Стабилизация платформы при помощи шаговых двигателей

Установка изображена на *рис. 1.* Платформа имеет одну фиксированную точку и две точки, в которых закрепляются шаговые двигатели. Стабилизация работает на принципе P-регулятора. Шаговые двигатели производят смещение только вдоль вертикальной оси. Данная модель симулирует работу двигателей и подсчитывает количество шагов, необходимых до стабилизации платформы. Модель работает корректно при углах Эйлера меньше

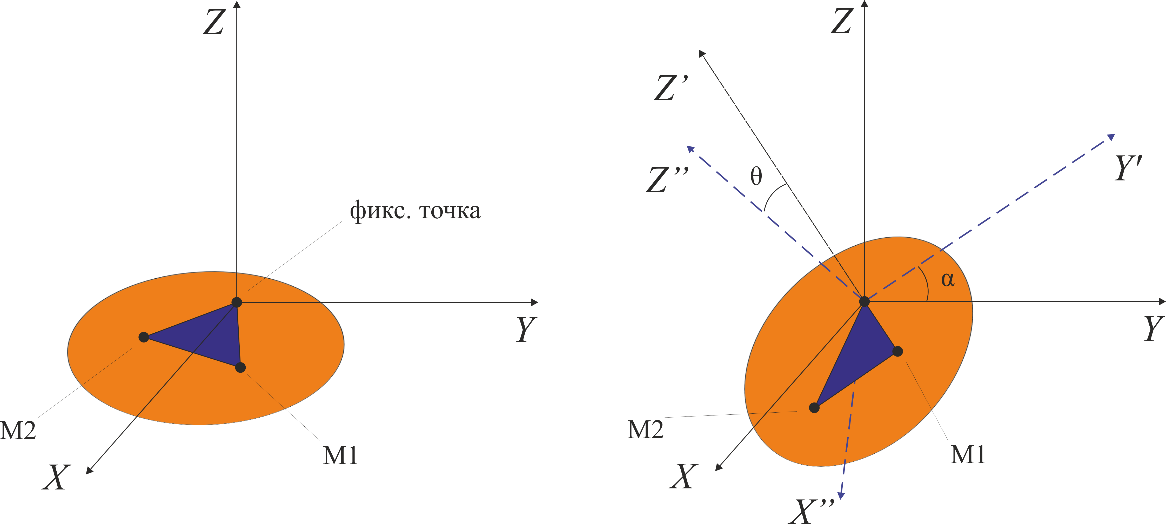


Рис. 1. Схема установки до и после поворота. Оранжевым цветом изображена платформа. Углы Эйлера: крен – α, тангаж – θ.

Реализация

Программа состоит из 3 классов: *EulerToMistake*, *MistakeToOffset*, *OffsetToEngine*.

Алгоритм работы:

1. На вход класса *EulerToMistake* подаются углы Эйлера, которые описывают поворот платформы. Далее считается ошибка: разность между углами Эйлера и положением равновесия.
2. Далее ошибка из класса *EulerToMistake* подается на вход класса *MistakeToOffset*. Метод *computeOffset* вычисляет смещения, которые нужно задать двум двигателям. Логика основана на принципе P-регулятора.
3. Далее вызывается класс *OffsetToEngine*, на вход которого подаются смещения для двигателей и текущая ошибка. В классе *OffsetToEngine* симулируется работа двигателей и после смещения двигателей вычисляется ошибка, которая передается обратно классу *MistakeToOffset*. Затем процесс происходит рекурсивно до тех пор, пока платформа не стабилизируется или наоборот, начнет раскачиваться все сильнее.

Подробнее про реализацию классов и математику

Класс *EulerToMistake*

Конструктор:

**EulerToMistake(vector<double> euler\_coords\_t)** – на вход подаются углы Эйлера;

Методы:

**void computeMistake()** – метод, вычисляющий ошибку;

**vector<double> getEulerCoords() const** – метод, возвращающий углы Эйлера;

**vector<double> getMistake() const** – метод, возвращающий ошибку;

Поля:

**vector<double> euler\_coords** - углы Эйлера, {крен, тангаж}, положительное направление углов выбирается по правилу правой руки;

**vector<double> mistake** - ошибка, {крен, тангаж};

В нашем случае задаются два угла: крен и тангаж, рыскание отсутствует. Крен – поворот плоскости относительно оси , тангаж – поворот плоскости относительно (т.е. ось , после поворота системы координат относительно ). Например, если , то тангаж будет происходить относительно оси , так как после поворота ось будет совпадать с осью . Если то ось будет совпадать с осью .

Положение равновесия считается при углах Эйлера . Другими словами, ошибка – это отрицательные углы Эйлера. Необходимость создания отдельного класса для получения ошибки заключается в том, что в будущем в этом классе будет реализовываться механизм получения углов Эйлера из датчика.

Класс *MistakeToOffset*

Конструктор:

**MistakeToOffset(vector<double> mistake\_t, double P\_t, double I\_t, double D\_t, int step, ostream& output\_t)** – на вход подаются ошибка, коэффициенты P, I, D, шаг, поток вывода;

Методы:

**void computeOffset()**– метод, вычисляющий смещение моторов;

**vector<double> getOffset() const** – метод, возвращающий смещение моторов;

**vector<double> getMistake() const** – метод, возвращающий ошибку;

Поля:

**vector<double> mistake** - ошибка, {крен, тангаж};

**double P, I, D** - коэффициенты PID-регулятора;

**vector<double> offset** - смещение на двух моторах;

**ostream& output** – поток вывода;

**int step** – шаг стабилизации;

В нашем случае задаются . Расположение моторов изображено на *рис. 1.*

Логика вычисления смещения для двух моторов:

Это и есть P-регулятор. Например, если заданы ,, то оба двигателя будут делать смещение вниз. Если ,, то первый двигатель делает смещение вверх, второй двигатель делает смещение вниз.

В методе **computeOffset** вычисляются смещения для моторов по формулам выше и вместе с текущей ошибкой передаются на вход класса *OffsetToEngine.* Далее рекурсивно создается класс *MistakeToOffset* и заданы условия, когда рекурсия должна закончиться: если углы Эйлера меньше , то считается, что платформа стабилизировалась; если хотя бы один угол Эйлера достигает считается, что стабилизация произошла неудачно. Поле **step** увеличивается на единицу на каждый шаг рекурсии, чтобы понять, сколько требуется шагов для стабилизации. В консоль выводится информация о начальных данных: крен, тангаж, коэффициент P и результат стабилизации: успех и количество шагов, или неудача. В текстовый файл записываются данные в формате: *шаг крен тангаж.*

Класс *OffsetToEngine*

Конструктор:

**OffsetToEngine(vector<double> offset\_t, vector<double> old\_mistake)** – на вход подаются смещение и текущая ошибка;

Методы:

**void computeMatrix(vector<double> mistake)** – метод, вычисляющий матрицу поворота;

**void OffsetToMistakeModel(vector<double> old\_mistake)** – метод, вычисляющий ошибку после смещения моторов (модель);

**void OffsetToMistake()** – метод, вычисляющий ошибку после смещения моторов (истинная ошибка, полученная с датчика; метод пока реализован);

**vector<double> getMistake() const** – метод, возвращающий ошибку;

**Matrix getMatrix() const** – метод, возвращающий матрицу поворота;

Поля:

**vector<double> offset** - смещение на двух моторах;

**vector<double> new\_mistake** - ошибка, {крен, тангаж};

**Matrix matrix** – матрица поворота;

Алгоритм работы модели:

1. Задаем изначальное положение двух точек, к которым крепятся двигатели, так, чтобы они с началом координат составляли равносторонний треугольник (*рис.1.*). Длина радиус векторов единичная. Задаем нормаль к платформе в состоянии равновесия.
2. Вычисляем матрицу поворота, которая поворачивает систему координат, связанную с платформой в положении равновесия, вместе с платформой, т.е. новая система координат будет повернута так, как и платформа.

Матрица поворота вычисляется через произведение двух матриц поворота: матрица задает поворот относительно оси , матрица задает поворот относительно оси . и [известны](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%BE%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0) по формулам:

где – угол крена, – угол тангажа, – единичный вектор оси вращения. В нашем случае направлен вдоль оси . Учитывая это, получаем итоговую матрицу :

1. Преобразуем радиус-векторы двух точек в соответствии с поворотом плоскости. Теперь точки у нас находятся там, где и должны быть, если платформа отклонилась от положения равновесия.

Преобразование векторов задается по формуле:

где – радиус-вектор точки, к которой прикреплен мотор.

1. Задаем смещение двух точек по оси , так как моторы могут двигаться только в этом направлении. После этого нормируем радиус-векторы двух точек, так как их длина изначально единичная. Из-за нормировки, реальное смещение меньше, чем то, что мы изначально задали, но логики процесса это не меняет. При углах меньше смещение корректно, так равносторонний треугольник остается равносторонним.
2. Вычисляем через векторное произведение нормаль новой плоскости, построенной уже по двум смещенным радиус-векторам.
3. Зная нормаль, находим новые углы Эйлера. Здесь алгоритм пока работает только для углов от до .

Новые углы Эйлера находятся из решения обратной задачи:

где – нормаль платформы в положении равновесия, – нормаль платформы, смещенной от положения равновесия. Так как новые углы Эйлера

1. Новые углы Эйлера возвращаются в класс *MistakeToOffset* и процесс продолжается рекурсивно, пока платформа не стабилизируется.

Примеры работы программы

Начальные данные: крен = , тангаж = .

Вывод в консоль:

Начальные данные: крен = 30 градусов, тангаж = 30 градусов

P = 0.1

Стабилизация прошла успешно! Количество шагов: 47

P = 0.25

Стабилизация прошла успешно! Количество шагов: 16

P = 0.5

Стабилизация прошла успешно! Количество шагов: 6

P = 0.75

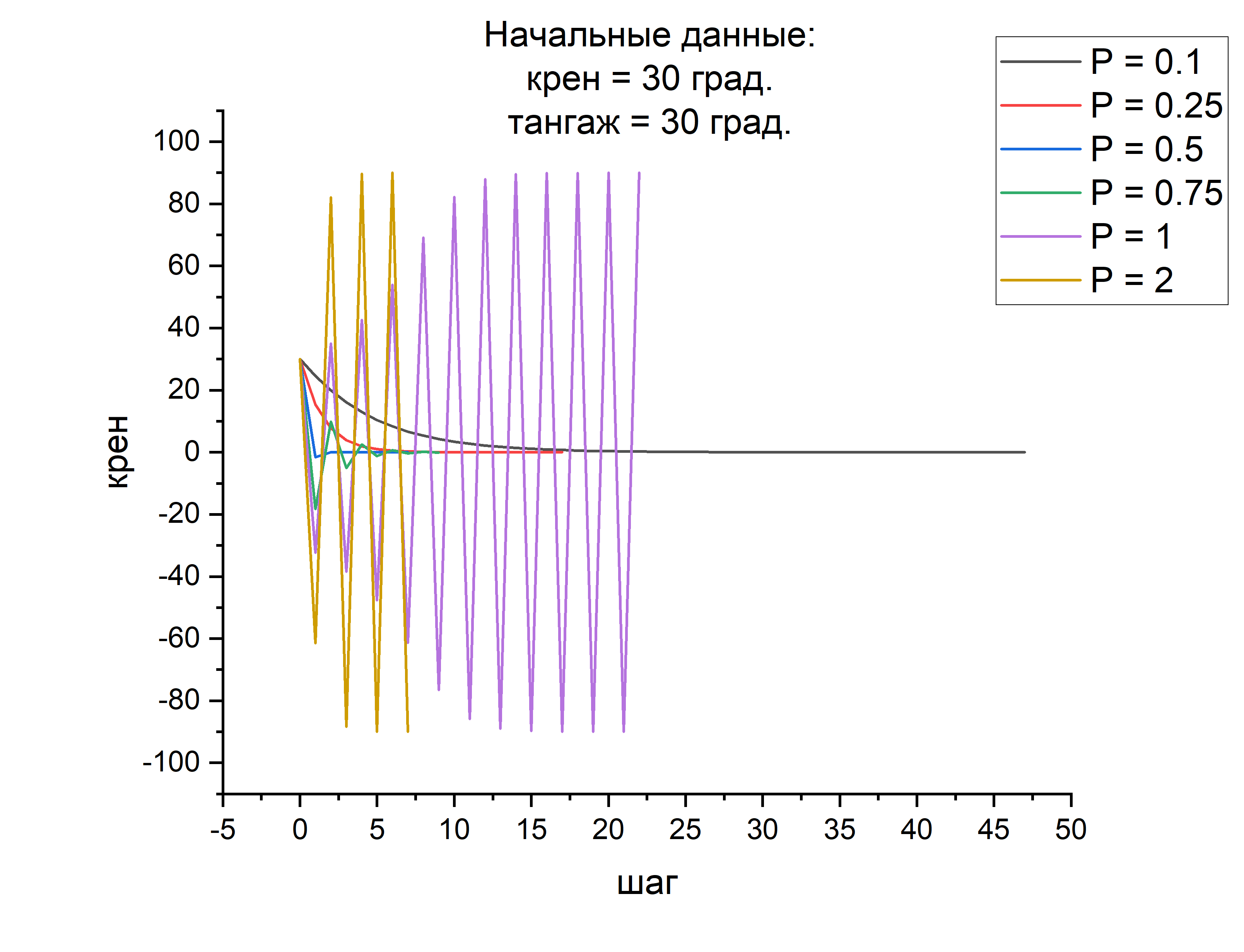
Стабилизация прошла успешно! Количество шагов: 8

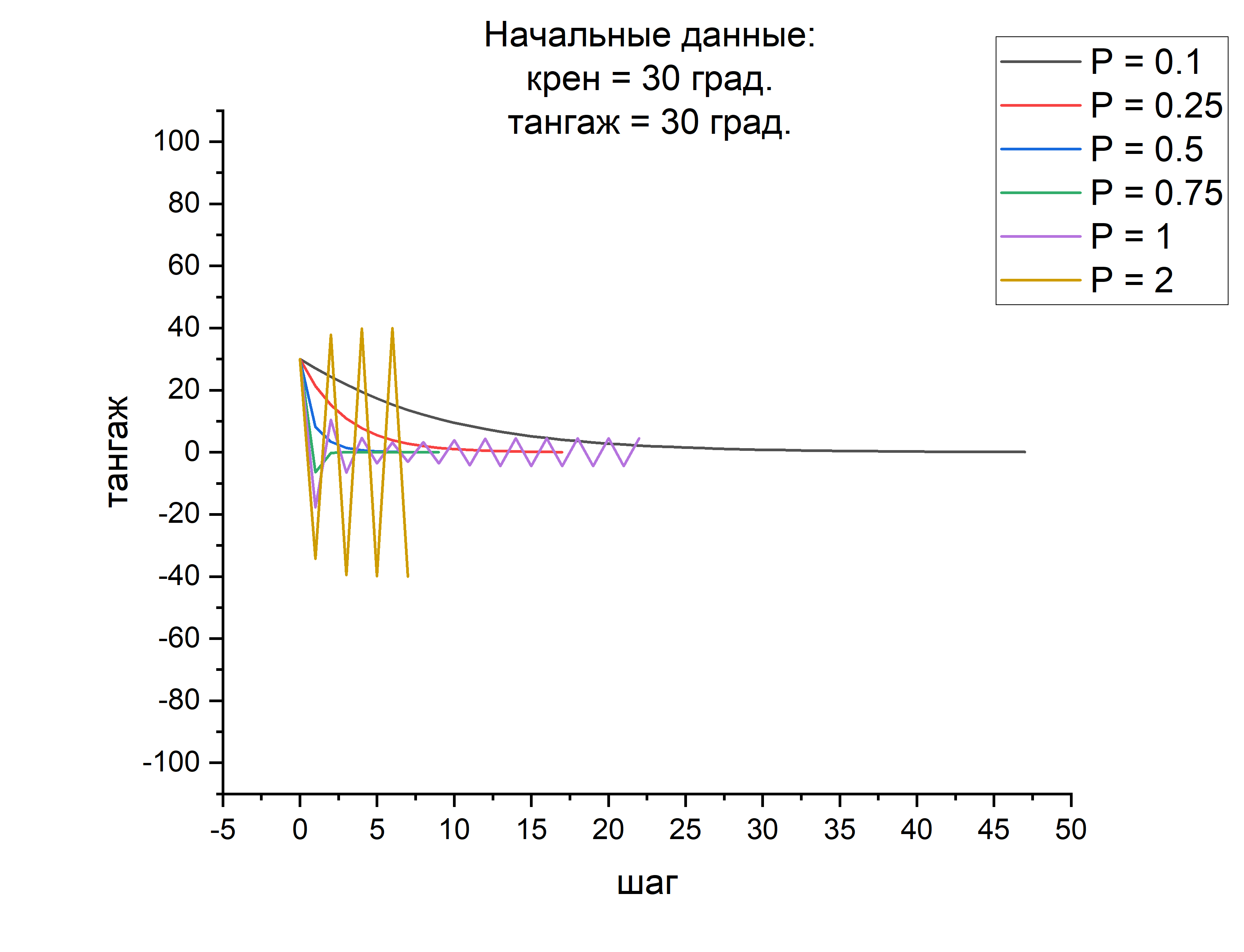
P = 1

Стабилизация прошла неудачно!

P = 2

Стабилизация прошла неудачно!





Начальные данные: крен = 18, тангаж = 12.

Вывод в консоль:

Начальные данные: крен = 18 градусов, тангаж = 12 градусов

P = 0.1

Стабилизация прошла успешно! Количество шагов: 39

P = 0.25

Стабилизация прошла успешно! Количество шагов: 14

P = 0.5

Стабилизация прошла успешно! Количество шагов: 5

P = 0.75

Стабилизация прошла успешно! Количество шагов: 7

P = 1

Стабилизация прошла неудачно!

P = 2

Стабилизация прошла неудачно!

