ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στον Προγραμματισμό
Introduction to Programming

Διάλεξη 14.β: Συναρτήσεις και Γραφικά

Γ. Παπαγιαννάκης

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα

Αναφορά Δημιουργού 3.0 - Μη εισαγόμενο Ελλάδα (Attribution 3.0– Unported GR)

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

HY-150 Προγραμματισμός
CS-150 Programming

Lecture 14b:
Functions and graphing

G. Papagiannakis

Abstract

- Here we present ways of graphing functions and data and some of the programming techniques needed to do so, notably scaling.

Note

- This course is about programming
  - The examples – such as graphics – are simply examples of
    - Useful programming techniques
    - Useful tools for constructing real programs

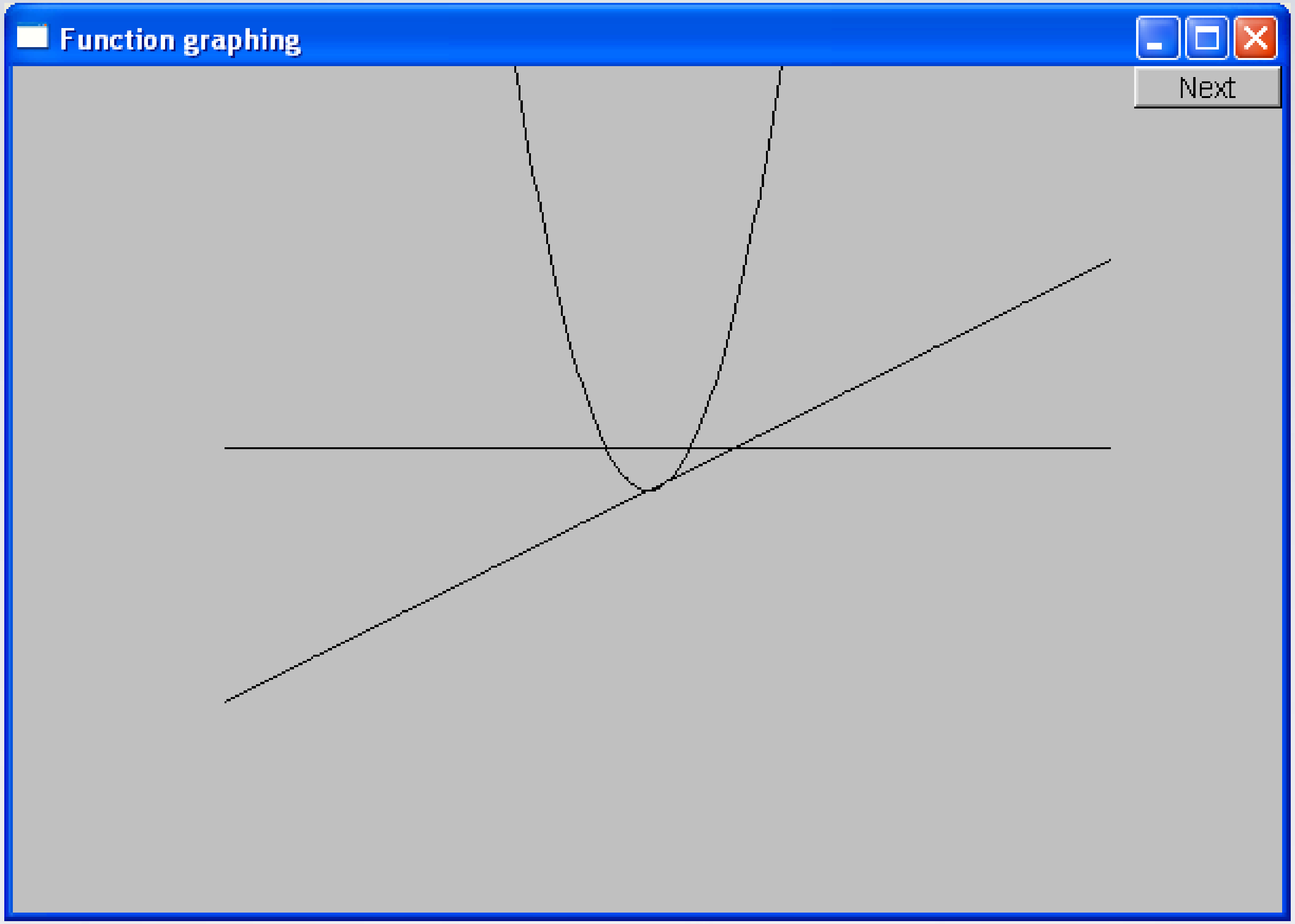
    Look for the way the examples are constructed
  - How are "big problems" broken down into little ones and solved separately?
  - How are classes defined and used
    - Do they have sensible data members?
    - Do they have useful member functions?
  - Use of variables
    - Are there too few?
    - Too many?
    - How would you have named them better?

Graphing functions

- Start with something really simple
  - Always remember “Hello, World!”
- We graph functions of one argument yielding a one value
  - Plot (x, f(x)) for values of x in some range [r1,r2)
- Let’s graph three simple functions

```
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
```

Functions




```
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
```

How do we write code to do this?

Function to be graphed

```
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( );
```

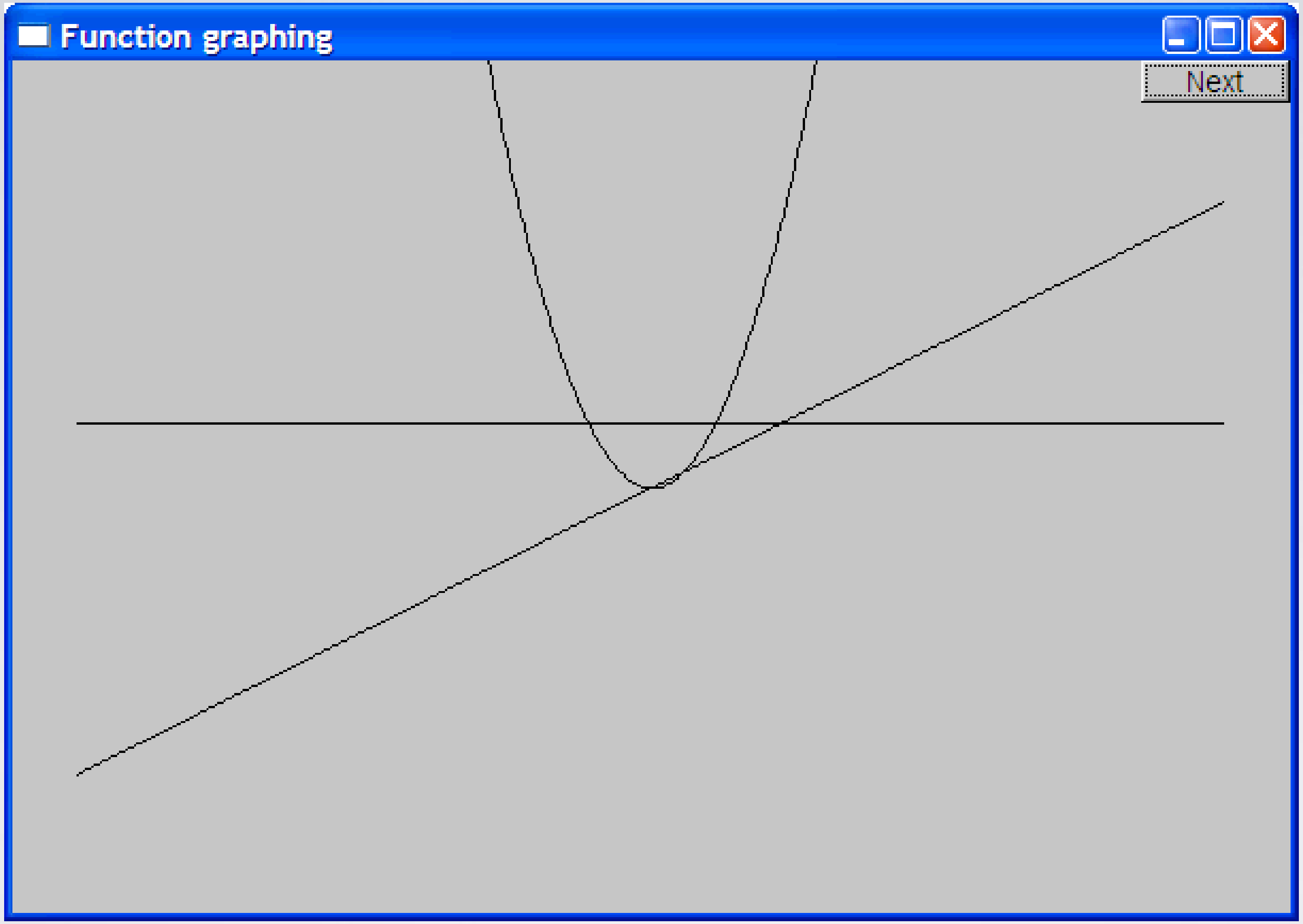
Range in which to graph

“stuff” to make the graph fit into the window

We need some Constants

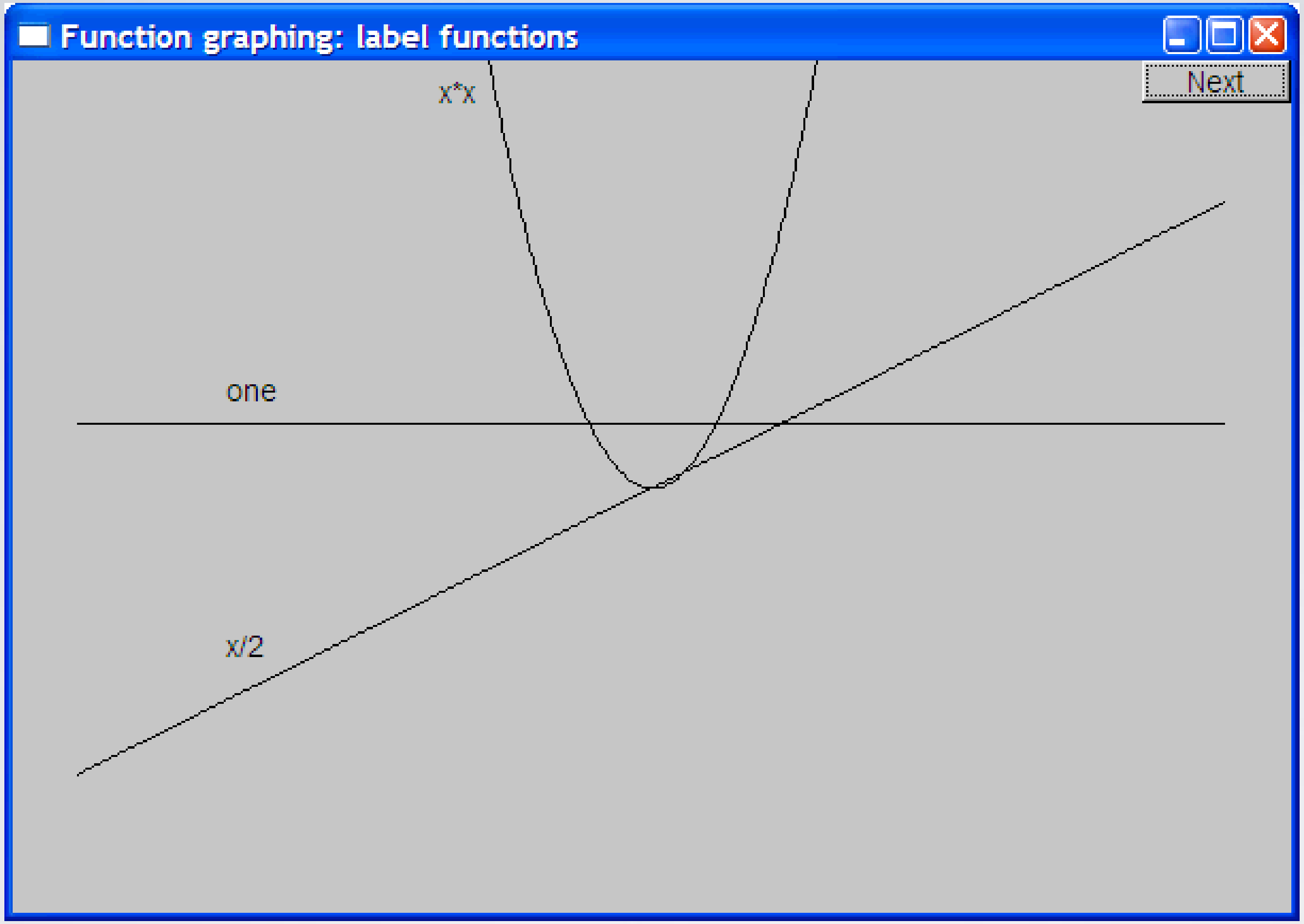
```
const int xmax = 600;                       // window size
const int ymax = 400;

const int x_orig = xmax/2;
const int y_orig = ymax/2;
const Point orig(x_orig, y_orig);   // position of (0,0) in window

const int r_min = -10;                      // range [-10:11) == [-10:10] of x
const int r_max = 11;

const int n_points = 400;                   // number of points used in range

const int x_scale = 20;                     // scaling  factors
const int y_scale = 20;

// Choosing a center (0,0), scales, and number of points can be fiddly
// The range usually comes from the definition of what you are doing
```

Functions – but what does it mean?



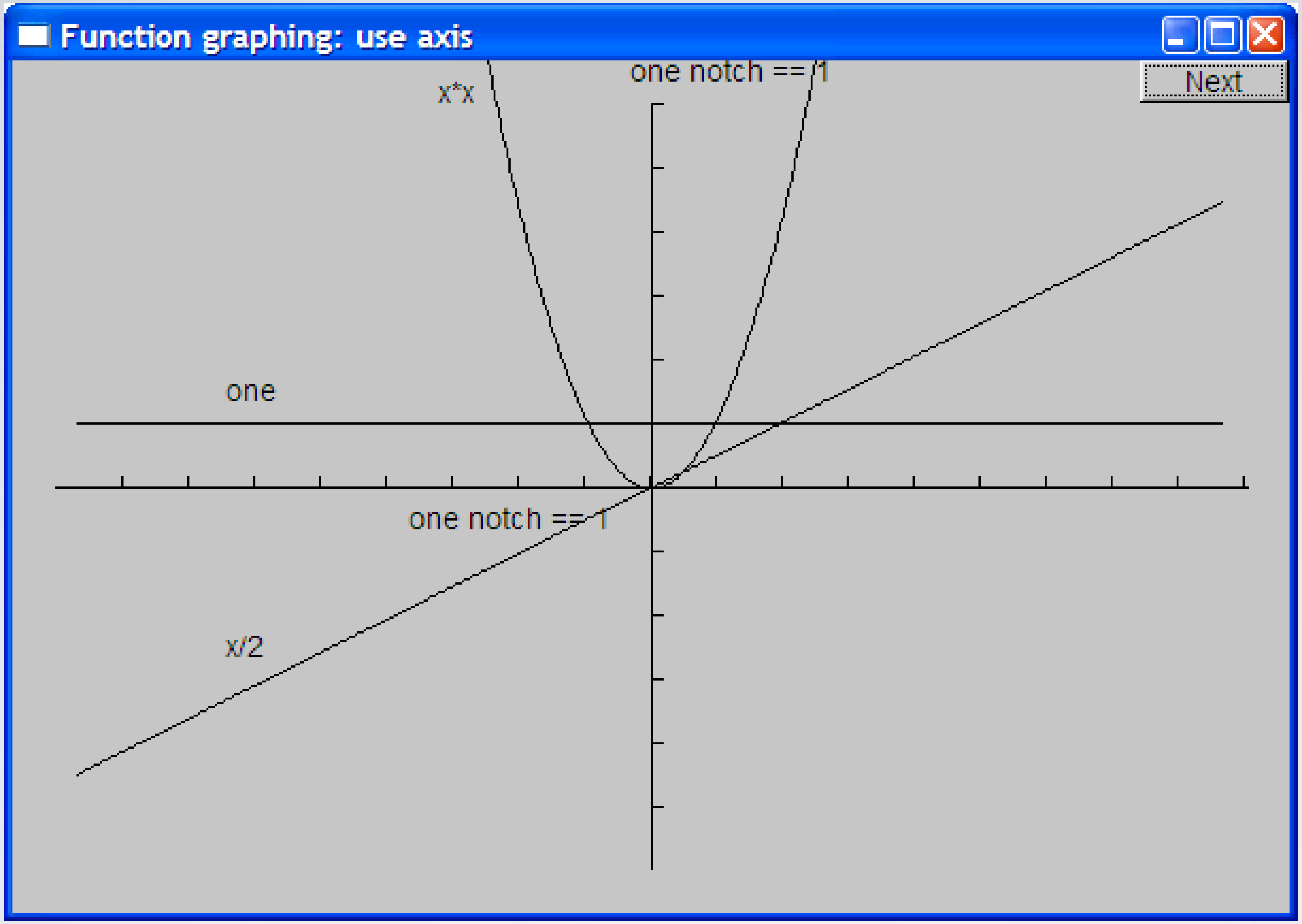
- What's wrong with this?
- No axes (no scale)
- No labels

Label the functions




```
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");
```

Add x-axis and y-axis

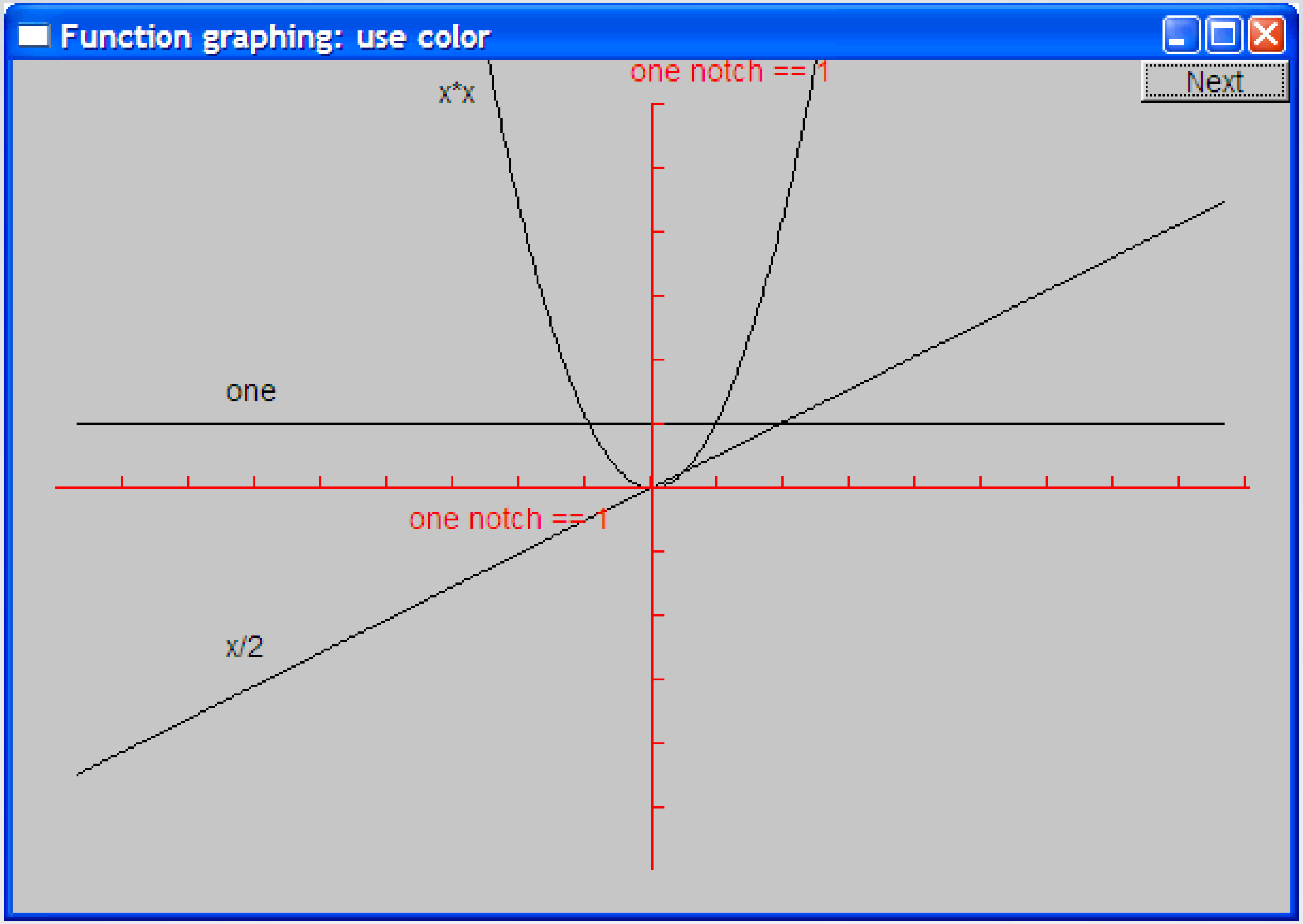


- We can use axes to show (0,0) and the scale

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");

Use color (in moderation)



s.set_color(Color::green); **x.set_color(Color::red);**

y.set_color(Color::red); **ts.set_color(Color::green);**

The implementation of Function

- We need a type for the argument specifying the function to graph
  - **typedef** can be used to declare a new name for a type
    - **typedef int Color;** *// now Color means int*

  - Define the type of our desired argument, **Fct**
    - **typedef double Fct(double);** *// now Fct means function*
      // taking a double argument
      // and returning a double

  - Examples of functions of type **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
```

Now Define "Function"

```
struct Function : Shape                    // Function is derived from Shape
{
   // all it needs is a constructor:
   Function(
      Fct f,                        // f is a Fct  (takes a double, returns a double)

      double r1,    // the range of x values (arguments to f) [r1:r2)
      double r2,
      Point orig,   // the screen location of (0,0)
      int count,                    // number of points used to draw the function
                                    // (number of line segments used is count-1)

      double xscale ,               // the location (x,f(x)) is (xscale*x,yscale*f(x))
      double yscale
   );
};
```

Implementation of Function

```
Function::Function( Fct f,
                    double r1, double r2,   // range
                    Point xy,
                    int 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;
    }
}
```

Default arguments

- Seven arguments are too many!
  - Many too many
  - We're just asking for confusion and errors
  - Provide defaults for some (trailing) arguments
    - Default arguments are often useful for constructors

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

Function f1(sqrt, 0, 11, orig, 100, 25, 25 );  // ok (obviously)
Function f2(sqrt, 0, 11, orig, 100, 25);       // ok: exactly the same as f1
Function f3(sqrt, 0, 11, orig, 100);           // ok: exactly the same as f1
Function f4(sqrt, 0, 11, orig);                // ok: exactly the same as f1
```

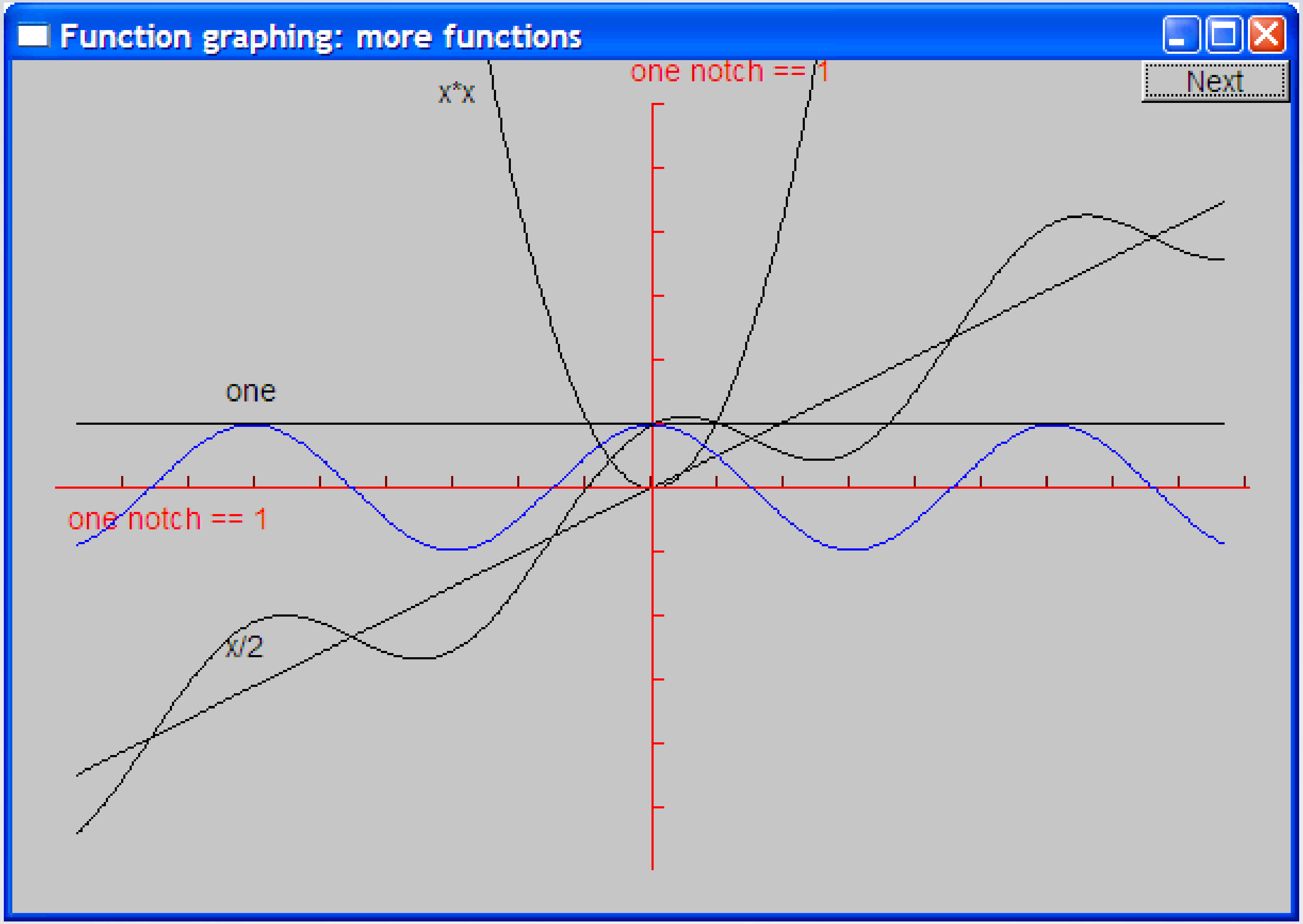
Function

- Is **Function** a “pretty class”?
  - No
    - Why not?
  - What could you do with all of those position and scaling arguments?
    - See 15.6.3 for one minor idea
  - If you can’t do something genuinely clever, do something simple, so that the user can do anything needed
    - Such as adding parameters so that the caller can control

Some more functions

```
#include<cmath>        // standard mathematical functions


// You can combine functions (e.g., by addition):
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);
```

Cos and sloping-cos



Standard mathematical functions (<cmath>)

- **double abs(double);** *// absolute value*

- **double ceil(double d);** *// smallest integer* **>= d**
- **double floor(double d);** *// largest integer* **<= d**

- **double sqrt(double d);** *//* ***d*** *must be non-negative*

- **double cos(double);**
- **double sin(double);**
- **double tan(double);**
- **double acos(double);** *// result is non-negative; "a" for "arc"*
- **double asin(double);** *// result nearest to 0 returned*
- **double atan(double);**
- **double sinh(double);** *// "h" for "hyperbolic"*
- **double cosh(double);**
- **double tanh(double);**

Standard mathematical functions (<cmath>)

- **double exp(double);** *// base e*
- **double log(double d);** *// natural logarithm (base e)* **; d** *must be positive*
- **double log10(double);** *// base 10 logarithm*

- **double pow(double x, double y);** *//* ***x*** *to the power of* ***y***
- **double pow(double x, int y);** *//* ***x*** *to the power of* ***y***
- **double atan2(double x, double y);** *// atan(x/y)*
- **double fmod(double d, double m);** *// floating-point remainder*
  // same sign as d%m
- **double ldexp(double d, int i);** *// d*pow(2,i)*

Why graphing?

- Because you can see things in a graph that are not obvious from a set of numbers
  - How would you understand a sine curve if you couldn’t (ever) see one?
- Visualization is
  - key to understanding in many fields
  - Used in most research and business areas
    - Science, medicine, business, telecommunications, control of large systems

An example: e^x

e^x == **1**

+ x

+ x^2/2!

+ x^3/3!

+ x^4/4!

+ x^5/5!

+ x^6/6!

+ x^7/7!

+ …

Where ! Means factorial (e.g. 4!==4*3*2*1)

Simple algorithm to approximate e^x

```
double fac(int n) { /* … */ }          // factorial

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

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

Simple algorithm to approximate e^x

- But we can only graph functions of one argument, so how can we get graph **expe(x,n)** for various **n**?

```
int expN_number_of_terms = 6;        // nasty sneaky argument to expN

double expN(double x)            // sum of expN_number_of_terms terms of x
{
   return expe(x,expN_number_of_terms);
}
```

“Animate” approximations to e^x

```
Simple_window win(Point(100,100),xmax,ymax,"");
// the real exponential :
Function real_exp(exp,r_min,r_max,orig,200,x_scale,y_scale);
real_exp.set_color(Color::blue);
win.attach(real_exp);

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

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

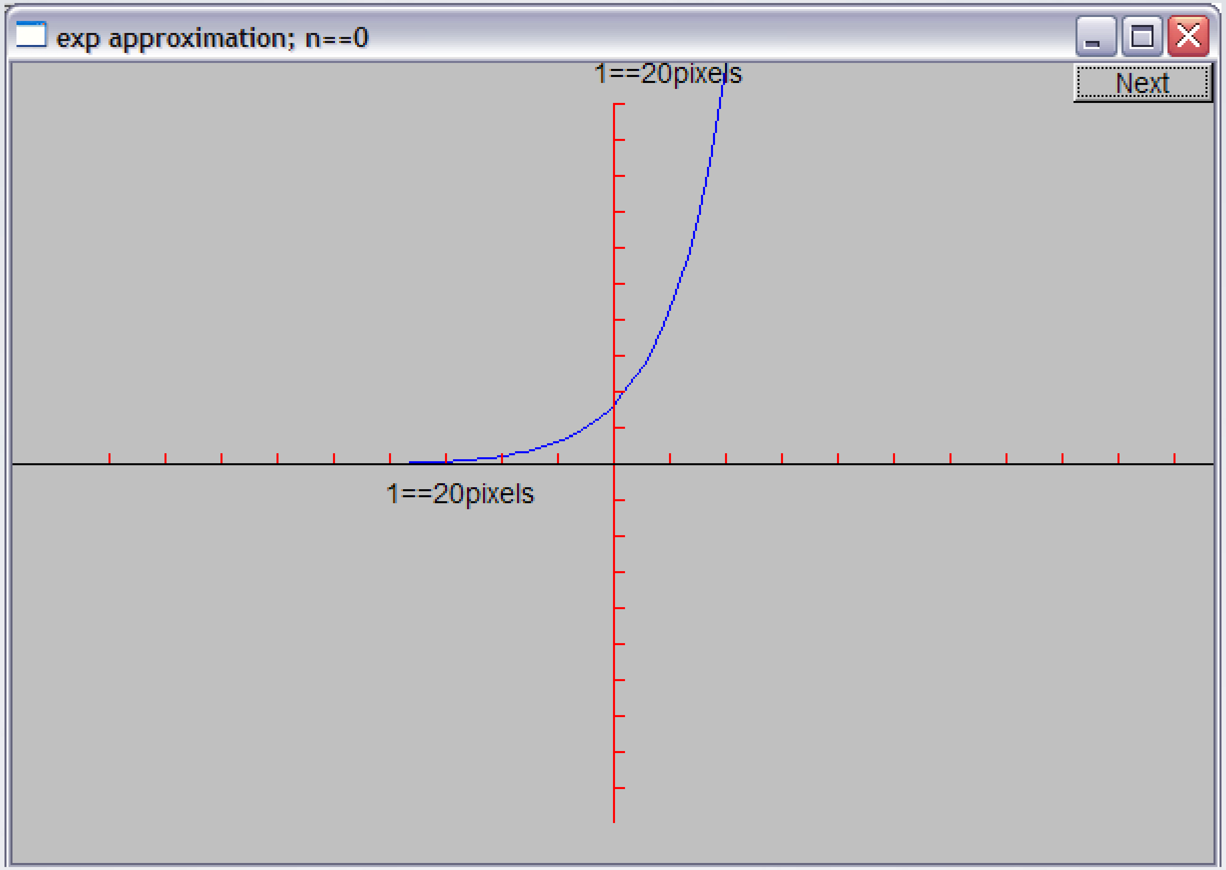
"Animate" approximations to e^x

```
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;    // nasty sneaky argument to expN

    // next approximation:
    Function e(expN,r_min,r_max,orig,200,x_scale,y_scale);

    win.attach(e);
    wait_for_button();    // give the user time to look
    win.detach(e);
}
```

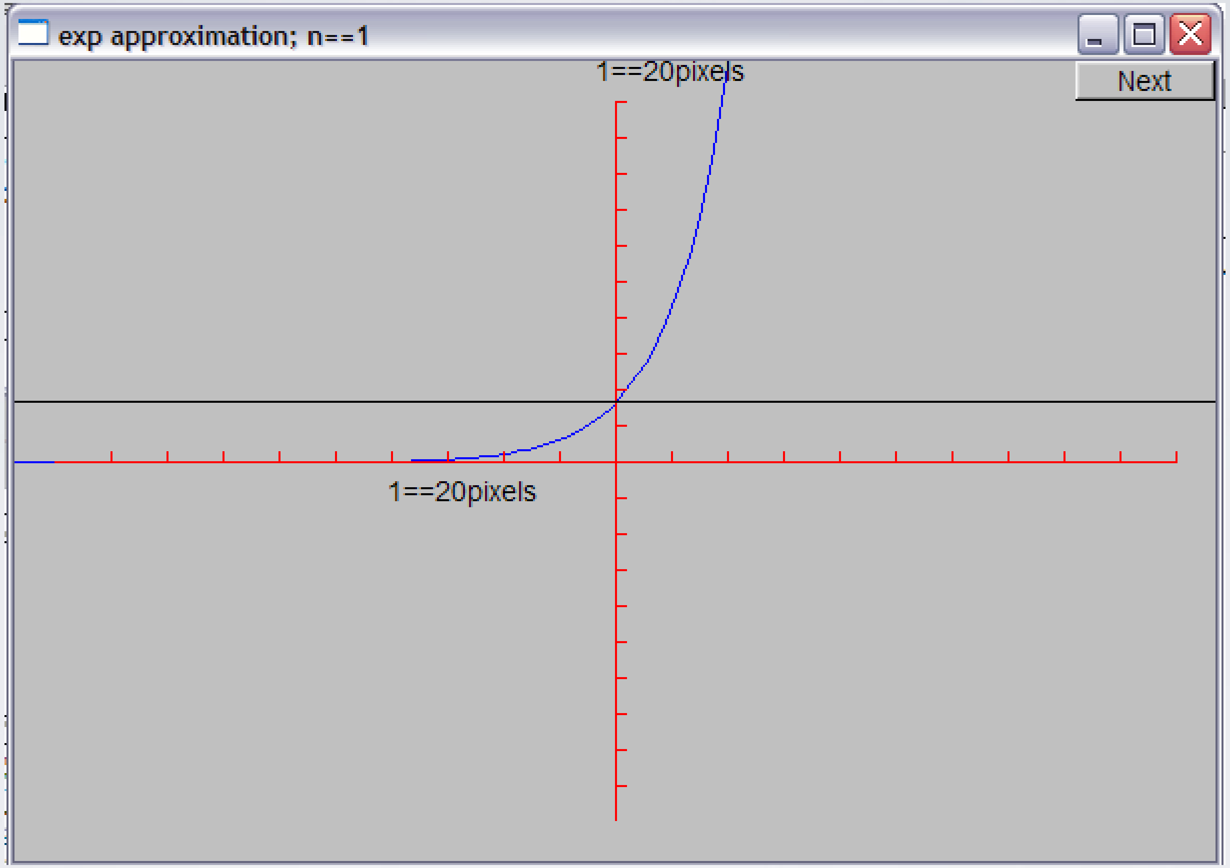
Demo

- The following screenshots are of the successive approximations of **exp(x)** using **expe(x,n)**

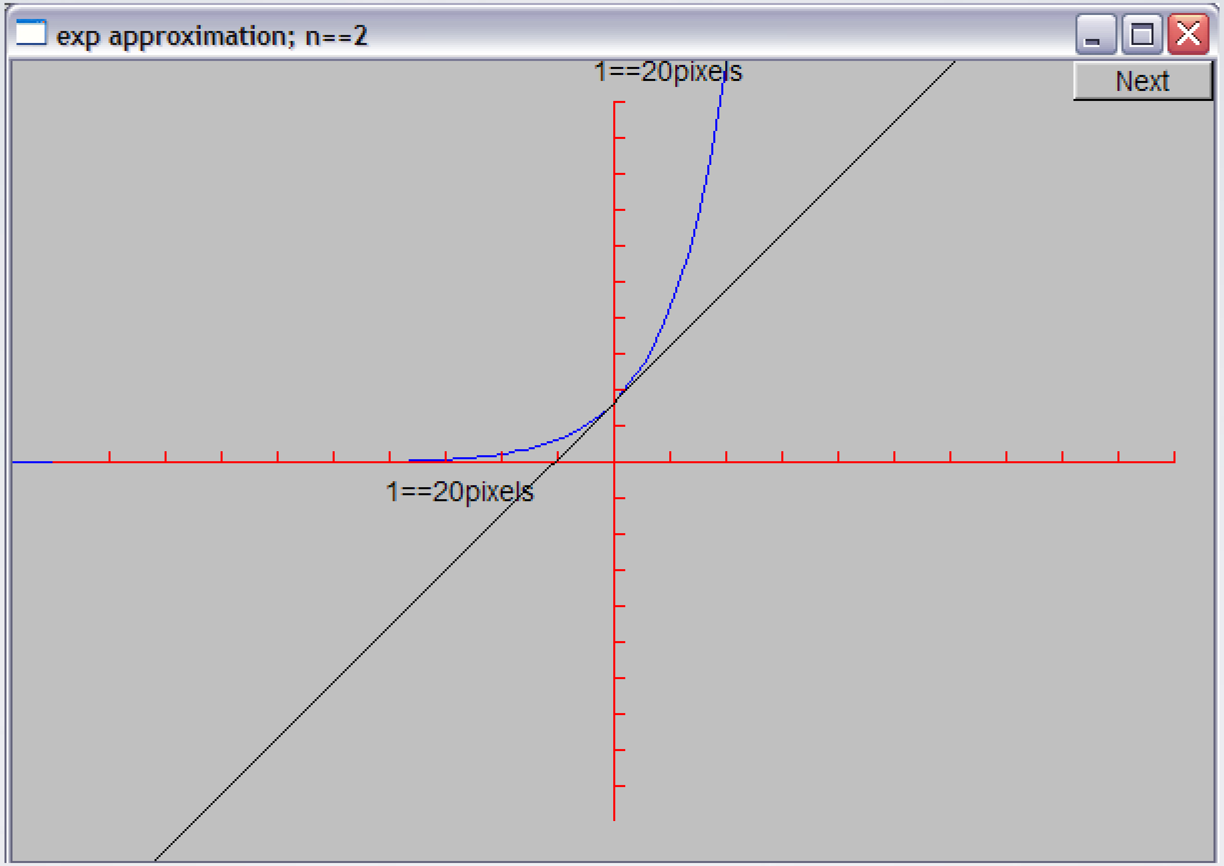
Demo n = 0



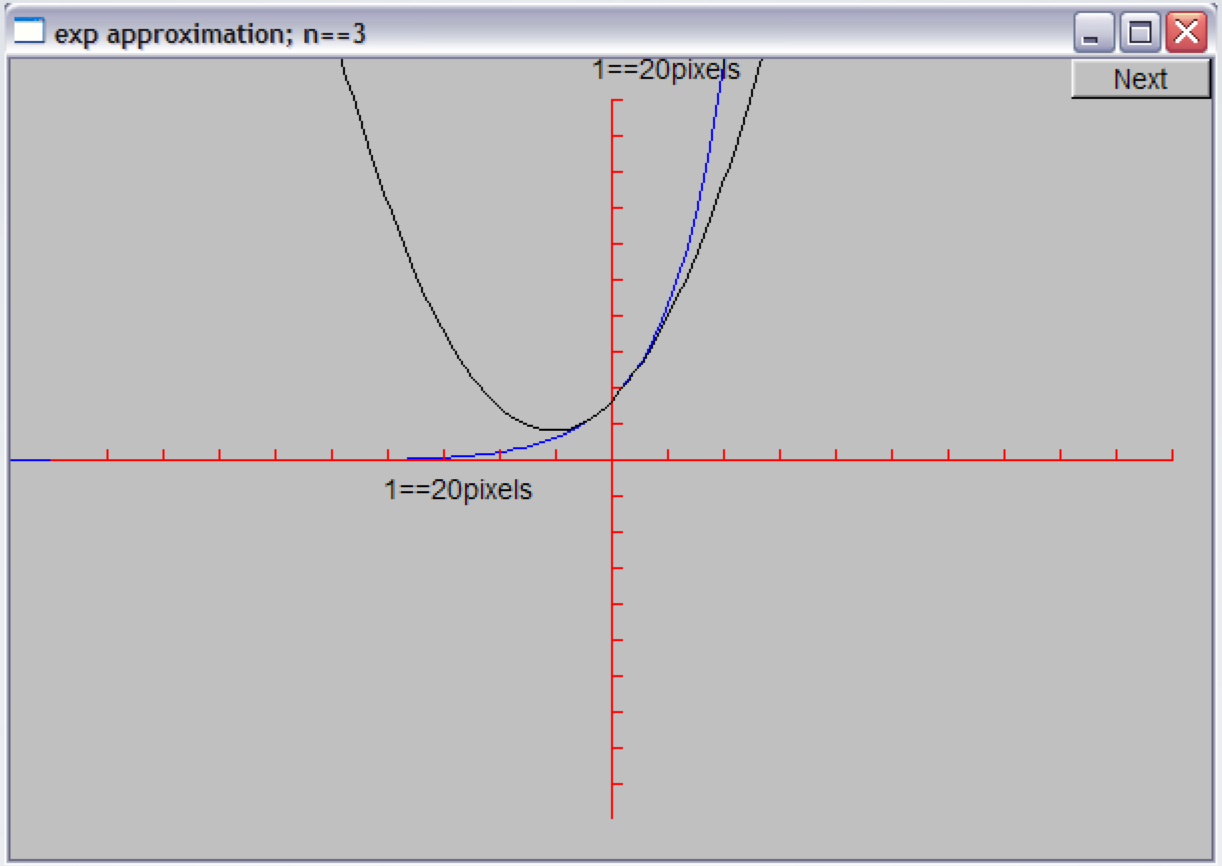
Demo n = 1



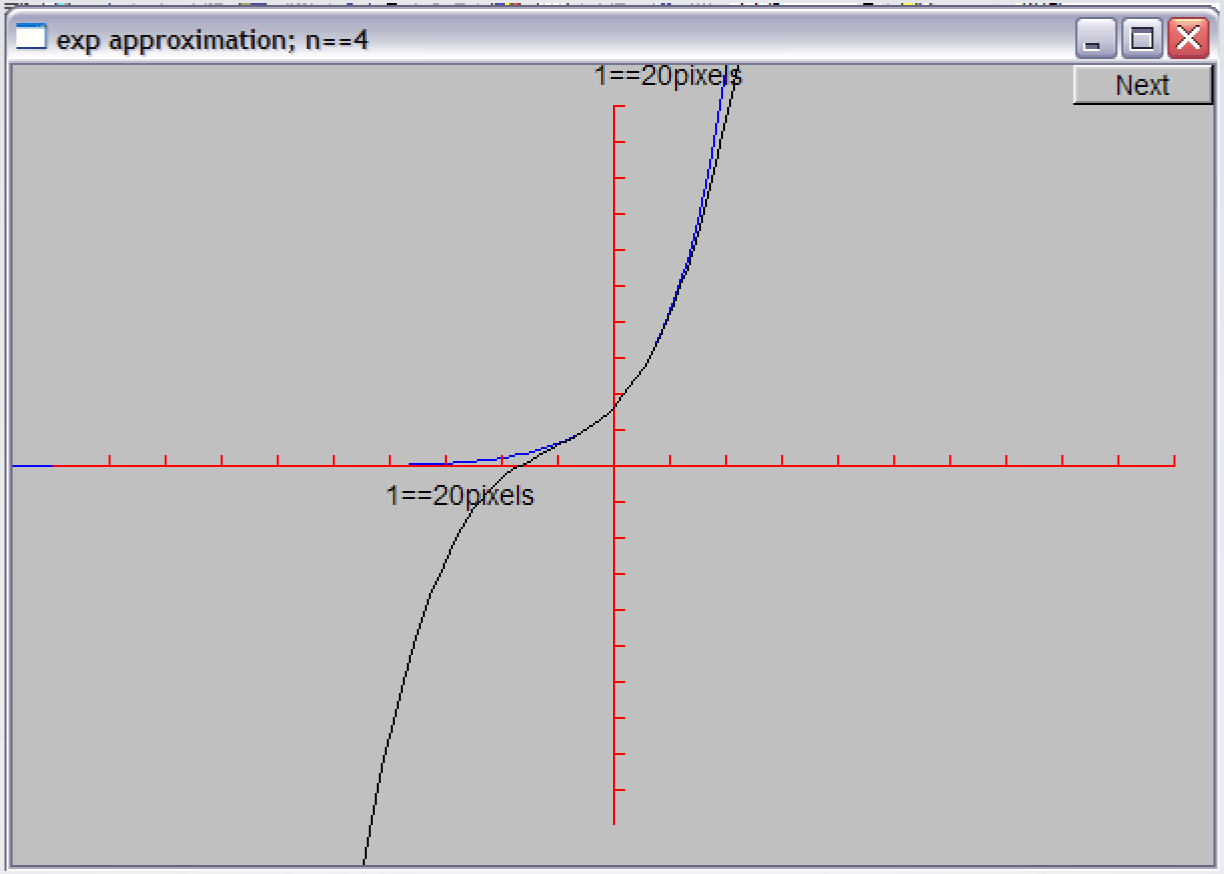
Demo n = 2



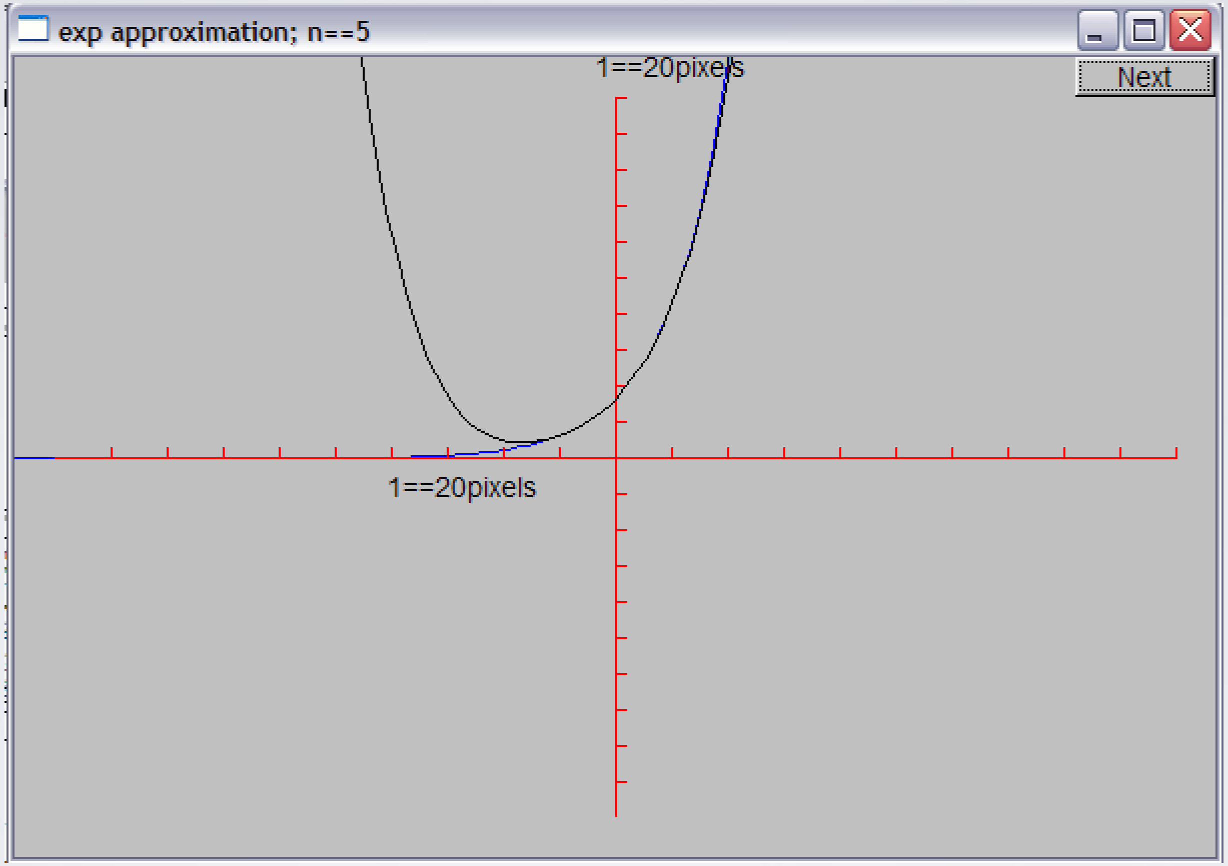
Demo n = 3



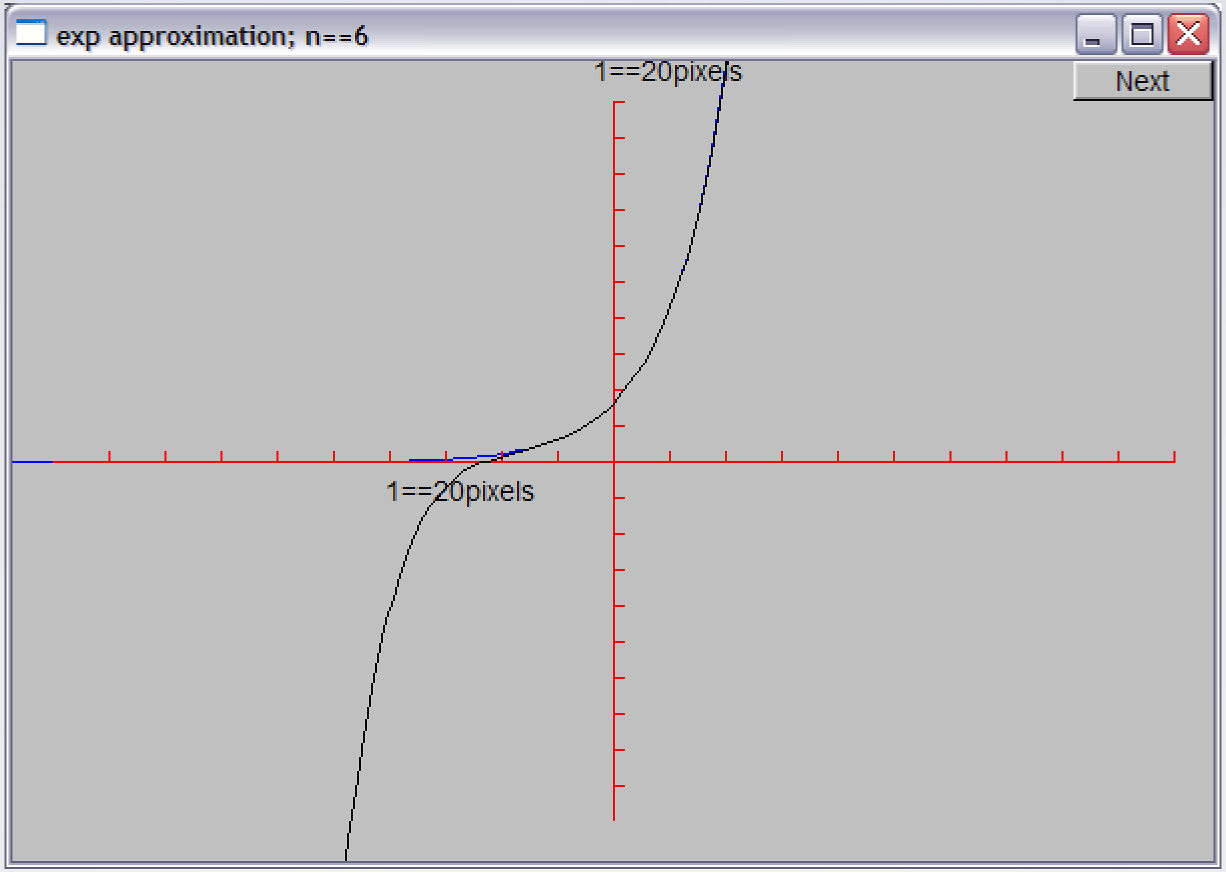
Demo n = 4



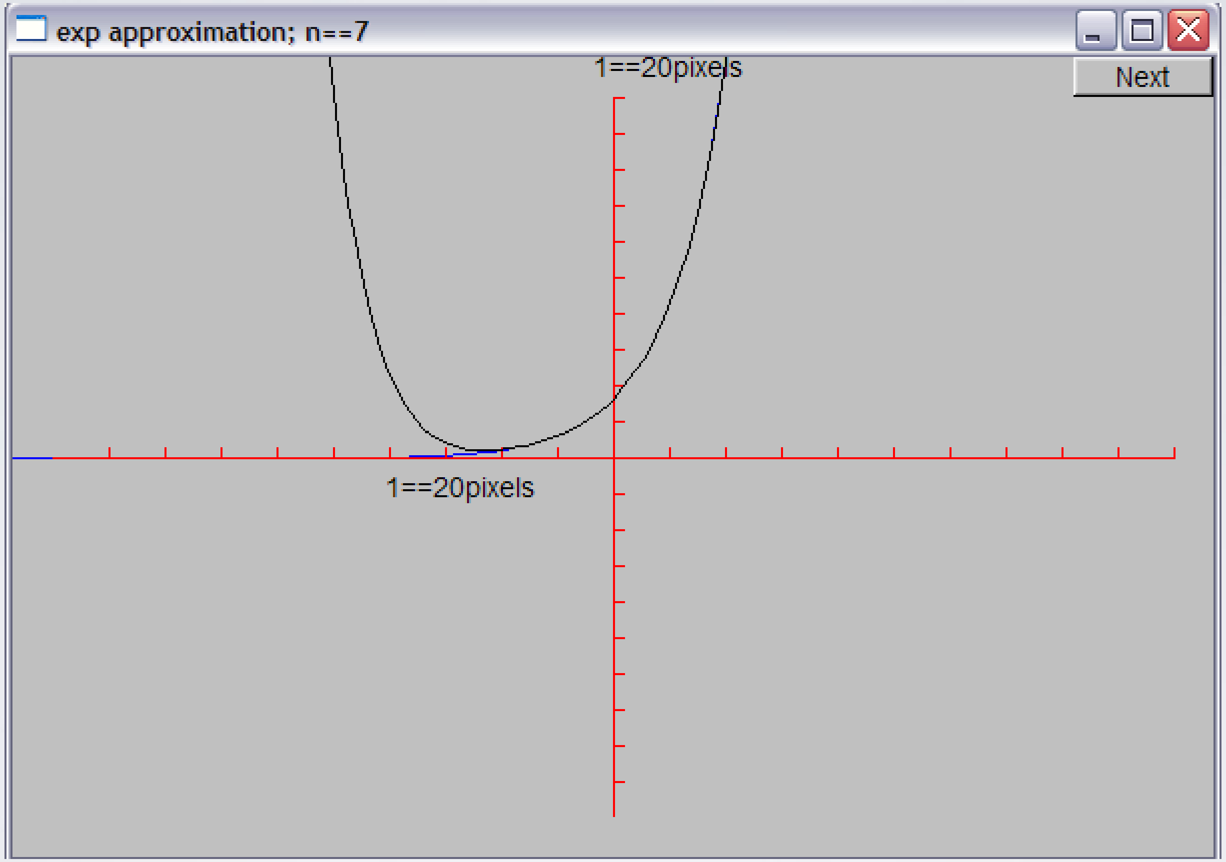
Demo n = 5



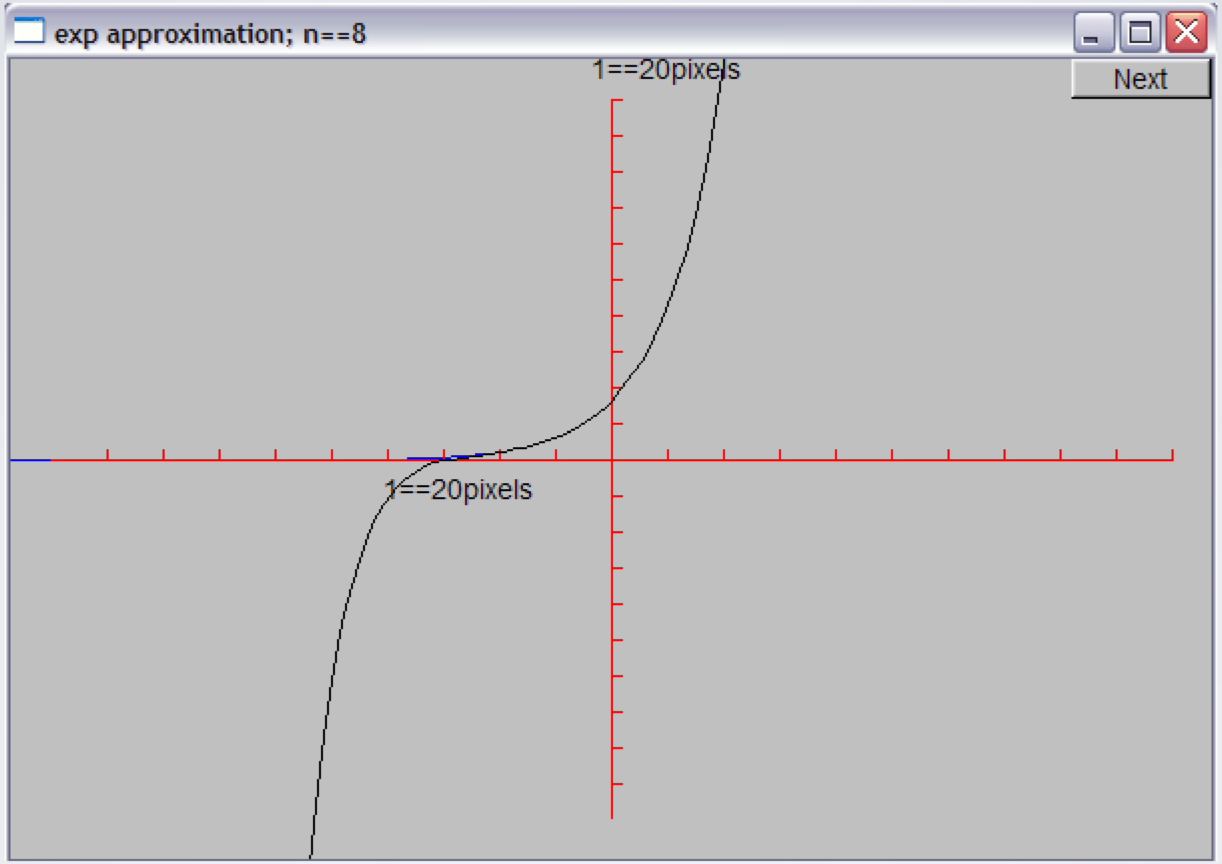
Demo n = 6



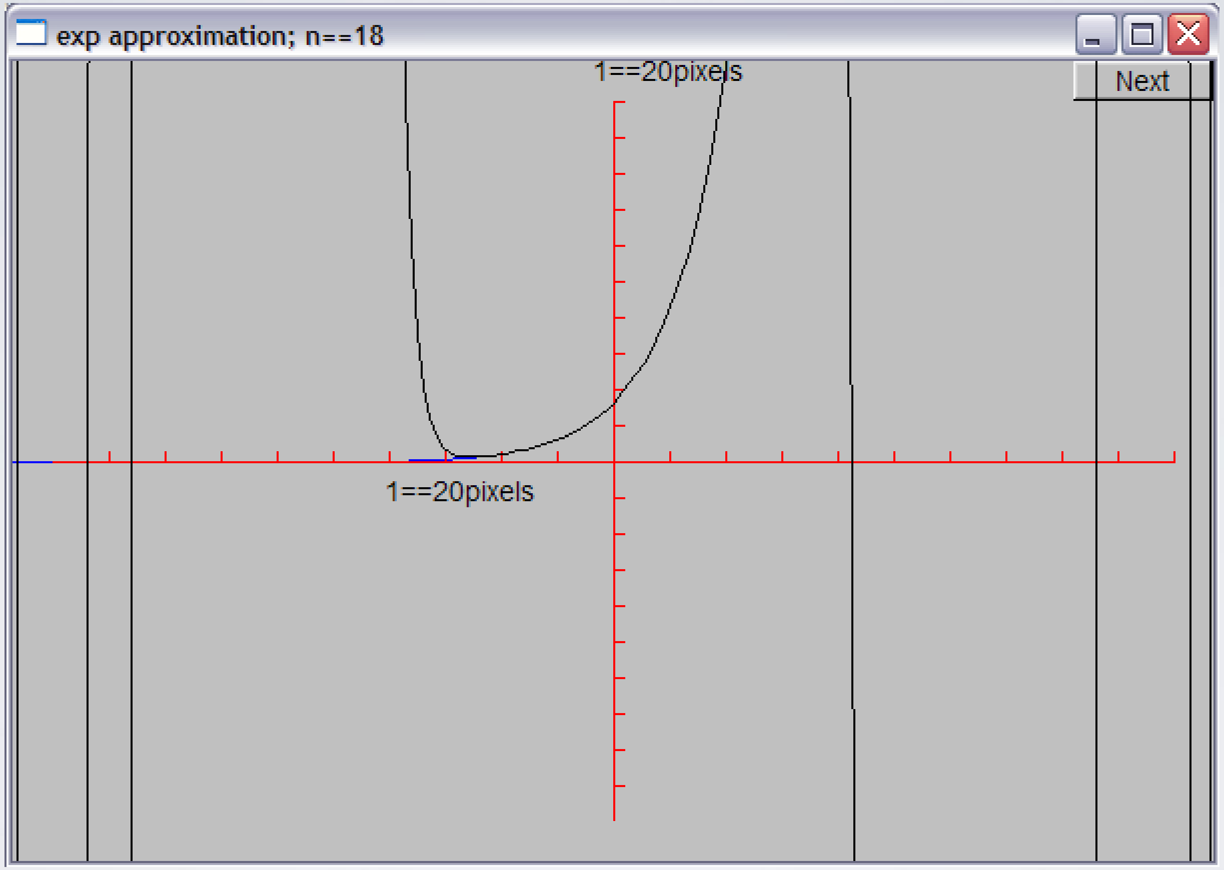
Demo n = 7



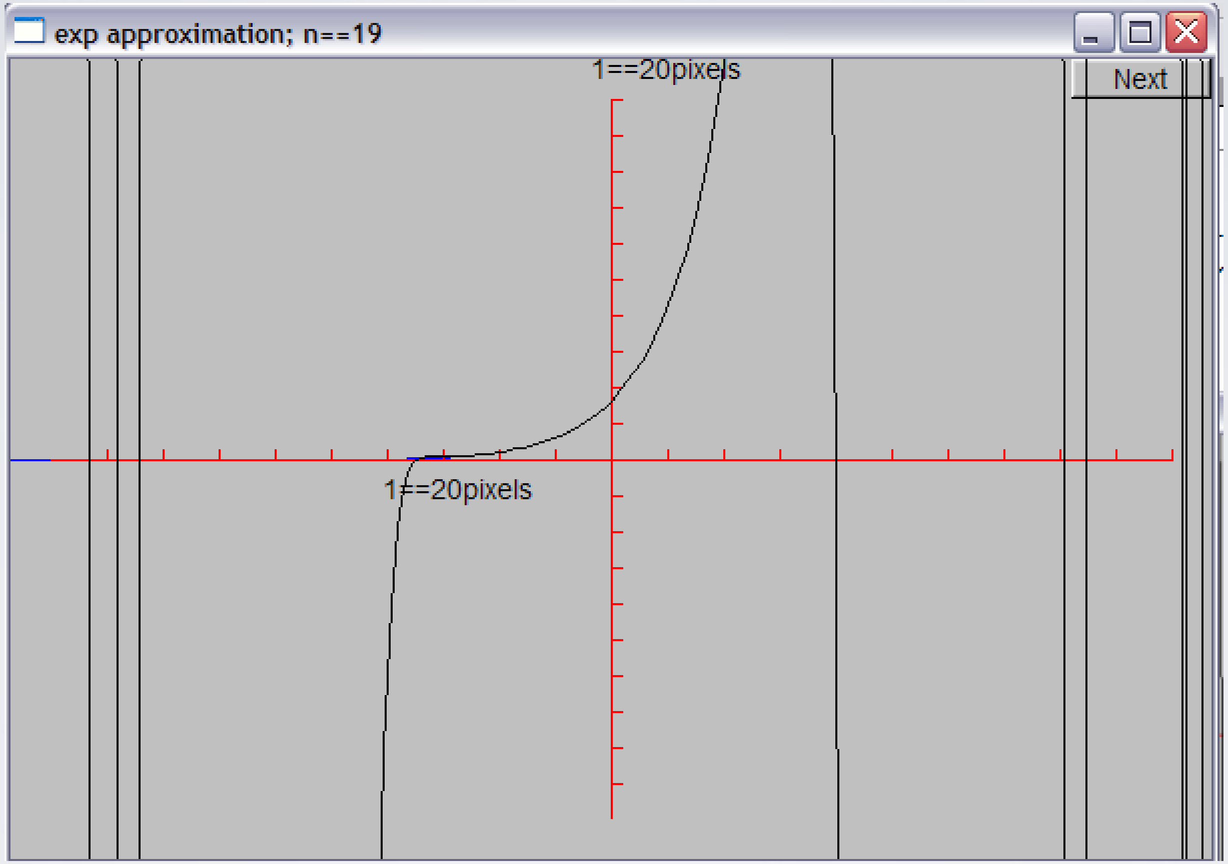
Demo n = 8



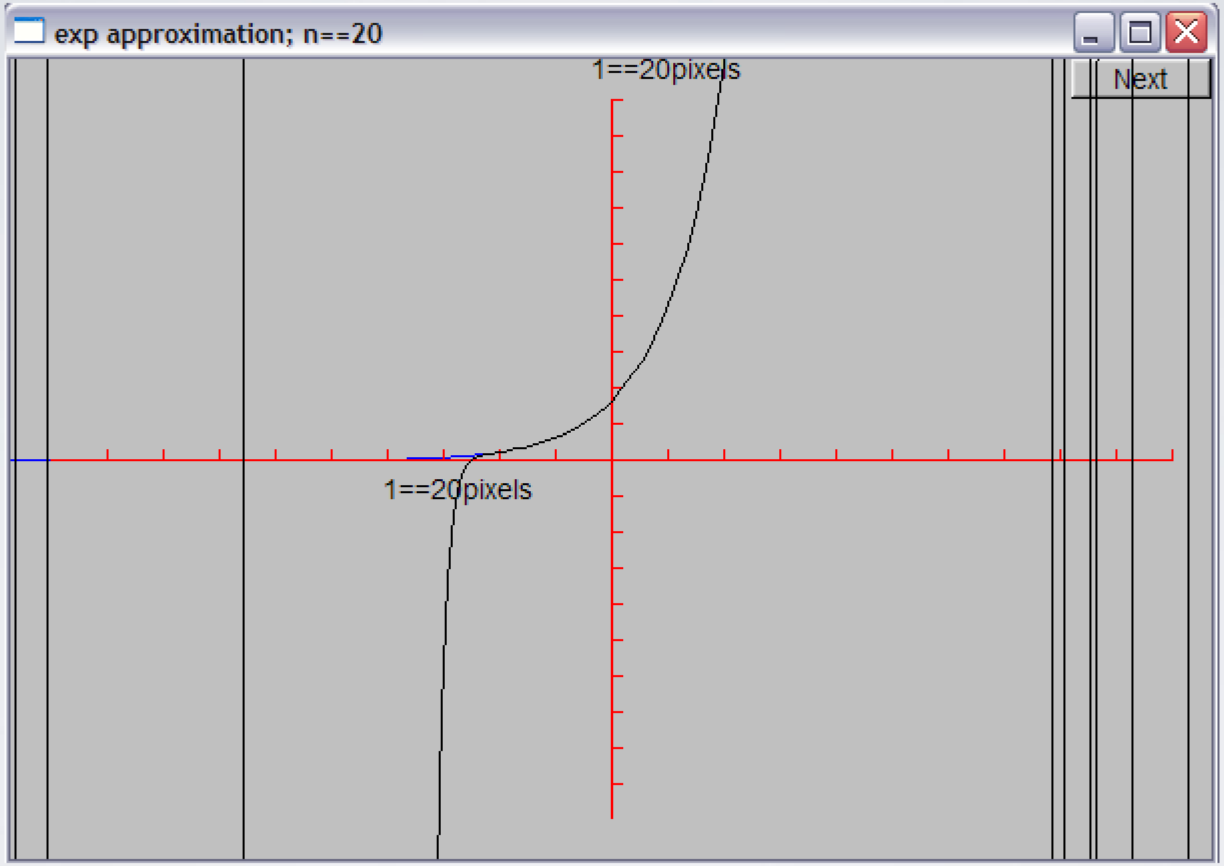
Demo n = 18



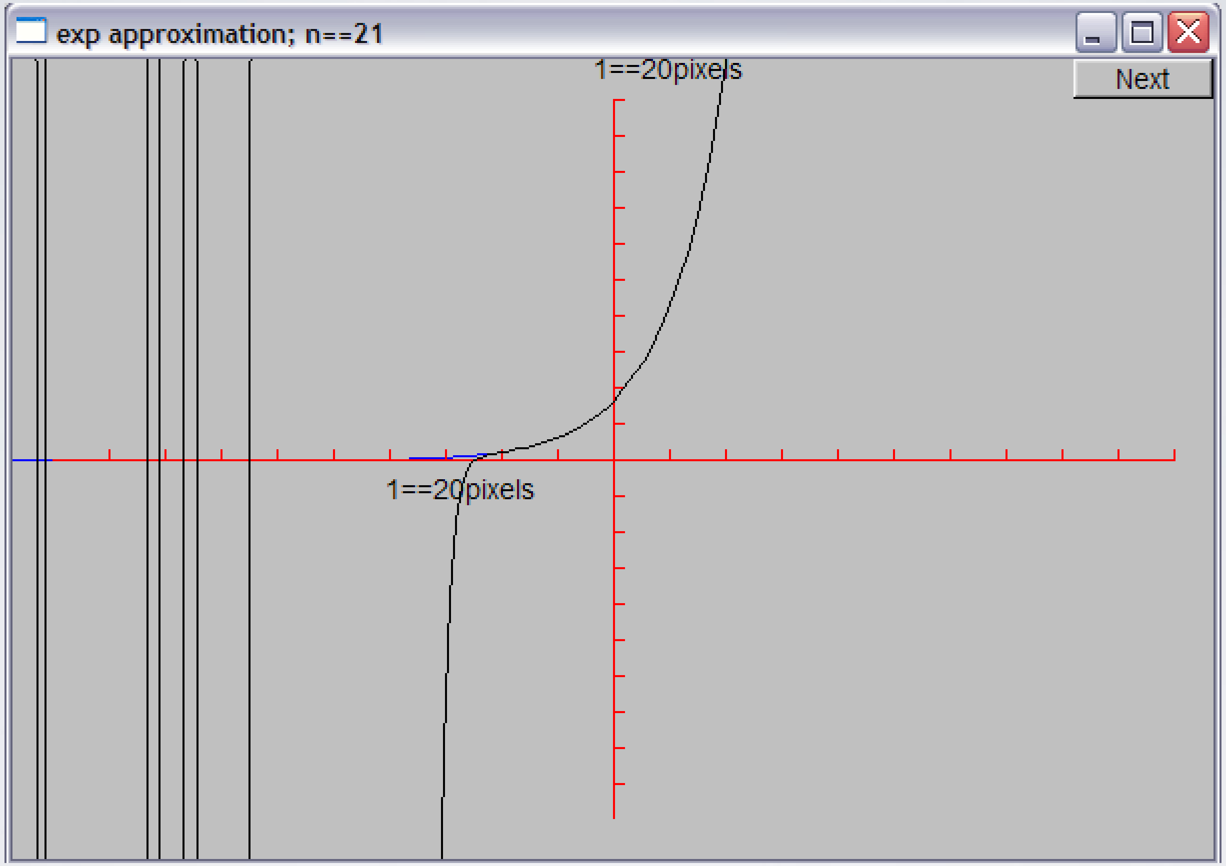
Demo n = 19



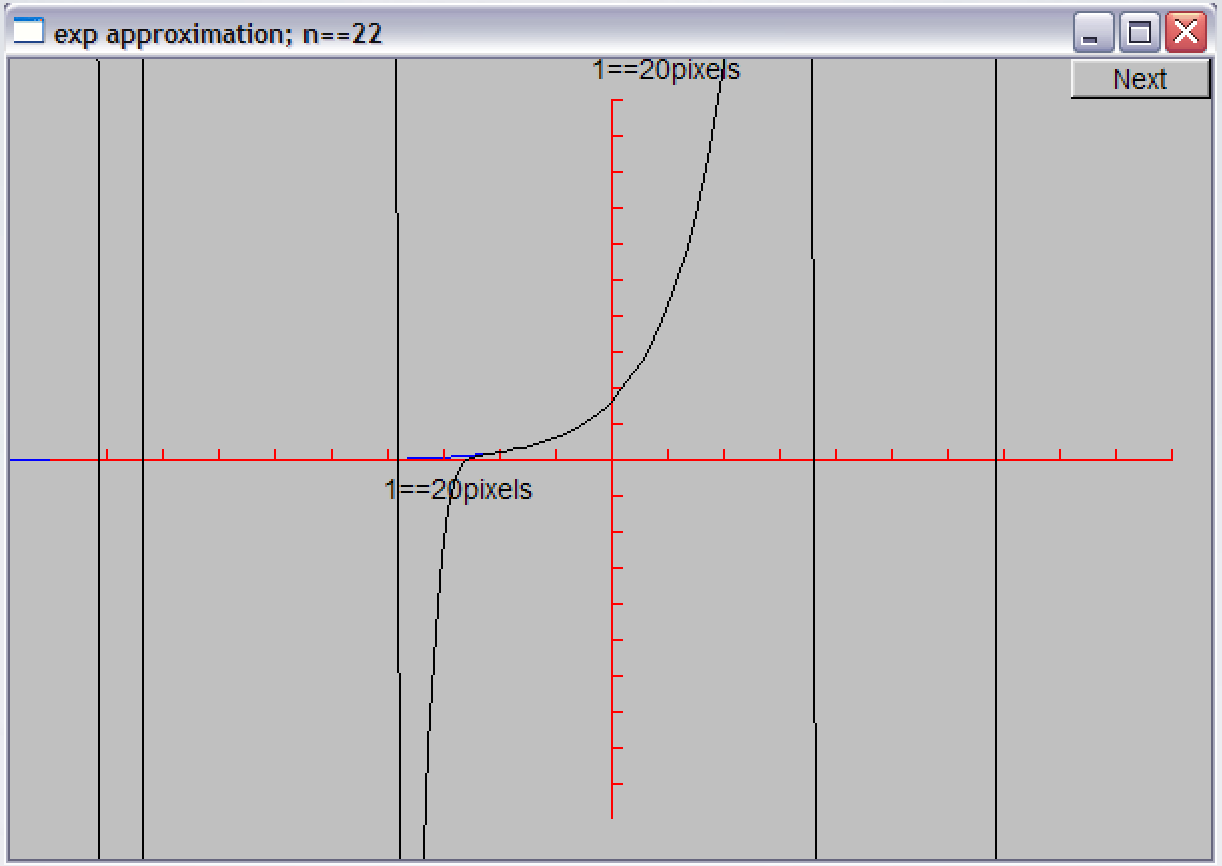
Demo n = 20



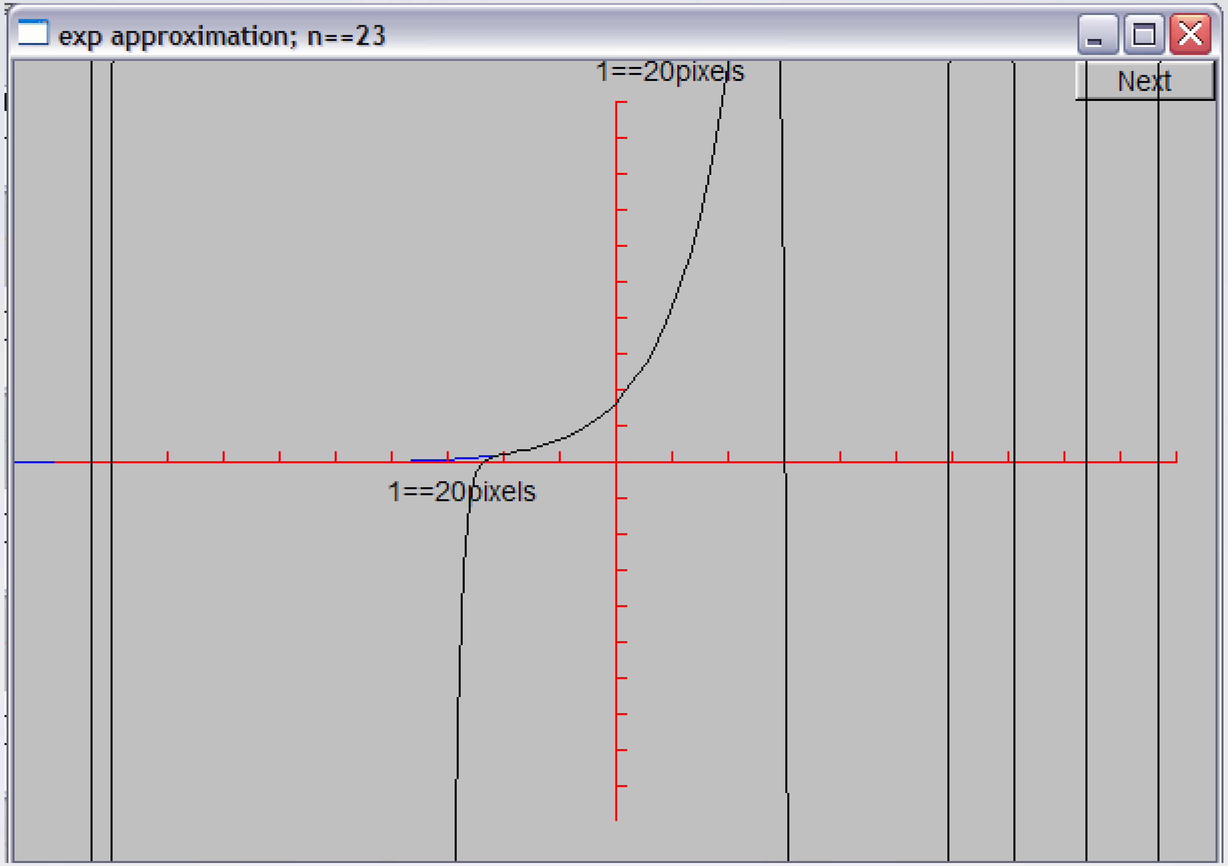
Demo n = 21



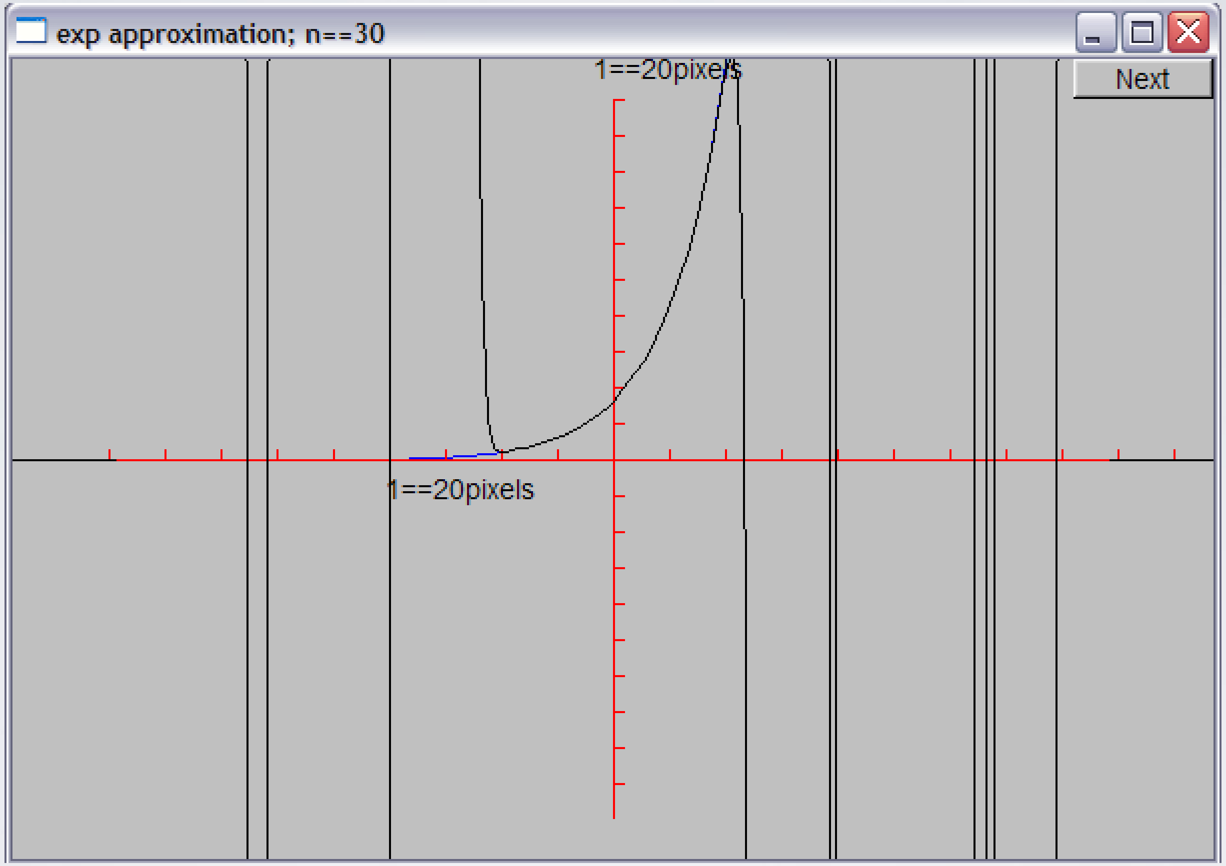
Demo n = 22



Demo n = 23



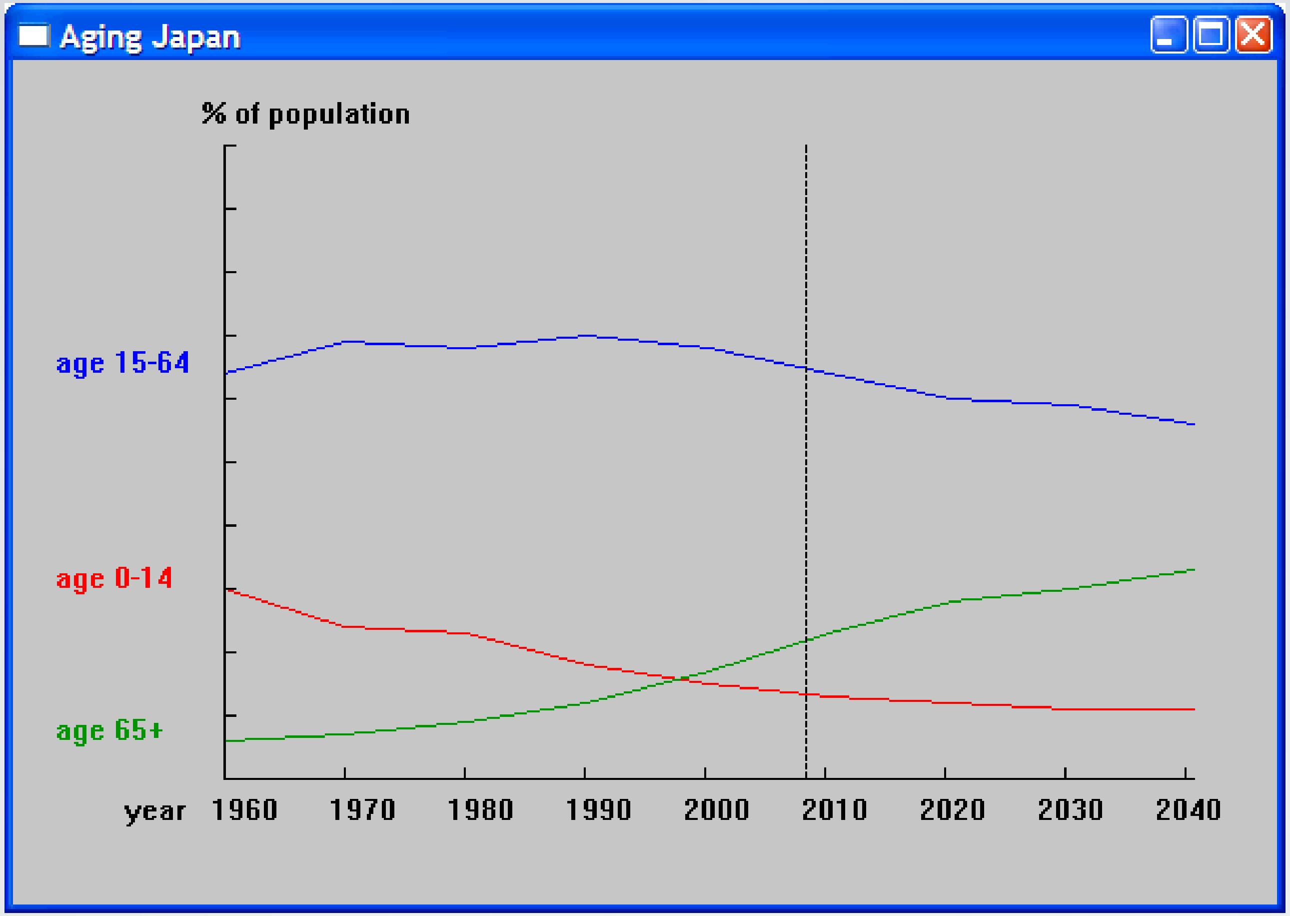
Demo n = 30



Why did the graph "go wild"?

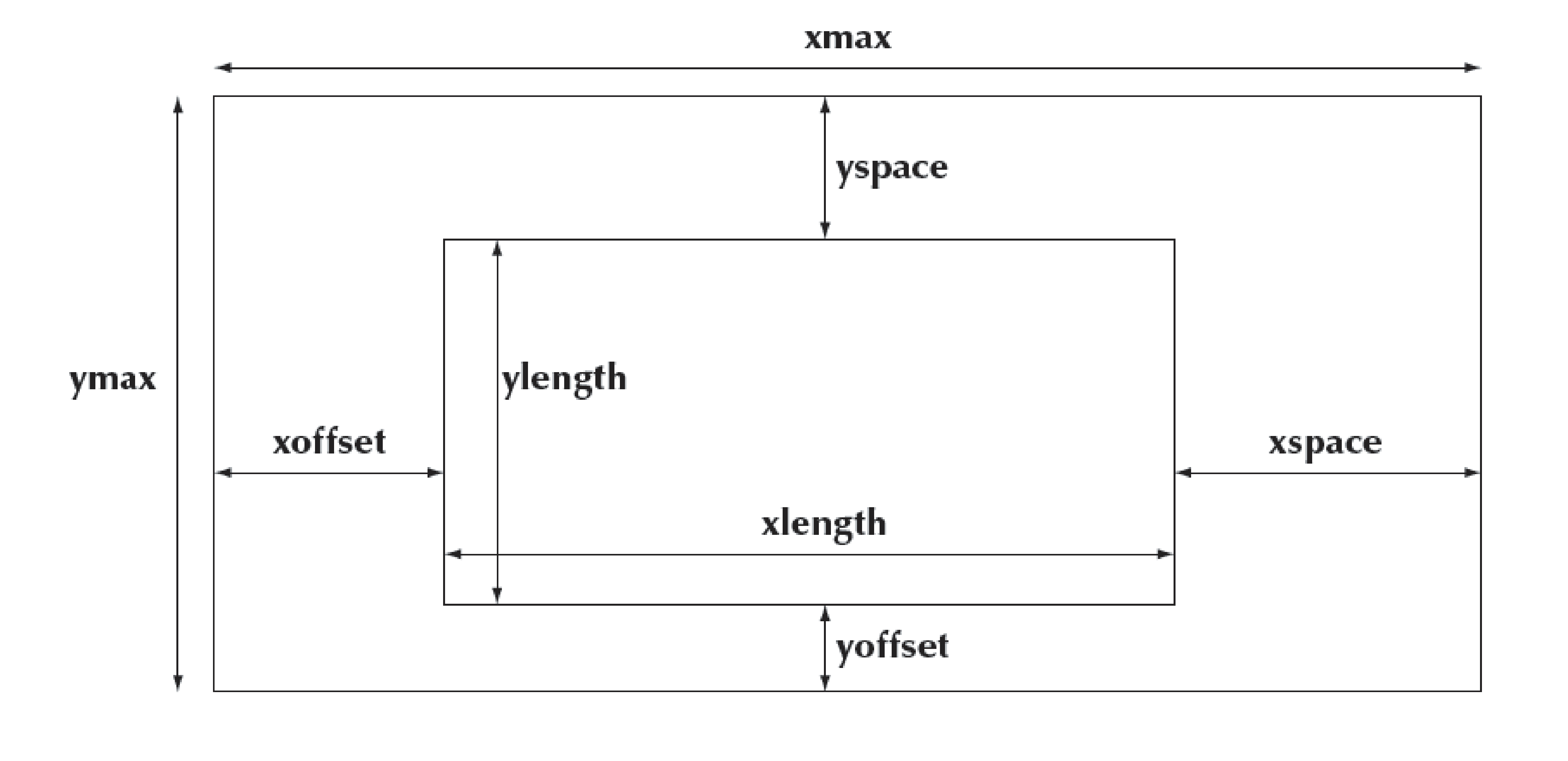
- Floating-point numbers are an approximations of real numbers
  - Just approximations
  - Real numbers can be arbitrarily large and arbitrarily small
    - Floating-point numbers are of a fixed size
  - Sometimes the approximation is not good enough for what you do
  - Here, small inaccuracies (rounding errors) built up into huge errors
- Always
  - be suspicious about calculations
  - check your results
  - hope that your errors are obvious
    - You want your code to break early – before anyone else gets to use it

Graphing data



- Often, what we want to graph is data, not a well-defined mathematical function
  - Here, we used three **Open_polyline**s

Graphing data



- Carefully design your screen layout

Code for Axis

```
struct Axis : Shape {
    enum Orientation { x, y, z };
    Axis(Orientation d, Point xy, int length,
            int number_of_notches=0,        // default: no notches
            string label = ""               // default : no label
            );

    void draw_lines() const;
    void move(int dx, int dy);

    void set_color(Color);         // in case we want to change the color of all parts at once

    // line stored in Shape
    // orientation not stored (can be deduced from line)
    Text label;
    Lines notches;
};
```

```
Axis::Axis(Orientation d, Point xy, int length, int n, string lab)
    :label(Point(0,0),lab)
{
    if (length<0) error("bad axis length");
    switch (d){
    case Axis::x:
    {       Shape::add(xy);                                 // axis line begin
            Shape::add(Point(xy.x+length,xy.y));            // axis line end
            if (1<n) {
                    int dist = length/n;
                    int x = xy.x+dist;
                    for (int i = 0; i<n; ++i) {
                            notches.add(Point(x,xy.y),Point(x,xy.y-5));
                            x += dist;
                    }
            }
            label.move(length/3,xy.y+20);      // put label under the line
            break;
    }
    // …
}
```

Axis implementation

```
void Axis::draw_lines() const
{
    Shape::draw_lines();   // the line
    notches.draw_lines();  // the notches may have a different color from the line
    label.draw();          // the label may have a different color from the line
}

void Axis::move(int dx, int dy)
{
    Shape::move(dx,dy);   // the line
    notches.move(dx,dy);
    label.move(dx,dy);
}

void Axis::set_color(Color c)
{
    // … the obvious three lines …
}
```

Next Lecture

- Graphical user interfaces
- Windows and Widgets
- Buttons and dialog boxes

Acknowledgements

Bjarne Stroustrup

Programming -- Principles and Practice Using C++

http://www.stroustrup.com/Programming/

Thank you!

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης