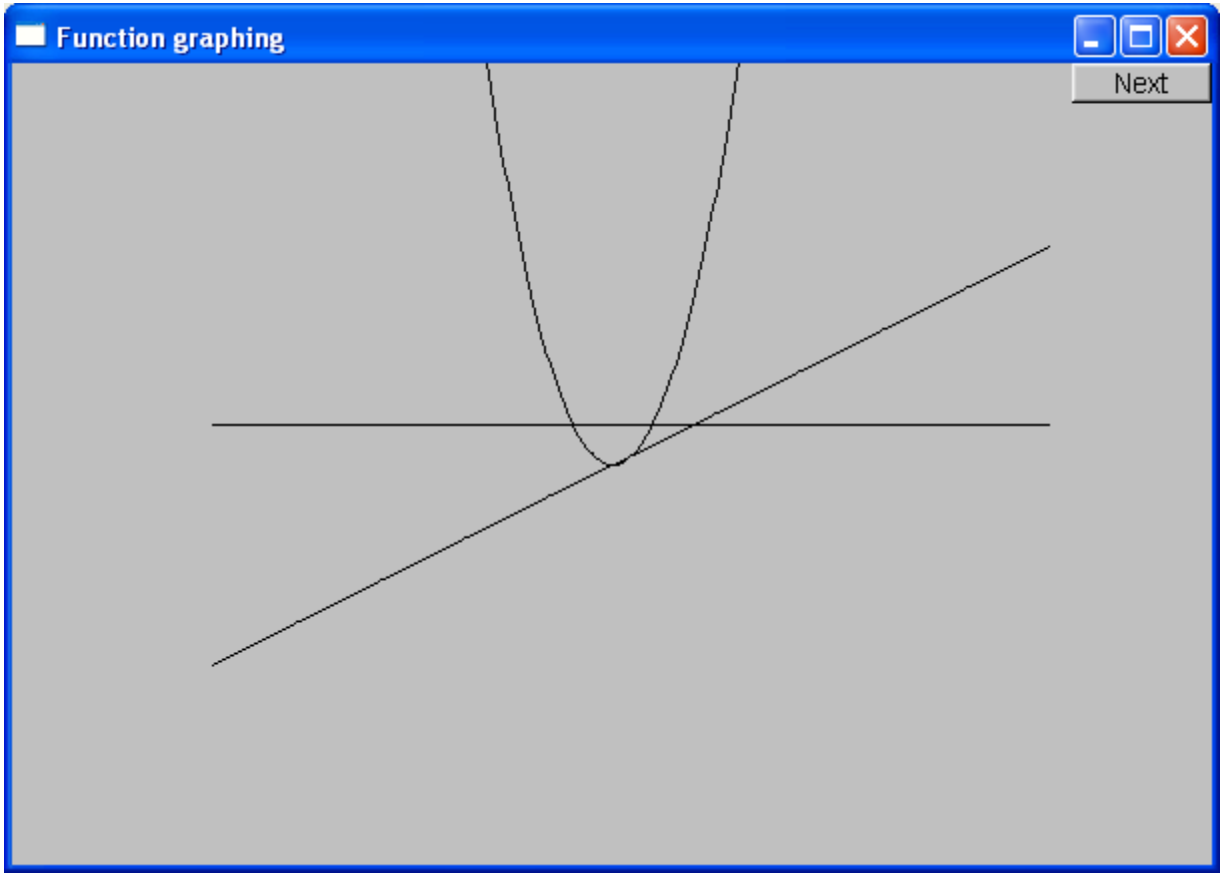
Исцртавање функција

Три једноставне функције




```
double one(double x) { return 1; }        // y==1
double slope(double x) { return x/2; }    // y==x/2
double square(double x) { return x*x; }   // y==x*x
```

Како их исцртавамо?

Функција коју исцртавамо

```
Simple_window win0(Point(100,100),xmax,ymax,"Function graphing");

Function s(one, -10,11, orig, n_points, x_scale,y_scale);
Function s2(slope, -10,11, orig, n_points, x_scale,y_scale);
Function s3(square, -10,11, orig, n_points, x_scale,y_scale);

win0.attach(s);
win0.attach(s2);
win0.attach(s3);

win0.wait_for_button( );
```

Број тачака

Координатни почетак

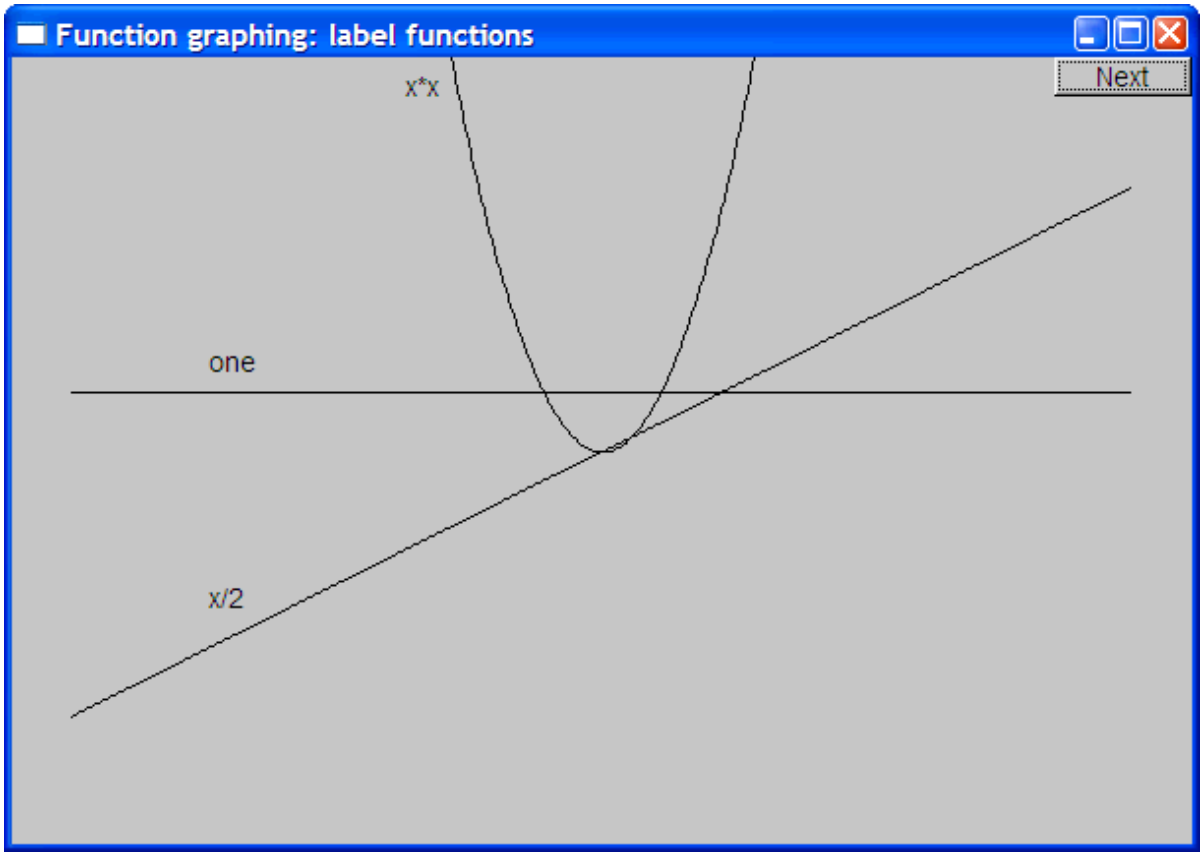
Тиче се приказа, тј. смештања у ограничени простор прозора

Опсег у којем исцртавамо [x0:xN)

Корисне константе

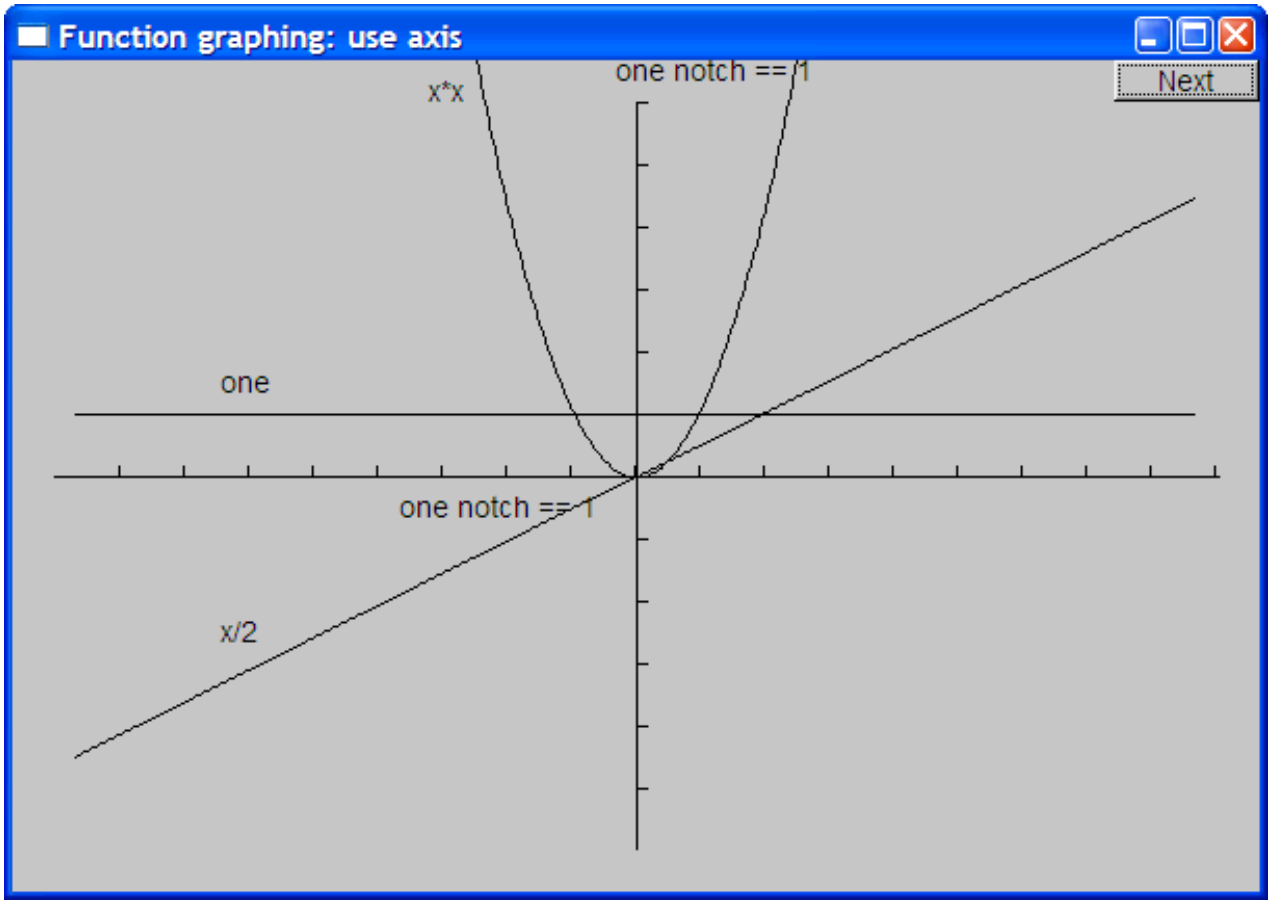
```
const int xmax = win0.x_max();    // величина прозора (600 пута 400)
const int ymax = win0.y_max();

const int x_orig = xmax / 2;
const int y_orig = ymax / 2;
const Point orig(x_orig, y_orig); // координатни почетак у прозору

const int r_min = -10;       // опсег [-10:11) по x
const int r_max = 11;

const int n_points = 400;    // број тачака који се користи за приказ

const int x_scale = 20;      // коефицијенти скалирања
const int y_scale = 20;
```

Означимо функције



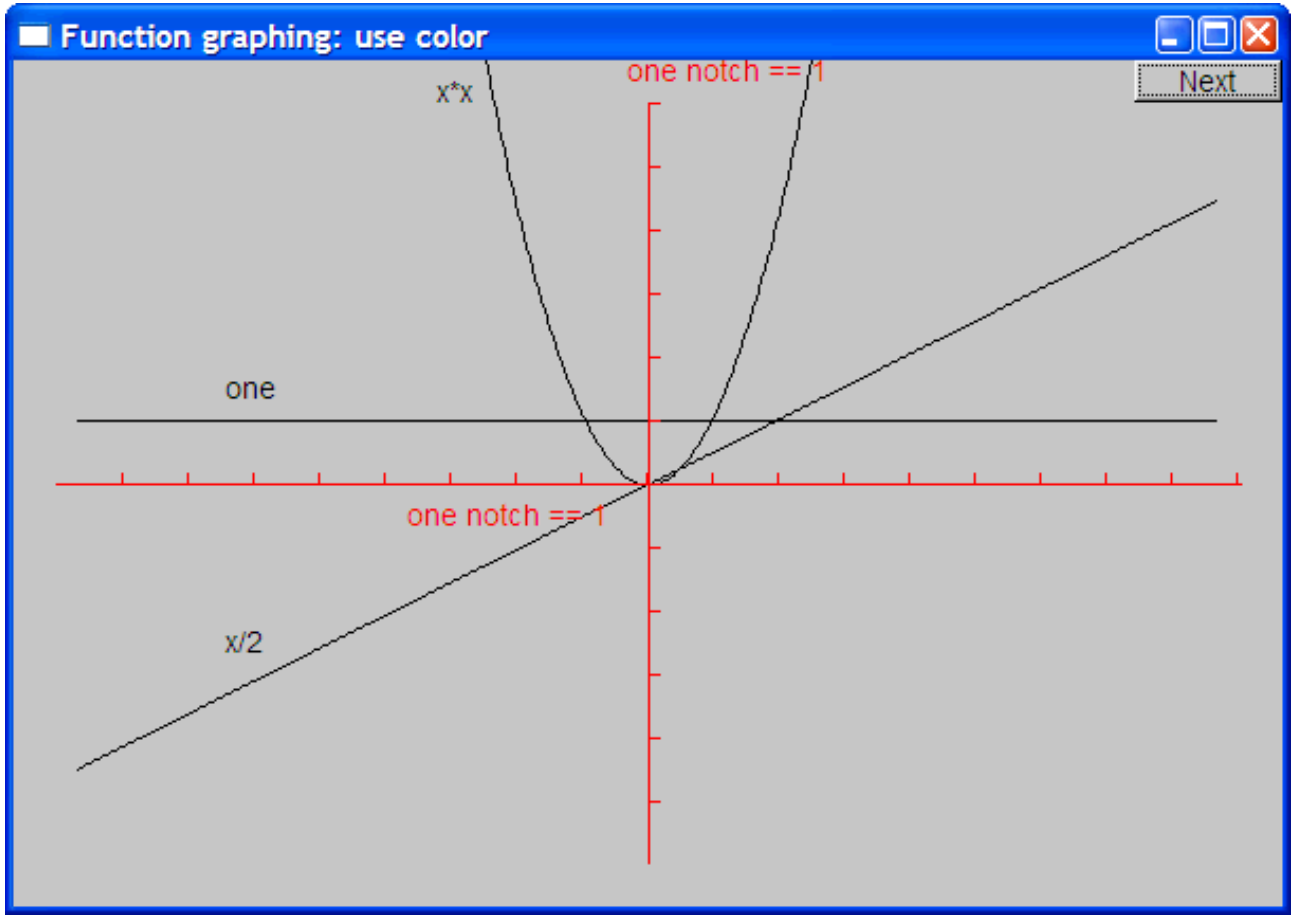

```
Text ts(Point(100, y_orig-30), "one");
Text ts2(Point(100, y_orig+y_orig/2-10), "x/2");
Text ts3(Point(x_orig-90, 20), "x*x");
```

Додајмо x и y осу




```
Axis x(Axis::x, Point(20, y_orig), xlength/x_scale, "one notch == 1 ");
Axis y(Axis::y, Point(x_orig, ylength+20), ylength/y_scale,
  "one notch == 1");
```

Да се шарени




```
x.set_color(Color::red);
y.set_color(Color::red);
```

Дефиниција **Function** класе

- Треба нам, најпре, тип података који представља функцију коју можемо исцртати
  - **using** је механизам за придруживање имена типу
    - `using Count = int; // сада је Count исто што и int`
  - постоји и алтернативни (старији) механизам:
    - `typedef int Count; // сада је Count исто што и int`

  - Укратко **using**: Иза = иде декларација типа, а назив пре = ће представљати назив тог типа.
  - Укратко **typedef**: Иза **typedef** иде редовна декларација, али сада назив „променљиве“ у тој декларацји ће представљати назив тог типа.
  - Искористићемо те механизме и за функцију
    - `using Fct = double(double);`
    - `typedef double Fct(double);`
  - Функције типа **Fct**:

```
double one(double x) { return 1; } // y==1
double slope(double x) { return x/2; }  // y==x/2
double square(double x) { return x*x; } // y==x*x
```

Дефиниција **Function** класе

```
class Function : public Shape
{
public:
  // Довољан нам је само конструктор
  Function(
     Fct f,

     double r1,
     double r2,

     Point orig,
     Count count,

     double xscale,  // позиција (x,f(x)) је (xscale*x, -yscale*f(x))
     double yscale
   );

   ...
 };
```

Имплементација Function класе

```
Function::Function( Fct f,
          double r1, double r2, // опсег
          Point xy,
          Count count,
          double xscale, double yscale )
{
 if (r2-r1 <= 0) error("bad graphing range");
 if (count <= 0) error("non-positive graphing count");
 double dist = (r2-r1)/count;
 double r = r1;
 for (int i = 0; i < count; ++i)
 {
     add(Point(xy.x+int(r*xscale), xy.y-int(f(r)*yscale)));
     r += dist;
 }
}
```

Подразумевани аргументи

- Седам аргумената је прилично много.
  - Грешка само чека да се деси
  - Треба додати подразумеване вредности за неке аргументе. Те аргументе обично ваља ставити на крај.

```
class Function : public Shape
{
public:
  Function(Fct f, double r1, double r2, Point xy,
    Count count = 100, double xscale = 25, double yscale = 25);
};

Function f1(sqrt, 0, 11, orig, 100, 25, 25 );
Function f2(sqrt, 0, 11, orig, 100, 25);      // исто као f1
Function f3(sqrt, 0, 11, orig, 100);          // исто као f1
Function f4(sqrt, 0, 11, orig);               // исто као f1
```

Function класа

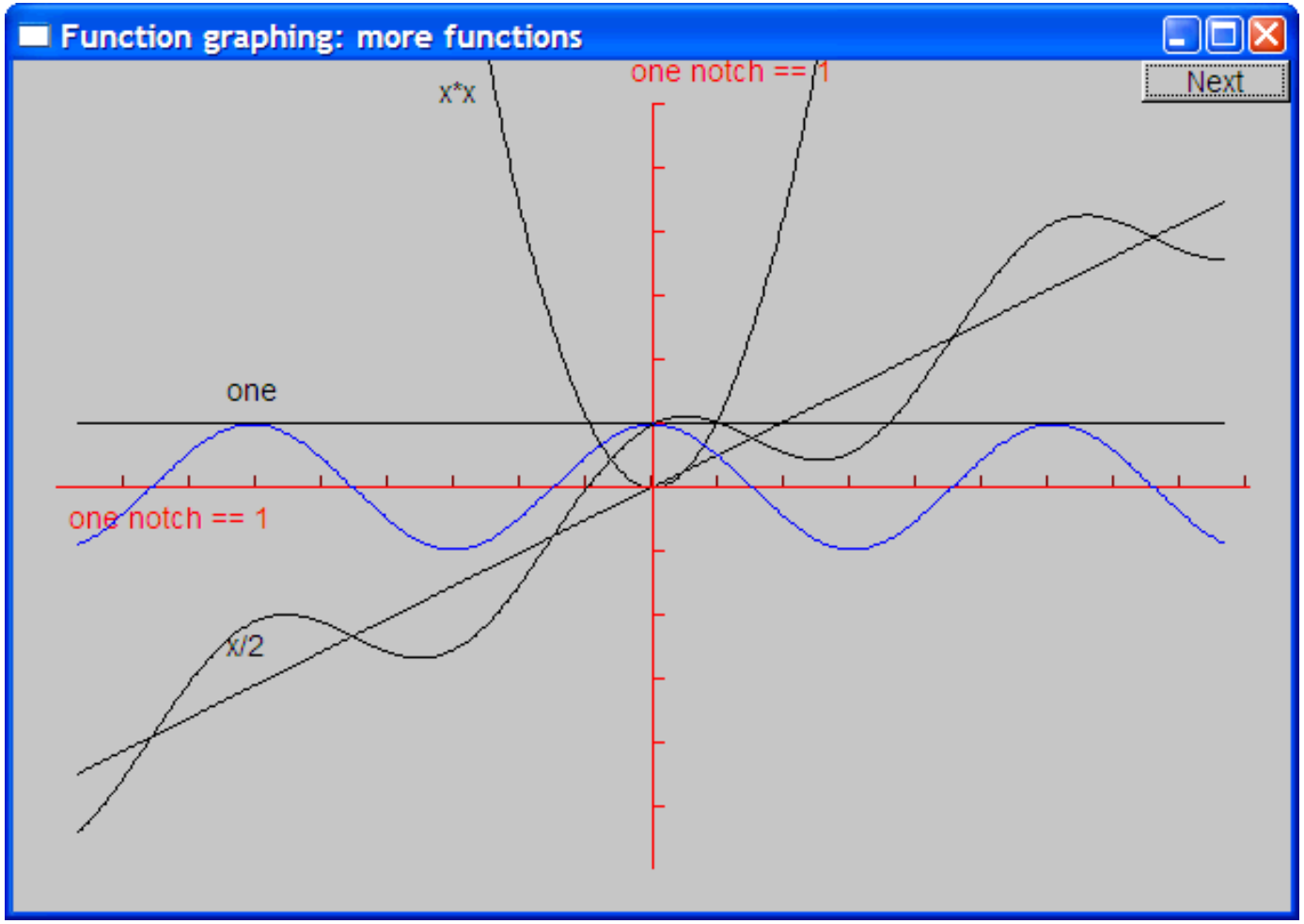
- Да ли можемо побољшати класу **Function**?
  - Позиција и скалирање се можда могу издвојити у посебну целину.
    - Погледати поглавље 15.6.3 у књизи за један приступ.
    - Укратко:

```
class Scale
{
      int cbase;
      int vbase;
      double scale;
public:
      Scale(int b, int vb, double s) : cbase(b), vbase(vb), scale(s) {}
      int operator()(int v) const { return cbase+(v-vbase)*scale; }
};

Scale sc(0, 10, 2);
int scaledValue = sc(value);
```

Још функција

```
#include<cmath>// стандардне математичке функције

// Можемо комбиновати функције
double sloping_cos(double x) { return cos(x)+slope(x); }

Function s4(cos, -10,11, orig, 400,20,20);
s4.set_color(Color::blue);
Function s5(sloping_cos, -10,11, orig, 400,20,20);
```

Косинус и коси косинус



Стандардне математичке функције (<cmath>)

- `double abs(double);`

- `double ceil(double d);` `// већи цео део`
- `double floor(double d);` `// цео део`

- `double sqrt(double d);`

- `double cos(double);`
- `double sin(double);`
- `double tan(double);`
- `double acos(double);`
- `double asin(double);`
- `double atan(double);`
- `double sinh(double);`
- `double cosh(double);`
- `double tanh(double);`

Стандардне математичке функције (<cmath>)

- **double exp(double d);** // основа e
- **double log(double d);** // природни логаритам (основа e)
- **double log10(double d);**

- **Логаритам са основом 2?**

- **double pow(double x, double y);** // ***x*** на ***y***
- **double pow(double x, int y);**
- **double atan2(double y, double x);** // *atan(y/x)*
- **double fmod(double d, double m);**
- **double ldexp(double d, int i);** // *d*pow(2,i)*

Пример: e^x

e^x **== 1**

+ x

+ $x^2/2!$

+ $x^3/3!$

+ $x^4/4!$

+ $x^5/5!$

+ $x^6/6!$

+ $x^7/7!$

+ …

(Тејлоров ред око x==0)

Једноставан алгоритам за срачунавање e^x

```
double fac(int n) { /* … */ } // !
double my_pow(double x, int n) { /* … */ }   // !

double term(double x, int n)  // x^n/n!
{
 return my_pow(x, n)/fac(n);
}

double expe(double x, int n)
{
 double sum = 0;
 for (int i = 0; i < n; ++i)
   sum += term(x, i);
 return sum;
}
```

Једноставан алгоритам за срачунавање e^x

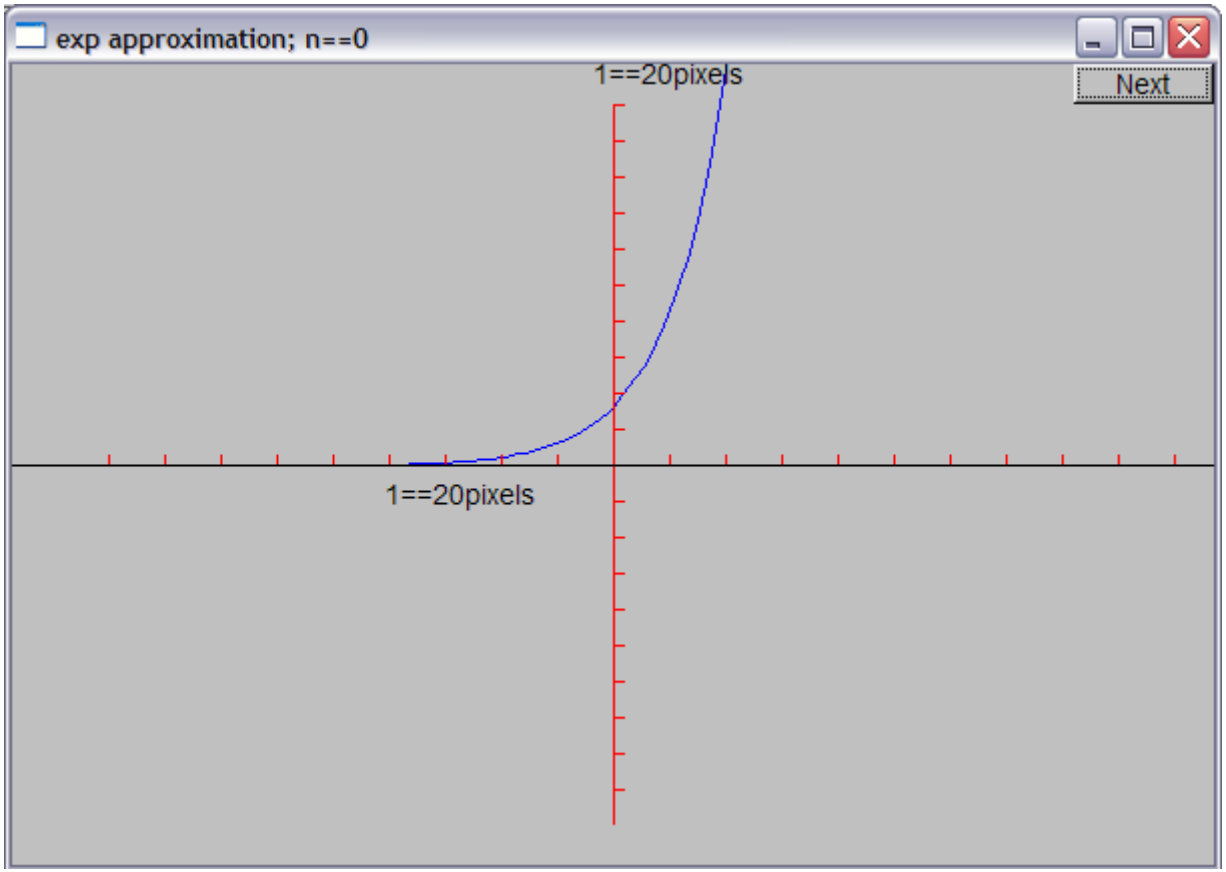
- **expe(x, n)** нам се не уклапа у тип функције коју можемо нацртати.
- Једно решење:

```
int expN_number_of_terms = 6; // супер тајни аргумент функције expN

double expN(double x)
{
  return expe(x, expN_number_of_terms);
}
```

Исцртавање апроксимација e^x

```
Simple_window win(Point(100,100), xmax, ymax, "");

// Постављање координатног система:
const int xlength = xmax-40;
const int ylength = ymax-40;
Axis x(Axis::x, Point(20,y_orig),
       xlength, xlength/x_scale, "1==20pixels");
Axis y(Axis::y, Point(x_orig,ylength+20),
       ylength, ylength/y_scale, "1==20pixels");

win.attach(x);
win.attach(y);
x.set_color(Color::red);
y.set_color(Color::red);

// Референтна функције из <cmath>:
Function real_exp(exp, r_min,r_max, orig, 200, x_scale,y_scale);
real_exp.set_color(Color::blue);
win.attach(real_exp);
```

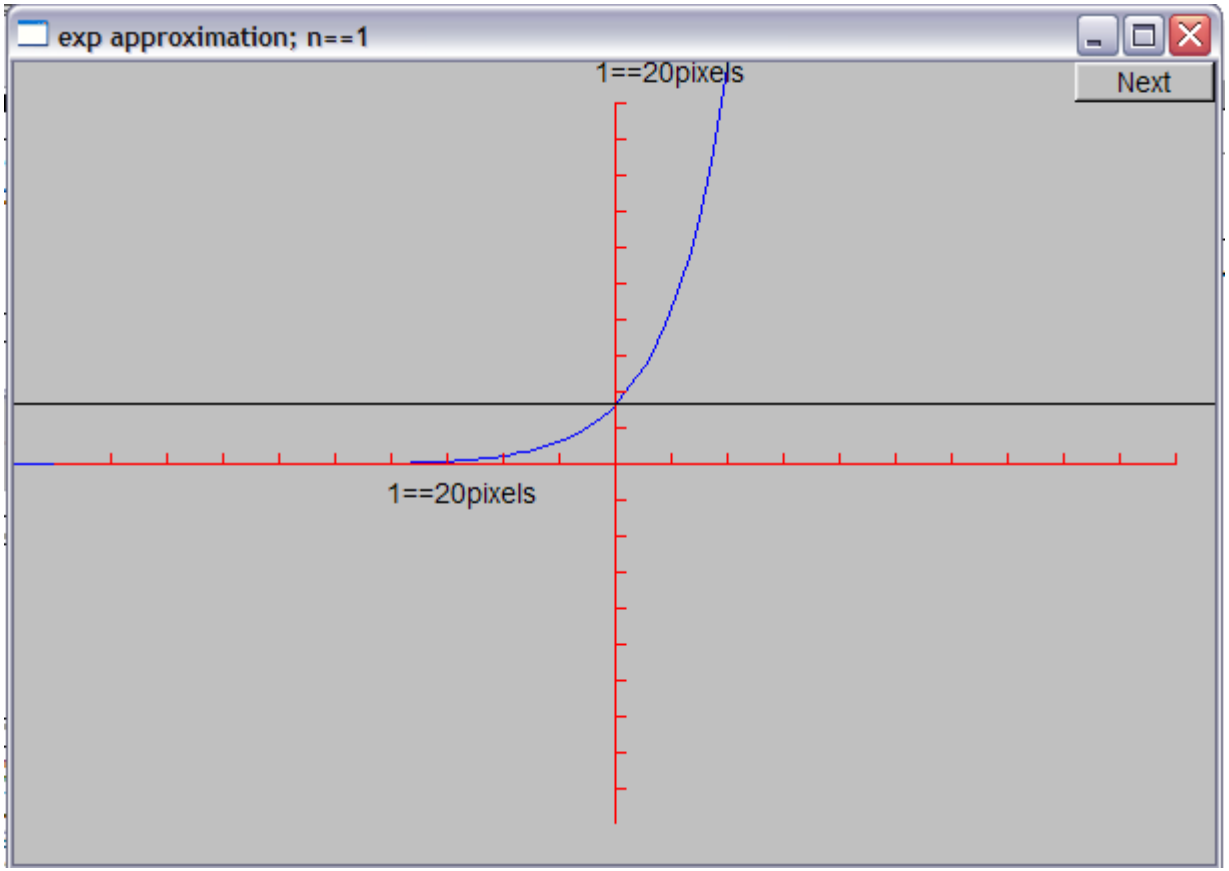
Референтна функција e^x



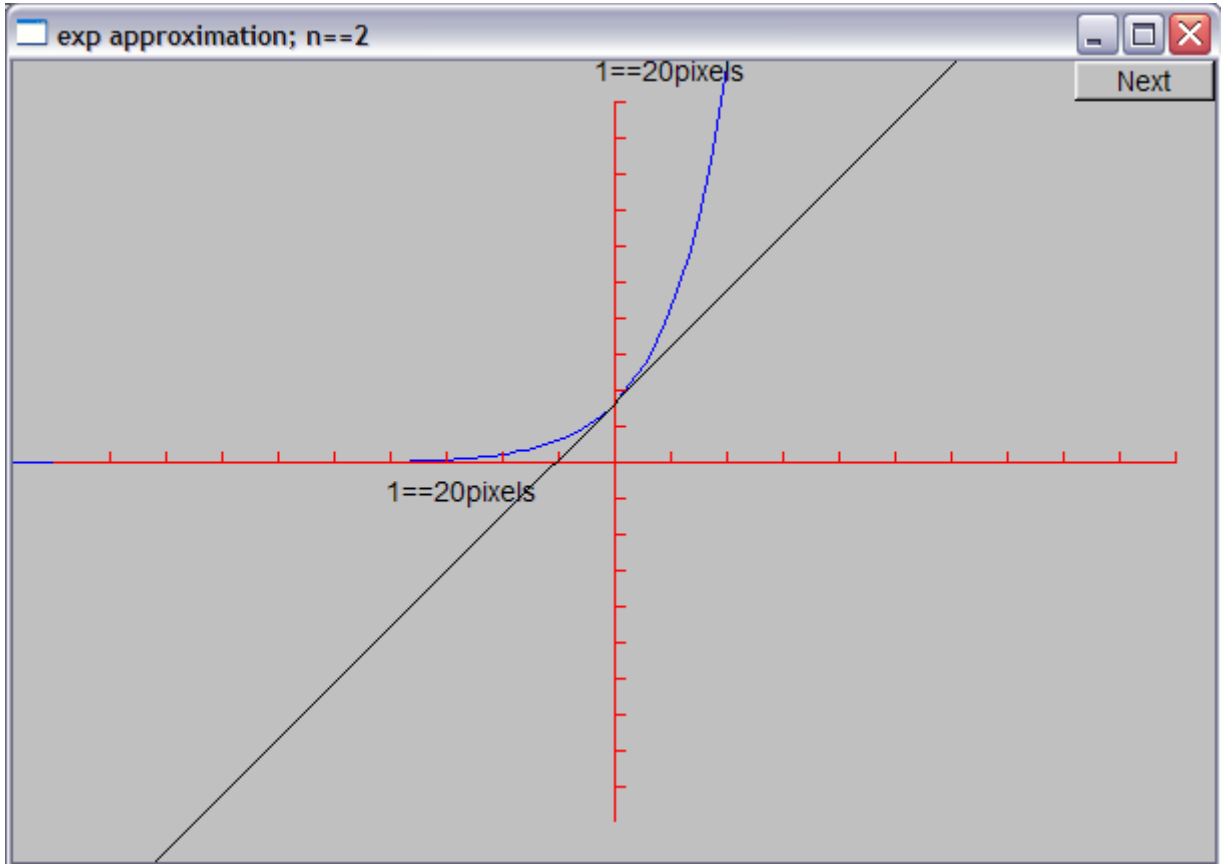
Исцртавање апроксимација e^x

```
for (int n = 0; n < 50; ++n)
{
 ostringstream ss;
 ss << "exp approximation; n==" << n ;
 win.set_label(ss.str().c_str());
 expN_number_of_terms = n; // супер тајни аргумент функције expN

 // наредна апроксимација:
 Function e(expN, r_min,r_max, orig, 200, x_scale,y_scale);

 win.attach(e);
 win.wait_for_button();
 win.detach(e);
}
```

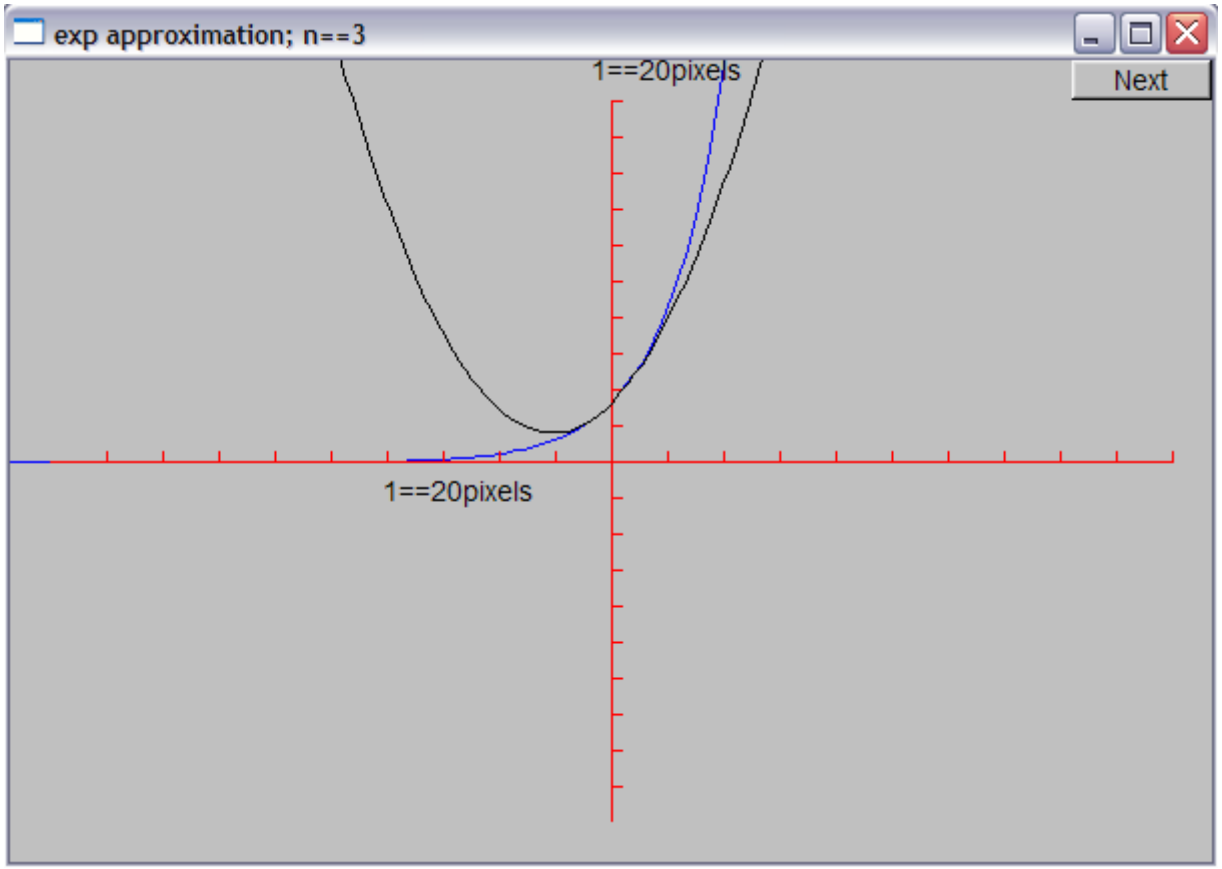
n = 1



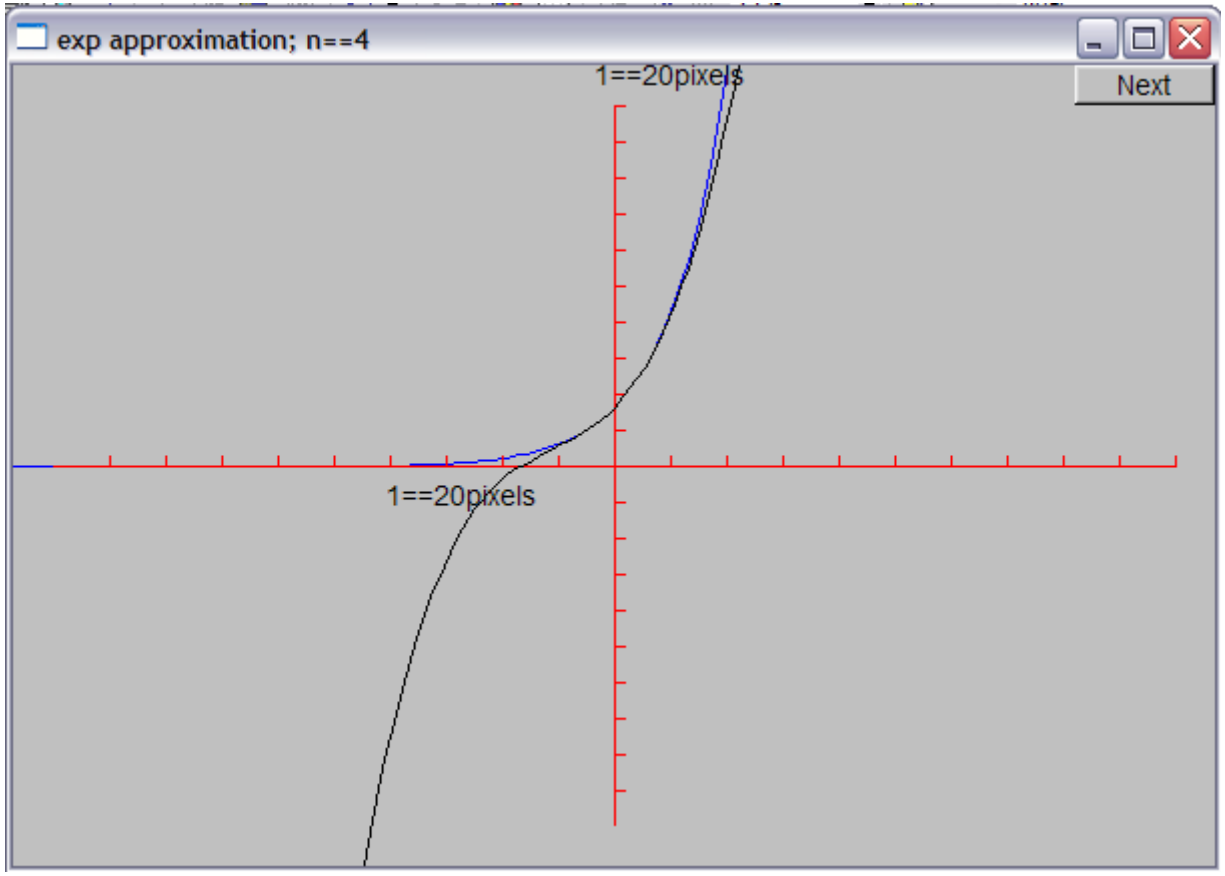
n = 2



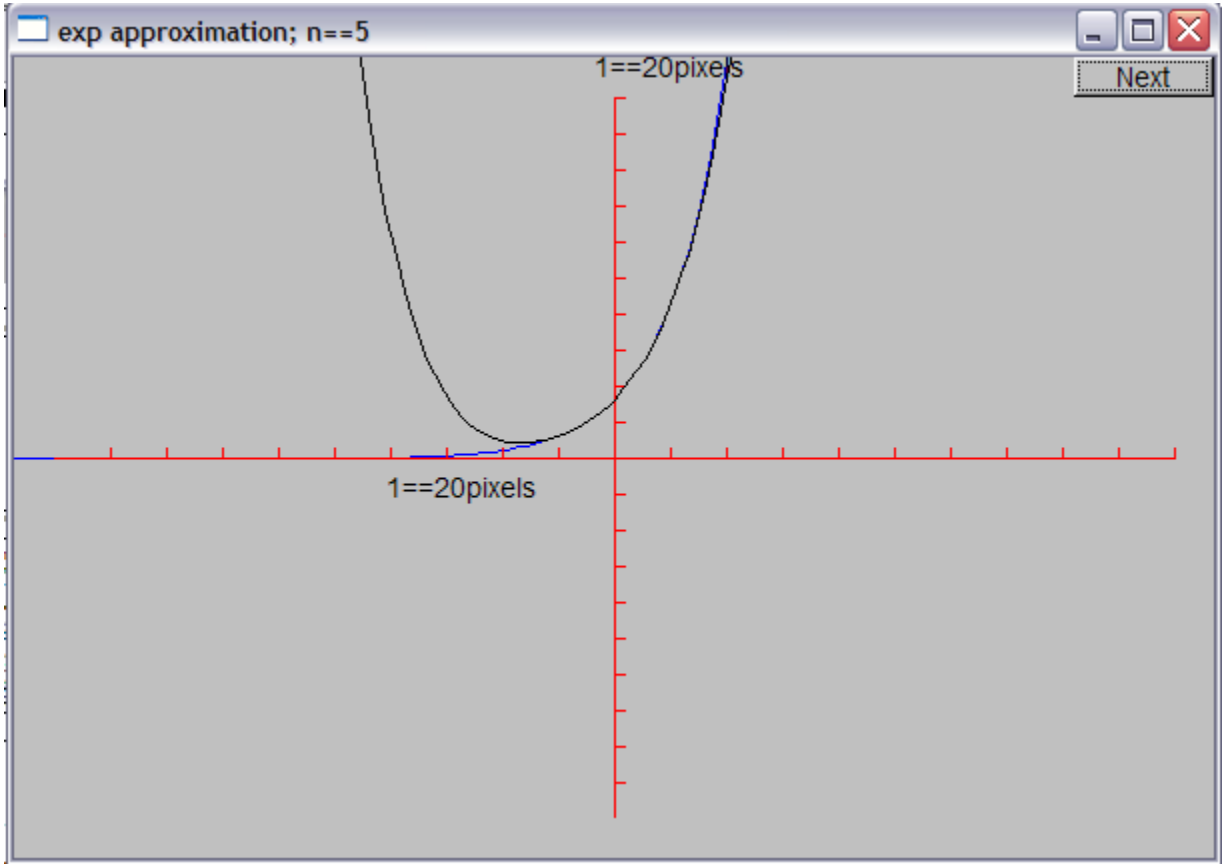
n = 3



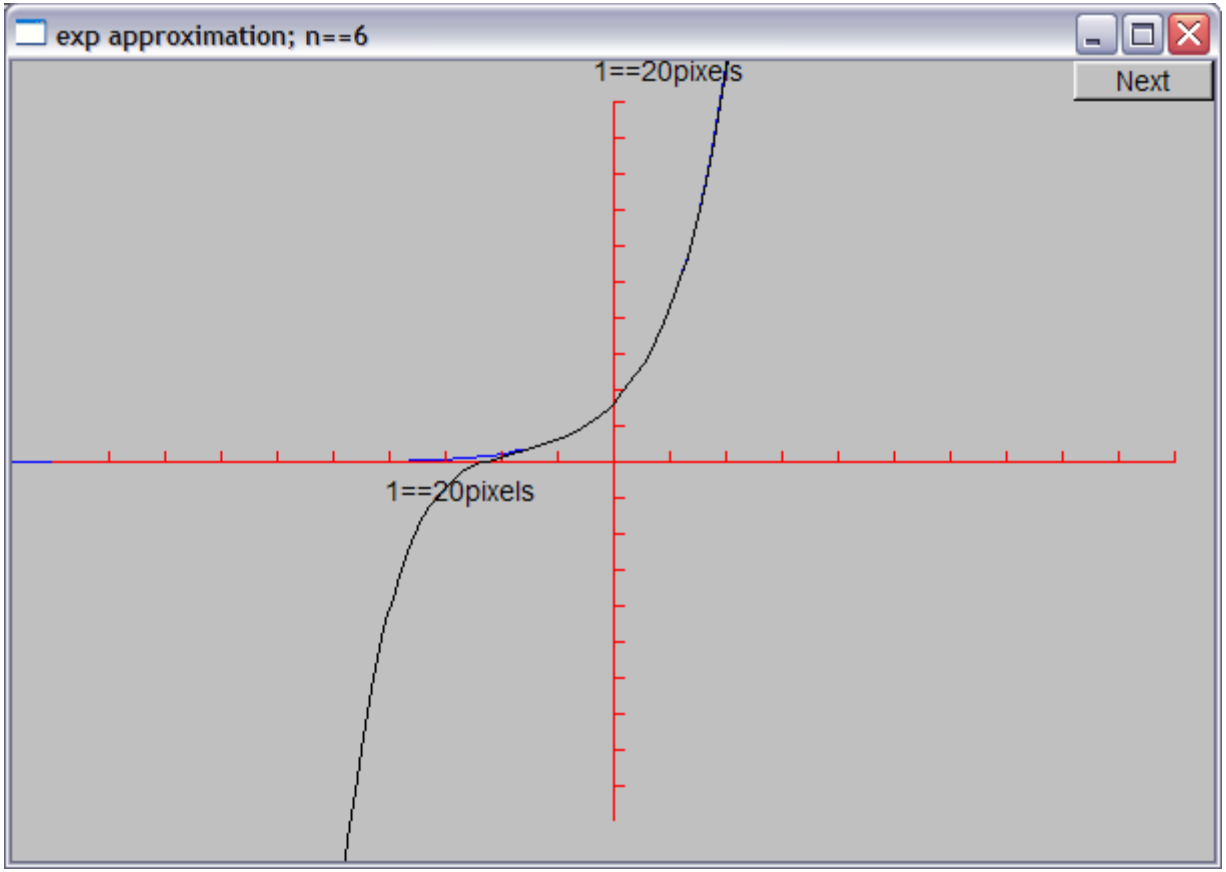
n = 4



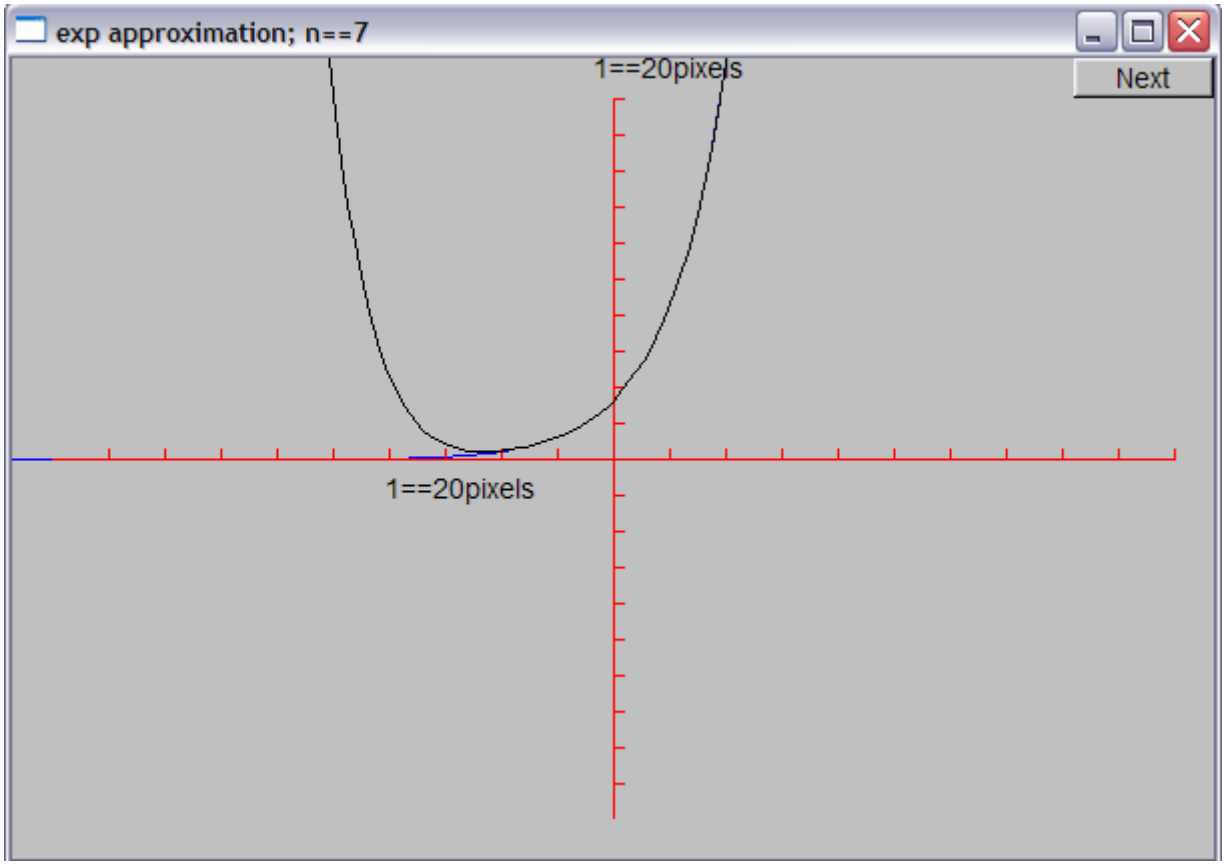
n = 5



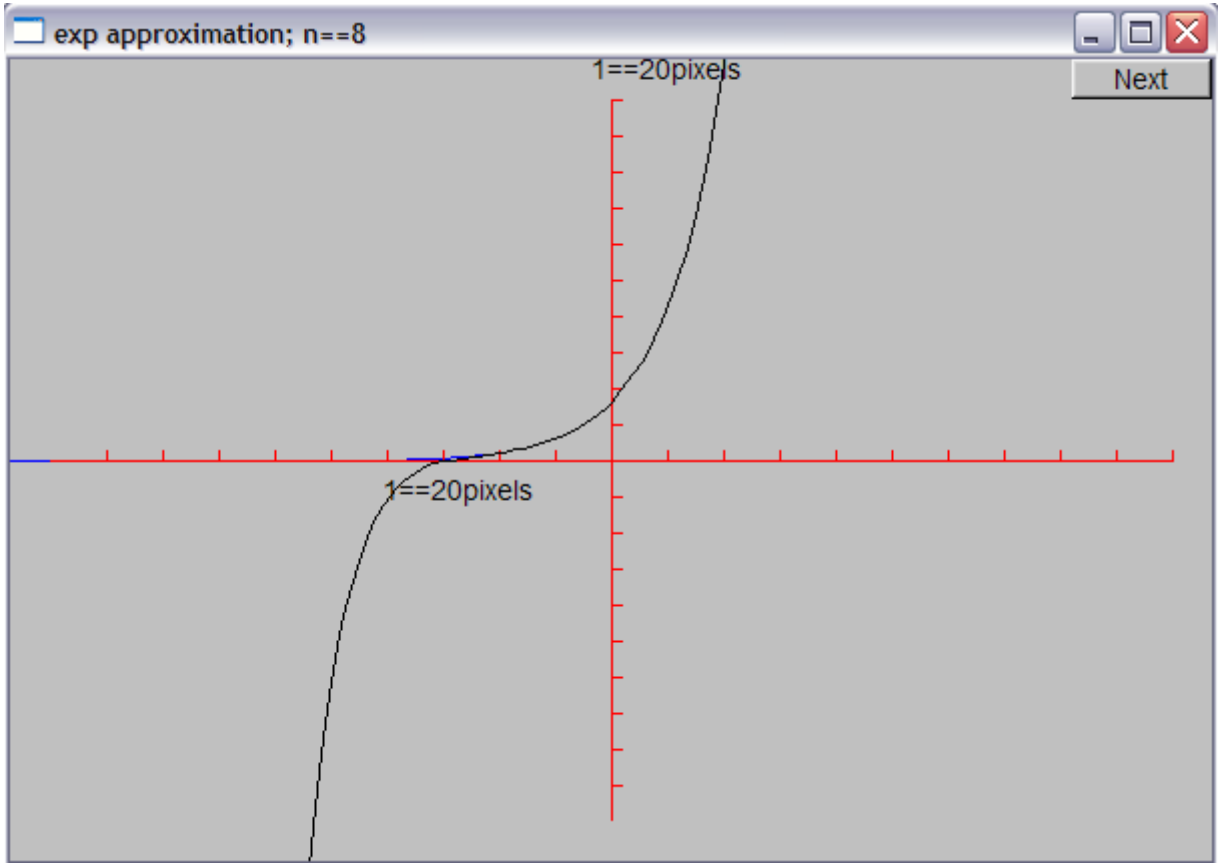
n = 6



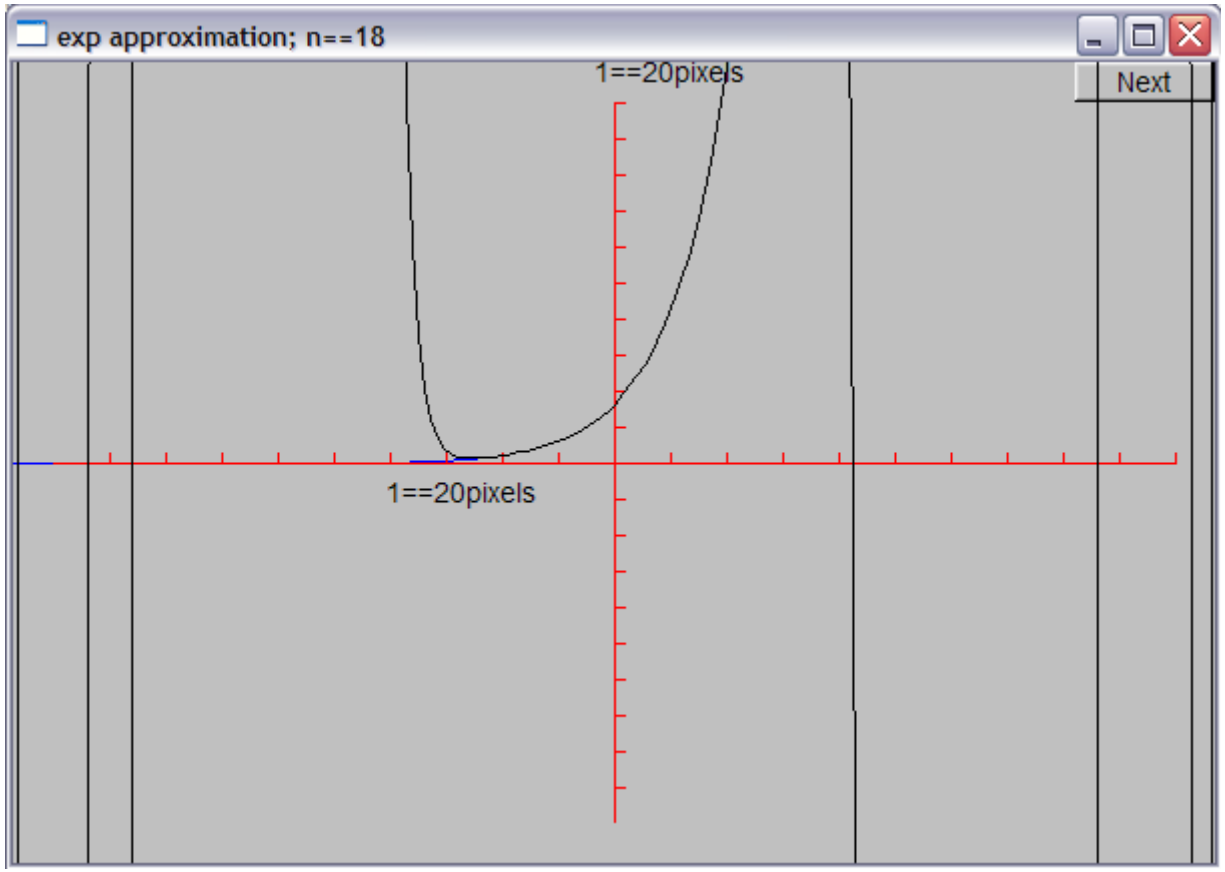
n = 7



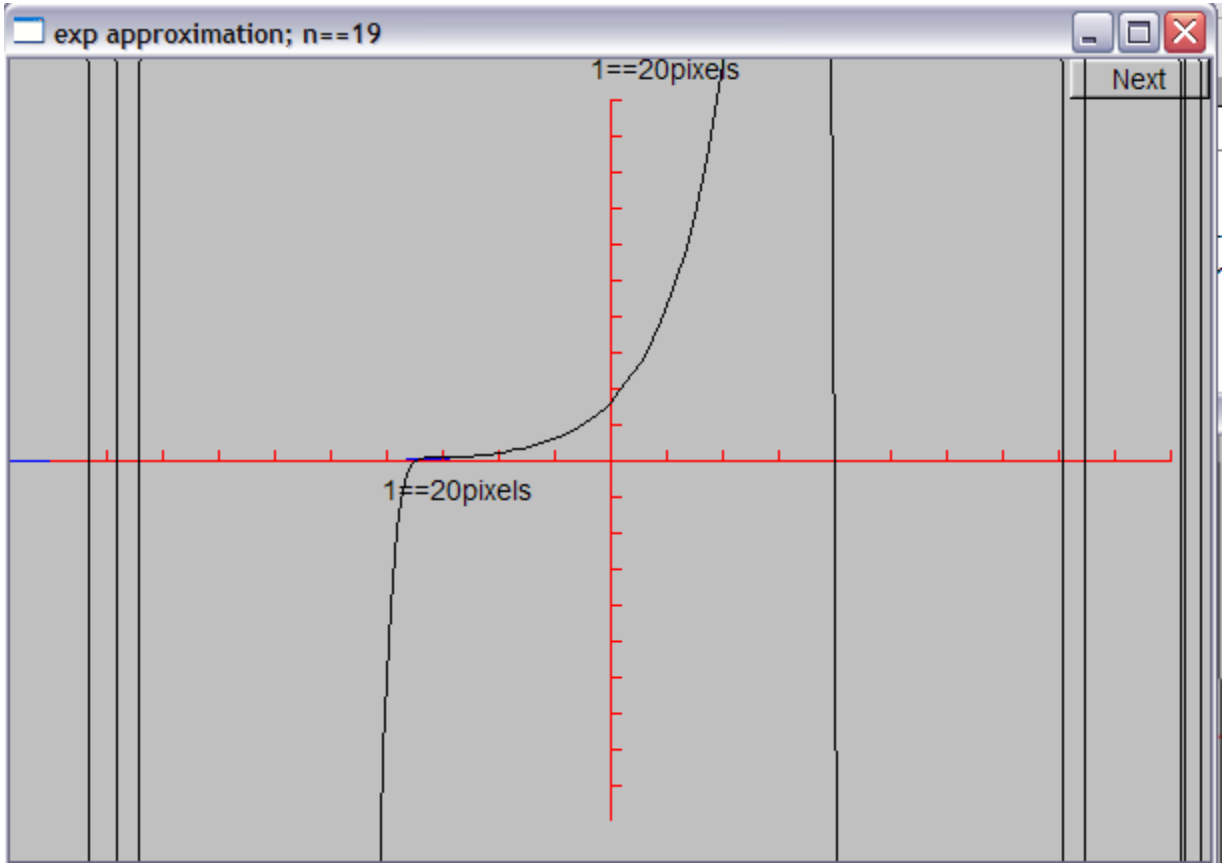
n = 8



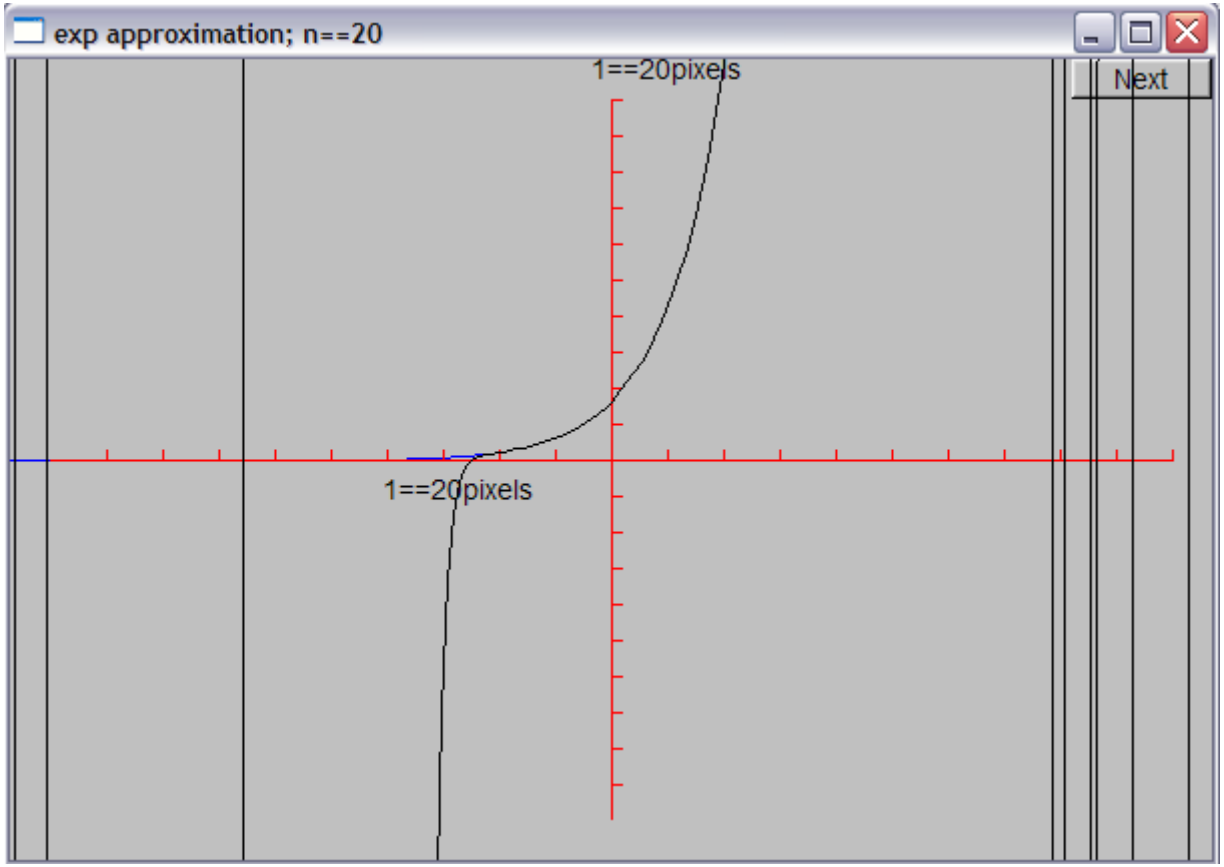
n = 18



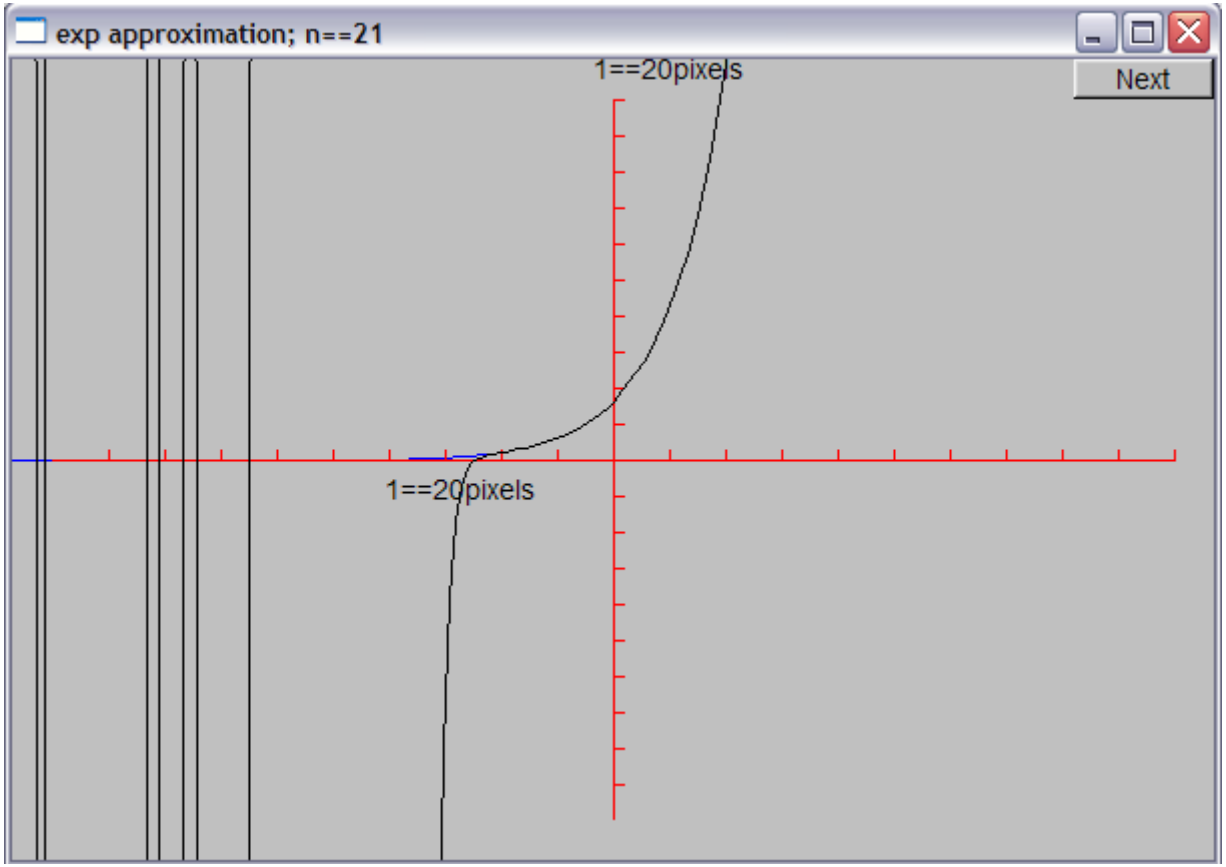
n = 19



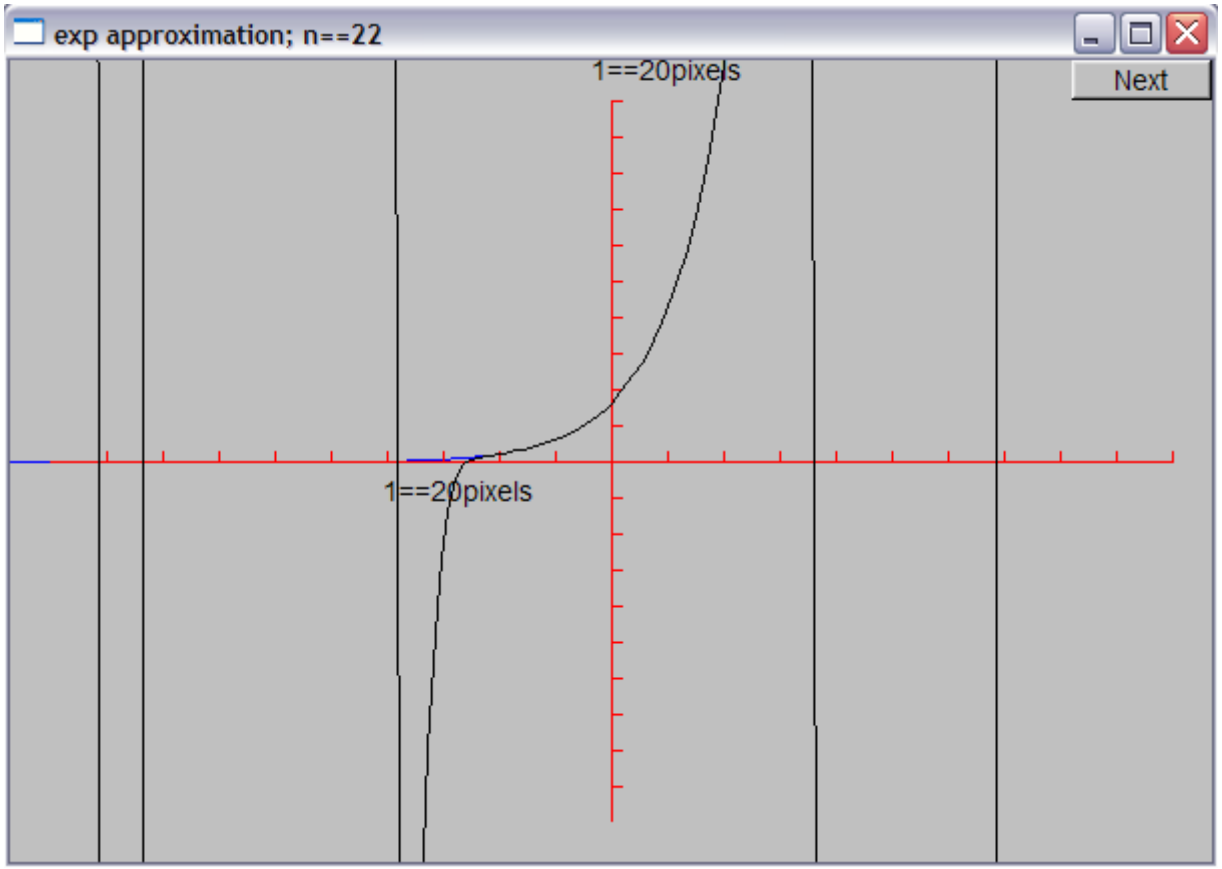
n = 20



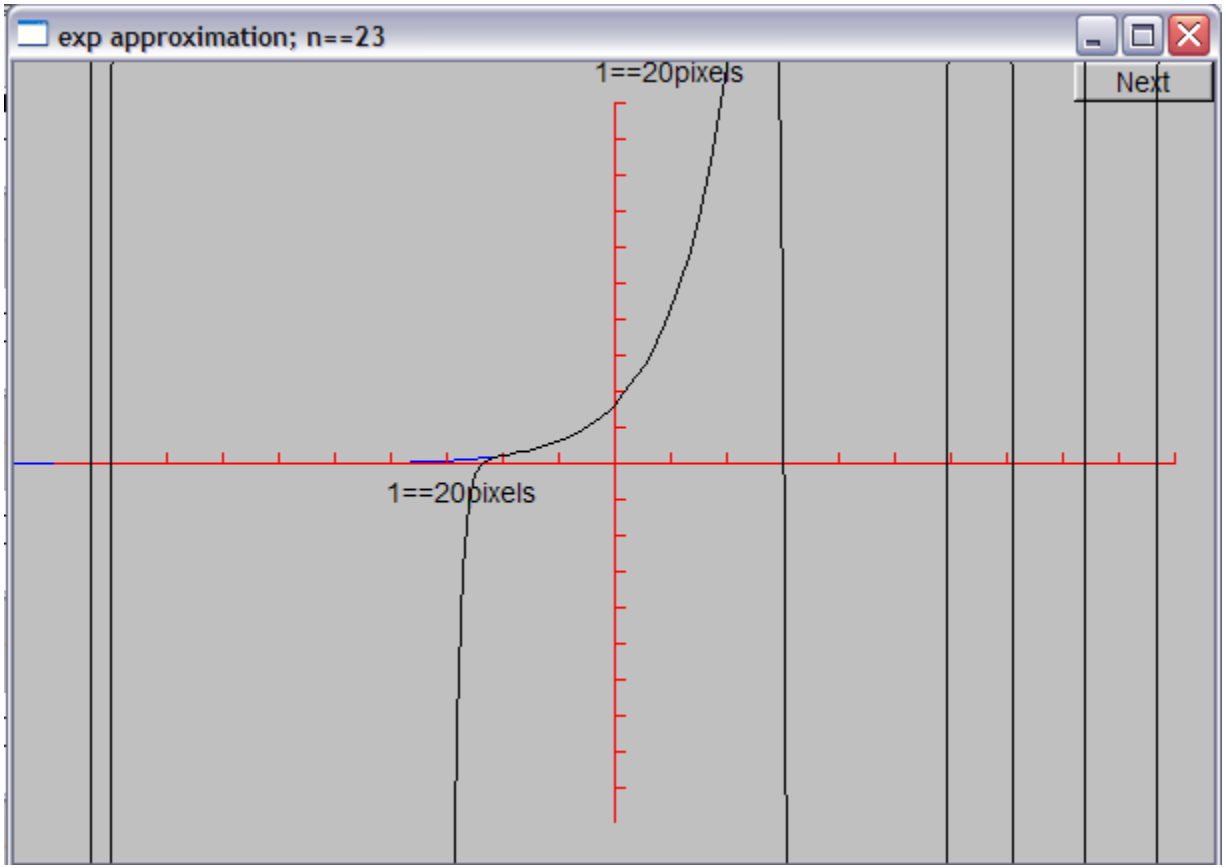
n = 21



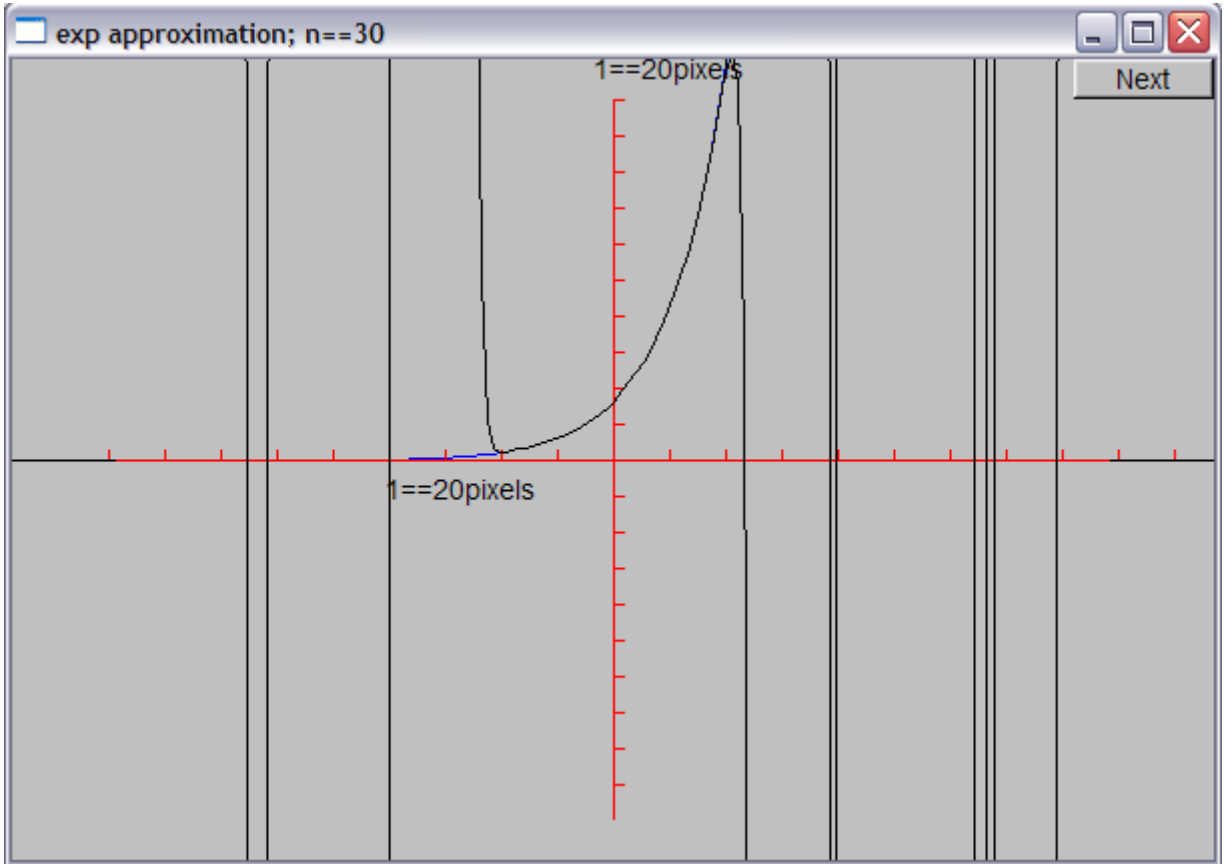
n = 22



n = 23



n = 30



Шта се десило?!

- Бројеви са покретним зарезом нису прави реални бројеви, већ њихова апроксимација
  - Реални бројеви могу бити произвољно велики и произвољно прецизни
    - Бројеви у покретном зарезу су ограничени и по питању величине и по питању прецизности. (Уједно, прецизност зависи од величине броја: за мале бројеве је већа)
  - Понекад прецизност није довољно велика за наше потребе
  - Мале грешке при заокруживању се могу нагомилати и направити велике грешке на излазу
- Добра пракса је да:
  - пажљиво размислите о математичким рачунањима
  - проверите резултате
  - трудите се да грешка буде очевидна
    - боље је да се грешка испољи раније током коришћења, док је још не користи велики број људи