоскости ξ . Нужно выполнить некоторые манипуляции.

Скорости в плоскостях z и ξ записываются в следующем виде:

$$\begin{cases} u_z = \frac{\partial \psi}{\partial y} \\ v_z = -\frac{\partial \psi}{\partial x} \end{cases} \quad \text{и} \quad \begin{cases} u_\xi = \frac{\partial \psi}{\partial \xi_y} \\ v_\xi = -\frac{\partial \psi}{\partial \xi_x} \end{cases} \quad (8)$$

Как получить u_ξ и v_ξ , имея u_z и v_z ? Это можно сделать, используя правило дифференцирования сложной функции. Самое время вспомнить о функции потенциала ϕ , которая тоже является решением уравнения Лапласа. Таким образом, значения потенциала и функции тока остаются неизменными при конформном отображении. То же самое справедливо для комплексного потенциала $F(\xi) = F(\xi(z)) = \phi + i\psi$.

По правилу дифференцирования сложной функции $dF/d\xi = dF/dz \times dz/d\xi$. Таким образом,

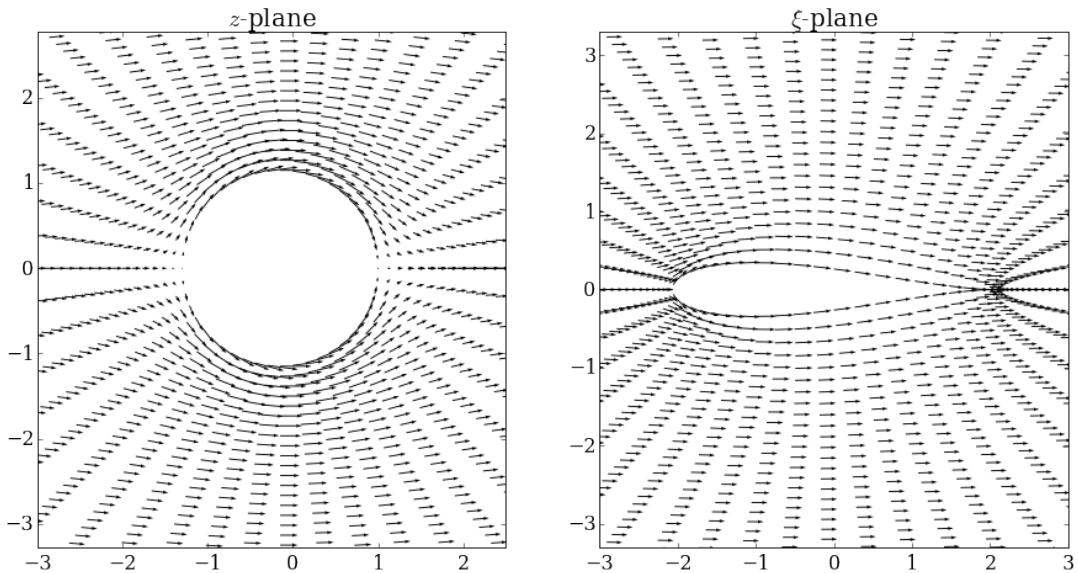
$$W_\xi = u_\xi - iv_\xi = \frac{dF}{d\xi} = \frac{dF}{dz} \times \frac{dz}{d\xi} = \frac{dF}{dz} / \frac{dz}{d\xi} = (u_z - iv_z) / \frac{dz}{d\xi} \quad (9)$$

и

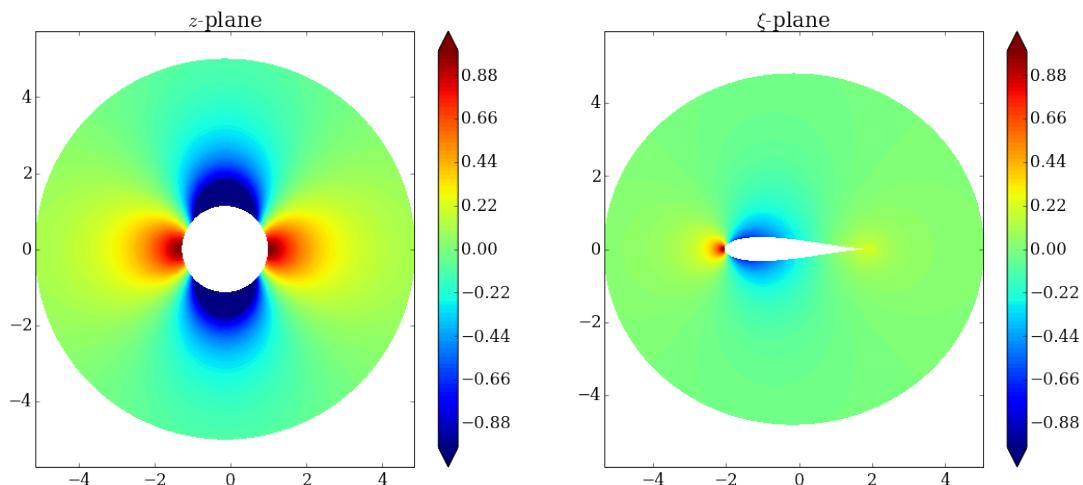
$$\frac{dz}{d\xi} = \frac{d(z + c^2/z)}{dz} = 1 - \left(\frac{c}{z}\right)^2 \quad (10)$$

Теперь, используя два полученных выше соотношения, можно получить скорости u_ξ и v_ξ на плоскости ξ .

Если воспользоваться функцией `pyplot.quiver()` для визуализации векторных полей, результат должен быть таким:



А поле коэффициента давления в расчётной области — так:



Упражнения

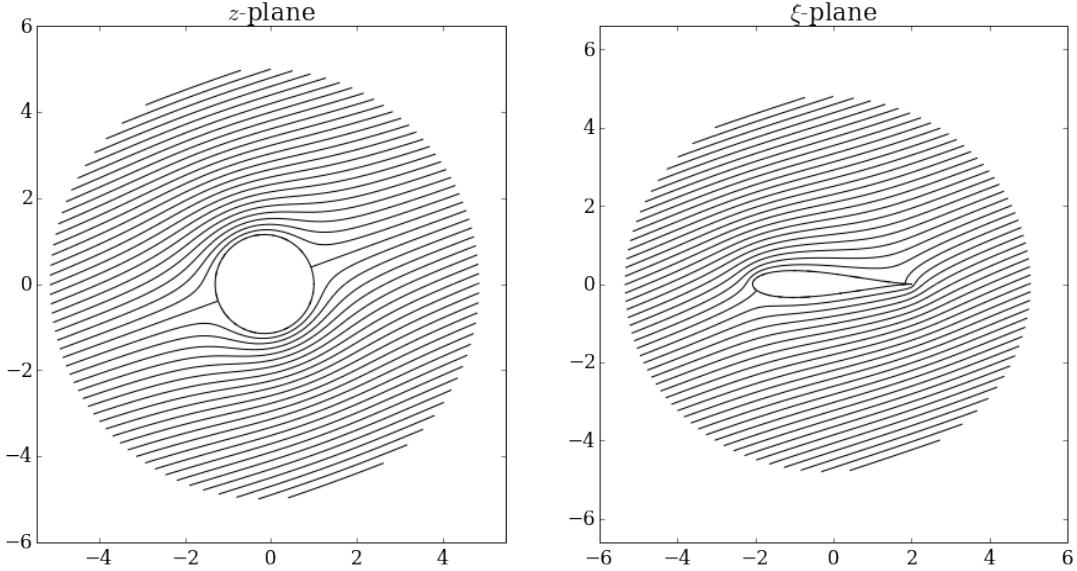
- Напишите код на Python для получения следующих картин: линий тока, векторов скорости и распределения коэффициента давления.
- Ответьте на следующие вопросы:
 1. Какова интенсивность диполя?
 2. Чему равна скорость в 62-й точке на поверхности профиля? Предположим, что задняя кромка имеет индекс 1 и что значения индекса возрастают при движении против часовой стрелки.
 3. Каково минимальное значение коэффициента давления на поверхности профиля?

Обтекание симметричного профиля Жуковского под ненулевым углом атаки без циркуляции

Теперь мы хотим расположить профиль под углом атаки (AoA — angle of attack) относительно набегающего потока. Конечно, для этого можно использовать преобразование Жуковского, применив его к обтеканию цилиндра. Но как это сделать? Можно ли достичь желаемого результата,

сложив безграничное течение и диполь? На самом деле, нет. Если пойти этим путем, нам не удастся выделить замкнутую линию тока, как мы делали это раньше — а такая замкнутая линия тока выступает в качестве поверхности цилиндра.

Для того, чтобы получить равномерный поток под углом атаки, нужно просто повернуть систему координат. Сначала создадим новую систему координат (или еще одну комплексную плоскость) z' с началом в центре цилиндра, в которой ось x' (действительная часть z') параллельна потоку, как показано на картинке.



Взаимосвязь между плоскостями z' и z :

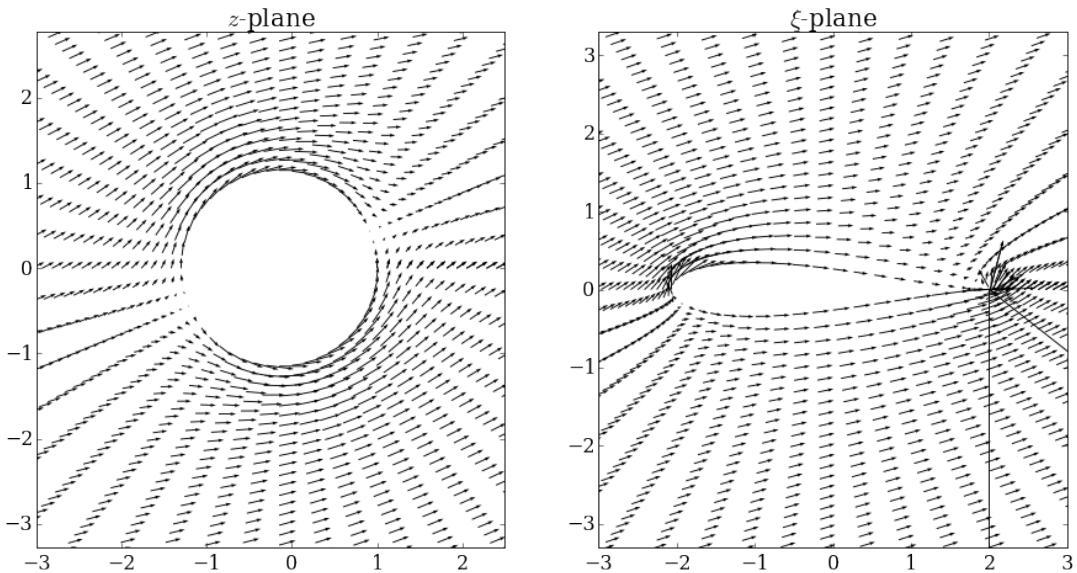
$$z' = [z - (x_c + iy_c)] e^{-i \times AoA} \quad (11)$$

Или для координат x , y , x' и y' :

$$\begin{cases} x' = (x - x_c) \cos(AoA) + (y - y_c) \sin(AoA) \\ y' = -(x - x_c) \sin(AoA) + (y - y_c) \cos(AoA) \end{cases} \quad (12)$$

где (x_c, y_c) — положение центра цилиндра, а AoA — угол атаки.

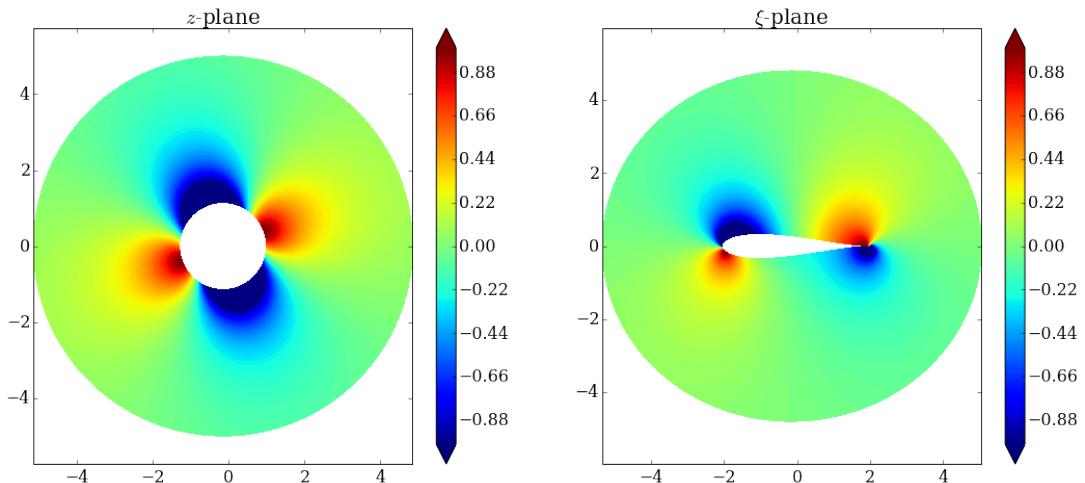
Теперь в новой плоскости z' можно получить обтекание цилиндра, сложив безграничный поток с *нулевым углом атаки* и диполь, расположенный в начале системы координат. Затем получим течение в плоскостях z и ξ . Опять же, функция потока остается той же самой в заданной точке во всех трёх различных системах координат (z' , z и ξ). В плоскостях z и ξ у вас должны получиться такие линии тока:



Векторы скоростей нужно развернуть обратно при переходе от плоскости z' к z .

$$u - iv = \frac{dF}{dz} = \frac{dF}{dz'} \times \frac{dz'}{dz} = (u' - iv')e^{-i \times AoA} \quad (13)$$

Конечно же, можно использовать явную запись для компонент x , y , x' и y' . Выведите соответствующие соотношения самостоятельно, если вам удобнее пользоваться явной формой записи. Получив скорости в плоскости z , вы можете воспользоваться навыками, приобретёнными при выполнении предыдущего упражнения, для того, чтобы выписать скорости в плоскости ξ . Итоговые поля скорости и коэффициента давления должны выглядеть следующим образом:



Упражнения

- Напишите код на Python, чтобы получить показанные выше картины. Используйте угол атаки $AoA = 20^\circ$.
- Ответьте на следующие вопросы:
 1. Как вы думаете, физично ли полученное решение? Обоснуйте свой ответ.
 2. Где на профиле расположены точки торможения? Предположим, что задняя кромка имеет индекс 1 и что значение индекса возрастает при движении против часовой стрелки.

3. Чему равна подъёмная сила?
4. Чему равен коэффициент сопротивления?
5. Чему равна скорость в 50-й точке на поверхности профиля?
6. Чему равен коэффициент давления в 75-й точке на поверхности профиля?

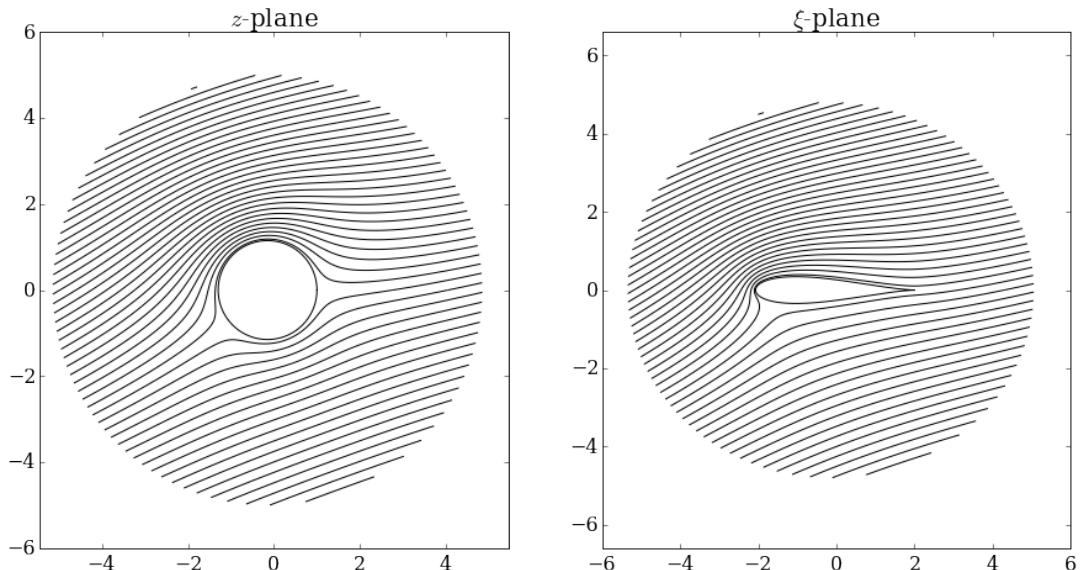
Обтекание симметричного профиля Жуковского под ненулевым углом атаки при наличии циркуляции

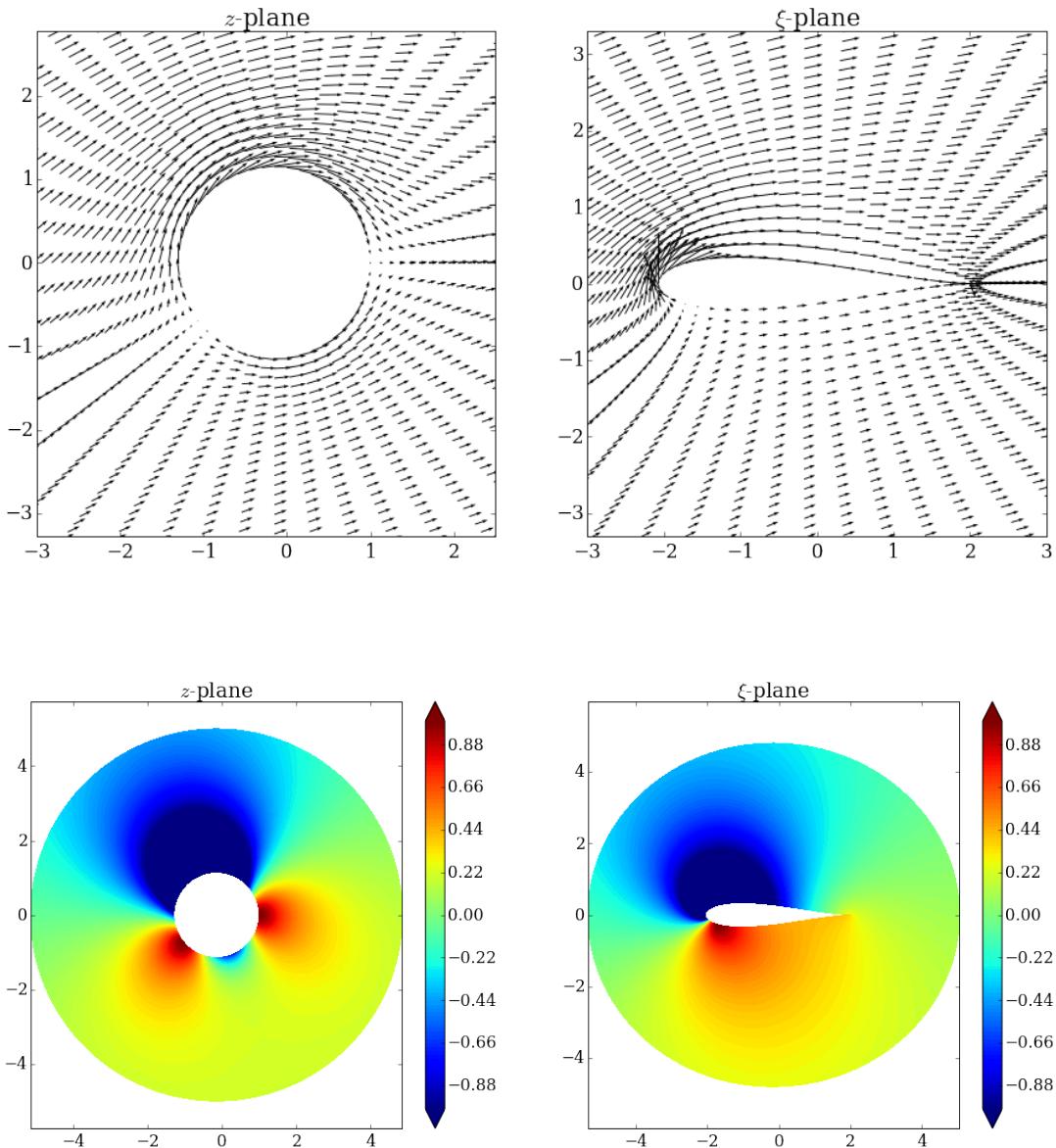
Результат, полученный в предыдущем упражнении, не имеет физического смысла. Нам нужен **вихрь**. Как известно из [Занятия 6: Подъёмная сила цилиндра](#), добавление вихря (другими словами циркуляции) к потенциальному обтеканию цилиндра приводит к изменению положения точек торможения, а также создает подъёмную силу.

Для того, чтобы сделать решение более физичным, нужно, чтобы выполнялось [условие Кутты-Жуковского](#), «Из всех возможных обтеканий крыла с задней острой кромкой в природе реализуется только то, в котором скорость в заднем острие конечна».

Это утверждение позволяет нам подобрать нужную интенсивность вихря. Она должна быть такой, чтобы задняя точка торможения на цилиндре переместилась из положения $\theta = \text{AoA}$ в положение $\theta = 0^\circ$ на плоскости z . Знаний, полученных на Занятии 6, должно быть достаточно для того, чтобы самостоятельно рассчитать необходимую интенсивность.

Линии тока, поля скоростей и коэффициента давления в плоскостях z и ξ должны выглядеть так:





Упражнения

- Напишите код на Python, чтобы получить показанные выше картины.
- Ответьте на следующие вопросы:
 1. Какова интенсивность вихря?
 2. Каково значение подъёмной силы? (Подсказка: подъёмная сила, как нам известно из Занятия 6, действует в направлении, нормальном к набегающему потоку, в нашем случае в направлении оси y' .)
 3. Попробуйте вычислить значения подъёмной силы и сопротивления напрямую, по формулам $L = -\oint p \times \sin \theta dA$ и $D = \oint p \times \cos \theta dA$. Удовлетворяет ли полученное значение подъёмной силы теореме Кутты–Жуковского? Каково значение коэффициента сопротивления?
 4. Где на профиле расположены точки торможения? Предположим, что задняя кромка имеет индекс 1 и что значение индекса возрастает при движении против часовой стрелки.
 5. Чему равна скорость в 92-й точке на поверхности профиля?
 6. Чему равен коэффициент давления в 111-й точке на поверхности профиля?
 7. Что происходит с коэффициентом давления на задней кромке профиля?

7 Метод зеркального отражения

На предыдущих занятиях курса *AeroPython* вы узнали, как при помощи комбинации элементарных потенциальных решений можно получить картины линий тока, описывающие течение вокруг некоторых объектов, например [овала Рэнкина](#) или [кругового цилиндра](#).

Вы спросите, а можно ли воспроизвести обтекание плоской поверхности? Да, можно!

Коротко, метод зеркального отражения заключается в следующем: поместим особенность рядом со стенкой, добавив *отражение* этой особенности по другой сторону от стенки. Иногда такой подход называют «аэродинамической интерференцией».

Кроме того, этот блокнот также дает нам возможность ввести понятие **классов** в Python. Это очень полезный способ организовать ваш код, который становится по-настоящему необходимым по мере того, как программы становятся сложнее.

Как обычно, мы начнём с импорта библиотек и создании расчётной сетки. И быстренько разберёмся со всем этим.

```
In [1]: import numpy
        import math
        from matplotlib import pyplot
        # помещаем графику внутрь блокнота
        %matplotlib inline

In [2]: N = 50                                # Число точек в каждом направлении
        x_start, x_end = -2.0, 2.0                 # граница по x
        y_start, y_end = -1.0, 1.0                 # граница по y
        x = numpy.linspace(x_start, x_end, N)      # вычисляем одномерный массив x
        y = numpy.linspace(y_start, y_end, N)      # вычисляем одномерный массив y
        X, Y = numpy.meshgrid(x, y)                # создаём расчётную сетку
```

7.1 Источник вблизи плоской поверхности

Если расположить источник вблизи поверхности, то эта поверхность внесёт возмущения в течение. Представим, что источник расположен в точке $y = y_{\text{source}}$ вблизи поверхности $y = 0$. Для выполнения граничного условия на поверхности нужно, чтобы поток был ей параллелен, для горизонтальной плоскости это означает, что на стенке $v = 0$. Стенка оказывает на течение от источника такое же воздействие, как и равный по интенсивности источник (*отражение*), расположенный в точке $y = -y_{\text{source}}$.

Теперь настало время стать хитрее и перестать множить код, тратя наше драгоценное время! В предыдущем блокноте мы уже ввели понятие функции в Python, а теперь пора двигаться дальше и познакомиться с **классами**.

Класс — это набор данных (параметры и переменные) и «методов» или функций, которые работают с этими данными. Это очень аккуратный способ организации кода. По мере создания больших объёмов и усложнения кода, это помогает нам справиться со сложностями. Код будет легче поддерживать, изменять и расширять.

Мы определим класс под названием `Source`, который будет содержать информацию, относящуюся к источнику. Такие особенности, как источник, определяются интенсивностью и положением в расчётной области. Поэтому у нашего класса `Source` будут три атрибута:

- `strength`: интенсивность источника.
- `x`: координата положения источника по горизонтали.
- `y`: координата положения источника по вертикали.

Что бы мы хотели сделать после определения нашего источника? Мы бы хотели вычислить поле скоростей, а также функцию тока. Таким образом, в нашем классе нужно реализовать два метода (функции, принадлежащие классу): для вычисления скорости (функция `velocity()`) и функции тока (функция `stream_function()`).

Давайте рассмотрим их подробнее:

- метод `velocity()`: это функция для расчёта скорости, порождаемой источником, в узлах сетки (`X,Y`). Поэтому у этого метода будет два аргумента, `X` и `Y`, и он будет создавать два новых атрибута источника: компоненты скорости `u` и `v`.

- метод `stream_function()`: это функция для вычисления функции тока порождаемой источником, в узлах сетки (X, Y). Поэтому у этого метода будет два аргумента, X и Y , и он будет создавать один новый атрибут источника: `psi`, функцию тока.

Помимо этих двух методов, у каждого класса есть *конструктор*. По сути, это способ инициализации данных. Это функция, которая всегда называется `__init__()`:

```
In [3]: class Source:
    """
    Contains information related to a source (or sink).
    """

    def __init__(self, strength, x, y):
        """
        Sets the location and strength of the singularity.

        Parameters
        -----
        strength: float
            Strength of the singularity.
        x: float
            x-coordinate of the singularity.
        y: float
            y-coordinate of the singularity.
        """

        self.strength = strength
        self.x, self.y = x, y

    def velocity(self, X, Y):
        """
        Computes the velocity field generated by the singularity.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """

        self.u = self.strength/(2*math.pi)*(X-self.x)/((X-self.x)**2+\n                                         (Y-self.y)**2)
        self.v = self.strength/(2*math.pi)*(Y-self.y)/((X-self.x)**2+\n                                         (Y-self.y)**2)

    def stream_function(self, X, Y):
        """
        Computes the stream-function generated by the singularity.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """

        self.psi = self.strength/(2*math.pi)*numpy.arctan2((Y-self.y), \
                                                          (X-self.x))
```

А для чего нужен параметр `self`? Создание объекта класса `Source`, выглядит как вызов функции. Например: `source = Source(1, 0, 0)` создаст источник интенсивностью 1, расположенный в

начале координат. Python автоматически вызывает функцию *конструктор* с параметрами (`self`, `1, 0, 0`), т.е. интерпретатор добавляет `self` в список параметров, что означает "источник, который должен быть создан сейчас с такими параметрами".

Как только мы создали `source`, можно вызвать два метода для вычисления поля скоростей и функции тока на сетке (`X, Y`). Как и при использовании библиотек, мы укажем на метод класса с помощью точечной нотации (как показано ниже).

Компоненты скорости в декартовой системе координат:

$$u = \frac{\sigma}{2\pi} \frac{x - x_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

$$v = \frac{\sigma}{2\pi} \frac{y - y_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2}$$

а функция тока записывается в виде:

$$\psi = \frac{\sigma}{2\pi} \arctan \left(\frac{y - y_{\text{source}}}{x - x_{\text{source}}} \right)$$

Теперь давайте посмотрим, как это работает.

```
In [4]: strength_source = 1.0          # интенсивность источника
       x_source, y_source = 0.0, 0.5      # положение источника

       # создаём источник (объект класса Source)
       source = Source(strength_source, x_source, y_source)

       # вычисляем поле скорости и функцию тока для источника
       # в узлах расчётной сетки
       source.velocity(X, Y)
       source.stream_function(X, Y)
```

Обратили внимание на точку? Используя её, мы говорим: «возьми функцию `velocity()` объекта `source`, который мы только что создали, и выполнни её». Как можно убедиться, функция `velocity()` связана с данными конкретного источника, поскольку является частью класса.

Отражение источника мы тоже создадим при помощи класса `Source`, с такой же интенсивностью, но другим положением. Вы, вероятно, оценили эффективность использования классов.

```
In [5]: # создаём отражение источника и вычисляем для него
       # скорости
       # и функцию тока
       source_image = Source(strength_source, x_source, -y_source)
       source_image.velocity(X, Y)
       source_image.stream_function(X, Y)
```

Используя принцип суперпозиции, можно вычислить линии тока источника вблизи от плоской поверхности. Наложение двух источников приводит к следующему полю скоростей:

$$u = \frac{\sigma}{2\pi} \left(\frac{x - x_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2} + \frac{x - x_{\text{source}}}{(x - x_{\text{source}})^2 + (y + y_{\text{source}})^2} \right)$$

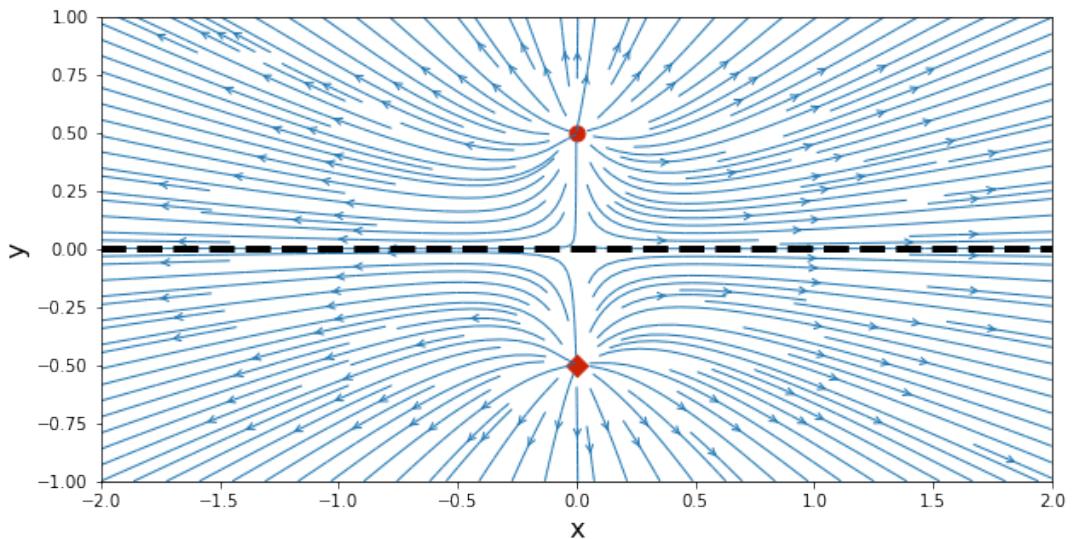
$$v = \frac{\sigma}{2\pi} \left(\frac{y - y_{\text{source}}}{(x - x_{\text{source}})^2 + (y - y_{\text{source}})^2} + \frac{y + y_{\text{source}}}{(x - x_{\text{source}})^2 + (y + y_{\text{source}})^2} \right)$$

и такой функции тока:

$$\psi = \frac{\sigma}{2\pi} \left(\arctan \left(\frac{y - y_{\text{source}}}{x - x_{\text{source}}} \right) + \arctan \left(\frac{y + y_{\text{source}}}{x - x_{\text{source}}} \right) \right)$$

```
In [6]: # суперпозиция источника и его отражения
u = source.u + source_image.u
v = source.v + source_image.v
psi = source.psi + source_image.psi

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1,\arrowstyle='->')
pyplot.scatter(source.x, source.y, color='#CD2305', s=80, marker='o')
pyplot.scatter(source_image.x, source_image.y, color='#CD2305', s=80,\marker='D')
pyplot.axhline(0., color='k', linestyle='--', linewidth=4);
```



7.2 Вихрь вблизи плоской поверхности

Аналогичным образом мы создадим ещё один класс под названием `Vortex`, требуя в качестве данных его интенсивность и расположение. Класс будет иметь два метода: один для вычисления скорости вихря и другой, чтобы вычислить функцию тока.

И нам еще потребуется *конструктор*, функция с именем `__init__()` для того, чтобы инициализировать параметры вихря.

```
In [7]: class Vortex:
    """
    Contains information related to a vortex.
    """
    def __init__(self, strength, x, y):
        """
        Sets the location and strength of the vortex.

        Parameters
        -----
    
```

```

        strength: float
            Strength of the vortex.
        x: float
            x-coordinate of the vortex.
        y: float
            y-coordinate of the vortex.
    """
    self.strength = strength
    self.x, self.y = x, y

def velocity(self, X, Y):
    """
    Computes the velocity field generated by a vortex.

    Parameters
    -----
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.
    """
    self.u = +self.strength/(2*math.pi)*(Y-self.y)/((X-self.x)**2+\n
                                                    (Y-self.y)**2)
    self.v = -self.strength/(2*math.pi)*(X-self.x)/((X-self.x)**2+\n
                                                    (Y-self.y)**2)

def stream_function(self, X, Y):
    """
    Computes the stream-function generated by a vortex.

    Parameters
    -----
    X: 2D Numpy array of floats
        x-coordinate of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.
    """
    self.psi = -self.strength/(4*math.pi)*numpy.log((X-self.x)**2+\n
                                                    (Y-self.y)**2)

```

Теперь очень легко можно создать два объекта типа `Vortex`: собственно вихрь и его отражение.

```

In [8]: strength_vortex = 1.0                      # интенсивность вихря
         x_vortex, y_vortex = 0.0, 0.5                # положение вихря

        # создаём вихрь и вычисляем для него скорость и функцию тока
        vortex = Vortex(strength_vortex, x_vortex, y_vortex)
        vortex.velocity(X, Y)
        vortex.stream_function(X, Y)

        # создаём отражение вихря и вычисляем для него скорость и функцию тока
        vortex_image = Vortex(-strength_vortex, x_vortex, -y_vortex)
        vortex_image.velocity(X, Y)
        vortex_image.stream_function(X, Y)

```

Применяя принцип суперпозиции, мы можем получить линии тока, описывающие вихрь возле стенки.

```

In [9]: # суперпозиция вихря и его отражения
         u = vortex.u + vortex_image.u

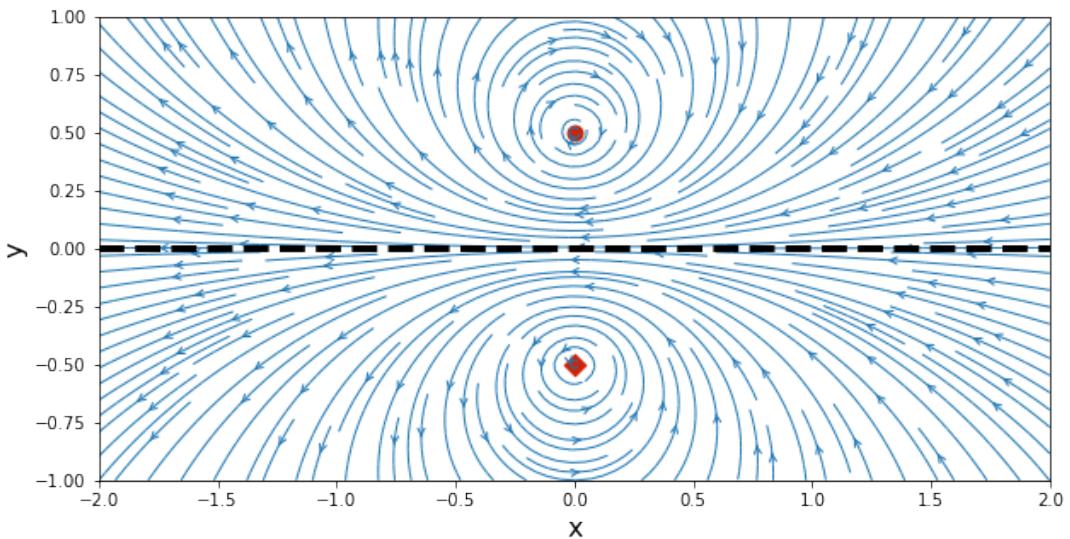
```

```

v = vortex.v + vortex_image.v
psi = vortex.psi + vortex_image.psi

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowstyle='->')
pyplot.scatter(vortex.x, vortex.y, color='#CD2305', s=80, marker='o')
pyplot.scatter(vortex_image.x, vortex_image.y, color='#CD2305', s=80, \
               marker='D')
pyplot.axhline(0., color='k', linestyle='--', linewidth=4);

```



7.3 Движение вихревой пары у земли

Продолжаем веселье. Нужно больше особенностей! От пары вихрей может закружиться голова...

```

In [10]: strength_vortex = 1.0                      # абсолютное значение интенсивности каждого вихря
          x_vortex1, y_vortex1 = -0.1, 0.5            # положение первого вихря
          x_vortex2, y_vortex2 = +0.1, 0.5            # положение второго вихря

# создаём два вихря в разных точках
vortex1 = Vortex(+strength_vortex, x_vortex1, y_vortex1)
vortex2 = Vortex(-strength_vortex, x_vortex2, y_vortex2)

# вычисляем скорости и функцию тока для каждого из вихрей
vortex1.velocity(X, Y)
vortex1.stream_function(X, Y)
vortex2.velocity(X, Y)
vortex2.stream_function(X, Y)

# создаём отражение каждого вихря
vortex1_image = Vortex(-strength_vortex, x_vortex1, -y_vortex1)

```

```

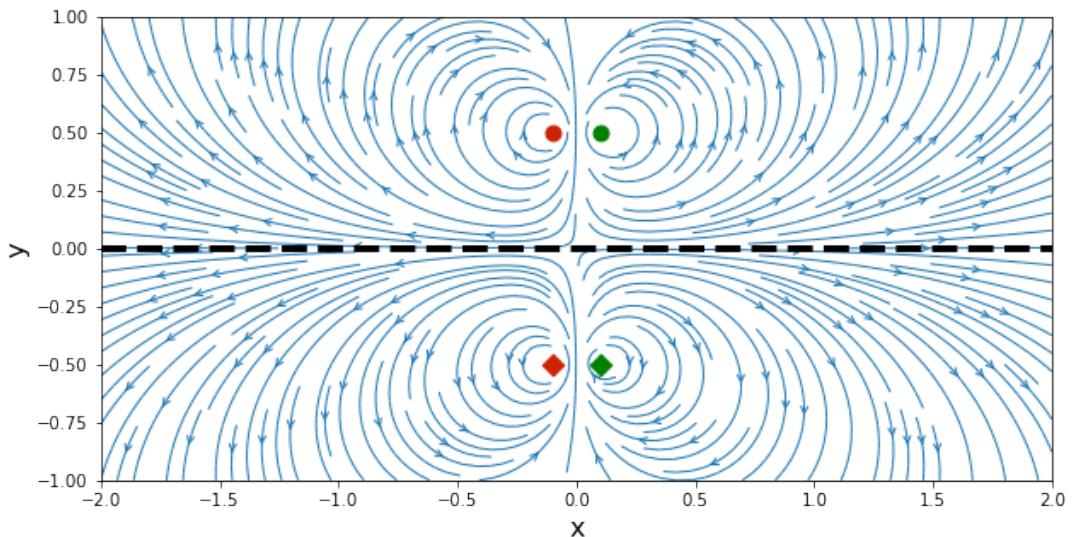
vortex2_image = Vortex(+strength_vortex, x_vortex2, -y_vortex2)

# вычисляем скорости и функцию тока для каждого из отражений
vortex1_image.velocity(X, Y)
vortex1_image.stream_function(X, Y)
vortex2_image.velocity(X, Y)
vortex2_image.stream_function(X, Y)

In [11]: # суперпозиция пары вихрей и ее отражения
u = vortex1.u + vortex2.u + vortex1_image.u + vortex2_image.u
v = vortex1.v + vortex2.v + vortex1_image.v + vortex2_image.v
psi = vortex1.psi + vortex2.psi + vortex1_image.psi + vortex2_image.psi

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1, \
                   arrowstyle='->')
pyplot.scatter(vortex1.x, vortex1.y, color='#CD2305', s=80, marker='o')
pyplot.scatter(vortex2.x, vortex2.y, color='g', s=80, marker='o')
pyplot.scatter(vortex1_image.x, vortex1_image.y, color='#CD2305', \
               s=80, marker='D')
pyplot.scatter(vortex2_image.x, vortex2_image.y, color='g', \
               s=80, marker='D')
pyplot.axhline(0., color='k', linestyle='--', linewidth=4);

```



7.4 Диполь в потоке, параллельном плоскости

И напоследок ... диполь у плоской поверхности. Нам нужен новый класс, на этот раз под названием `Doublet`. Всё так же, как и прежде!

```

In [12]: u_inf = 1.0      # скорость равномерного течения

# вычисляем компоненты скорости и функцию тока для равномерного потока

```

```

u_freestream = u_inf * numpy.ones((N, N), dtype=float)
v_freestream = numpy.zeros((N, N), dtype=float)
psi_freestream = u_inf * Y

In [14]: class Doublet:
    """
    Contains information related to a doublet.
    """
    def __init__(self, strength, x, y):
        """
        Sets the location and strength of the doublet.

        Parameters
        -----
        strength: float
            Strength of the doublet.
        x: float
            x-coordinate of the doublet.
        y: float
            y-coordinate of the doublet.
        """
        self.strength = strength
        self.x, self.y = x, y

    def velocity(self, X, Y):
        """
        Computes the velocity field generated by a doublet.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """
        self.u = -self.strength/(2*math.pi)*\
            ((X-self.x)**2-(Y-self.y)**2)/((X-self.x)**2+\
            (Y-self.y)**2)**2
        self.v = -self.strength/(2*math.pi)*\
            2*(X-self.x)*(Y-self.y)/((X-self.x)**2+\
            (Y-self.y)**2)**2

    def stream_function(self, X, Y):
        """
        Computes the stream-function generated by a doublet.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """
        self.psi = -self.strength/(2*math.pi)*(Y-self.y)/((X-self.x)**2+\
            (Y-self.y)**2)

```

```

In [15]: strength_doublet = 1.0          # интенсивность диполя
          x_doublet, y_doublet = 0.0, 0.3      # положение диполя

```

```

# создаём диполь (объект класса Doublet)
doublet = Doublet(strength_doublet, x_doublet, y_doublet)

# вычисляем поле скорости
# и функцию тока для диполя в узлах расчётной сетки
doublet.velocity(X, Y)
doublet.stream_function(X, Y)

# создаём отражение диполя
doublet_image = Doublet(strength_doublet, x_doublet, -y_doublet)

# вычисляем поле скорости
# и функцию тока для отражения диполя в узлах расчётной сетки
doublet_image.velocity(X, Y)
doublet_image.stream_function(X, Y)

```

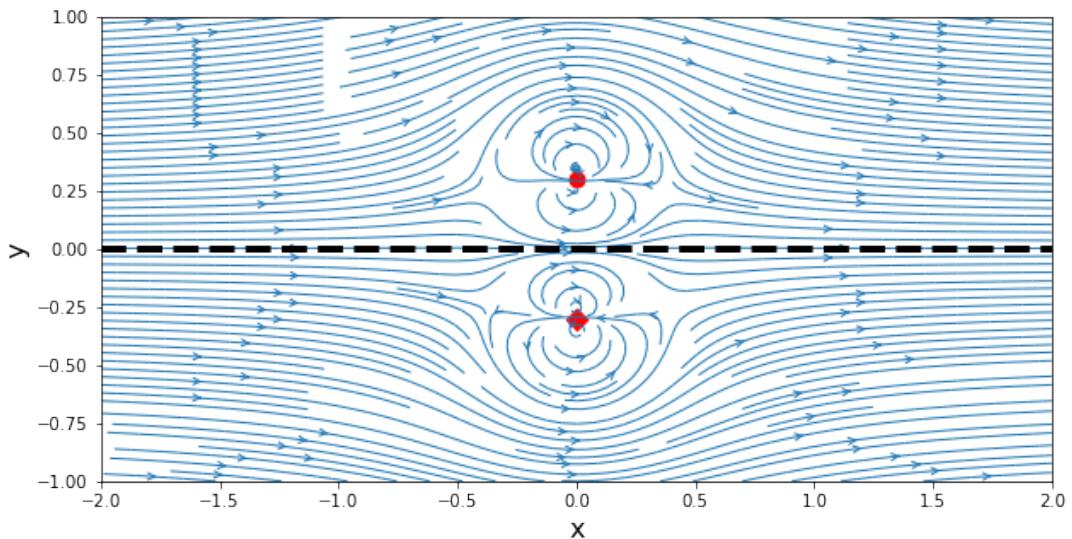
In [16]: # суперпозиция диполя, его отражения и равномерного потока

```

u = u_freestream + doublet.u + doublet_image.u
v = v_freestream + doublet.v + doublet_image.v
psi = psi_freestream + doublet.psi + doublet_image.psi

# рисуем линии тока
size = 10
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.streamplot(X, Y, u, v, density=2, linewidth=1, arrowsize=1,\ 
                  arrowstyle='->')
pyplot.scatter(doublet.x, doublet.y, color='r', s=80, marker='o')
pyplot.scatter(doublet_image.x, doublet_image.y, color='r', s=80,\ 
               marker='D')
pyplot.axhline(0., color='k', linestyle='--', linewidth=4);

```



8 Слой источников

Слой источников это бесконечная цепочка источников, выстроенных в одну линию. Вы уже делали что-то похожее — слой, составленный из бесконечной [цепочки вихрей](#).

Убедитесь, что вы изучили [7-е занятие AeroPython](#), посвященное методу зеркальных отражений и научились пользоваться классами в Python. С этого момента классы будут нас постоянно сопровождать!

Начнём, как мы уже много раз делали, с импорта библиотек и создания расчётной сетки.

```
In [1]: import numpy
        import math
        from matplotlib import pyplot
        # помещаем графику внутрь блокнота
        %matplotlib inline

In [2]: N = 100                                # Число точек в каждом направлении
        x_start, x_end = -1.0, 1.0                 # граница по x
        y_start, y_end = -1.5, 1.5                 # граница по y
        x = numpy.linspace(x_start, x_end, N)      # вычисляем одномерный массив x
        y = numpy.linspace(y_start, y_end, N)      # вычисляем одномерный массив y
        X, Y = numpy.meshgrid(x, y)                 # создаём расчётную сетку
```

Добавим также равномерный поток со скоростью $U_\infty = 1$, параллельный горизонтальной оси. Массивы `u_freestream` и `v_freestream` содержат значения компонент вектора скорости равномерного потока в узлах расчётной сетки. Давайте заполним их:

```
In [3]: u_inf = 1.0      # скорость набегающего потока

        # рассчитываем компоненты вектора скорости набегающего потока
        u_freestream = u_inf * numpy.ones((N, N), dtype=float)
        v_freestream = numpy.zeros((N, N), dtype=float)
```

8.1 Конечное число источников, расположенных на линии

Сперва рассмотрим конечное число источников, расположенных вдоль вертикальной линии по нормали к потоку. Линии тока будут истекать из каждого источника и отклоняться потоком, набегающим из бесконечности.

С вычислительной точки зрения, конечное число источников может быть представлено как одномерный массив, состоящий из объектов класса `Source`. У этого класса должны быть атрибуты, отвечающие за интенсивность источника — `strength`, и его местоположение (`x,y`). Методы класса должны вычислять компоненты скорости и функцию тока в узлах заданной расчётной сетки, и, конечно, у него должен быть [конструктор](#):

```
In [4]: class Source:
        """
        Contains information related to a source/sink.
        """

        def __init__(self, strength, x, y):
            """
            Sets the location and strength of the singularity.

            Parameters
            -----
            strength: float
                Strength of the source/sink.
            x: float
                x-coordinate of the source/sink.
            y: float
                y-coordinate of the source/sink.
            """


```

```

        self.strength = strength
        self.x, self.y = x, y

    def velocity(self, X, Y):
        """
        Computes the velocity field generated by the source/sink.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """
        self.u = self.strength/(2*math.pi)*(X-self.x)/((X-self.x)**2+\n
                                                       (Y-self.y)**2)
        self.v = self.strength/(2*math.pi)*(Y-self.y)/((X-self.x)**2+\n
                                                       (Y-self.y)**2)

    def stream_function(self, X, Y):
        """
        Computes the stream-function generated by the source/sink.

        Parameters
        -----
        X: 2D Numpy array of floats
            x-coordinate of the mesh points.
        Y: 2D Numpy array of floats
            y-coordinate of the mesh points.
        """
        self.psi = self.strength/(2*math.pi)*numpy.arctan2((Y-self.y), \
                                                       (X-self.x))

```

Теперь вы увидите, какая полезная штука — классы! Мы воспользуемся созданным классом `Source` для создания источников, из которых будет состоять рассматриваемый слой. Выберем число `N_sources` и вызовем конструктор класса `N_sources` раз, при этом изменения координату y в положении источника. Затем вычислим скорости, наведённые источниками, используя соответствующую функцию.

```

In [5]: N_sources = 11                      # количество источников
        strength = 5.0                         # суммарная интенсивность источников
        strength_source = strength/N_sources   # интенсивность единичного источника
        # положение источников по горизонтали (одномерный массив)
        x_source = numpy.zeros(N_sources, dtype=float)
        # положение источников по вертикали (одномерный массив)
        y_source = numpy.linspace(-1.0, 1.0, N_sources)

        # создаём линию из источников (Нumpy массив объектов типа Source)
        sources = numpy.empty(N_sources, dtype=object)
        for i in range(N_sources):
            sources[i] = Source(strength_source, x_source[i], y_source[i])
            sources[i].velocity(X, Y)

        # суперпозиция всех источников и набегающего потока
        u = u_freestream.copy()
        v = v_freestream.copy()
        for source in sources:
            u += source.u
            v += source.v

```

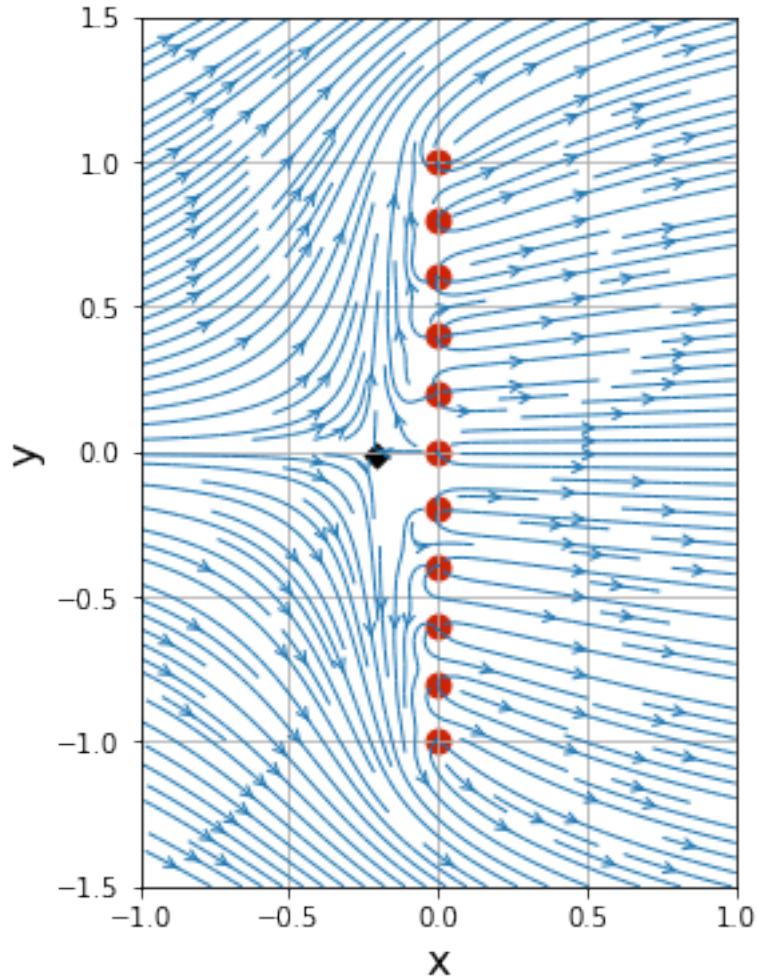
Обратите внимание, что переменная `sources` обозначает массив NumPy, содержащий набор элементов одного типа. Какого? Вы, вероятно, уже привыкли к массивам чисел, но можно также создавать массивы объектов любого типа, в нашем случае — объектов класса `Source`.

Мы создаём пустой массив NumPy с именем `sources`, и говорим Python, что его элементами будут объекты, которые не являются встроенным типом данных, как `int` или `float`. Число элементов массива равно `N_sources`.

В первом цикле мы заполняем массив, вызывая *конструктор* класса `Source` для каждого элемента. Кроме этого мы еще и вычисляем поле скоростей, наведённое каждым источником, используя метод `velocity()`. Во втором цикле, создав предварительно массивы для компонент скоростей `u` и `v` и инициализировав их компонентами набегающего потока, мы прибавляем к ним вклад от скоростей источников.

Окончательная картина линий тока описывает суперпозицию равномерного потока и `N_sources` источников одинаковой интенсивности `strength_source`, равномерно распределённых вдоль вертикальной линии по нормали к потоку. На рисунке мы также обозначим красными точками положение источников, а чёрным ромбом — положение точки торможения, которая определяется как точка с наименьшим значением абсолютной величины скорости.

```
In [6]: # рисуем линии тока
size = 4
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.streamplot(X, Y, u, v,
                   density=2, linewidth=1, arrowsize=1, arrowstyle='->')
# рисуем источники
pyplot.scatter(x_source, y_source,
               color='#CD2305', s=80, marker='o')
# вычисляем величину скорости и индексы точки торможения
# внимание: за точку торможения принимается точка,
# в которой скорость минимальна
magnitude = numpy.sqrt(u**2 + v**2)
j_stagn, i_stagn = numpy.unravel_index(magnitude.argmin(),
                                         magnitude.shape)
# рисуем точку торможения
pyplot.scatter(x[i_stagn], y[j_stagn],
               color='black', s=40, marker='D')
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end);
```



Попробуйте изменить интенсивность и положение источников. При каком минимальном значении суммарной интенсивности источников разделительная линия тока ($\psi = 0$) приближается к ним вплотную?

8.2 Бесконечная линия источников

По определению, *слой источников* — это набор расположенных вплотную источников бесконечно малой и равной интенсивности, распределённых вдоль заданной кривой.

Обозначим погонную интенсивность (или интенсивность на единицу длины) как $\sigma = \sigma(s)$, где s — параметр кривой.

Вспомним, что интенсивность одиночного источника — это его удельный расход. Тогда σ является удельным расходом на единицу длины в направлении s . А σds — это интенсивность бесконечно малого отрезка ds слоя источников. Этот отрезок настолько мал, что его можно рассматривать как отдельный источник интенсивности σds .

Согласно этим рассуждениям, функция тока в точке (r, θ) от такого источника будет равна

$$d\psi(r, \theta) = \frac{\sigma ds}{2\pi} \theta$$

Интегрируя вдоль s , найдем функцию тока слоя источников:

$$\psi(r, \theta) = \frac{\sigma}{2\pi} \int_{\text{sheet}} \theta(s) ds$$

В предыдущем разделе мы рассмотрели конечное распределение источников вдоль вертикальной прямой. Аналогично, функция тока вертикального слоя источников, расположенного в $x = 0$ между точками y_{\min} и y_{\max} в декартовой системе координат записывается в виде:

$$\psi(x, y) = \frac{\sigma}{2\pi} \int_{y_{\min}}^{y_{\max}} \tan^{-1} \left(\frac{y - \xi}{x} \right) d\xi$$

А компоненты скорости —

$$u(x, y) = \frac{\sigma}{2\pi} \int_{y_{\min}}^{y_{\max}} \frac{x}{x^2 + (y - \xi)^2} d\xi$$

$$v(x, y) = \frac{\sigma}{2\pi} \int_{y_{\min}}^{y_{\max}} \frac{y - \xi}{x^2 + (y - \xi)^2} d\xi$$

Используем SciPy Для определения скоростей нам нужно вычислить два интеграла, полученные выше. Ну, приехали, скажете вы. Вычислять интегралы — это не модно. Без паники! У нас есть SciPy: набор мощных математических алгоритмов и функций. В нем имеется модуль `integrate`, базовый набор функций для научных вычислений, включающий простейшее интегрирование, квадратурные формулы и реализации методов численного интегрирования для обыкновенных дифференциальных уравнений. Насколько это полезно?

Давайте импортируем нужный модуль из SciPy:

```
In [7]: from scipy import integrate
```

Мы воспользуемся функцией `quad(func, a, b)` модуля `integrate` для вычисления значения определённого интеграла функции одной переменной на отрезке:

$$I = \int_a^b f(x) dx$$

Нужно, чтобы первым аргументом функции `quad` была питоновская функция. Вы уже знаете, как создавать функцию в Python при помощи ключевого слова `def`, а теперь вы узнаете, как это сделать, используя выражение `lambda`. Зачем, вы спросите, нужны два способа для одного и того же действия? Ответ на этот вопрос довольно тонкий, а на эту тему написан не один пост в блогах!

Если коротко, функция, создаваемая при помощи `lambda`-выражения задается одним выражением, которое возвращает какое-либо значение (при этом `return` не используется!). Такую функцию часто называют «анонимной», так как ей не нужно давать имя.

Давайте разберемся, как можно использовать `lambda`-выражения для интегрирования математических функций. Предположим, что мы хотим вычислить интеграл $f : x \rightarrow x^2$ от 0 до 1. Вы же можете это сделать вручную, правда?

Получится $\frac{1}{3}$.

Чтобы использовать функцию `quad()`, мы подаем ей в качестве первого аргумента выражение `lambda x : x**2`:

```
In [8]: print(integrate.quad(lambda x: x**2, 0.0, 1.0))
```

```
(0.3333333333333337, 3.700743415417189e-15)
```

Как видите, для передачи подынтегрального выражения в функцию `quad()` мы воспользовались лямбда-выражением, не определяя функцию отдельно (мы её никак не задавали, поэтому она *анонимная*).

Заметим, что функция `quad()` возвращает список: его первый элемент — результат интегрирования, а второй — оценка ошибки вычисления результата. Если вам нужно только значение определенного интеграла, нужно указать индекс `[0]`, чтобы получить первый элемент списка.

Заметим также, что лямбда-функция может принимать несколько аргументов:

```
In [9]: a = 3.0
print(integrate.quad(lambda x, a: a * x**2, 0.0, 1.0, args=a))
b = 2.0
print(integrate.quad(lambda x, a, b: a * b * x**2, 0.0, 1.0, args=(a, b)))
```

```
(1.0, 1.1102230246251565e-14)
(2.0, 2.220446049250313e-14)
```

Теперь мы готовы к использованию нового знания в определении поля скорости от слоя источников.

Ещё один момент! Функция `quad()` возвращает число с плавающей точкой в качестве значения интеграла. Для того, чтобы вычислить интеграл в каждом узле расчётной сетки, нужно пробежать его в цикле. Это может стать затратным с вычислительной точки зрения при измельчении сетки. Если только не воспользоваться векторизацией — `numpy.vectorize()`. В результате получится функция, принимающая массив NumPy в качестве аргумента, и возвращающая другой массив NumPy!

```
In [10]: sigma = 2.5      # интенсивность слоя источников

# границы слоя
y_min, y_max = -1.0, 1.0

# создаём анонимные функции
integrand_u = lambda s, x, y: x / (x**2 + (y - s)**2)
integrand_v = lambda s, x, y: (y - s) / (x**2 + (y - s)**2)

# создаём шаблонную функцию для векторизации
def integration(x, y, integrand):
    return integrate.quad(integrand, y_min, y_max, args=(x, y))[0]

vec_integration = numpy.vectorize(integration)

# рассчитываем поле скорости, создаваемое слоем источников
u_sheet = sigma / (2.0 * numpy.pi) * vec_integration(X, Y, integrand_u)
v_sheet = sigma / (2.0 * numpy.pi) * vec_integration(X, Y, integrand_v)

# суперпозиция слоя источников и набегающего потока
u = u_freestream + u_sheet
v = v_freestream + v_sheet
```

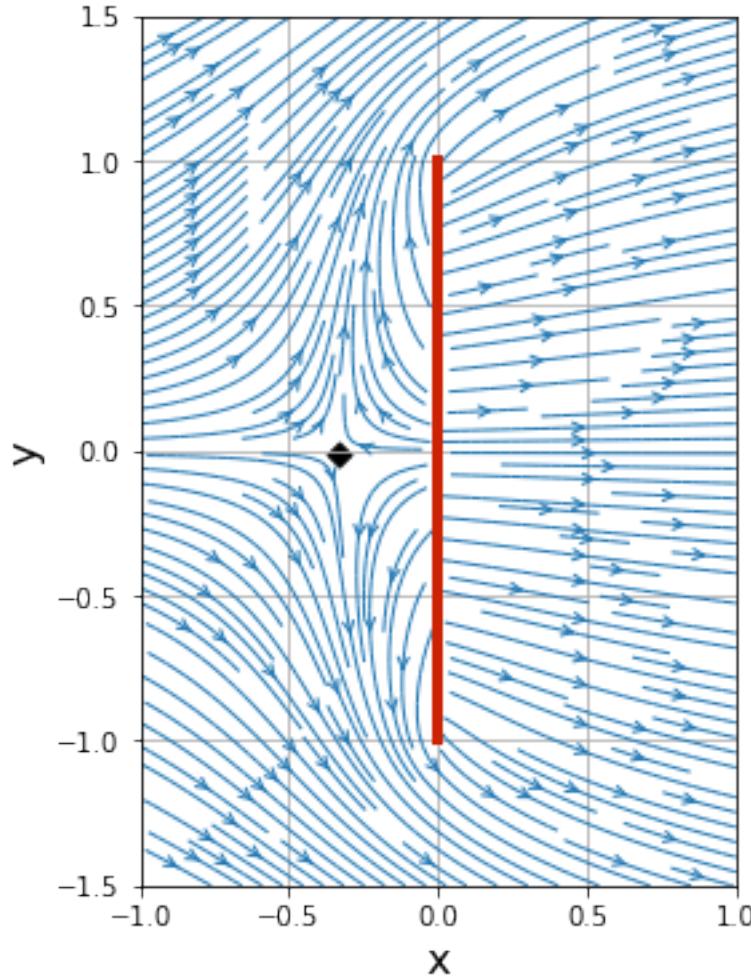
Теперь давайте визуализируем получившиеся линии тока. Слой источников обозначим красной линией, а положение точки торможения покажем при помощи контурной заливки.

```
In [11]: # рисуем линии тока
size = 4
pyplot.figure(figsize=(size, (y_end-y_start)/(x_end-x_start)*size))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.streamplot(X, Y, u, v,
                  density=2, linewidth=1, arrowsize=1, arrowstyle='->')
# рисуем слой источников
pyplot.axvline(0.0,
               (y_min-y_start)/(y_end-y_start),
               (y_max-y_start)/(y_end-y_start),
               color='#CD2305', linewidth=4)
# вычисляем величину скорости и индексы точки торможения
# внимание: за точку торможения принимается точка,
# в которой скорость минимальна
magnitude = numpy.sqrt(u**2 + v**2)
j_stagn, i_stagn = numpy.unravel_index(magnitude.argmin(), \
                                         magnitude.shape)
# рисуем точку торможения
```

```

pyplot.scatter(x[i_stagn], y[j_stagn],
               color='black', s=40, marker='D')
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end);

```



Как было сказано, интенсивность σ соответствует расходу жидкости, вытекающей из слоя источников. Если «поиграть» с этим параметром, то можно увидеть, что при его уменьшении точка торможения становится всё ближе и ближе к слою.

Чтобы использовать комбинацию нескольких слоёв источников для получения обтекания тела заданной формы, нужно, чтобы каждый такой слой был частью разделительной линии тока. Возникает вопрос: *Какой должна быть интенсивность источников, чтобы так получилось?*

Объёмный расход с левой стороны от слоя равен $\frac{\sigma}{2}$, и эта жидкость вытекает навстречу потоку со скоростью U_∞ . Следовательно, условие непротекания требует, чтобы $\frac{\sigma}{2} = U_\infty$.

Теперь вернитесь к коду, написанному выше, и замените в нем `sigma` на правильное значение. Где теперь точка торможения? Где проходит разделительная линия тока?

8.3 Дополнительные материалы

Лямбда-функции кажутся особенно запутанными, если вы только начинаете программировать на Python. Вот несколько мест, чтобы копнуть глубже эту тему:

- интересный пост, в котором рассматриваются тонкости лямбда-выражений: [Yet Another Lambda Tutorial](#), блог «Python Conquers the Universe» (29 August 2011)
- глава «Anonymous functions: `lambda`» в книге *Learning Python* Марка Лутца.

9 Обтекание цилиндра, составленного из панелей

На предыдущих занятиях для представления тел простой формы, таких как [овал Рэнкина](#) или [круговой цилиндр](#) использовались точечные особенности, помещённые в равномерный поток. Надо сказать, это большая удача, что при помощи комбинации нескольких фундаментальных решений уравнения Лапласа удается получить картины течения, в которых разделительная линия тока выглядит как замкнутое тело.

Но что, если нужно получить картину течения вокруг произвольной геометрии? Удастся ли подобрать такую комбинацию фундаментальных решений, чтобы получился нужный результат? *И как это сделать?* Методом проб и ошибок? Потребуется невероятная удача и масса усилий, чтобы получилась необходимая геометрия.

Цель этого занятия — получить распределения положений источников и их интенсивностей, при которых создается потенциальное обтекание заданной геометрии, а именно, кругового цилиндра. Известно, что течение вокруг цилиндра моделируется при помощи комбинации диполя и равномерного потока, но нам нужно разработать общий подход к моделированию течений, который можно будет применить к *различным* геометриям.

Метод, которым мы воспользуемся, состоит в следующем. Геометрия тела представляется в виде набора небольших отрезков, *панелей*, которые являются [слоями источников](#), изученными на прошлом занятии.

В результате мы должны получить алгоритм, согласно которому, вначале задается геометрия объекта, а затем определяются интенсивности источников каждой панели, при которых разделительная линия тока совпадает с границей тела. Искомые значения интенсивностей определяются из условия непротекания.

Начнем с импорта необходимых библиотек.

```
In [1]: import math
      import numpy
      from scipy import integrate
      from matplotlib import pyplot
      # помещаем графику внутрь блокнота
%matplotlib inline
```

Добавим равномерный горизонтальный поток со скоростью `u_inf`, равной 1:

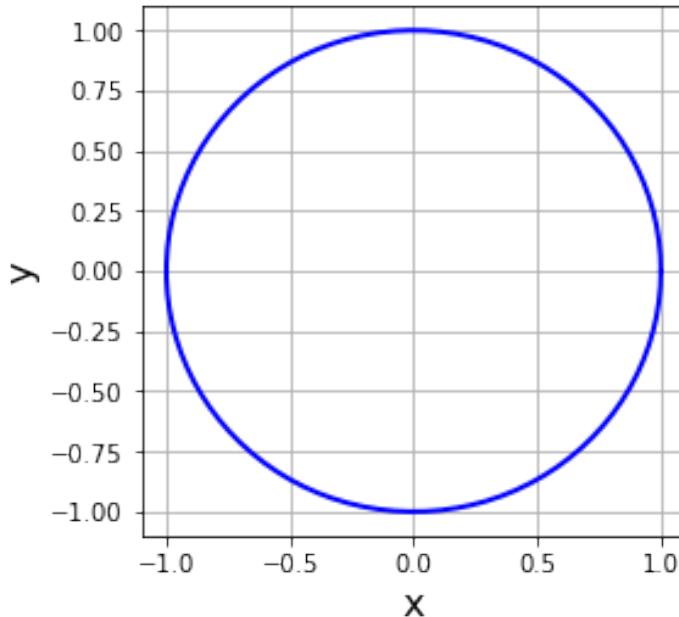
```
In [2]: u_inf = 1.0          # скорость набегающего потока
```

9.1 Задание геометрии

Рассмотрим круговой цилиндр единичного радиуса. Такая геометрия задается множеством точек в диапазоне углов от 0 до 2π .

```
In [3]: # задаём параметры цилиндра
R = 1.0                                # радиус
# угловая координата в радианах
theta = numpy.linspace(0, 2*math.pi, 100)
# цилиндр в декартовых координатах
x_cylinder, y_cylinder = R*numpy.cos(theta), R*numpy.sin(theta)

# рисуем цилиндр
size = 4
pyplot.figure(figsize=(size, size))
pyplot.grid(True)
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x_cylinder, y_cylinder, color='b', linestyle='-', linewidth=2)
pyplot.xlim(-1.1, 1.1)
pyplot.ylim(-1.1, 1.1);
```



9.2 Разбиение на панели

Панель, представляющая собой слой источников, определяются координатами начала и конца (`xa`, `ya`) и (`xb`, `yb`), и интенсивностью `sigma`. В дальнейшем нам также понадобятся её центр (`xc`, `yc`) и длина. Положение панели определяется углом между нормалью и осью x . Положительным считается направление против часовой стрелки.

Что нам нужно вычислить для каждой панели? Во-первых, интенсивность слоя источников, которая нужна для получения линий тока. Во-вторых, касательную компоненту скорости на панели (нормальная компонента равна нулю) и коэффициент давления.

На этом занятии вы в полной мере оцените пользу классов. Они сильно облегчат процесс управления кодом. Создадим класс с именем `Panel`, в котором будет содержаться все геометрические данные, относящиеся к одиночной панели. Имея начальную и конечную точку, внутри класса вычислим координаты центра, длину и нормаль. Кроме того, интенсивность источника, касательная компонента скорости и коэффициент давления инициализируются нулевыми значениями. (Потом они изменятся.)

```
In [4]: class Panel:
    """
    Contains information related to a panel.

    def __init__(self, xa, ya, xb, yb):
        """
        Initializes the panel.

        Sets the end-points and calculates the center, length, and angle
        (with the x-axis) of the panel.
        Initializes the strength of the source-sheet,
        the tangential velocity,
        and the pressure coefficient to zero.

        Parameters
        -----
        xa: float
            x-coordinate of the first end-point.
    
```

```

ya: float
    y-coordinate of the first end-point.
xb: float
    x-coordinate of the second end-point.
yb: float
    y-coordinate of the second end-point.
"""
self.xa, self.ya = xa, ya
self.xb, self.yb = xb, yb

# контрольная точка (центр панели)
self.xc, self.yc = (xa+xb)/2, (ya+yb)/2
# длина панели
self.length = math.sqrt((xb-xa)**2+(yb-ya)**2)

# ориентация панели (угол между нормалью и осью x)
if xb-xa <= 0.:
    self.beta = math.acos((yb-ya)/self.length)
elif xb-xa > 0.:
    self.beta = math.pi + math.acos(-(yb-ya)/self.length)

self.sigma = 0.                      # интенсивность источника
self.vt = 0.                          # касательная скорость
self.cp = 0.                          # коэффициент давления

```

Для хранения разбиения создадим массив NumPy длины `N_panels`, в котором каждый элемент будет объектом класса `Panel`.

```

In [5]: N_panels = 10                  # желаемое число панелей

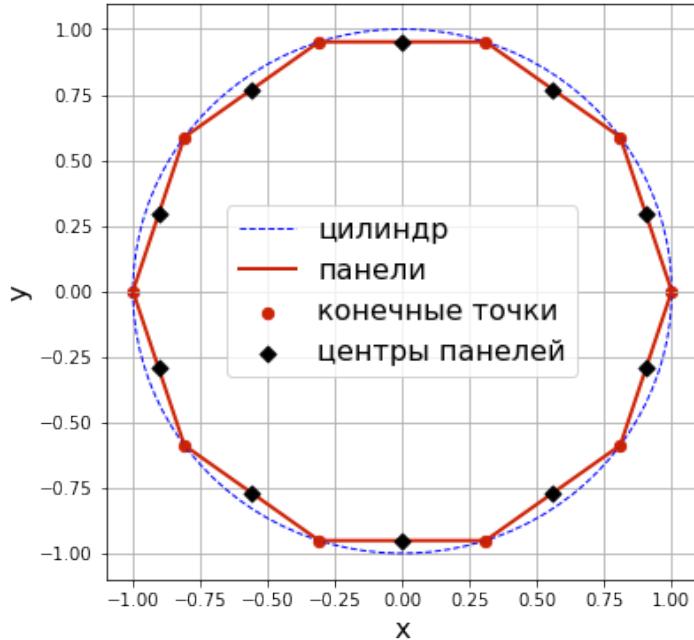
# задаём конечные точки панелей
x_ends = R*numpy.cos(numpy.linspace(0, 2*math.pi, N_panels+1))
y_ends = R*numpy.sin(numpy.linspace(0, 2*math.pi, N_panels+1))

# определяем панели
panels = numpy.empty(N_panels, dtype=object)
for i in range(N_panels):
    panels[i] = Panel(x_ends[i], y_ends[i], x_ends[i+1], y_ends[i+1])

# рисуем панели
size = 6
pyplot.figure(figsize=(size, size))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x_cylinder, y_cylinder,
            label='цилиндр',
            color='b', linestyle='--', linewidth=1)
pyplot.plot(x_ends, y_ends,
            label='панели',
            color='#CD2305', linestyle='-', linewidth=2)
pyplot.scatter([p.xa for p in panels], [p.ya for p in panels],
              label='конечные точки',
              color='#CD2305', s=40)
pyplot.scatter([p.xc for p in panels], [p.yc for p in panels],
              label='центры панелей', marker = 'D',
              color='k', s=40, zorder=3)
pyplot.legend(loc='best', prop={'size':16})
pyplot.xlim(-1.1, 1.1)

```

```
pyplot.ylim(-1.1, 1.1);
```



9.3 Граничное условие для потока на поверхности тела

В заключительном задании [первого занятия](#) вы получили потенциал скорости для одиночного источника. При интегрировании радиальной компоненты скорости $u_r = \frac{\sigma}{2\pi r}$ получим

$$\phi = \frac{\sigma}{2\pi} \ln r$$

(При интегрировании ещё появляется функция от θ , которая, однако, является константой, поскольку $u_\theta = 0$; положим эту константу равной нулю.)

Мы воспользуемся потенциалом скорости, чтобы сделать вектор скорости касательным к панели. Для этого нужно, чтобы $u_n = 0$, или

$$u_n(x, y) = \frac{\partial \phi}{\partial n}(x, y)$$

в заданной точке панели. Выберем в качестве точки, в которой будет обеспечиваться выполнение условия непротекания, центр панели и будем называть её *контрольная точка*.

Потенциал скорости для [слоя источников](#) одиночной панели в декартовой системе координат:

$$\phi(x, y) = \frac{\sigma}{2\pi} \int_{\text{панель}} \ln \sqrt{(x - x(s))^2 + (y - y(s))^2} ds$$

где s — текущая координата вдоль панели, а $(x(s), y(s))$ — декартовы координаты s .

Потенциал в произвольной точке (x, y) есть суперпозиция потенциалов каждой панели, поэтому просуммируем вклад от каждой панели (при этом вынесем логарифм корня из-под знака интеграла):

$$\phi(x, y) = \sum_{j=1}^{N_p} \frac{\sigma_j}{4\pi} \int \ln \left((x - x_j(s_j))^2 + (y - y_j(s_j))^2 \right) ds_j$$

Если добавить к полученному выражению невозмущенный поток, то течение в окрестности кругового цилиндра будет иметь следующий потенциал:

$$\phi(x, y) = U_\infty x + \sum_{j=1}^{N_p} \frac{\sigma_j}{4\pi} \int \ln \left((x - x_j(s_j))^2 + (y - y_j(s_j))^2 \right) ds_j$$

Обеспечение касательности скорости в *контрольных точках* делает контур тела разделительной линией тока в некотором приближении, при этом точность такого приближения увеличивается по мере роста количества панелей. Так, для каждой панели i , в точке (x_{c_i}, y_{c_i}) должно выполняться $u_n = 0$:

$$u_{n_i} = \frac{\partial}{\partial n_i} \{ \phi(x_{c_i}, y_{c_i}) \} = 0,$$

что приводит к

$$0 = U_\infty \cos \beta_i + \sum_{j=1}^{N_p} \frac{\sigma_j}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \frac{\partial x_{c_i}}{\partial n_i} + (y_{c_i} - y_j(s_j)) \frac{\partial y_{c_i}}{\partial n_i}}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j,$$

где β_i — угол, который нормаль к панели образует с осью x , так что

$$\frac{\partial x_{c_i}}{\partial n_i} = \cos \beta_i \quad \text{and} \quad \frac{\partial y_{c_i}}{\partial n_i} = \sin \beta_i$$

и

$$\begin{aligned} x_j(s_j) &= x_{a_j} - \sin(\beta_j) s_j \\ y_j(s_j) &= y_{a_j} + \cos(\beta_j) s_j \end{aligned}$$

В случае, когда $i = j$, возникает проблема. Из предыдущего блокнота мы знаем, что интенсивность [слоя источников](#) должна принимать определенное значение, чтобы линии тока не проникали через панель. Исходя из этого соображения, обозначим вклад i -ой панели как $\frac{\sigma_i}{2}$.

Тогда граничное условие в центре i -ой панели можно записать в следующем виде:

$$0 = U_\infty \cos \beta_i + \frac{\sigma_i}{2} + \sum_{j=1, j \neq i}^{N_p} \frac{\sigma_j}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \cos \beta_i + (y_{c_i} - y_j(s_j)) \sin \beta_i}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j$$

Из приведенного выше уравнения, мы понимаем, что нам придется вычислять интегралы с помощью SciPy функции `integrate.quad()`. Определим функцию `integral_normal()`, которая сделает за нас всю сложную работу.

```
In [6]: def integral_normal(p_i, p_j):
    """
    Evaluates the contribution of a panel at the center-point of another,
    in the normal direction.

    Parameters
    -----
    p_i: Panel object
        Panel on which the contribution is calculated.
    p_j: Panel object
        Panel from which the contribution is calculated.

    Returns
    -----
    Integral over the panel at the center point of the other.
    """
    def integrand(s):
        return ( (p_i.xc - (p_j.xa - math.sin(p_j.beta)*s))*math.cos(p_i.beta)
                +(p_i.yc - (p_j.ya + math.cos(p_j.beta)*s))*math.sin(p_i.beta))
               /((p_i.xc - (p_j.xa - math.sin(p_j.beta)*s))**2
                 +(p_i.yc - (p_j.ya + math.cos(p_j.beta)*s))**2) )
    return integrate.quad(integrand, 0.0, p_j.length)[0]
```

9.4 Решение системы линейных уравнений

Мы только что вывели уравнение, обеспечивающее выполнение условия непротекания на i -ой панели. У нас N_{panels} таких панелей, а значит N_{panels} неизвестных интенсивностей σ_i . Таким образом, задача сводится к решению системы линейных уравнений:

$$[A][\sigma] = [b],$$

где

$$A_{ij} = \begin{cases} \frac{1}{2}, & \text{если } i = j \\ \frac{1}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \cos \beta_i + (y_{c_i} - y_j(s_j)) \sin \beta_i}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j, & \text{если } i \neq j \end{cases}$$

и

$$b_i = -U_\infty \cos \beta_i$$

для $1 \leq i, j \leq N_p$. Заполним матрицу A и столбец правых частей b нужными значениями:

```
In [7]: # вычисляем матрицу влияния источников
A = numpy.empty((N_panels, N_panels), dtype=float)
numpy.fill_diagonal(A, 0.5)

for i, p_i in enumerate(panel):
    for j, p_j in enumerate(panel):
        if i != j:
            A[i,j] = 0.5/math.pi*integral_normal(p_i, p_j)

# вычисляем правую часть системы уравнений
b = - u_inf * numpy.cos([p.beta for p in panel])
```

Вуаля! Мы только что воспользовались новой (для нас) встроенной функцией `enumerate()`. С её помощью мы получаем доступ к каждому элементу `panel` массива `panels`, отслеживая его порядковый номер `i` (начинается с 0), что позволяет нам выбрать соответствующий элемент `A` для его заполнения.

Теперь можно с легкостью решить систему линейных уравнений при помощи функции `linalg.solve()` из библиотеки NumPy, и присвоить каждой панели источников нужную интенсивность:

```
In [8]: # решаем систему линейных уравнений
sigma = numpy.linalg.solve(A, b)

for i, panel in enumerate(panel):
    panel.sigma = sigma[i]
```

9.5 Коэффициент давления на поверхности

Теперь у нас есть нужное распределение интенсивностей источников, чтобы рассчитать поле линий тока в окрестности данной геометрии. Полезным измеримым следствием полученного результата является коэффициент давления на поверхности тела.

Согласно уравнению Бернулли, коэффициент давления на i -ой панели равен

$$C_{p_i} = 1 - \left(\frac{u_{t_i}}{U_\infty} \right)^2$$

где u_{t_i} — тангенциальная составляющая скорости в центре i -ой панели,

$$u_{t_i} = \frac{\partial}{\partial t_i} \{ \phi(x_{c_i}, y_{c_i}) \}$$

которую можно получить как

$$u_{t_i} = -U_\infty \sin \beta_i + \sum_{j=1}^{N_p} \frac{\sigma_j}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \frac{\partial x_{c_i}}{\partial t_i} + (y_{c_i} - y_j(s_j)) \frac{\partial y_{c_i}}{\partial t_i}}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j$$

учитывая, что

$$\frac{\partial x_{c_i}}{\partial t_i} = -\sin \beta_i \quad \text{и} \quad \frac{\partial y_{c_i}}{\partial t_i} = \cos \beta_i$$

Примем во внимание, что вклад от касательной компоненты скорости на выбранной панели от собственного потенциала скорости равен нулю, поскольку линии тока истекают из источника *наружу*.

Определим функцию `integral_tangential()`, которая вычислит полученный интеграл, опять используя `integrate.quad()`:

```
In [9]: def integral_tangential(p_i, p_j):
    """
    Evaluates the contribution of a panel at the center-point of another,
    in the tangential direction.

    Parameters
    -----
    p_i: Panel object
        Panel on which the contribution is calculated.
    p_j: Panel object
        Panel from which the contribution is calculated.

    Returns
    -----
    Integral over the panel at the center point of the other.
    """
    def integrand(s):
        return ((-(p_i.xc-(p_j.xa-math.sin(p_j.beta)*s))*math.sin(p_i.beta)
                  +(p_i.yc-(p_j.ya+math.cos(p_j.beta)*s))*math.cos(p_i.beta))
                /((p_i.xc-(p_j.xa-math.sin(p_j.beta)*s))**2
                  +(p_i.yc-(p_j.ya+math.cos(p_j.beta)*s))**2) )
    return integrate.quad(integrand, 0.0, p_j.length)[0]
```

```
In [10]: # вычисляем матрицу системы линейных уравнений
A = numpy.empty((N_panels, N_panels), dtype=float)
numpy.fill_diagonal(A, 0.0)

for i, p_i in enumerate(panels):
    for j, p_j in enumerate(panels):
        if i != j:
            A[i,j] = 0.5/math.pi*integral_tangential(p_i, p_j)

# вычисляем столбец -- правую часть
b = - u_inf * numpy.sin([panel.beta for panel in panels])

# вычисляем касательную скорость в центрах панелей
vt = numpy.dot(A, sigma) + b

for i, panel in enumerate(panels):
    panel.vt = vt[i]
```

Теперь, когда мы вычислили касательную компоненту скорости на каждой панели, можно рассчитать коэффициент давления.

```
In [11]: # рассчитываем коэффициент давления
for panel in panels:
    panel.cp = 1.0 - (panel.vt/u_inf)**2
```

Отлично! Пора рисовать график распределения коэффициента давления.

Но перед этим вспомним, что на занятии, посвященном [диполю](#), мы получили, что точное значение коэффициента давления на поверхности цилиндра равно

$$Cp = 1 - 4 \sin^2 \theta,$$

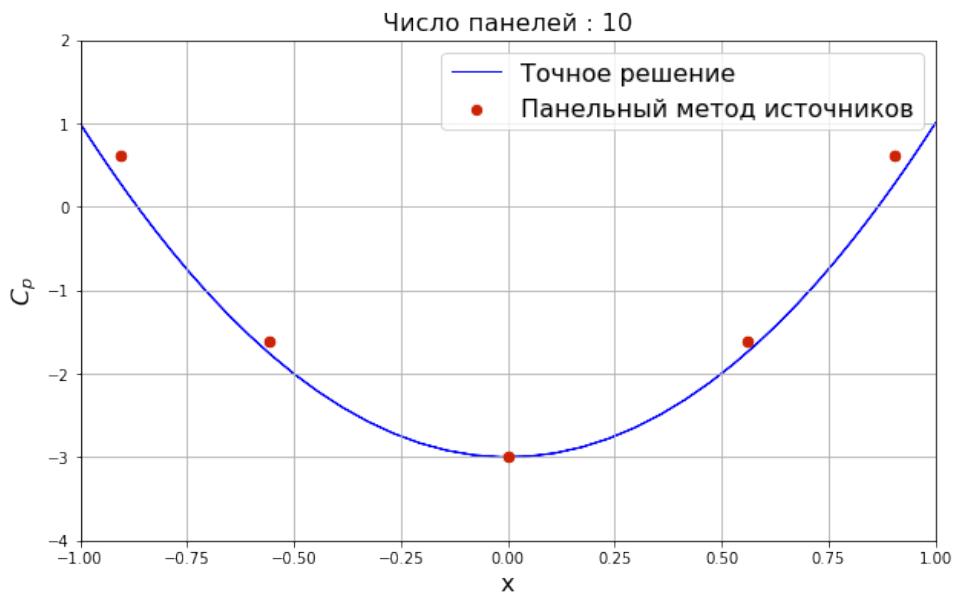
то есть

$$Cp = 1 - 4 \left(\frac{y}{R} \right)^2$$

Это можно использовать для сравнения с результатом, полученным при помощи нашего панельного метода.

```
In [12]: # вычисляем аналитическое значение коэффициента давления на цилиндре
cp_analytical = 1.0 - 4*(y_cylinder/R)**2

# рисуем распределение коэффициента давления на поверхности
pyplot.figure(figsize=(10, 6))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('$C_p$', fontsize=16)
pyplot.plot(x_cylinder, cp_analytical,
            label='Точное решение',
            color='b', linestyle='-', linewidth=1, zorder=1)
pyplot.scatter([p.xc for p in panels], [p.cp for p in panels],
               label='Панельный метод источников',
               color='#CD2305', s=40, zorder=2)
pyplot.title('Число панелей : %d' % N_panels, fontsize=16)
pyplot.legend(loc='best', prop={'size':16})
pyplot.xlim(-1.0, 1.0)
pyplot.ylim(-4.0, 2.0);
```



Контрольное задание Теперь, когда мы вычислили коэффициент давления на поверхности цилиндра, было бы интересно узнать, как ведут себя линии тока.

Для этого воспользуемся функцией `streamplot()` библиотеки Matplotlib, которой требуются декартовы компоненты скорости (u, v) в узлах расчётной сетки (X, Y). Поэтому первый шаг — это вывод уравнения для компонент скорости.

Потенциал N_p слоёв источников в равномерном потоке в точке (x, y) задается формулой

$$\phi(x, y) = U_\infty x + \sum_{j=1}^{N_p} \frac{\sigma_j}{4\pi} \int \ln \left((x - x_j(s_j))^2 + (y - y_j(s_j))^2 \right) ds_j$$

А поле скоростей в точке (x, y) :

$$u(x, y) = \frac{\partial}{\partial x} \{\phi(x, y)\}$$

$$v(x, y) = \frac{\partial}{\partial y} \{\phi(x, y)\}$$

Ваша задача:

- определить компоненты скорости в декартовой системе координат
 - создать сетку
 - вычислить поле скоростей в каждом узле сетки
 - нарисовать результат
 - изменить количество панелей для улучшения визуализации
-

10 Панельный метод источников

Курс *AeroPython* выходит на финишную прямую! На первых занятиях мы рассмотрели несколько элементарных потенциальных течений и научились использовать всю мощь принципа суперпозиции. И при помощи этих знаний получили некоторые полезные с точки зрения аэродинамики результаты.

При наложении [диполя](#) на равномерный поток получилось обтекание кругового цилиндра, а мы узнали о [парадоксе Д'Аламбера](#): сопротивление цилиндра в потоке жидкости равно нулю. Добавив в центр круга [вихрь](#) мы познакомились с подъёмной силой и [теоремой Кутты-Жуковского](#), согласно которой подъёмная сила пропорциональна циркуляции: $L = \rho U G$. Это важнейший результат!

Сложение элементарных потенциальных течений и интерпретация получившейся картины течения как обтекания твёрдого тела, контуры которого образованы разделительной линией тока, обычно называют *непрямым методом*. Этот метод восходит к Рэнкину, а это 1871 год! Но он имеет ограниченную применимость, поскольку невозможно найти походящее решение для наперед заданной геометрии.

В [девятом занятии](#) мы убедились, что произвольную геометрию можно разбить на панели, поместить на них распределенные источники и подобрать их интенсивности таким образом, чтобы направление потока было касательным к границе тела. Такой подход называется *прямым методом*, он появился в 1960-х в работах Гесса и Смита, выполненных для Douglas Aircraft Company.

Набором панелей (в двумерном случае, отрезков) и связанными с ними слоями источников, интенсивность которых определяется из условия равенства нулю нормальной компоненты скорости, можно представить поверхность любого твёрдого тела, помещённого в потенциальное течение. Это очень мощная идея! Нужно только помнить, что интенсивности панелей связаны между собой, и это приводит к необходимости решать систему линейных уравнений.

Для произвольной геометрии, мы должны построить набор панелей по некоторым точкам, которые определяют геометрию. На этом занятии мы зачитаем из файла геометрию профиля **NACA0012**, создадим набор панелей и решим систему уравнений для определения интенсивностей слоёв источников, что позволит нам получить обтекание профиля.

- Убедитесь, что вы изучили [Занятие 9](#) тщательно прежде чем продолжить! Мы не будем повторять все математические выкладки в этом блокноте, так что обращайтесь к пройденному материалу по мере необходимости.

Для начала загрузим наши любимые библиотеки, а также модуль `integrate` из SciPy:

```
In [1]: import os
        import math
        import numpy
        from scipy import integrate
        from matplotlib import pyplot
        # помещаем графику внутрь блокнота
%matplotlib inline
```

Затем зачитаем геометрию профиля из файла, используя функцию `loadtxt()`. Файл с геометрией был взят с сайта [Airfoil Tools](#) и содержит набор координат стандартного симметричного профиля NACA0012. Локальная копия файла с геометрией находится в директории `resources`.

Точки геометрии загружаются в один массив NumPy, а мы разделим эти данные на два массива: `x, y` для удобства. При помощи следующих строчек кода мы нарисуем профиль.

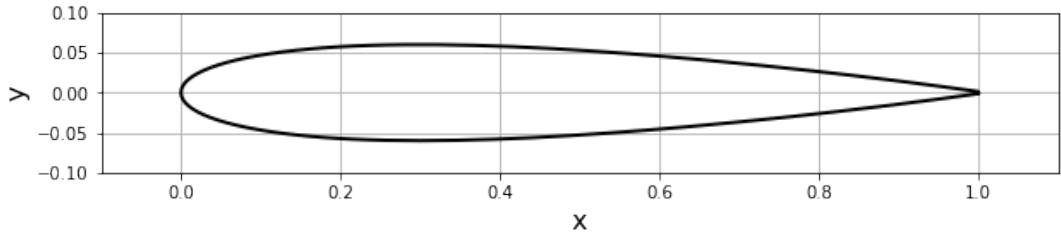
```
In [2]: # зачитываем геометрию из файла с данными
naca_filepath = os.path.join('resources', 'naca0012.dat')
with open(naca_filepath, 'r') as file_name:
    x, y = numpy.loadtxt(file_name, dtype=float,
                         delimiter='\t', unpack=True)

    # рисуем геометрию
width = 10
pyplot.figure(figsize=(width, width))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
```

```

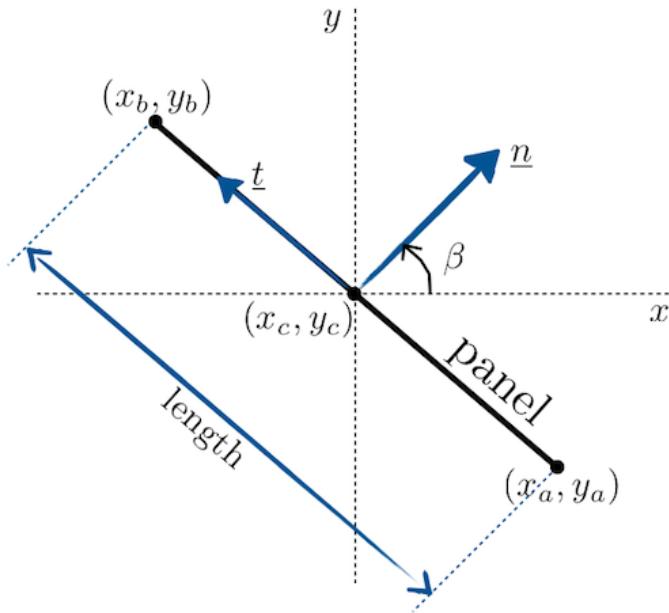
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x, y, color='k', linestyle='-', linewidth=2)
pyplot.axis('scaled', adjustable='box')
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(-0.1, 0.1);

```



10.1 Разбиение на панели

Как и в [Занятии 9](#), создадим разбиение геометрии на панели (в нашем, двумерном, случае — на отрезки). У панелей будут такие характеристики (атрибуты): начальная и конечная точки, середина, длина и ориентация. На рисунке ниже приведены условные обозначения — имена переменных, которые мы будем использовать в коде.



Мы слегка изменим класс `Panel` из предыдущего блокнота, подгоним его под новую задачу — изучение обтекания профиля. Единственное изменение — принадлежность точки к верхней или нижней поверхности обозначим признаками `upper` и `lower`. Это нам пригодится позже, мы сможем нарисовать результаты для верхней и нижней поверхностей разными цветами.

```

In [3]: class Panel:
    """
    Contains information related to a panel.
    """
    def __init__(self, xa, ya, xb, yb):
        """

```

Initializes the panel.

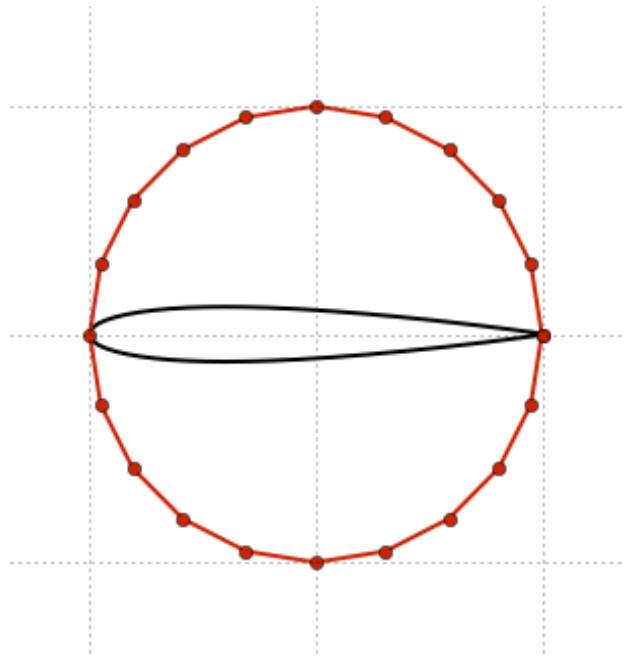
*Sets the end-points and calculates the center, length, and angle (with the x-axis) of the panel.
Defines if the panel is on the lower or upper surface of the geometry.
Initializes the source-sheet strength, tangential velocity, and pressure coefficient to zero.*

Parameters

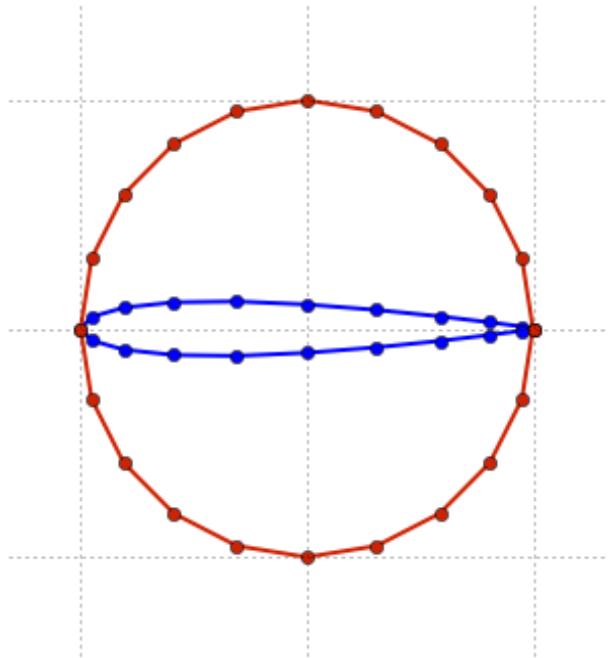
```
xa: float  
    x-coordinate of the first end-point.  
ya: float  
    y-coordinate of the first end-point.  
xb: float  
    x-coordinate of the second end-point.  
yb: float  
    y-coordinate of the second end-point.  
"""  
  
self.xa, self.ya = xa, ya  
self.xb, self.yb = xb, yb  
  
# контрольная точка (центр панели)  
self.xc, self.yc = (xa+xb)/2, (ya+yb)/2  
# длина панели  
self.length = math.sqrt((xb-xa)**2+(yb-ya)**2)  
  
# ориентация панели (угол между нормалью и осью x)  
if xb-xa <= 0.:  
    self.beta = math.acos((yb-ya)/self.length)  
elif xb-xa > 0.:  
    self.beta = math.pi + math.acos(-(yb-ya)/self.length)  
  
# положение панели  
if self.beta <= math.pi:  
    self.loc = 'upper'  
else:  
    self.loc = 'lower'  
  
self.sigma = 0.                      # интенсивность источника  
self.vt = 0.                          # касательная скорость  
self.cp = 0.                           # коэффициент давления
```

В случае кругового цилиндра разбиение на панели выглядело очень просто. Именно эта часть усложняется при переходе к произвольной геометрии, в то время как часть метода, ответственная за решение, остается такой же, как и в [Занятии 9](#).

Функция, приведённая ниже, создает панели, исходя из геометрических данных, которые были прочитаны из файла. На передней и задней кромках, где большая кривизна, лучше сделать панели помельче. Один из способов получения неравномерного распределения панелей по профилю — сначала дискретизировать круг с диаметром, равным хорде профиля, касающейся передней и задней кромок, как показано на следующем рисунке.



Затем, сохранить значения x -координат точек окружности, `x_circle`, которые станут x -координатами узлов панелей, и спроектировать y -координаты точек окружности на профиль при помощи интерполяции. В итоге получится распределение узлов на профиле, со сгущением к передней и задней кромкам. Выглядит это так:



Описанный способ разбиения реализован в функции `define_panels()`, она получает на вход желаемое число панелей и набор координат, а выдает массив объектов класса `Panel`, содержащих всю информацию о каждой панели.

Несколько замечаний о реализации функции `define_panels()`:

- только x -координаты точек окружности (`x_circle`) нужно вычислить, поскольку y -координаты узлов панелей будут определены при помощи интерполяции;

- для построения окружности создается $N+1$ точка, но первая и последняя точки совпадают;
- расширение массивов происходит простым добавлением дополнительных элементов с известными значениями (они равны значениям нулевых элементов), поэтому в циклах элементы $x[i+1]$ не участвуют;
- цикл *while* используется для поиска двух соседних точек на профиле, ($x[I]$, $y[I]$) и ($x[I+1]$, $y[I+1]$), таких, чтобы в интервале $[x[I], x[I+1]]$ лежало значение координаты $x_ends[i]$. Для выхода из цикла используется ключевое слово **break**;
- когда определены две нужные точки, значение $y_ends[i]$ рассчитывается при помощи интерполяции.

```
In [4]: def define_panels(x, y, N=40):
    """
    Discretizes the geometry into panels using the 'cosine' method.

    Parameters
    -----
    x: 1D array of floats
        x-coordinate of the points defining the geometry.
    y: 1D array of floats
        y-coordinate of the points defining the geometry.
    N: integer, optional
        Number of panels;
        default: 40.

    Returns
    -----
    panels: 1D Numpy array of Panel objects
        The discretization of the geometry into panels.
    """

    R = (x.max()-x.min())/2                                # радиус окружности
    x_center = (x.max()+x.min())/2                          # x-координата центра
    # x-координаты точек окружности
    x_circle = x_center + R*numpy.linspace(0, 2*math.pi, N+1)
    # проекция x-координаты на поверхность профиля
    x_ends = numpy.copy(x_circle)
    # инициализация массива с y-координатами
    y_ends = numpy.empty_like(x_ends)
    # добавляем к массиву точки, используя numpy.append()
    x, y = numpy.append(x, x[0]), numpy.append(y, y[0])

    # вычисляем y-координаты концов отрезков
    I = 0
    for i in range(N):
        while I < len(x)-1:
            if (x[I] <= x_ends[i] <= x[I+1]) or \
               (x[I+1] <= x_ends[i] <= x[I]):
                break
            else:
                I += 1
        a = (y[I+1]-y[I])/(x[I+1]-x[I])
        b = y[I+1] - a*x[I+1]
        y_ends[i] = a*x_ends[i] + b
    y_ends[N] = y_ends[0]

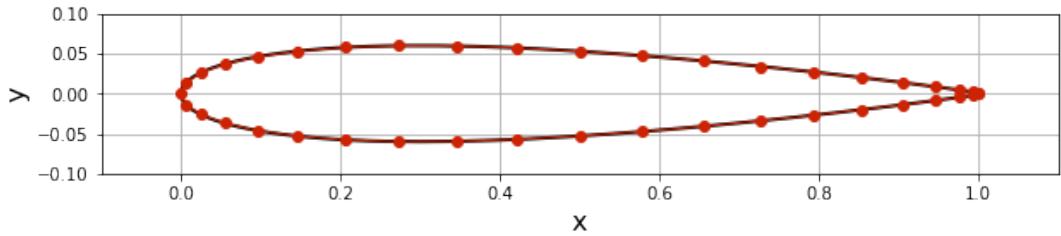
    panels = numpy.empty(N, dtype=object)
    for i in range(N):
        panels[i] = Panel(x_ends[i], y_ends[i], x_ends[i+1], y_ends[i+1])

    return panels
```

Теперь можно воспользоваться только что созданной функцией. В следующем фрагменте кода она вызывается с необходимым числом панелей в качестве параметра. И ещё нарисуем конечную геометрию.

```
In [5]: N = 40 # число панелей
panels = define_panels(x, y, N) # разбиваем геометрию на панели

# рисуем исходную геометрию и панели
width = 10
pyplot.figure(figsize=(width, width))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x, y, color='k', linestyle='-', linewidth=2)
pyplot.plot(numpy.append([panel.xa for panel in panels], panels[0].xa),
            numpy.append([panel.ya for panel in panels], panels[0].ya),
            linestyle='--', linewidth=1, marker='o', markersize=6,
            color='#CD2305')
pyplot.axis('scaled', adjustable='box')
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(-0.1, 0.1);
```



10.2 Параметры набегающего потока

Профиль NACA0012 будет помещён в равномерный поток со скоростью U_∞ и углом атаки $\alpha = 0$. Хотя создание отдельного класса для невозмущённого течения может показаться лишним, мы все равно это сделаем. Создавая класс, мы предполагаем, что понадобится несколько объектов такого класса. Здесь же нам нужен только один набегающий поток, зачем же такие сложности? Вообще, это делает код более читабельным и позволяет программисту использовать имена переменных `u_inf` и `alpha` для чего-то еще за пределами класса. Кроме того, каждый раз, когда нам понадобится передать параметры набегающего потока в качестве входных данных для функции, мы просто можем передать объект в качестве аргумента, а не все параметры течения по отдельности.

```
In [6]: class Freestream:
    """
    Freestream conditions.
    """
    def __init__(self, u_inf=1.0, alpha=0.0):
        """
        Sets the freestream speed and angle (with the x-axis).

        Parameters
        -----
        u_inf: float, optional
            Freestream speed;
            default: 1.0.
        alpha: float, optional
```

```

    Angle of attack in degrees;
    default: 0.0.
"""
self.u_inf = u_inf
self.alpha = alpha*math.pi/180           # градусы --> радианы

In [7]: # задаём параметры набегающего потока
        # и определяем ими объект нужного класса
u_inf = 1.0                                # скорость потока
alpha = 0.0                                   # угол атаки (в градусах)
# инициализация объекта класса freestream
freestream = Freestream(u_inf, alpha)

```

10.3 Граничное условие для потока на поверхности тела

Обеспечение касательности скорости в *контрольных точках* делает контур тела разделительной линией тока в некотором приближении, при этом точность такого приближения увеличивается по мере роста количества панелей. Итак, для каждой панели i нужно обеспечить $u_n = 0$ в точке (x_{c_i}, y_{c_i}) , что приводит к системе уравнений, которую мы вывели на прошлом занятии:

$$u_{n_i} = \frac{\partial}{\partial n_i} \{ \phi(x_{c_i}, y_{c_i}) \} = 0, \quad (14)$$

то есть

$$0 = U_\infty \cos \beta_i + \frac{\sigma_i}{2} + \sum_{j=1, j \neq i}^{N_p} \frac{\sigma_j}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \cos \beta_i + (y_{c_i} - y_j(s_j)) \sin \beta_i}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j \quad (15)$$

В приведённом выше уравнении мы вычисляем производную потенциала по нормали для обеспечения касательности потока на каждой панели. Но позже нам предстоит вычислять еще и производную по касательной для определения коэффициента давления. А если нам потребуется отобразить поле скоростей в узлах расчётной сетки, то нужно будет еще вычислить и производные по x - и y -координатам.

Поэтому функция, приведённая ниже, аналогична реализованной в [Занятии 9](#) для вычисления значений интеграла на каждой панели, но теперь мы обобщили её на случай разных направлений производных (при помощи двух новых аргументов `dxdz` and `dydz`, которые представляют значения $\frac{\partial x_{c_i}}{\partial z_i}$ and $\frac{\partial y_{c_i}}{\partial z_i}$, где z_i — нужное направление).

Более того, новая функция более общая ещё и потому, что позволяет получать значения в любой точке панели, а не только в центре (вместо аргумента `p_i` появились координаты контрольной точки `x` и `y`, а `p_j` заменили на `panel`).

```

In [8]: def integral(x, y, panel, dxdz, dydz):
        """
        Evaluates the contribution of a panel at one point.

        Parameters
        -----
        x: float
            x-coordinate of the target point.
        y: float
            y-coordinate of the target point.
        panel: Panel object
            Source panel which contribution is evaluated.
        dxdz: float
            Derivative of x in the z-direction.
        dydz: float
            Derivative of y in the z-direction.

        Returns
        -----

```

```

Integral over the panel of the influence at the given target point.
"""
def integrand(s):
    return ((x - (panel.xa - math.sin(panel.beta)*s))*dxdz
            +(y - (panel.ya + math.cos(panel.beta)*s))*dydz)
    / ((x - (panel.xa - math.sin(panel.beta)*s))**2
        +(y - (panel.ya + math.cos(panel.beta)*s))**2) )
return integrate.quad(integrand, 0.0, panel.length)[0]

```

10.4 Построение системы линейных уравнений

В этом разделе мы составим и решим систему линейных уравнений в виде

$$[A][\sigma] = [b]. \quad (16)$$

При создании матрицы, мы будем вызывать функцию `integral()` с нужными значениями последних параметров: $\cos \beta_i$ и $\sin \beta_i$, соответствующим производным по нормали.

Наконец, мы воспользуемся `linalg.solve()` из NumPy для решения системы уравнений и определим интенсивность для каждой панели.

```

In [9]: def build_matrix(panels):
    """
    Builds the source matrix.

    Parameters
    -----
    panels: 1D array of Panel object
        The source panels.

    Returns
    -----
    A: 2D Numpy array of floats
        The source matrix (NxN matrix; N is the number of panels).
    """
    N = len(panels)
    A = numpy.empty((N, N), dtype=float)
    numpy.fill_diagonal(A, 0.5)

    for i, p_i in enumerate(panels):
        for j, p_j in enumerate(panels):
            if i != j:
                A[i,j] = 0.5/math.pi*integral(p_i.xc, p_i.yc,\n
                                                p_j, math.cos(p_i.beta),\n
                                                math.sin(p_i.beta))

    return A

def build_rhs(panels, freestream):
    """
    Builds the RHS of the linear system.

    Parameters
    -----
    panels: 1D array of Panel objects
        The source panels.
    freestream: Freestream object
        The freestream conditions.

    Returns
    """

```

```

-----
b: 1D Numpy array of floats
    RHS of the linear system.
"""
b = numpy.empty(len(panels), dtype=float)

for i, panel in enumerate(panels):
    b[i] = -freestream.u_inf * math.cos(freestream.alpha - panel.beta)

return b

In [10]: A = build_matrix(panels)          # вычисляем матрицу особенностей
         b = build_rhs(panels, freestream)  # вычисляем столбец правых частей

In [11]: # решаем систему уравнений
         sigma = numpy.linalg.solve(A, b)

         for i, panel in enumerate(panels):
             panel.sigma = sigma[i]

```

10.5 Коэффициент давления на поверхности

Согласно уравнению Бернуlli, коэффициент давления на i -ой панели равен

$$C_{p_i} = 1 - \left(\frac{u_{t_i}}{U_\infty} \right)^2, \quad (17)$$

где u_{t_i} — касательная компонента скорости в центре i -ой панели,

$$u_{t_i} = -U_\infty \sin \beta_i + \sum_{j=1}^{N_p} \frac{\sigma_j}{2\pi} \int \frac{(x_{c_i} - x_j(s_j)) \frac{\partial x_{c_i}}{\partial t_i} + (y_{c_i} - y_j(s_j)) \frac{\partial y_{c_i}}{\partial t_i}}{(x_{c_i} - x_j(s))^2 + (y_{c_i} - y_j(s))^2} ds_j \quad (18)$$

учитывая, что

$$\frac{\partial x_{c_i}}{\partial t_i} = -\sin \beta_i \quad \text{и} \quad \frac{\partial y_{c_i}}{\partial t_i} = \cos \beta_i \quad (19)$$

Обратите внимание, что ниже мы вызываем функцию `integral()` с другими аргументами: $-\sin \beta_i$ и $\cos \beta_i$, чтобы получить производную по касательной.

```

In [12]: def get_tangential_velocity(panels, freestream):
        """
        Computes the tangential velocity on the surface of the panels.

        Parameters
        -----
        panels: 1D array of Panel objects
            The source panels.
        freestream: Freestream object
            The freestream conditions.
        """

        N = len(panels)
        A = numpy.empty((N, N), dtype=float)
        numpy.fill_diagonal(A, 0.0)

        for i, p_i in enumerate(panels):
            for j, p_j in enumerate(panels):
                if i != j:
                    A[i,j] = 0.5/math.pi*integral(p_i.xc, p_i.yc, p_j,\n                                         -math.sin(p_i.beta),\n                                         math.cos(p_i.beta))

```

```

    b = freestream.u_inf * \
        numpy.sin([freestream.alpha - panel.beta for panel in panels])

    sigma = numpy.array([panel.sigma for panel in panels])

    vt = numpy.dot(A, sigma) + b

    for i, panel in enumerate(panels):
        panel.vt = vt[i]

In [13]: # вычисляем касательную скорость в центрах панелей
    get_tangential_velocity(panels, freestream)

In [14]: def get_pressure_coefficient(panels, freestream):
    """
    Computes the surface pressure coefficients on the panels.

    Parameters
    -----
    panels: 1D array of Panel objects
        The source panels.
    freestream: Freestream object
        The freestream conditions.
    """
    for panel in panels:
        panel.cp = 1.0 - (panel.vt/freestream.u_inf)**2

In [15]: # вычисляем коэффициент давления на поверхности
    get_pressure_coefficient(panels, freestream)

```

10.5.1 Теоретическое решение

Существует классический способ получить теоретические характеристики крыльевых профилей, известный как метод Теодорсена. Метод основан на преобразовании Жуковского, но может применяться к настоящим профилям, так как в нём имеется дополнительное отображение из «почти окружности» в окружность. Метод довольно кучерякий! Но итоговые значения коэффициента давления для некоторых профилей затабулированы в 1945 году в [отчете NACA №824](#) и доступны на сайте NASA. Нужные нам данные находятся на странице 71.

В таблице представлены значения $(u/U_\infty)^2$ в нескольких точках по хорде профиля. Выведем их здесь и сохраним в массив:

```

In [16]: voverVsquared=numpy.array([0, 0.64, 1.01, 1.241,\n
                                         1.378, 1.402, 1.411, \n
                                         1.411, 1.399, 1.378, \n
                                         1.35, 1.288, 1.228, \n
                                         1.166, 1.109, 1.044, 0.956, 0.906, 0])\n
print(voverVsquared)

```

```
[0.      0.64  1.01  1.241 1.378 1.402 1.411 1.411 1.399 1.378 1.35   1.288\n 1.228 1.166 1.109 1.044 0.956 0.906 0.      ]
```

```

In [17]: xtheo=numpy.array([0, 0.5, 1.25, 2.5, 5.0, 7.5,\n
                                         10, 15, 20, 25, 30, 40, 50, 60,\n
                                         70, 80, 90, 95, 100])\n
xtheo = xtheo/100\n
print(xtheo)

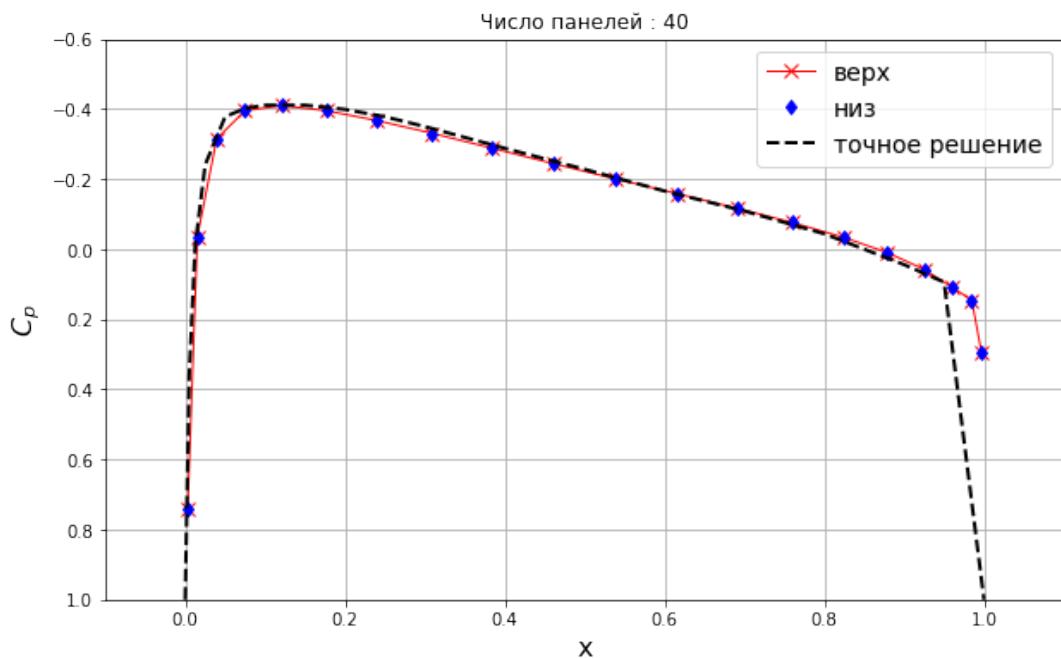
```

```
[0.      0.005 0.0125 0.025 0.05   0.075 0.1     0.15   0.2     0.25\n 0.3     0.4     0.5     0.6     0.7     0.8     0.9     0.95   1.      ]
```

10.5.2 И нарисуем результат!

Воспользуемся данными из отчета NASA (они также опубликованы в книге Эббота и Доэнхорфа "Theory of Wing Sections," 1949), чтобы сопоставить их с результатами расчёта по панельному методу источников. Давайте посмотрим, как это выглядит!

```
In [18]: # рисуем коэффициент давления на поверхности профиля
pyplot.figure(figsize=(10, 6))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('$C_p$', fontsize=16)
pyplot.plot([panel.xc for panel in panels if panel.loc == 'upper'],
            [panel.cp for panel in panels if panel.loc == 'upper'],
            label='верх',
            color='r', linewidth=1, marker='x', markersize=8)
pyplot.plot([panel.xc for panel in panels if panel.loc == 'lower'],
            [panel.cp for panel in panels if panel.loc == 'lower'],
            label='низ',
            color='b', linewidth=0, marker='d', markersize=6)
pyplot.plot(xtheo, 1-voverVsquared,
            label='точное решение',
            color='k', linestyle='--', linewidth=2)
pyplot.legend(loc='best', prop={'size':14})
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(1.0, -0.6)
pyplot.title('Число панелей : %d' % N);
```



Довольно неплохо! Единственное место, где панельный метод не совсем соответствует табличным данным, полученным по методу Теодорсена, это задняя кромка. Но учтите, что граничное условия непротекания в панельном методе выполняется в контрольных точках панели (а не на концах), поэтому такое расхождение не удивительно.

Проверка точности Для замкнутого тела суммарная интенсивность всех источников должна равняться нулю. В противном случае тело добавляло бы в поток какую-то массу, или наоборот, что-то забирало бы. Поэтому, должно выполняться

$$\sum_{j=1}^N \sigma_j l_j = 0,$$

где l_j — длина j -ой панели.

Зная это, можно оценить точность панельного метода источников.

```
In [19]: # вычисляем точность
accuracy = sum([panel.sigma*panel.length for panel in panels])
print('--> суммарная интенсивность источников/стоков: {}'.format(accuracy))

--> суммарная интенсивность источников/стоков: 0.004617031175283105
```

10.6 Линии тока

Чтобы построить линии тока, нужно создать сетку (как мы делали на каждом занятии курса *AeroPython!*) и вычислить значения скоростей в её узлах. Зная интенсивность каждой панели, мы находим x -компоненту скорости, взяв производную от потенциала скорости по переменной x , и y -компоненту путем взятия производной по y :

$$u(x, y) = \frac{\partial}{\partial x} \{\phi(x, y)\}$$

$$v(x, y) = \frac{\partial}{\partial y} \{\phi(x, y)\}$$

Обратите внимание, что при вычислении производных по переменной x мы вызываем функцию `integral()` с 1, 0 в качестве последних аргументов, и 0, 1 — для производных по y .

Кроме того, мы используем функцию `numpy.vectorize()` (как и в [Занятии 8](#)), чтобы избавиться от вложенного цикла по всей расчётной области.

```
In [20]: def get_velocity_field(panels, freestream, X, Y):
    """
    Computes the velocity field on a given 2D mesh.

    Parameters
    -----
    panels: 1D array of Panel objects
        The source panels.
    freestream: Freestream object
        The freestream conditions.
    X: 2D Numpy array of floats
        x-coordinates of the mesh points.
    Y: 2D Numpy array of floats
        y-coordinate of the mesh points.

    Returns
    -----
    u: 2D Numpy array of floats
        x-component of the velocity vector field.
    v: 2D Numpy array of floats
        y-component of the velocity vector field.
    """
    # вклад от набегающего потока
    u = freestream.u_inf * math.cos(freestream.alpha) * \
        numpy.ones_like(X, dtype=float)
    v = freestream.u_inf * math.sin(freestream.alpha) * \
        numpy.ones_like(X, dtype=float)
    # добавляем вклад от каждой панели (суперпозиция рулит!!!)
```

```

vec_integral = numpy.vectorize(integral)
for panel in panels:
    u += panel.sigma / (2.0 * math.pi) * \
        vec_integral(X, Y, panel, 1, 0)
    v += panel.sigma / (2.0 * math.pi) * \
        vec_integral(X, Y, panel, 0, 1)

return u, v

```

In [21]: # определяем параметры сетки

```

nx, ny = 20, 20 # количество узлов в x и y направлениях
x_start, x_end = -1.0, 2.0
y_start, y_end = -0.3, 0.3
X, Y = numpy.meshgrid(numpy.linspace(x_start, x_end, nx),
                      numpy.linspace(y_start, y_end, ny))

# рассчитываем поле скоростей в узлах расчётной сетки
u, v = get_velocity_field(panels, freestream, X, Y)

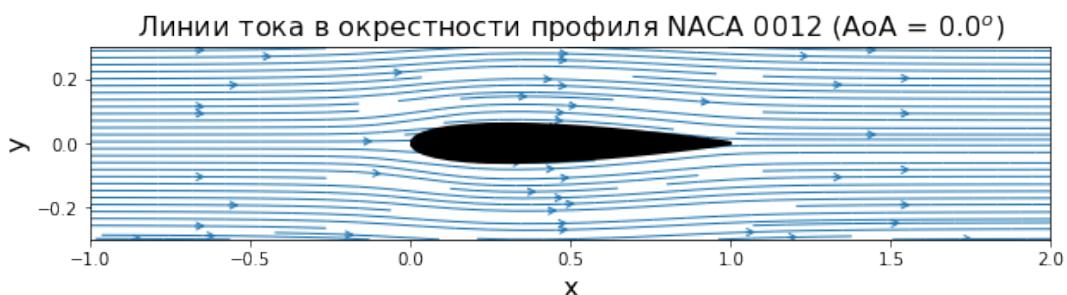
```

In [22]: # рисуем поле скорости

```

width = 10
pyplot.figure(figsize=(width, width))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.streamplot(X, Y, u, v,
                   density=1, linewidth=1, arrowstyle='->')
pyplot.fill([panel.xc for panel in panels],
            [panel.yc for panel in panels],
            color='k', linestyle='solid', linewidth=2, zorder=2)
pyplot.axis('scaled', adjustable='box')
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
title = 'Линии тока в окрестности профиля NACA 0012 (AoA = ${}^{\circ}$)'
pyplot.title(title.format(alpha), fontsize=16);

```



Теперь мы можем рассчитать коэффициент давления. На занятии 9 мы вычислили коэффициент давления на поверхности кругового цилиндра. Это было полезно, потому что у нас есть аналитическое решение для давления на поверхности цилиндра в потенциальном потоке. В случае профиля интересно было бы посмотреть, как выглядит давление в окрестности профиля, поэтому мы построим контурный график в расчётной области.

In [23]: # рассчитываем поле давления

```

cp = 1.0 - (u**2+v**2)/freestream.u_inf**2

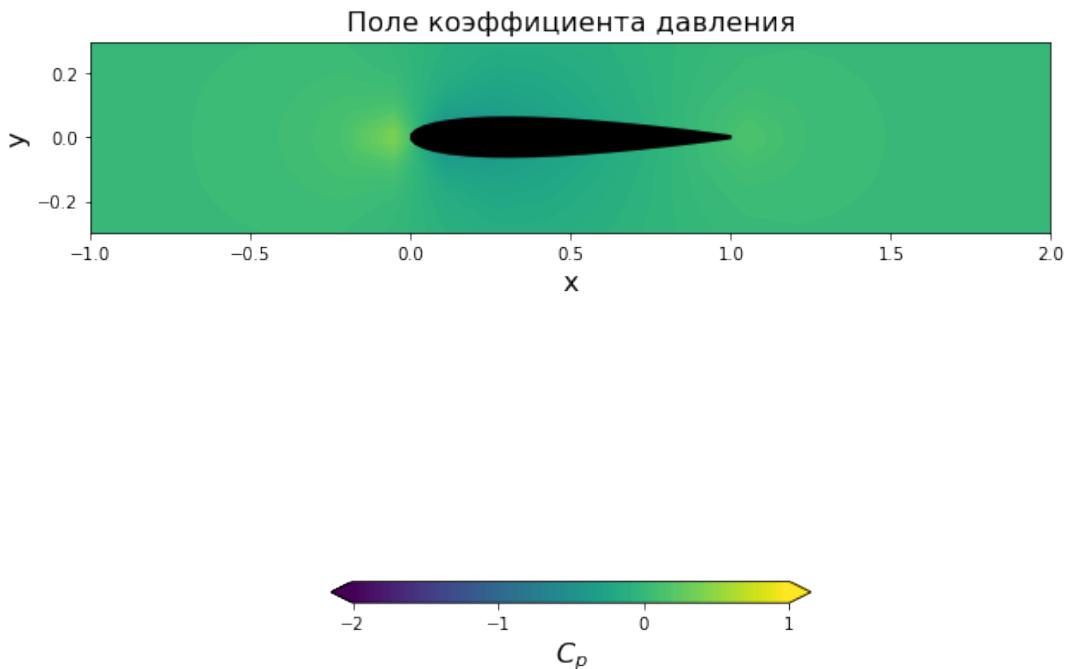
# рисуем поле давления
width = 10

```

```

pyplot.figure(figsize=(width, width))
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
contf = pyplot.contourf(X, Y, cp,
                        levels=numpy.linspace(-2.0, 1.0, 100), \
                        extend='both')
cbar = pyplot.colorbar(contf,
                       orientation='horizontal',
                       shrink=0.5, pad = 0.1,
                       ticks=[-2.0, -1.0, 0.0, 1.0])
cbar.set_label('$C_p$', fontsize=16)
pyplot.fill([panel.xc for panel in panels],
            [panel.yc for panel in panels],
            color='k', linestyle='solid', linewidth=2, zorder=2)
pyplot.axis('scaled', adjustable='box')
pyplot.xlim(x_start, x_end)
pyplot.ylim(y_start, y_end)
pyplot.title('Поле коэффициента давления', fontsize=16);

```



10.7 Заключительные замечания

Мы научились использовать слой источников для представления любого твёрдого тела: сначала [кругового цилиндра](#) (который, как известно, можно было бы получить путем суперпозиции диполя и равномерного потока), а теперь и аэродинамического профиля.

Но в чем особенность профилей, что делает их интересными? Ну, то, что мы можем использовать их для создания подъёмной силы, собственно то, что заставляет их летать, конечно! Но что же нам нужно для создания подъёмной силы? Подумайте, подумайте ... ничего не приходит на ум?

10.8 Литература

1. [Airfoil Tools](#), сайт с данными по профилям.
2. Ira Herbert Abbott, Albert Edward Von Doenhoff and Louis S. Stivers, Jr. (1945), “Summary of Airfoil Data”, NACA Report No.824, [PDF on the NASA web server](#) (см. стр. 71)

3. Ira Herbert Abbott, Albert Edward Von Doenhoff, "Theory of Wing Sections, Including a Summary of Airfoil Data" (1949), Dover Press.

Ссылка на метод Теодорсена:

- Roland Schinzinger, Patricio A. A. Laura (1991), "Conformal Mapping: Methods and Applications." Dover edition in 2003. [Доступно на Google Books](#)
-

11 Панельный метод вихрей-источников

В [девятом занятии](#) курса *AeroPython* вы узнали, как при помощи панельного метода источников получить обтекание кругового цилиндра, а в [десяттом занятии](#) использовали тот же метод для симметричного профиля под нулевым углом атаки. Но что делать, если нужно, чтобы профиль создавал подъёмную силу? Если расположить профиль под углом атаки к набегающему потоку, подъёмная сила обязательно появится, но получим ли мы её, используя панельный метод источников? Поминте [теорему Кутты-Жуковского](#)?

Исторически, первым панельным методом был метод слоя источников. В то время компания Douglas Aircraft была озабочена расчётом течения около тел вращения, и только позднее этот способ был обобщён для несущих поверхностей. (Ниже приведена ссылка на исторический обзор.)

Панельный метод источников дает решение без циркуляции, а значит и без подъёмной силы. Цель этого занятия состоит в том, чтобы, начав с панельного метода источников, реализованного на предыдущем занятии, добавить в него немного *циркуляции*. Будет введено важное понятие — **условие Кутты-Жуковского**, благодаря которому можно определить необходимое значение циркуляции.

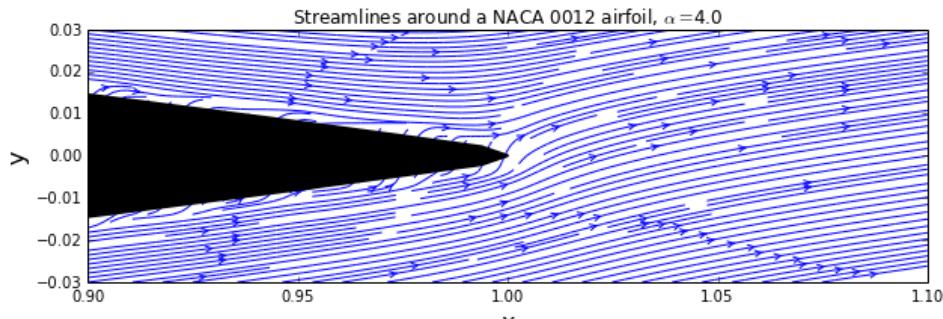
Ссылки

- Smith, A.M.O., The Panel Method: Its Original Development. Глава в *Applied Computational Aerodynamics*, Vol. 125, edited by P.A. Henne, published by AIAA (1990). [Доступно на Google Books](#).

11.1 Панельный метод для несущего тела

Если просто увеличить угол атаки набегающего потока и рассчитать течение панельным методом источников, задняя точка торможения не будет располагаться на задней кромке профиля. Вместо этого, поток будет огибать заднюю кромку, и точка торможения переместится куда-то на верхнюю поверхность профиля. Такое решение нефизично.

Например, если воспользоваться методом источников из [занятия 10](#) с углом атаки $\alpha = 4^\circ$ (используем 40 панелей) и изобразить получившиеся линии тока в районе задней кромки, то мы увидим такую картину:



Как видно, на задней кромке линии тока ведут себя довольно странно. Экспериментально установлено, что поток обтекает заднюю кромку профиля плавно, поэтому полученное нами решение неверно. Что не так? Для расчёта потенциального обтекания профиля под ненулевым углом атаки нельзя использовать только источники — нужна циркуляция. Но как её получить?

11.2 Условие Кутты-Жуковского

Согласно условию Кутты-Жуковского, давление на верхней и нижней поверхностях профиля в районе задней кромки должно быть одинаковым, при этом поток не огибает заднюю кромку, а сходит с неё по касательной. Задняя точка торможения располагается точно на задней кромке профиля.

Это, естественно, слегка озадачивает. Как обосновать такое, казалось бы, произвольное условие? Вспомним, что теория потенциальных течений полностью игнорирует вязкость жидкости, и

поскольку мы оставляем за бортом важный физический эффект, нас не должно смущать, что получившаяся из таких предпосылок теория, требует некоторой настройки в случаях, когда вязкость играет важную роль. В реальности вязкая жидкость не может безотрывно развернуться вокруг остrego угла, как в случае с задней кромкой крыла. Условие Кутты-Жуковского подправляет теорию потенциальных течений таким образом, чтобы решение приблизилось к реальности.

Вспомните [шестое занятие](#), в котором мы изучали подъёмную силу цилиндра, комбинируя диполь с набегающим потоком и вихрем. Тогда мы выучили, что **для создания подъёмной силы всегда нужна циркуляция**. Если вы экспериментировали с величиной циркуляции точечного вихря (а вы экспериментировали, правда же?), то обнаружили, что точки торможения перемещаются по цилиндру.

В случае обтекания профиля, так же как и в случае кругового цилиндра, добавление циркуляции будет перемещать точку торможения по поверхности. И если добавить нужное количество циркуляции, можно сделать так, чтобы точка торможения оказалась на задней кромке профиля. Такая величина циркуляции сделает решение физичным. А также создаст нужную подъёмную силу!

Для реализации условия Кутты-Жуковского в нашем панельном методе нужно добавить в систему уравнений ещё одно уравнение, решение которого позволит получить значение циркуляции, необходимое для перемещения точки торможения на заднюю кромку профиля. Поместив на каждую панель вихревой слой с одинаковой постоянной интенсивностью, мы добавим к течению циркуляцию, при этом получив одну дополнительную неизвестную.

Как это оформить в виде кода? Мы можем повторно использовать большую часть кода из [урока 10](#), и применять условие Кутты-Жуковского при добавлении циркуляции к потоку. Прежде мы разбивали геометрию на N панелей с источниками постоянной интенсивности (которая, однако, менялась от панели к панели) и требовали выполнения граничного условия Неймана для касательной компоненты скорости в каждом из N центров панелей. В результате получалась система N линейных уравнений с N неизвестными, которую мы решали при помощи функции `linalg.solve` библиотеки SciPy. В случае же несущего тела, мы получим $N+1$ уравнение с $N+1$ неизвестной. Читайте дальше, чтобы узнать как мы это сделаем!

11.2.1 Разбиение на панели

Давайте быстро выполним все приготовления. Нужно импортировать наши любимые библиотеки и функцию `integrate` из SciPy, как и в занятии 10.

```
In [1]: # импортируем нужные библиотеки и модули
import os
import numpy
from scipy import integrate, linalg
from matplotlib import pyplot

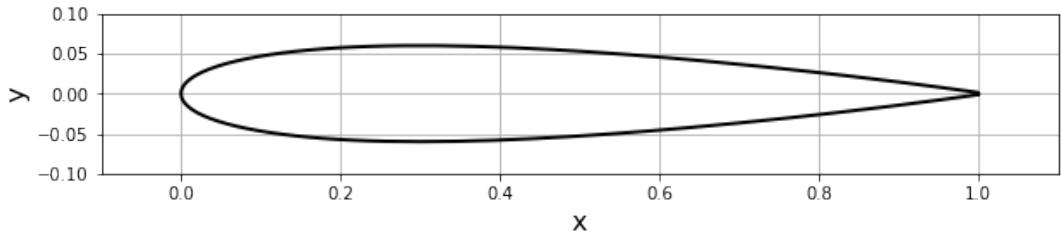
# вставляем графику в блокнот
%matplotlib inline
```

Начнем с зачтывания геометрии профиля NACA0012 из файла с данными и нарисуем получившийся профиль:

```
In [2]: # загружаем геометрию из файла с данными
naca_filepath = os.path.join('resources', 'naca0012.dat')
with open(naca_filepath, 'r') as infile:
    x, y = numpy.loadtxt(infile, dtype=float, unpack=True)
```

```
In [3]: # рисуем геометрию
width = 10
pyplot.figure(figsize=(width, width))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x, y, color='k', linestyle='-', linewidth=2)
pyplot.axis('scaled', adjustable='box')
```

```
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(-0.1, 0.1);
```



Линию, определяющую профиль, разобьём на N панелей, используя тот же подход, что и в занятии 10.

Далее определим класс `Panel`, в котором будет содержаться вся информация о панели: начальная и конечная точки, центр, ориентация, длина, интенсивность источника, касательная компонента скорости и коэффициент давления. Интенсивность вихревого слоя мы хранить не будем, поскольку она одинакова для всех панелей.

```
In [4]: class Panel:
    """
    Contains information related to a panel.
    """
    def __init__(self, xa, ya, xb, yb):
        """
        Initializes the panel.

        Sets the end-points and calculates the center-point, length,
        and angle (with the x-axis) of the panel.
        Defines if the panel is located on the upper or lower surface of the geometry.
        Initializes the source-strength, tangential velocity, and pressure coefficient
        of the panel to zero.

        Parameters
        -----
        xa: float
            x-coordinate of the first end-point.
        ya: float
            y-coordinate of the first end-point.
        xb: float
            x-coordinate of the second end-point.
        yb: float
            y-coordinate of the second end-point.
        """
        self.xa, self.ya = xa, ya # начальная точка панели
        self.xb, self.yb = xb, yb # конечная точка панели

        self.xc, self.yc = (xa+xb)/2, (ya+yb)/2 # центр панели
        self.length = numpy.sqrt((xb-xa)**2+(yb-ya)**2) # длина панели

        # ориентация панели (угол между нормалью и осью x)
        if xb-xa <= 0.0:
            self.beta = numpy.arccos((yb-ya)/self.length)
        elif xb-xa > 0.0:
            self.beta = numpy.pi + numpy.arccos(-(yb-ya)/self.length)

        # расположение панели
```

```

        if self.beta <= numpy.pi:
            self.loc = 'upper' # верхняя поверхность
        else:
            self.loc = 'lower' # нижняя поверхность

        self.sigma = 0.0 # интенсивность источника
        self.vt = 0.0    # касательная скорость
        self.cp = 0.0    # коэффициент давления

```

Как и ранее, для разбиения геометрии профиля на N панелей, вызовем функцию `define_panels()`. Эта функция возвращает массив NumPy, состоящий из N объектов типа `Panel`.

```
In [5]: def define_panels(x, y, N=40):
    """
    Discretizes the geometry into panels using 'cosine' method.

    Parameters
    -----
    x: 1D array of floats
        x-coordinate of the points defining the geometry.
    y: 1D array of floats
        y-coordinate of the points defining the geometry.
    N: integer, optional
        Number of panels;
        default: 40.

    Returns
    -----
    panels: 1D Numpy array of Panel objects.
        The list of panels.
    """

    R = (x.max()-x.min())/2.0          # радиус окружности
    x_center = (x.max()+x.min())/2.0 # x-координата центра

    theta = numpy.linspace(0.0, 2.0*numpy.pi, N+1) # массив углов
    x_circle = x_center + R*numpy.cos(theta)         # x-координаты точек окружности

    x_ends = numpy.copy(x_circle)       # x-координаты конечных точек панелей
    y_ends = numpy.empty_like(x_ends) # y-координаты конечных точек панелей

    # добавляем точку, чтобы получить замкнутую линию
    x, y = numpy.append(x, x[0]), numpy.append(y, y[0])

    # вычисляем y-координаты конечных точек при помощи проецирования
    I = 0
    for i in range(N):
        while I < len(x)-1:
            if (x[I] <= x_ends[i] <= x[I+1]) or (x[I+1] <= x_ends[i] <= x[I]):
                break
            else:
                I += 1
        a = (y[I+1]-y[I])/(x[I+1]-x[I])
        b = y[I+1] - a*x[I+1]
        y_ends[i] = a*x_ends[i] + b
    y_ends[N] = y_ends[0]

    # создаём панели
    panels = numpy.empty(N, dtype=object)
```

```

for i in range(N):
    panels[i] = Panel(x_ends[i], y_ends[i], x_ends[i+1], y_ends[i+1])

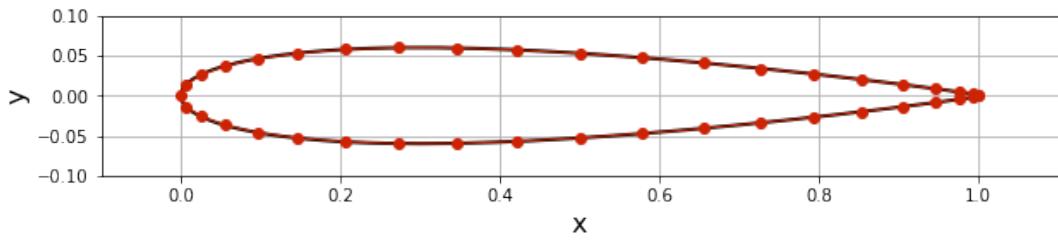
return panels

```

Теперь можно воспользоваться новыми функциями: создать геометрию, разбить её на панели и нарисовать получившееся распределение узлов.

```
In [6]: # разбиваем геометрию на панели
panels = define_panels(x, y, N=40)
```

```
In [7]: # рисуем дискретизированную геометрию
width = 10
pyplot.figure(figsize=(width, width))
pyplot.grid()
pyplot.xlabel('x', fontsize=16)
pyplot.ylabel('y', fontsize=16)
pyplot.plot(x, y, color='k', linestyle='-', linewidth=2)
pyplot.plot(numpy.append([panel.xa for panel in panels], panels[0].xa),
            numpy.append([panel.ya for panel in panels], panels[0].ya),
            linestyle='--', linewidth=1, marker='o', markersize=6, \
            color='#CD2305')
pyplot.axis('scaled', adjustable='box')
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(-0.1, 0.1);
```



11.2.2 Параметры набегающего потока

Профиль помещён в поток с характеристиками (U_∞, α) , где U_∞ и α — скорость на бесконечности и угол атаки соответственно. Как и раньше, мы создадим класс для свободного потока, хоть у нас и будет только один объект этого класса. Зато потом так будет проще передавать параметры набегающего потока в другие функции.

```

In [8]: class Freestream:
    """
    Freestream conditions.
    """
    def __init__(self, u_inf=1.0, alpha=0.0):
        """
        Sets the freestream speed and angle (in degrees).

        Parameters
        -----
        u_inf: float, optional
            Freestream speed;
            default: 1.0.
        alpha: float, optional
            Angle of attack in degrees;

```

```

    default 0.0.
"""
self.u_inf = u_inf
self.alpha = alpha*numpy.pi/180.0 # градусы в радианы

In [9]: # задаем параметры набегающего потока
freestream = Freestream(u_inf=1.0, alpha=4.0)

```

11.2.3 Граничное условие для потока на поверхности тела

К каждой панели добавляется вихрь с интенсивностью γ . Тогда, согласно принципу суперпозиции, потенциал скорости записывается в виде:

$$\begin{aligned}\phi(x_{c_i}, y_{c_i}) &= V_\infty x_{c_i} \cos \alpha + V_\infty y_{c_i} \sin \alpha \\ &+ \sum_{j=1}^N \frac{\sigma_j}{2\pi} \int_j \ln \left(\sqrt{(x_{c_i} - x_j(s_j))^2 + (y_{c_i} - y_j(s_j))^2} \right) ds_j \\ &- \sum_{j=1}^N \frac{\gamma}{2\pi} \int_j \tan^{-1} \left(\frac{y_{c_i} - y_j(s_j)}{x_{c_i} - x_j(s_j)} \right) ds_j\end{aligned}$$

Условие касательности потока записывается для центра каждой панели:

$$0 = \vec{V} \cdot \vec{n}_i = \frac{\partial}{\partial n_i} \{\phi(x_{c_i}, y_{c_i})\},$$

то есть

$$\begin{aligned}0 &= V_\infty \cos(\alpha - \beta_i) + \frac{\sigma_i}{2} \\ &+ \sum_{j=1, j \neq i}^N \frac{\sigma_j}{2\pi} \int_j \frac{\partial}{\partial n_i} \ln \left(\sqrt{(x_{c_i} - x_j(s_j))^2 + (y_{c_i} - y_j(s_j))^2} \right) ds_j \\ &- \sum_{j=1, j \neq i}^N \frac{\gamma}{2\pi} \int_j \frac{\partial}{\partial n_i} \tan^{-1} \left(\frac{y_{c_i} - y_j(s_j)}{x_{c_i} - x_j(s_j)} \right) ds_j\end{aligned}$$

На предыдущем занятии мы уже работали с первым интегралом:

$$\frac{\partial}{\partial n_i} \ln \left(\sqrt{(x_{c_i} - x_j(s_j))^2 + (y_{c_i} - y_j(s_j))^2} \right) = \frac{(x_{c_i} - x_j) \frac{\partial x_{c_i}}{\partial n_i} + (y_{c_i} - y_j) \frac{\partial y_{c_i}}{\partial n_i}}{(x_{c_i} - x_j)^2 + (y_{c_i} - y_j)^2},$$

где $\frac{\partial x_{c_i}}{\partial n_i} = \cos \beta_i$ и $\frac{\partial y_{c_i}}{\partial n_i} = \sin \beta_i$, и

$$x_j(s_j) = x_{b_j} - s_j \sin \beta_j$$

$$y_j(s_j) = y_{b_j} + s_j \cos \beta_j$$

Теперь нам нужно вывести последний интеграл в уравнении для граничного условия:

$$\frac{\partial}{\partial n_i} \tan^{-1} \left(\frac{y_{c_i} - y_j(s_j)}{x_{c_i} - x_j(s_j)} \right) = \frac{(x_{c_i} - x_j) \frac{\partial y_{c_i}}{\partial n_i} - (y_{c_i} - y_j) \frac{\partial x_{c_i}}{\partial n_i}}{(x_{c_i} - x_j)^2 + (y_{c_i} - y_j)^2},$$

где $\frac{\partial x_{c_i}}{\partial n_i} = \cos \beta_i$ и $\frac{\partial y_{c_i}}{\partial n_i} = \sin \beta_i$.

11.2.4 Выполнение условия Кутты-Жуковского

Для выполнения условия Кутты-Жуковского потребуем, чтобы коэффициенты давления на первой и последней панелях совпадали:

$$C_{p_1} = C_{p_N}$$

Исходя из определения коэффициента давления $C_p = 1 - \left(\frac{V}{U_\infty}\right)^2$, условие Кутты-Жуковского подразумевает, что величины скоростей в центрах первой и последней панелей должны совпадать:

$$V_1^2 = V_N^2$$

Поскольку условие непротекания требует, чтобы $V_{n_1} = V_{n_N} = 0$, мы приходим к следующей формулировке *условия Кутты-Жуковского*:

$$V_{t_1} = -V_{t_N}$$

(знак «минус» обусловлен выбором осей для определения нормального и касательного направлений).

Получим выражение для касательных компонент скорости на каждой панели, это потребуется нам позже для вычисления коэффициента давления.

$$V_{t_i} = \frac{\partial}{\partial t_i} (\phi(x_{c_i}, y_{c_i})),$$

то есть

$$\begin{aligned} V_{t_i} &= V_\infty \sin(\alpha - \beta_i) \\ &+ \sum_{j=1, j \neq i}^N \frac{\sigma_j}{2\pi} \int_j \frac{\partial}{\partial t_i} \ln \left(\sqrt{(x_{c_i} - x_j(s_j))^2 + (y_{c_i} - y_j(s_j))^2} \right) ds_j \\ &- \sum_{j=1, j \neq i}^N \frac{\gamma}{2\pi} \int_j \frac{\partial}{\partial t_i} \tan^{-1} \left(\frac{y_{c_i} - y_j(s_j)}{x_{c_i} - x_j(s_j)} \right) ds_j \\ &- \frac{\gamma}{2} \end{aligned},$$

что дает

$$\begin{aligned} V_{t_i} &= V_\infty \sin(\alpha - \beta_i) \\ &+ \sum_{j=1, j \neq i}^N \frac{\sigma_j}{2\pi} \int_j \frac{(x_{c_i} - x_j) \frac{\partial x_{c_i}}{\partial t_i} + (y_{c_i} - y_j) \frac{\partial y_{c_i}}{\partial t_i}}{(x_{c_i} - x_j)^2 + (y_{c_i} - y_j)^2} ds_j \\ &- \sum_{j=1, j \neq i}^N \frac{\gamma}{2\pi} \int_j \frac{(x_{c_i} - x_j) \frac{\partial y_{c_i}}{\partial t_i} - (y_{c_i} - y_j) \frac{\partial x_{c_i}}{\partial t_i}}{(x_{c_i} - x_j)^2 + (y_{c_i} - y_j)^2} ds_j \\ &- \frac{\gamma}{2} \end{aligned}$$

где $\frac{\partial x_{c_i}}{\partial t_i} = -\sin \beta_i$ и $\frac{\partial y_{c_i}}{\partial t_i} = \cos \beta_i$.

11.3 Построение системы линейных уравнений

В этом разделе мы составим и решим систему линейных уравнений в виде

$$[A][\sigma, \gamma] = [b],$$

где матрица $[A]$ размерности $N + 1 \times N + 1$ состоит из трех блоков: матрицы источников размерности $N \times N$ (такая же, как в занятии 10), массива вихрей $N \times 1$, в котором хранятся весовые коэффициенты переменной γ каждой панели и массива $1 \times N + 1$, соответствующего условию Кутты-Жуковского.

Мы собираемся повторно использовать функцию `integral()` из [занятия 10](#) для вычисления различных интегралов с помощью SciPy функции `integrate.quad()`:

```
In [10]: def integral(x, y, panel, dxdk, dydk):
    """
    Evaluates the contribution from a panel at a given point.
    
```

Parameters

```

-----
x: float
    x-coordinate of the target point.
y: float
    y-coordinate of the target point.
panel: Panel object
    Panel whose contribution is evaluated.
dxdk: float
    Value of the derivative of x in a certain direction.
dydk: float
    Value of the derivative of y in a certain direction.

Returns
-----
Contribution from the panel at a given point (x, y).
"""

def integrand(s):
    return ((x - (panel.xa - numpy.sin(panel.beta)*s))*dxdk
            +(y - (panel.ya + numpy.cos(panel.beta)*s))*dydk)
    / ((x - (panel.xa - numpy.sin(panel.beta)*s))**2
        +(y - (panel.ya + numpy.cos(panel.beta)*s))**2) )
return integrate.quad(integrand, 0.0, panel.length)[0]

```

Сначала определим функцию `source_contribution_normal()` для создания матрицы источников. Каждый её элемент вносит вклад в нормальную компоненту скорости каждой панели:

```

In [11]: def source_contribution_normal(pannels):
    """
Builds the source contribution matrix for the normal velocity.

Parameters
-----
pannels: 1D array of Panel objects
    List of panels.

Returns
-----
A: 2D Numpy array of floats
    Source contribution matrix.
"""

A = numpy.empty((pannels.size, pannels.size), dtype=float)
# вклад рассматриваемой панели в общее поле источников
numpy.fill_diagonal(A, 0.5)
# вклад остальных панелей
for i, panel_i in enumerate(pannels):
    for j, panel_j in enumerate(pannels):
        if i != j:
            A[i, j] = 0.5/numpy.pi*integral(panel_i.xc, panel_i.yc,
                                              panel_j,
                                              numpy.cos(panel_i.beta),
                                              numpy.sin(panel_i.beta))
return A

```

Затем, определим `vortex_contribution_normal()` для создания матрицы вихрей:

```

In [12]: def vortex_contribution_normal(pannels):
    """
Builds the vortex contribution matrix for the normal velocity.

```

```

Parameters
-----
panels: 1D array of Panel objects
    List of panels.

Returns
-----
A: 2D Numpy array of floats
    Vortex contribution matrix.
"""

A = numpy.empty((panels.size, panels.size), dtype=float)
# вклад от рассматриваемой панели в поле вихря
numpy.fill_diagonal(A, 0.0)
# вихревой вклад от всех остальных панелей
for i, panel_i in enumerate(panels):
    for j, panel_j in enumerate(panels):
        if i != j:
            A[i, j] = -0.5/numpy.pi*integral(panel_i.xc, panel_i.yc,
                                                panel_j,
                                                numpy.sin(panel_i.beta),
                                                -numpy.cos(panel_i.beta))
return A

```

Вызовем обе эти функции для создания матрицы источников `A_source` и вихрей `B_vortex`:

```
In [13]: A_source = source_contribution_normal(panels)
          B_vortex = vortex_contribution_normal(panels)
```

Остается добавить условие Кутты-Жуковского в нашу систему уравнений.

После этого занятия будет ещё [упражнение](#), которое поможет вам убедиться в том, что:

- матрица, соответствующая вкладам вихрей в нормальную компоненту скорости B^n , **в точности** равна матрице, соответствующей вкладам источников в касательную компоненту скорости A^t , то есть:

$$B_{ij}^n = A_{ij}^t \quad \forall (i, j) \in \{1, \dots, N\}^2$$

- матрица, соответствующая вкладам вихрей в касательную компоненту скорости B^t , **противоположна** по знаку матрице, соответствующей вкладам источников в нормальную компоненту скорости A^n , то есть:

$$B_{ij}^t = -A_{ij}^n \quad \forall (i, j) \in \{1, \dots, N\}^2$$

где индексы n и t обозначают нормальную и касательную компоненты скорости соответственно. Таким образом, условие Кутты-Жуковского можно сформулировать в следующем виде:

$$\begin{bmatrix} (A_{11}^t + A_{N1}^t) \\ \vdots \\ (A_{1N}^t + A_{NN}^t) \\ \left(\sum_{j=1}^N (B_{1j}^t + B_{Nj}^t)\right) \end{bmatrix}^T \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_N \\ \gamma \end{bmatrix} = -(b_1^t + b_N^t)$$

Определим функцию `kutta_condition`, которая сделает всю работу за нас:

```
In [14]: def kutta_condition(A_source, B_vortex):
        """
        Builds the Kutta condition array.
```

Parameters

```

-----
A_source: 2D Numpy array of floats
    Source contribution matrix for the normal velocity.
B_vortex: 2D Numpy array of floats
    Vortex contribution matrix for the normal velocity.

>Returns
-----
b: 1D Numpy array of floats
    The left-hand side of the Kutta-condition equation.
"""

b = numpy.empty(A_source.shape[0]+1, dtype=float)
# матрица вкладов источников в тангенциальную составляющую скорости
# равна
# матрице вкладов вихрей в нормальную составляющую скорости
b[:-1] = B_vortex[0, :] + B_vortex[-1, :]
# матрица вкладов вихрей в тангенциальную составляющую скорости
# противоположна по знаку
# матрице вкладов источников в нормальную составляющую скорости
b[-1] = - numpy.sum(A_source[0, :] + A_source[-1, :])
return b

```

Теперь у нас есть все необходимые ингредиенты. Функция `build_singularity_matrix()` со-берет вместе матрицы источников, вихрей и условия Кутты-Жуковского и сформирует матрицу, необходимую для решения системы линейных уравнений.

```

In [15]: def build_singularity_matrix(A_source, B_vortex):
    """
    Builds the left-hand side matrix of the system
    arising from source and vortex contributions.

>Parameters
-----
A_source: 2D Numpy array of floats
    Source contribution matrix for the normal velocity.
B_vortex: 2D Numpy array of floats
    Vortex contribution matrix for the normal velocity.

>Returns
-----
A: 2D Numpy array of floats
    Matrix of the linear system.
"""

A = numpy.empty((A_source.shape[0]+1, A_source.shape[1]+1), \
                dtype=float)
# матрица источников
A[:-1, :-1] = A_source
# матрица вихрей
A[:-1, -1] = numpy.sum(B_vortex, axis=1)
# условие Кутты-Жуковского
A[-1, :] = kutta_condition(A_source, B_vortex)
return A

```

В правой части системы уравнений содержится все, что не зависит от неизвестных интенсивностей.

```

In [16]: def build_freestream_rhs(panels, freestream):
    """
    Builds the right-hand side of the system

```

arising from the freestream contribution.

Parameters

panels: 1D array of Panel objects

List of panels.

freestream: Freestream object

Freestream conditions.

Returns

b: 1D Numpy array of floats

Freestream contribution on each panel and on the Kutta condition.

"""

```
b = numpy.empty(panels.size+1,dtype=float)
# вклад от набегающего потока в каждую панель
for i, panel in enumerate(panels):
    b[i] = -freestream.u_inf * numpy.cos(freestream.alpha - panel.beta)
    # вклад от набегающего потока в условие Кутты-Жуковского
    b[-1] = -freestream.u_inf*( numpy.sin(freestream.alpha-panels[0].beta)
                                +numpy.sin(freestream.alpha-panels[-1].beta) )
return b
```

```
In [17]: A = build_singularity_matrix(A_source, B_vortex)
         b = build_freestream_rhs(panels, freestream)
```

Как и на предыдущем занятии, систему линейных уравнений будем решать при помощи SciPy функции `linalg.solve()`. Затем мы сохраним результаты в атрибуте `sigma` каждого объектка класса `Panel`. Мы также создадим переменную `gamma`, в которой будет храниться постоянная интенсивность вихря.

```
In [20]: # рассчитываем интенсивности источников и вихря
         strengths = numpy.linalg.solve(A, b)

        # заполняем интенсивности на каждой панели
        for i, panel in enumerate(panels):
            panel.sigma = strengths[i]

        # запоминаем циркуляцию
        gamma = strengths[-1]
```

11.4 Коэффициент давления на поверхности

Коэффициент давления в центре i -ой панели:

$$C_{p_i} = 1 - \left(\frac{V_{t_i}}{U_\infty} \right)^2$$

Итак, нужно вычислить касательную компоненту скорости в центре каждой панели, при помощи функции `compute_tangential_velocity()`:

```
In [21]: def compute_tangential_velocity(panels, freestream, gamma, A_source, B_vortex):
        """
        Computes the tangential surface velocity.

        Parameters
        -----
        panels: 1D array of Panel objects
            List of panels.
        freestream: Freestream object
```

```

Freestream conditions.
gamma: float
    Circulation density.
A_source: 2D Numpy array of floats
    Source contribution matrix for the normal velocity.
B_vortex: 2D Numpy array of floats
    Vortex contribution matrix for the normal velocity.
"""
A = numpy.empty((panels.size, panels.size+1), dtype=float)
# матрица вкладов источников в тангенциальную составляющую скорости
# равна
# матрице вкладов вихрей в нормальную составляющую скорости
A[:, :-1] = B_vortex
# матрица вкладов вихрей в тангенциальную составляющую скорости
# противоположна по знаку
# матрице вкладов источников в нормальную составляющую скорости
A[:, -1] = -numpy.sum(A_source, axis=1)
# добавляем набегающий поток
b = freestream.u_inf*numpy.sin([freestream.alpha_panel.beta
                                 for panel in panels])

strengths = numpy.append([panel.sigma for panel in panels], gamma)

tangential_velocities = numpy.dot(A, strengths) + b

for i, panel in enumerate(panels):
    panel.vt = tangential_velocities[i]

```

In [22]: # касательная компонента скорости на каждой панели
`compute_tangential_velocity(panels, freestream, gamma, A_source, B_vortex)`

И ещё определим функцию `compute_pressure_coefficient()`, чтобы вычислить коэффициент давления на поверхности:

In [23]: `def compute_pressure_coefficient(panels, freestream):`
`"""`
Computes the surface pressure coefficients.
`Parameters`
`-----`
`panels: 1D array of Panel objects`
List of panels.
`freestream: Freestream object`
Freestream conditions.
`"""`
`for panel in panels:`
 `panel.cp = 1.0 - (panel.vt/freestream.u_inf)**2`

In [24]: # коэффициент давления на поверхности
`compute_pressure_coefficient(panels, freestream)`

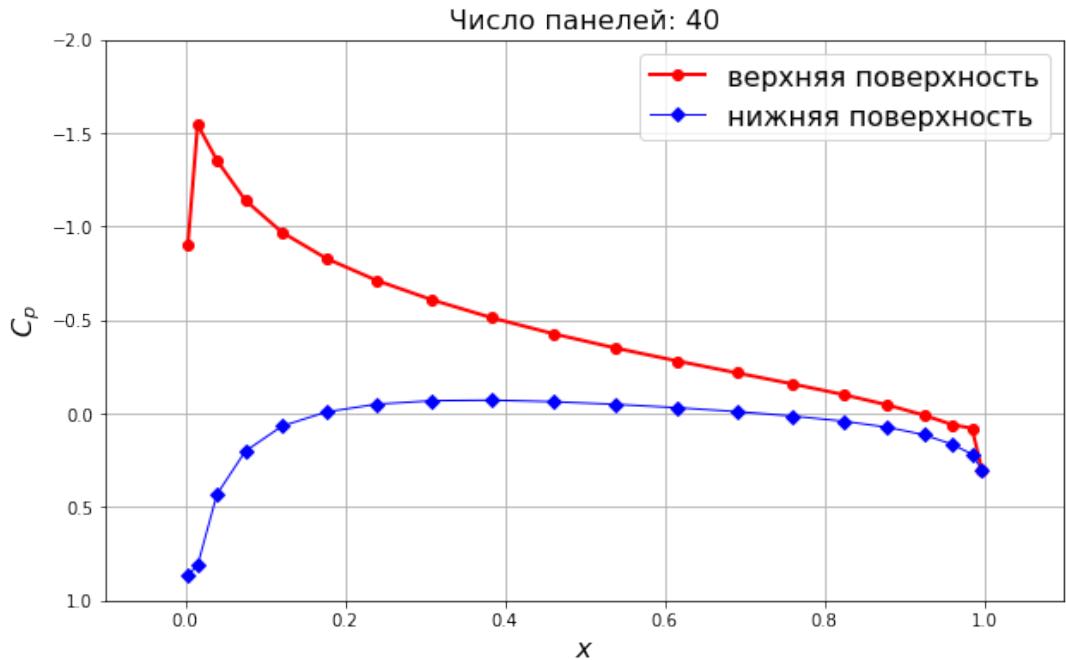
Пора рисовать результат!

In [25]: # рисуем коэффициент давления на поверхности
`pyplot.figure(figsize=(10, 6))`
`pyplot.grid()`
`pyplot.xlabel('x', fontsize=16)`
`pyplot.ylabel('C_p', fontsize=16)`
`pyplot.plot([panel.xc for panel in panels if panel.loc == 'upper'],
 [panel.cp for panel in panels if panel.loc == 'upper'],`

```

label='верхняя поверхность',
color='r', linestyle='-', linewidth=2, marker='o', markersize=6)
pyplot.plot([panel.xc for panel in panels if panel.loc == 'lower'],
            [panel.cp for panel in panels if panel.loc == 'lower'],
            label= 'нижняя поверхность',
            color='b', linestyle='-', linewidth=1, marker='D', markersize=6)
pyplot.legend(loc='best', prop={'size':16})
pyplot.xlim(-0.1, 1.1)
pyplot.ylim(1.0, -2.0)
pyplot.title('Число панелей: {}'.format(panels.size), fontsize=16);

```



11.4.1 Проверка точности

Для замкнутого тела суммарная интенсивность всех источников должна равняться нулю. В противном случае тело добавляло бы в поток какую-то массу, или наоборот, что-то забирало бы. Поэтому, должно выполняться

$$\sum_{i=1}^N \sigma_i l_i = 0$$

где l_i — длина i -ой панели.

Зная это, можно получить оценку точности панельного метода источников.

```
In [26]: # вычисляем точность
accuracy = sum([panel.sigma*panel.length for panel in panels])
print('суммарная интенсивность особенностей: {:.6f}'.format(accuracy))
```

суммарная интенсивность особенностей: 0.004606

11.5 Коэффициент подъёмной силы

Подъёмная сила определяется по теореме Кутты-Жукосского: $L = \rho \Gamma U_\infty$, где ρ — плотность жидкости. Общая циркуляция Γ получается как сумма циркуляций:

$$\Gamma = \sum_{i=1}^N \gamma l_i$$

Наконец, коэффициент подъёмной силы определяется по формуле:

$$C_l = \frac{\sum_{i=1}^N \gamma l_i}{\frac{1}{2} U_\infty c},$$

в которой c — хорда профиля.

```
In [27]: # вычисляем хорду и коэффициент подъёмной силы
c = abs(max(panel.xa for panel in panels)
           - min(panel.xa for panel in panels))
cl = (gamma*sum(panel.length for panel in panels)
      / (0.5*freestream.u_inf*c))
print('коэффициент подъёмной силы: CL = {:.3f}'.format(cl))
```

коэффициент подъёмной силы: CL = 0.506

11.5.1 Контрольное задание

Основываясь на том, что было сделано в предыдущем блокноте, рассчитайте и изобразите линии тока и коэффициент давления на декартовой сетке.

Упражнение: Вывод панельного метода вихрей-источников

Потенциал в точке (x, y) , создаваемый равномерным потоком, слоем источников и вихревым слоем, может быть записан в виде:

$$\begin{aligned}\phi(x, y) = & \phi_{\text{uniform flow}}(x, y) \\ & + \phi_{\text{source sheet}}(x, y) + \phi_{\text{vortex sheet}}(x, y)\end{aligned}\quad (20)$$

То есть

$$\begin{aligned}\phi(x, y) = & xU_\infty \cos(\alpha) + yU_\infty \sin(\alpha) \\ & + \frac{1}{2\pi} \int_{\text{sheet}} \sigma(s) \ln [(x - \xi(s))^2 + (y - \eta(s))^2]^{\frac{1}{2}} ds \\ & - \frac{1}{2\pi} \int_{\text{sheet}} \gamma(s) \tan^{-1} \frac{y - \eta(s)}{x - \xi(s)} ds\end{aligned}\quad (21)$$

где s — локальная координата слоя, а $\xi(s)$ и $\eta(s)$ — координаты бесконечного ряда источников и вихрей, из которых состоят слои. В записанном выше уравнении мы предполагаем, что слои источников и вихрей пересекаются.

Вопрос 1

Пусть слой разбит на N панелей, перепишите уравнение, полученное выше, в дискретном виде. Предположим, что l_j — длина панели j . И что

$$\begin{cases} \xi_j(s) = x_j - s \sin \beta_j \\ \eta_j(s) = y_j + s \cos \beta_j \end{cases}, \quad 0 \leq s \leq l_j \quad (22)$$

На следующей картинке показана панель j :

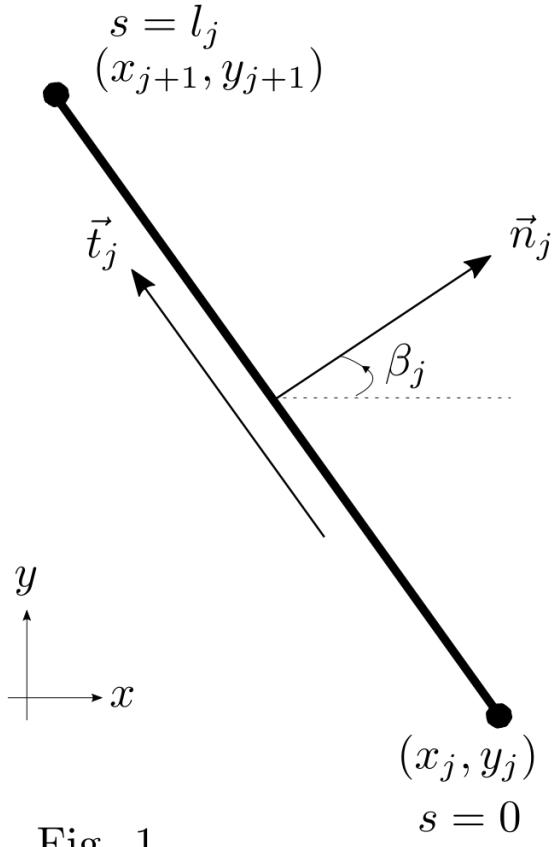


Fig. 1

Подсказка: например, рассмотрим интеграл $\int_0^L f(x)dx$. Если разбить отрезок $0 \sim L$ на три панели, его можно записать в виде:

$$\int_0^L f(x)dx = \int_0^{L/3} f(x)dx + \int_{L/3}^{2L/3} f(x)dx + \int_{2L/3}^L f(x)dx = \sum_{j=1}^3 \int_{l_j} f(x)dx$$

Теперь предположим, что

1. $\sigma_j(s) = \text{constant} = \sigma_j$
 2. $\gamma_1(s) = \gamma_2(s) = \dots = \gamma_N(s) = \gamma$
-

Вопрос 2

Примените изложенные выше предположения к уравнению для $\phi(x, y)$, полученному в Вопросе 1.

Нормальную компоненту скорости U_n можно получить, воспользовавшись правилом дифференцирования сложной функции:

$$\begin{aligned} U_n &= \frac{\partial \phi}{\partial \vec{n}} \\ &= \frac{\partial \phi}{\partial x} \frac{\partial x}{\partial \vec{n}} + \frac{\partial \phi}{\partial y} \frac{\partial y}{\partial \vec{n}} \\ &= \frac{\partial \phi}{\partial x} \nabla x \cdot \vec{n} + \frac{\partial \phi}{\partial y} \nabla y \cdot \vec{n} \\ &= \frac{\partial \phi}{\partial x} n_x + \frac{\partial \phi}{\partial y} n_y \end{aligned} \tag{23}$$

Касательную компоненту можно получить, используя тот же подход. Таким образом, в точке (x, y) можно записать выражения для компонент скорости:

$$\begin{cases} U_n(x, y) = \frac{\partial \phi}{\partial x}(x, y)n_x(x, y) + \frac{\partial \phi}{\partial y}(x, y)n_y(x, y) \\ U_t(x, y) = \frac{\partial \phi}{\partial x}(x, y)t_x(x, y) + \frac{\partial \phi}{\partial y}(x, y)t_y(x, y) \end{cases} \tag{24}$$

Вопрос 3

Используя выписанные выше уравнения, выведите соотношения для $U_n(x, y)$ и $U_t(x, y)$ из уравнения, полученного в Вопросе 2.

Вопрос 4

Рассмотрим нормальную компоненту скорости в центре i -ой панели, то есть в точке $(x_{c,i}, y_{c,i})$. Подставив $(x_{c,i}, y_{c,i})$ вместо (x, y) в уравнении, выведенном в Вопросе 3, можно переписать его в матричном виде:

$$\begin{aligned}
U_n(x_{c,i}, y_{c,i}) &= U_{n,i} \\
&= b_i^n + \begin{bmatrix} A_{i1}^n & A_{i2}^n & \dots & A_{iN}^n \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \end{bmatrix} + \left(\sum_{j=1}^N B_{ij}^n \right) \gamma \\
&= b_i^n + \begin{bmatrix} A_{i1}^n & A_{i2}^n & \dots & A_{iN}^n & \left(\sum_{j=1}^N B_{ij}^n \right) \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \\ \gamma \end{bmatrix}
\end{aligned} \tag{25}$$

$$\begin{aligned}
U_t(x_{c,i}, y_{c,i}) &= U_{t,i} \\
&= b_i^t + \begin{bmatrix} A_{i1}^t & A_{i2}^t & \dots & A_{iN}^t \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \end{bmatrix} + \left(\sum_{j=1}^N B_{ij}^t \right) \gamma \\
&= b_i^t + \begin{bmatrix} A_{i1}^t & A_{i2}^t & \dots & A_{iN}^t & \left(\sum_{j=1}^N B_{ij}^t \right) \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \\ \gamma \end{bmatrix}
\end{aligned} \tag{26}$$

Чему равны b_i^n , A_{ij}^n , B_{ij}^n , b_i^t , A_{ij}^t и B_{ij}^t ?

Учитывая, что (согласно Рис. 1)

$$\begin{cases} \vec{n}_i = n_{x,i}\vec{i} + n_{y,i}\vec{j} = \cos(\beta_i)\vec{i} + \sin(\beta_i)\vec{j} \\ \vec{t}_i = t_{x,i}\vec{i} + t_{y,i}\vec{j} = -\sin(\beta_i)\vec{i} + \cos(\beta_i)\vec{j} \end{cases} \tag{27}$$

получим

$$\begin{cases} n_{x,i} = t_{y,i} \\ n_{y,i} = -t_{x,i} \end{cases}, \text{ or } \begin{cases} t_{x,i} = -n_{y,i} \\ t_{y,i} = n_{x,i} \end{cases} \tag{28}$$

Вопрос 5

Применив вышеуказанные соотношения между \vec{n}_i и \vec{t}_i к ответу на Вопрос 4, найдите соотношения между B_{ij}^n и A_{ij}^t , а также между B_{ij}^t и A_{ij}^n . Наличие таких связей означает, что в коде не нужно вычислять значения B_{ij}^n и B_{ij}^t . Какие это соотношения?

Теперь, обратите внимание, что при $i = j$ в области интегрирования имеется особенность при вычислении A_{ii}^n и A_{ii}^t . Эта особенность возникает при $s = l_i/2$, то есть при $\xi_i(l_i/2) = x_{c,i}$ и $\eta_i(l_i/2) = y_{c,i}$. Это означает, что нужно вывести A_{ii}^n и A_{ii}^t аналитически.

Вопрос 6

Каковы точные значения A_{ii}^n и A_{ii}^t ?

В нашей задаче есть $N + 1$ неизвестных, то есть $\sigma_1, \sigma_2, \dots, \sigma_N, \gamma$. Нам понадобится $N + 1$ линейных уравнений для определения неизвестных. Первые N уравнений можно получить из условия непротекания на каждой панели. То есть

$$U_{n,i} = 0 \\ = b_i^n + \begin{bmatrix} A_{i1}^n & A_{i2}^n & \dots & A_{iN}^n & \left(\sum_{j=1}^N B_{ij}^n\right) \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \\ \gamma \end{bmatrix} \quad (29)$$

, $i = 1 \sim N$

или

$$\begin{bmatrix} A_{i1}^n & A_{i2}^n & \dots & A_{iN}^n & \left(\sum_{j=1}^N B_{ij}^n\right) \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_N \\ \gamma \end{bmatrix} = -b_i^n \quad (30)$$

, $i = 1 \sim N$

Для последнего уравнения воспользуемся условием Кутты-Жуковского.

$$U_{t,1} = -U_{t,N} \quad (31)$$

Вопрос 7

Используйте матрицы из соотношений для $U_{t,i}$ и $U_{t,N}$ для записи условия Кутты-Жуковского и получите последнее линейное уравнение. Перегруппируйте уравнения так, чтобы неизвестные оказались в левой части, а известные величины — в правой.

Вопрос 8

Теперь у вас есть $N + 1$ линейных уравнений для нахождения $N + 1$ неизвестной. Попробуйте скомбинировать первые N линейных уравнений и последнее (то, что получилось при наложении условия Кутты-Жуковского) из ответа на Вопрос 7, чтобы получить полную систему линейных уравнений в матричной форме.

Теперь можно решать уравнения! Это и есть панельный метод вихрей и источников.

Двумерный многозвездный профиль

Перед вами задание к четвертому модулю «**Вихревой панельный метод для несущих тел**» курса *AeroPython*. Вы рассмотрите двумерный многозвездный профиль, или сечение механизированного крыла с выпущенным закрылком, применив панельный метод вихрей и источников из [занятия 11](#) к расчёту обтекания пары профилей.

Главные особенности использования панельного метода для расчёта многозвездных профилей состоят в следующем:

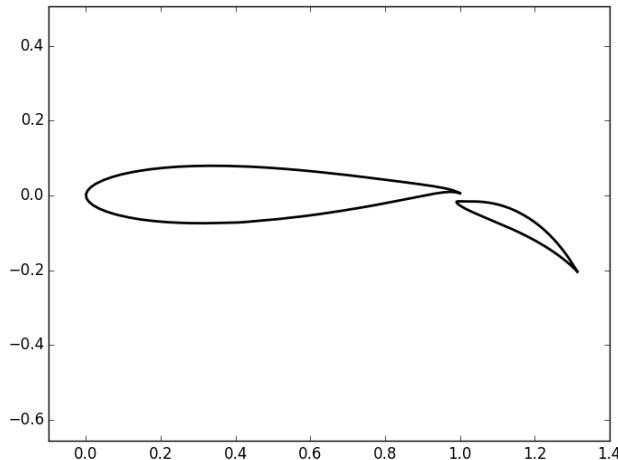
1. каждому профилю соответствует вихрь с собственной интенсивностью,
2. на задней кромке каждого профиля должно выполняться условие Кутты-Жуковского.

В первой части задания мы воспользуемся тестом, для которого существует теоретическое решение, полученное Вильямсом (1973). Посмотрите работу, на которую дана ссылка, и насладитесь полетом математической мысли! Во второй части вы будете иметь дело с более популярным профилем, NACA 23012, который будет использоваться в качестве основного и дополнительного звеньев. Для такой конфигурации есть экспериментальные данные, описанные в отчете Вензингера (1938). Это задание заставит вас задуматься о некоторых интересных, с точки зрения аэродинамики, вопросах.

Часть 1: Тест Вильямса

Ваша задача заключается в вычислении коэффициента давления на поверхностях многоэлементного профиля, состоящего из *основного профиля* и *внешнего закрылка*. Этот тест из работы Вильямса (1973), который получил красивое теоретическое решение при помощи конформных преобразований.

Профиль с выпущенным закрылком выглядит так:



Файлы CSV с x и y координатами узлов каждой панели содержатся в директории `resources` репозитория GitHub. Файлы имеют следующий формат наименования, в котором ******* представляет количество панелей N . `MainFoil_N=***.csv`, `FlapFoil_N=***.csv`.

Для отладки и тестирования можно использовать разбиение на меньшее число панелей, а для более серьезных вычислений — для ответа на вопросы задания — взять более подробное разбиение.

Также имеются два файла с именами `Cp_Main_theoretical.csv` и `Cp_Flap_theoretical.csv`, в которых содержатся теоретические значения — распределение коэффициента давления по оси x для рассматриваемой конфигурации (закрылок отклонен на 30° , основной профиль под нулевым углом атаки).

Теоретические значения коэффициентов подъёмной силы и сопротивления для этого случая: 3.7386 и 0 соответственно (обезразмерены на скоростной напор).

Подсказки

Вспомните [упражнение к занятию 11](#), в котором вас просили вывести математическую постановку панельного метода.

Теперь нужно повторить эти выкладки для двухэлементного профиля, начав с выражения для потенциала:

$$\begin{aligned}\phi(x, y) = & U_\infty x \cos \alpha + U_\infty y \sin \alpha \\ & + \int_{main} \frac{1}{2\pi} \sigma(s) \ln \sqrt{(x - \xi(s))^2 + (y - \eta(s))^2} ds \\ & + \int_{flap} \frac{1}{2\pi} \sigma(s) \ln \sqrt{(x - \xi(s))^2 + (y - \eta(s))^2} ds \\ & - \int_{main} \frac{1}{2\pi} \gamma(s) \tan^{-1} \frac{y - \eta(s)}{x - \xi(s)} ds \\ & - \int_{flap} \frac{1}{2\pi} \gamma(s) \tan^{-1} \frac{y - \eta(s)}{x - \xi(s)} ds\end{aligned}$$

Предположим следующее:

1. $\sigma(s)$ постоянны на каждой панели,
2. $\gamma(s)$ постоянны на каждом профиле,
3. к основному профилю относятся панели с 1-ой по N_a -ю, а к закрылку — панели с $(N_a + 1)$ -й по N -ю, при этом $N = N_a + N_b$, а N_b — это число панелей на закрылке.

Итого получится $N + 2$ неизвестных: $\sigma_1 \dots \sigma_N$, γ_a , и γ_b . Величины γ_a и γ_b соответствуют интенсивностям вихрей основного профиля и закрылка.

У вас должно получиться вывести матрицы для нормальной и касательной компонент скорости на i -ой панели:

$$\begin{aligned}U_i^n = b_i^n + & \left[A_{i1}^n \dots A_{iN}^n, \sum_{j=1}^{N_a} B_{ij}^n, \sum_{j=N_a+1}^N B_{ij}^n \right] \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_N \\ \gamma_a \\ \gamma_b \end{bmatrix} \\ U_i^t = b_i^t + & \left[A_{i1}^t \dots A_{iN}^t, \sum_{j=1}^{N_a} B_{ij}^t, \sum_{j=N_a+1}^N B_{ij}^t \right] \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_N \\ \gamma_a \\ \gamma_b \end{bmatrix}\end{aligned}$$

Используя граничное условие непротекания на панелях с 1-ой по N -ую, вы получите N линейных уравнений. Для нахождения всех $N + 2$ неизвестных нужно ещё два условия Кутты-Жуковского для двух профилей:

$$U_1^t = U_{N_a}^t U_{N_a+1}^t = U_N^t$$

И теперь вы готовы к расчёту потенциального течения вокруг двумерного многозвенного крыла!

Вопросы Запустите ваш код, используя 100 панелей на основном профиле и закрылке и ответьте на следующие вопросы.

Вопрос 1

- Чему равен коэффициент давления на 3-й панели *основного профиля*?

Вопрос 2

- Чему равен коэффициент давления на 10-й панели *закрылка*?

Вопрос 3

- Чему равна подъёмная сила?

Подсказка: $L = - \oint_{main} p \vec{n} \cdot \vec{j} dl - \oint_{flap} p \vec{n} \cdot \vec{j} dl$

Вопрос 4

- Какова величина сопротивления?

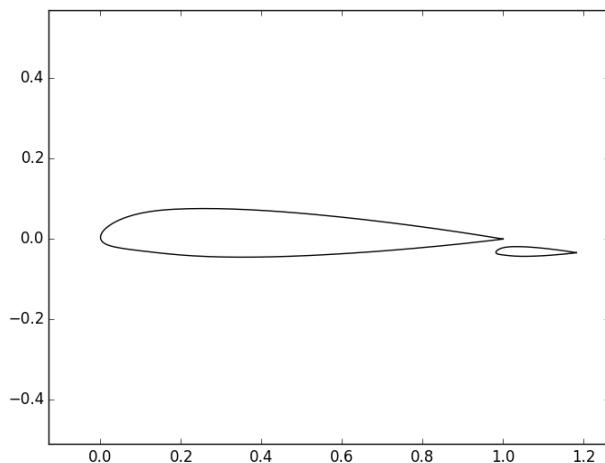
Вопрос 5

- Запустите программу с 200 панелями на основном профиле и закрылке. На сколько процентов уменьшилась ошибка в определении подъёмной силы?

Часть 2: Тест Вензингера

Теперь у вас должен появиться панельный солвер для двухкомпонентного профиля, который можно использовать повторно. Следующий шаг — применить его к более реалистичной задаче. Нам доступны экспериментальные данные (Wenzinger, 1938) для профиля NACA 23012 с закрылком той же формы, с которыми можно сравнить результаты расчётов.

На следующем рисунке показан профиль сечения крыла с 0° отклонением закрылка:



Файлы CSV с x и y координатами узлов каждой панели этого сечения крыла со 150 панелями на каждом профиле: `NACA23012_MainFoil.csv` и `NACA23012_FlapFoil.csv` находятся в директории `resources`.

Точка вращения закрылка расположена в точке $(1.03, -0.054)$ (см. конфигурацию на рисунке выше). Используя координаты точки вращения, можно получить различные конфигурации с разными углами отклонения закрылков.

Попробуйте использовать разные углы отклонения закрылков при различных углах атаки (по хорде основного профиля) и сравните с экспериментальными результатами Вензингера. Напомним, как определяется полный коэффициент подъёмной силы $C_L = L/(l_{main} + l_{flap})$, где L — подъёмная сила (на единицу размаха), как и в рассмотренной выше задаче, а l_{main} и l_{flap} — хорды обоих профилей. **Подумайте, что может быть источником расхождения между вашими результатами и экспериментальными данными.**

Вопросы

Вопрос 1

- Используйте [метод деления отрезка пополам](#) для определения угла атаки α , при котором подъёмная сила равна нулю при отклонении закрылка 20° . Округлите ответ до второго знака после плавающей точки.

Подсказка: можно начать поиск в интервале $-14^\circ \leq \alpha \leq 14^\circ$.

Вопрос 2

- Используйте [метод деления отрезка пополам](#) для определения угла отклонения закрылка, при котором подъёмная сила равна нулю при угле атаки $\alpha = 0^\circ$. Округлите ответ до второго знака после плавающей точки.

Вопрос 3

- Предположим, что основной профиль крепится к фюзеляжу самолета под углом атаки 4° (то есть строительный угол атаки на котором проходит крейсерский полет, равен 4°). Вычислите коэффициенты подъёмной силы с закрылком, отклоненным на 0° , а также на 5° , 10° и 15° . На сколько медленнее (в процентах) будет лететь самолет с выпущенными закрылками, по сравнению со случаем, когда они убраны?

Подумайте Каков физический смысл результата, полученного в **Вопросе 3**? Зачем нужен закрылок?

Ссылки

- B. R. Williams (1973), *An Exact Test Case for the Plane Potential Flow About Two Adjacent Lifting Aerofoils*, Reports & Memoranda No. 3717, Aeronautical Research Council of the United Kingdom // [PDF на сайте Крэнфилда](#)
 - C. J. Wenzinger (1938), *Pressure distribution over an NACA 23012 airfoil with an N.A.C.A. 23012 external-airfoil flap*, NACA Technical Report No.614 // [PDF на сайте NASA](#)
-