<https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130>

**Как нарисовать график**

* [Формируем интерфейс](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#формируем-интерфейс)
* [Добавляем обработчик события Paint](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#добавляем-обработчик-события-paint)
* [Рисуем график](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#рисуем-график)
* [Сдвигаем график в центр области](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#сдвигаем-график-в-центр-области)
* [Переворачиваем график](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#переворачиваем-график)
* [Масштабируем график](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#масштабируем-график)
* [Рисуем сетку](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#рисуем-сетку)
* [Решаем проблему перерисовки при изменении размеров формы](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#решаем-проблему-перерисовки-при-изменении-размеров-формы)
* [Сглаживаем график](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#сглаживаем-график)
* [Подключаем учет коэффициентов](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#подключаем-учет-коэффициентов)
* [Итоговый код](https://high.tealeaf.su/how-to-draw-graphic.html?ysclid=liuokz61j7225087130#итоговый-код)

При разработке более сложных приложений, в процессе обработки данных получаются результаты которые содержат много числовых данных. И хотя выводить эти данные в виде строк из циферок можно, но оценивать полученный результат в таком виде не очень удобно.

Поэтому человечество придумало выводить такие данные в виде графиков. Которые в свою очередь эволюционировало в инфографику.

Но для инфографики нам неплохо было бы еще и на художественном поучиться, а у нас времени даже не программирование не очень много получается, так что мы будем просто рисовать график.

Рисовать будем простую штуку, функцию вида

y=k⋅xn

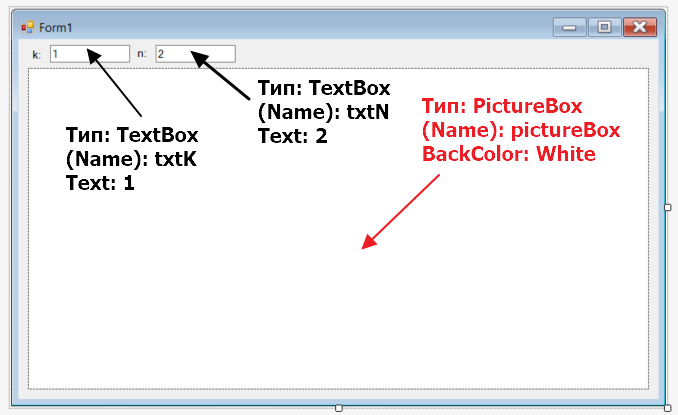
где коэффициенты k, n можно будет поменять.

**Формируем интерфейс**

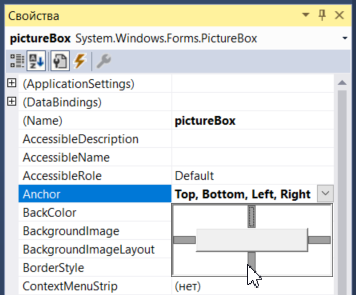
Добавим:

* **TextBox** под коэффициент k, и соответствующий **Label**, свойства установим
  + (Name): **txtK**
  + Text: 1
* **TextBox** под коэффициент n, и соответствующий **Label**, свойства установим
  + (Name): **txtN**
  + Text: 2 – чтоб парабола получилась
* **PictureBox**, на нем будем рисовать график
  + (Name): **pictureBox**
  + BackColor: White – чтобы область под график выглядела белой

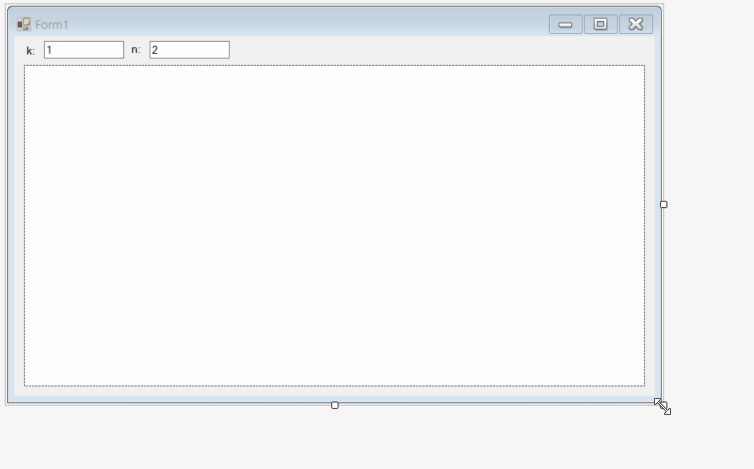
Получим



Чтобы область под график **pictureBox** изменяла свои размеры с изменением размеров формы, установим ей свойство **Anchor** как на картинке:



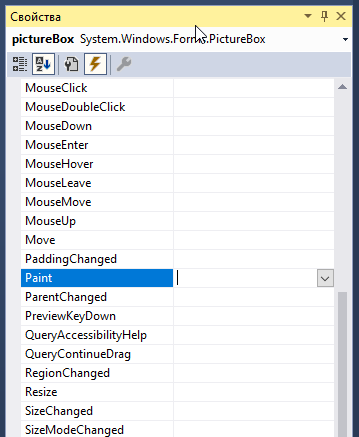
теперь если менять размеры формы, и область под график будет меняться соответственно:



**Добавляем обработчик события Paint**

Всю отрисовку на PictureBox, полагается выполнять внутри функции привязанного к событию Paint

Выбираем на форме **pictureBox** переходим в список событий, и кликаем дважды на свойство **Paint**



оказываемся в новосозданной функции

namespace GraphicApp

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ТОЛЬКО ЧТО СОЗДАННЫЙ ХЭНДЛЕР

}

}

}

**Рисуем график**

Попробуем нарисовать график, пока без учета коэффициентов.

Чтобы нарисовать график нам надо сформировать список точек, подготовим его

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// для отрисовки придется использовать тип данных PointF,

// который представляет собой точку в пространстве,

// с двумя координатами X и Y, поэтому и список инициализируем

// под хранение этого типа

var points = new List<PointF>();

// формируем точки на промежутке [-30; 30]

for(var x=-30;x<=30;++x)

{

// добавляем очередную точку с y = x \* x

points.Add(new PointF(x, x \* x));

}

}

и так, точки у нас есть, теперь их надо нарисовать. Прежде чем вызвать функцию для отрисовки, надо указать как будет выглядеть линия, Для этого надо создать объект типа **Pen** (то бишь перо, ручка), у которого есть два основных параметра: цвет и ширина линии, создадим такой объект:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

var points = new List<PointF>();

for(var x=-30;x<=30;++x)

{

points.Add(new PointF(x, x \* x));

}

// если смотреть на это более высокоуровнево

// тут мы, типа, создали ручку, которой сейчас будем рисовать

var blackPen = new Pen(Color.Black, 1);

}

ну а теперь можно нарисовать график, для этого воспользуемся функций DrawLines, объекта Graphics (так называемый графический контекст устройства) который привязан к аргументу **PaintEventArgs e**

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

var points = new List<PointF>();

for(var x=-30;x<=30;++x)

{

points.Add(new PointF(x, x \* x));

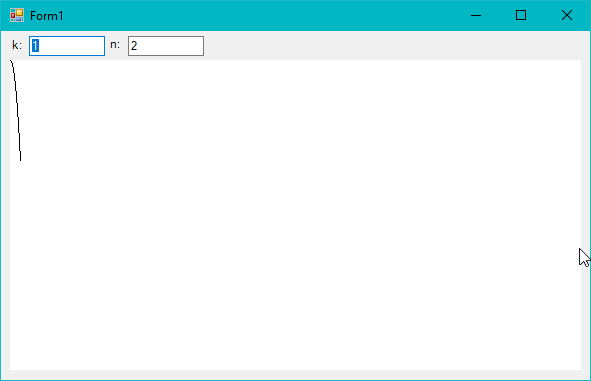
}

var blackPen = new Pen(Color.Black, 1);

e.Graphics.DrawLines(blackPen, points.ToArray());

}

Увидим что-то несуразное:



Если попытаться угадать в этом параболу, то увидим сразу несколько проблем:

* видно только правую ветвь параболы
* парабола перевёрнута
* график сильно маленький

**Сдвигаем график в центр области**

Решить все перечисленные выше проблемы можно используя матрицы перехода [https://ru.wikipedia.org/wiki/Матрица\_перехода](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D0%B0)

Матрицы перехода являются одним из основополагающих математических инструментов в создании 3D графики, который в упрощённом виде работает и в 2D графике (с чем мы собственно сейчас и работаем).

Ко всякому объекту типа **Graphics** привязана матрица переходов (доступная через свойство **Transform**). По умолчанию она представляет собой единичную матрицу. Мы можем изменять матрицу используя методы объекта типа **Graphics**

* **TranslateTransform** – для перемещения центра координат
* **ScaleTransform** – для масштабирования
* **RotateTransform** – для поворота вокруг центра координат

Применяя эти методы в разных порядках можно перемещать, масштабировать и крутить объекты и даже группы объектов малой кровью.

Мы конечно особо ничего крутить ничего не планируем. Но давайте перенесем центр координат в центр **pictureBox**, добавим строчку:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

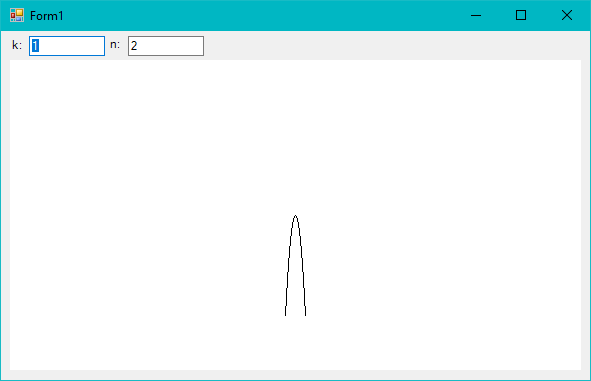
// добавили вызов метода TranslateTransform, у которого два параметра: сдвиг по оси x и y

e.Graphics.TranslateTransform(pictureBox.Width / 2, pictureBox.Height / 2);

e.Graphics.DrawLines(blackPen, points.ToArray());

}

получим:



**Переворачиваем график**

Очевидно, что у параболы с коэффициентом **1**, ветви параболы должны быть направлены вверх. И хотя мы абсолютно верно формируем список точек:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

for(var x=-30;x<=30;++x)

{

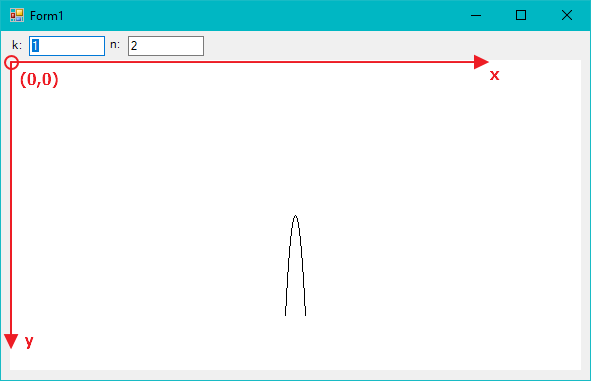
points.Add(new PointF(x, x \* x));

}

// ...

}

ошибка возникает из-за того, что центр координат, у большинства системных объектов, находится в левом верхнем углу:



чтобы перевернуть график воспользуемся функцией **ScaleTransform**

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

e.Graphics.TranslateTransform(pictureBox.Width / 2, pictureBox.Height / 2);

// у функции два параметра, скалирование по x и скалирование по y

// использую 1 для скалирования по x, мы никак не меняем масштаб по x

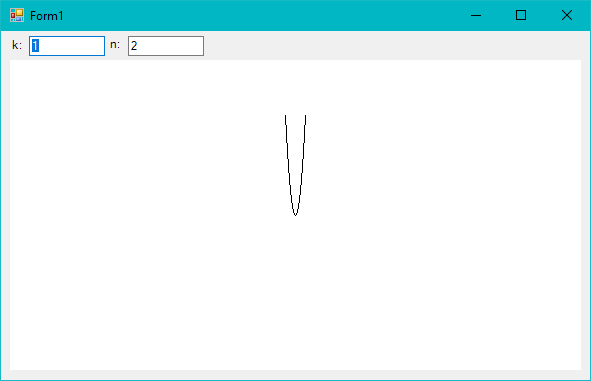
// а вот используя -1 для скалирования по y, мы как раз добивается эффекта переворота

e.Graphics.ScaleTransform(1, -1);

e.Graphics.DrawLines(blackPen, points.ToArray());

}

красота



**Масштабируем график**

И вот вроде все отлично, но график рисуется в пиксельной системе координат. Что на небольших мониторах выглядит еще куда ни шло, но на современных Ultra HD и 4K придется использовать лупу. Мы конечно не хотим заставлять пользователя доставать лупу, но зато мы можем увеличить масштаб графика, снова воспользовавшись функций **ScaleTransform**. Добавим строчку:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

e.Graphics.TranslateTransform(pictureBox.Width / 2, pictureBox.Height / 2);

e.Graphics.ScaleTransform(1, -1);

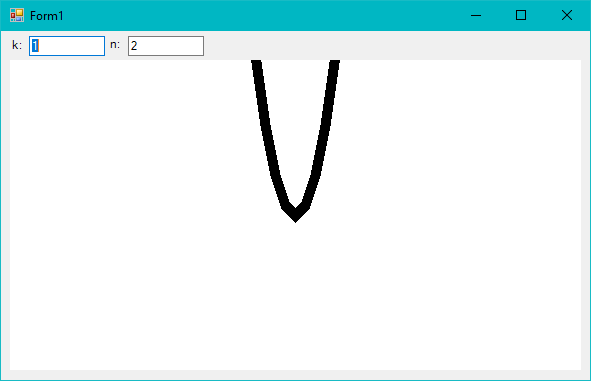
// увеличиваем масштаб

e.Graphics.ScaleTransform(10, 10)

e.Graphics.DrawLines(blackPen, points.ToArray());

}

проверяем:



неплохо, но линия какая-та толстая. Она масштабируется вместе со всем остальным, что не есть хорошо. Чтобы избавится от масштабирования линии придется модифицировать ее матрицу перехода. Да-да, у объекта типа **Pen** тоже есть своя матрица перехода, также доступная через свойство Transform.

Какую же матрицу перехода применять к линии? Очевидно, обратную к матрице основного **Graphics**. Сделаем это:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

e.Graphics.ScaleTransform(10, 10)

// придется склонировать матрицу перехода e.Graphics, используя метода Clone

// а клонировали, так как все объекты в C# ссылочные,

// а метод Invert, изменяет матрицу прям внутри объекта, у которой он был вызван

// и вот чтобы случайно не изменить матрицу перехода e.Graphics

// мы создаем копию исходной матрицы

var penTransform = e.Graphics.Transform.Clone();

penTransform.Invert(); // обращаем матрицу penTransform

// фиксируем матрицу перехода у пера,

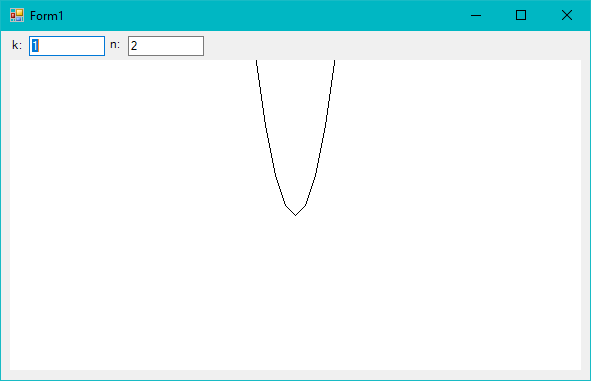
// как обратную к матрице перехода e.Graphics

blackPen.Transform = penTransform;

e.Graphics.DrawLines(blackPen, points.ToArray());

}

вот теперь другое дело



**Рисуем сетку**

Чтобы лучше ориентироваться где какая точка находится нарисуем сетку с размером ячейки в одну единицу

blackPen.Transform = penTransform;

// добавляем серое перо

var grayPen = new Pen(Color.LightGray, 1);

grayPen.Transform = penTransform; // матрица перехода та же, что и у blackPen

for (var x = -10; x <= 10; ++x) // рисуем сетку 10x10

{

var pen = x == 0 ? blackPen : grayPen; // чтобы центральные оси рисовались черным пером

e.Graphics.DrawLine(pen, x, -10, x, 10);

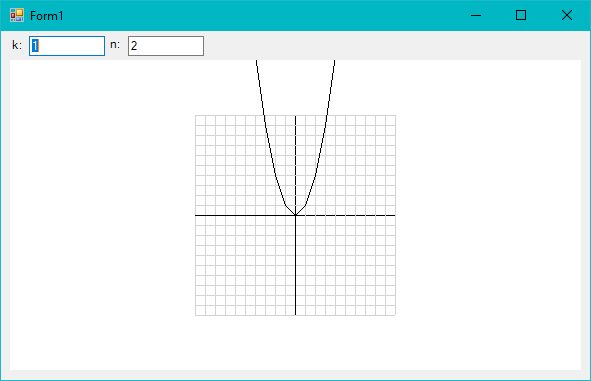
e.Graphics.DrawLine(pen, -10, x, 10, x);

}

e.Graphics.DrawLines(blackPen, points.ToArray());

}

проверяем:



кстати можно сделать чтобы единица на экране соответствовала одному сантиметру в реальной жизни и получить своего рода экранную линейку. Для этого надо воспользоваться свойством DpiX и DpiY объекта Graphics, пробуем. Dpi определяет количество точек на дюйм. В одном дюйме примерно 2.54 сантиметра, следовательно, нам надо отредактировать вызов функции скалирования следующим образом:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// ...

e.Graphics.ScaleTransform(1, -1);

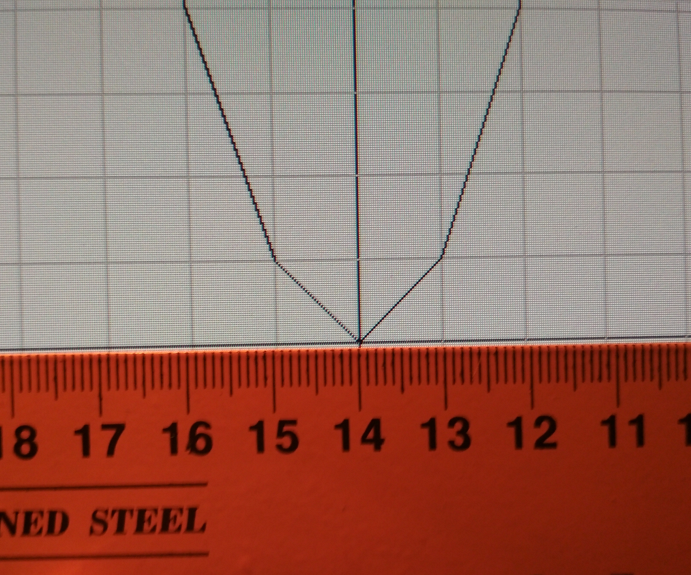
e.Graphics.ScaleTransform(e.Graphics.DpiX / 2.54f, e.Graphics.DpiY / 2.54f); // <<< МЕНЯЕМ ЭТУ СТРОЧКУ

var penTransform = e.Graphics.Transform.Clone();

// ...

}

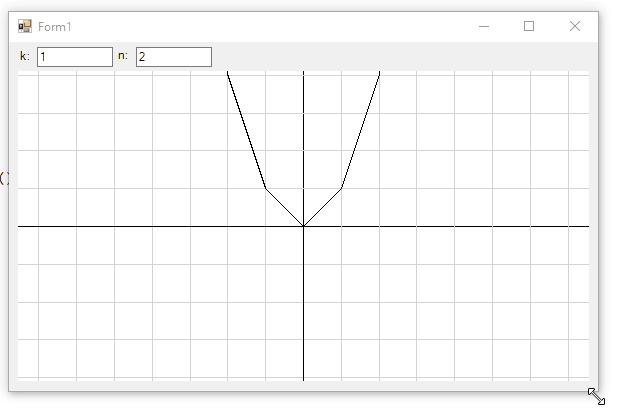
достаем линейку, прикладываем к экрану:



так себе точность, конечно, получается, но результат все-таки радует.

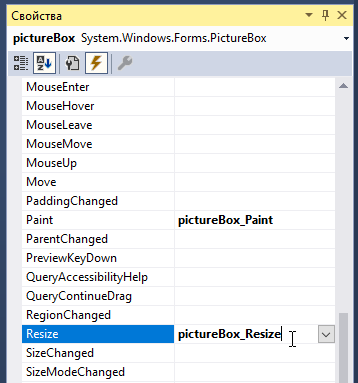
**Решаем проблему перерисовки при изменении размеров формы**

Возможны вы уже пробовали менять размеры формы, и заметили, что получаемый результат далек от ожидаемого.



связано такое поведение с особенностью перерисовки объектов на форме. Перерисовывается только изменённая часть. А так как при разных размерах формы положение графика и сетки разное получается наложение десятка разных изображений, что приводит к таким ужасным артефактам.

Чтобы при изменении размера формы изображение перерисовывалось всегда целиком, добавьте обработчик событию **Resize**, объекта **pictureBox**



и в обработчике вставьте строчку:

private void pictureBox\_Resize(object sender, EventArgs e)

{

// строчка обозначает что при изменении размеров формы

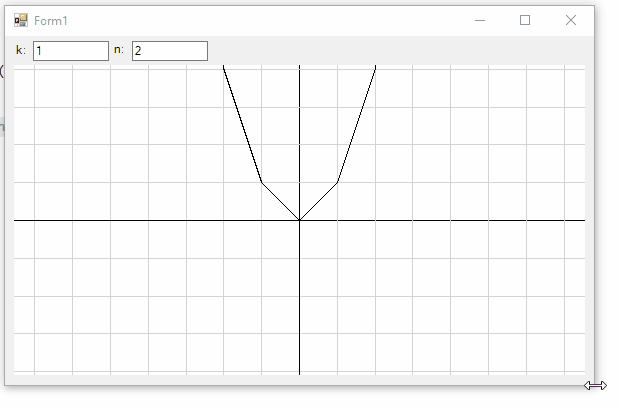
// вся область pictureBox стоит признать невалидной

// и затребовать полную перерисовку объекта

pictureBox.Invalidate();

}

проверяем:



**Сглаживаем график**

Сейчас у нас графи рисуется по целым координатам, из-за чего выглядит слегка уродливо и сильно угловато. Давайте увеличим количество точек. В этот раз, при формировании списка точек, я, вместо цикла, воспользуюсь Linq функциями, которые позволят мне сформировать список точек в функциональной манере с использованием лямбда-выражений.

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

/\*

!!! этот код убираем

var points = new List<PointF>();

for(var x=-10;x<=10;++x)

{

points.Add(new PointF(x, x \* x));

}\*/

var count = 100; // хочу сто точек

var step = 0.1f; // шаг 0.1

var points = Enumerable.Range(0, count) // хитрый метод, вернет массив чисел от 0 до 99

.Select(x => step \* x - step \* count / 2) // пересчитаю x, получится промежуток [-5;5) с шагом 0.1

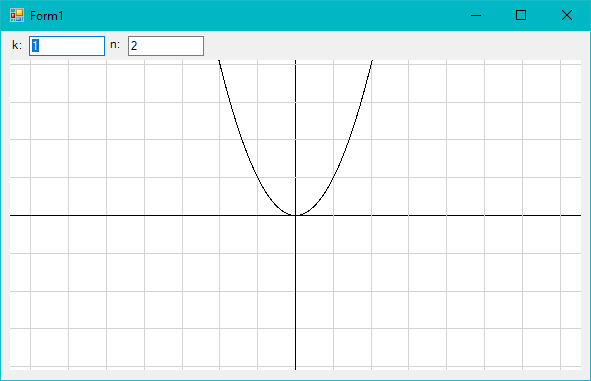
.Select(x => new PointF(x, x \* x)); // считаю y, и формирую массив точек в декартовой системе координат

var blackPen = new Pen(Color.Black, 1);

// ....

}

запускаем, получаем гладенький график:



**Подключаем учет коэффициентов**

Играясь с графиками совсем забыли про коэффициенты. Посчитаем их значения и добавим в учет при формировании точек:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

float k, power;

try

{

k = float.Parse(txtK.Text);

power = float.Parse(txtN.Text);

}

catch (FormatException)

{

return;

}

var count = 100;

var step = 0.1f;

var points = Enumerable.Range(0, count)

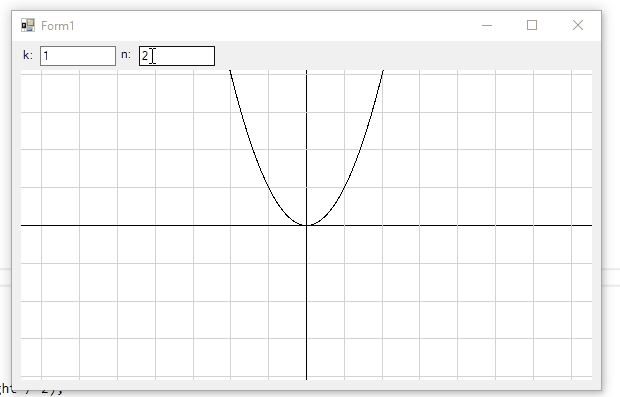
.Select(x => step \* x - step \* count / 2)

.Select(x => new PointF(x, k \* (float) Math.Pow(x, power)));

// ....

}

запускаем:



хм, чего-то не работает…

А ну да, нам же надо чтобы при изменении значения, вызывался метод **pictureBox.Invalidate()**, тот самый которые отправляет запрос на перерисовку всего pictureBox. Переключаемся на форму, и кликаем два разу на **txtK**, а затем на **txtN**, добавляем код в соответствующие обработчики:

private void txtK\_TextChanged(object sender, EventArgs e)

{

pictureBox.Invalidate();

}

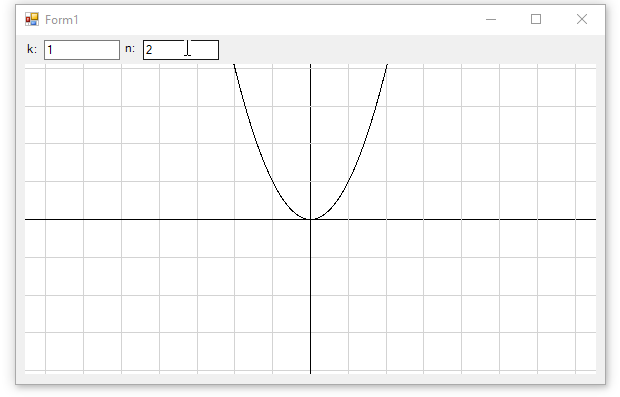
private void txtN\_TextChanged(object sender, EventArgs e)

{

pictureBox.Invalidate();

}

проверяем:



**Итоговый код**

Если вы в какой-то момент запутались, то вот вам итоговый код функции отрисовки:

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

float k, power;

try

{

k = float.Parse(txtK.Text);

power = float.Parse(txtN.Text);

}

catch (FormatException)

{

return;

}

var count = 100;

var step = 0.1f;

var points = Enumerable.Range(0, count)

.Select(x => step \* x - step \* count / 2)

.Select(x => new PointF(x, k \* (float)Math.Pow(x, power)));

var blackPen = new Pen(Color.Black, 1);

e.Graphics.TranslateTransform(pictureBox.Width / 2, pictureBox.Height / 2);

e.Graphics.ScaleTransform(1, -1);

e.Graphics.ScaleTransform(e.Graphics.DpiX / 2.54f, e.Graphics.DpiY / 2.54f);

var penTransform = e.Graphics.Transform.Clone();

penTransform.Invert();

blackPen.Transform = penTransform;

var grayPen = new Pen(Color.LightGray, 1);

grayPen.Transform = penTransform;

for (var x = -10; x <= 10; ++x)

{

var pen = x == 0 ? blackPen : grayPen; //

e.Graphics.DrawLine(pen, x, -10, x, 10);

e.Graphics.DrawLine(pen, -10, x, 10, x);

}

e.Graphics.DrawLines(blackPen, points.ToArray());

}

Конец.