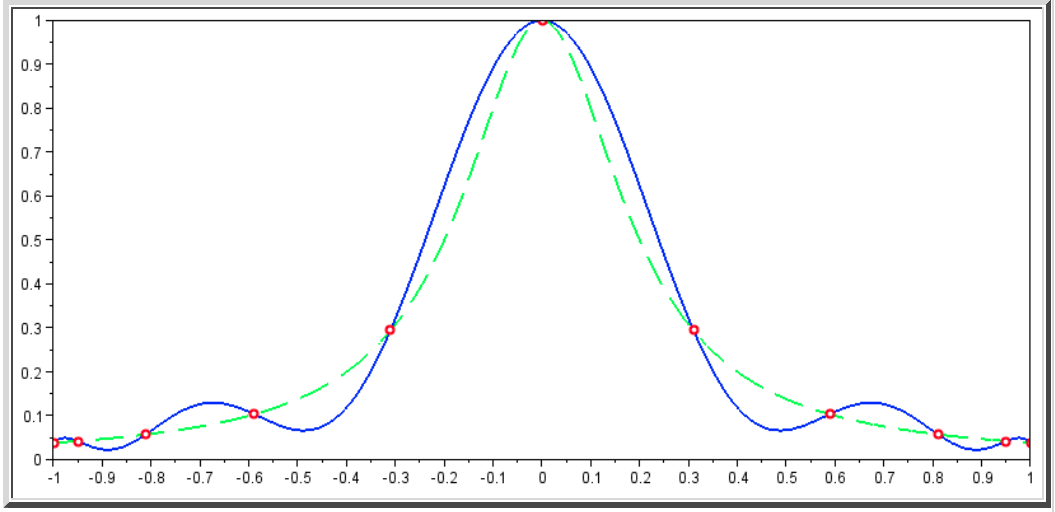
## How about more complex functions ?

- **Instruction set of a CPU**
  - The **instructions** that a CPU can execute can e divided into 3 categories:
    - **Data movement** (= copy)
    - **Arithmetic** (+, -, *, %, /), **logic** (and, or, not, including bitwise operations), **shift** or **rotate** instructions
    - **Branching** (including call and return)
  - Take a look at the list of **instructions** that the popular **Intel** can execute: **click here**

- **How does a computer compute more complex values, like "sin(x)"**
  - **Answer**:
    - Use ***interpolation* !!!!**



From **Mathematics**:

- We can **approximate** a (any) function with a **polynomial** to **arbitrary accuracy**

  (And we can **compute (evaulate)** a **polynomial** using ***only*** +, -, * and / operations !!!!)

(Google **"Taylor serie"** and **"Lagrange interpolation"** for more details.)

  - I found a **highly optimized** (= very good approximation with **very few operations**) of the **sin(x)** function:

$$sin(x)\ \sim=\ 0.775 * ( (4/\pi)*x + (4/\pi^2)*x^2 ) + 0.225 * ( (4/\pi)*x + (4/\pi^2)*x^2 )^2$$

  - Here is the code in **C**:

```
#define pi 3.14159265358979l

double mySine(double x)
{
    const double B = 4.0/pi;       // 2 special "interpolation constants
```

```
     const double C = 4.0/(pi*pi);

     double y;

     y = B*x - C*x*x;                // Highly optimized (= hokus pokus)
     y = 0.775*y + 0.225*y*y;        // approximation of sin(x)
     return y;
 }
```

- **Example Program:** (Demo above code)

  Example

  - Prog file: click here

**How to run the program:**

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc sin-appr.c -lm`
- To run: `./a.out`

**Output:**

```
x = 0.0, sin(x) = 0.000000, mySine(x) = 0.000000, Diff = 0.000000 ( NaN%)
x = 0.1, sin(x) = 0.099833, mySine(x) = 0.098954, Diff = 0.000879 (0.88%)
x = 0.2, sin(x) = 0.198669, mySine(x) = 0.197580, Diff = 0.001089 (0.55%)
x = 0.3, sin(x) = 0.295520, mySine(x) = 0.294617, Diff = 0.000903 (0.31%)
x = 0.4, sin(x) = 0.389418, mySine(x) = 0.388895, Diff = 0.000524 (0.13%)
x = 0.5, sin(x) = 0.479426, mySine(x) = 0.479329, Diff = 0.000097 (0.02%)
x = 0.6, sin(x) = 0.564642, mySine(x) = 0.564926, Diff = -0.000284 (0.05%)
x = 0.7, sin(x) = 0.644218, mySine(x) = 0.644781, Diff = -0.000564 (0.09%)
x = 0.8, sin(x) = 0.717356, mySine(x) = 0.718077, Diff = -0.000721 (0.10%)
x = 0.9, sin(x) = 0.783327, mySine(x) = 0.784086, Diff = -0.000759 (0.10%)
x = 1.0, sin(x) = 0.841471, mySine(x) = 0.842168, Diff = -0.000697 (0.08%)
x = 1.1, sin(x) = 0.891207, mySine(x) = 0.891773, Diff = -0.000565 (0.06%)
x = 1.2, sin(x) = 0.932039, mySine(x) = 0.932438, Diff = -0.000399 (0.04%)
x = 1.3, sin(x) = 0.963558, mySine(x) = 0.963792, Diff = -0.000234 (0.02%)
x = 1.4, sin(x) = 0.985450, mySine(x) = 0.985549, Diff = -0.000099 (0.01%)
x = 1.5, sin(x) = 0.997495, mySine(x) = 0.997513, Diff = -0.000018 (0.00%)
```

As you know, sin(x) is **periodic**.

The values compared are between **[0..π/2]**; which is the main period.

You can always **reduce any x value** to **some value inside this range** (and then to obtain the function value).

Doing so little work to get to **< 1% error** in sin(x) for **any value of x** is not too shaby !!!!

- **Experiment**
  - I found this page on a **high accurate** approximation of **sin/cos**:

```
 //always wrap input angle to -PI..PI
 if (x < -3.14159265)
     x += 6.28318531;
 else
 if (x >  3.14159265)
     x -= 6.28318531;

 //compute sine
 if (x < 0)
 {
     sin = 1.27323954 * x + .405284735 * x * x;

     if (sin < 0)
         sin = .225 * (sin *-sin - sin) + sin;
     else
         sin = .225 * (sin * sin - sin) + sin;
 }
 else
 {
     sin = 1.27323954 * x - 0.405284735 * x * x;

     if (sin < 0)
         sin = .225 * (sin *-sin - sin) + sin;
     else
```

```
        sin = .225 * (sin * sin - sin) + sin;
}

//compute cosine: sin(x + PI/2) = cos(x)
x += 1.57079632;
if (x >  3.14159265)
    x -= 6.28318531;

if (x < 0)
{
    cos = 1.27323954 * x + 0.405284735 * x * x;

    if (cos < 0)
        cos = .225 * (cos *-cos - cos) + cos;
    else
        cos = .225 * (cos * cos - cos) + cos;
}
else
{
    cos = 1.27323954 * x - 0.405284735 * x * x;

    if (cos < 0)
        cos = .225 * (cos *-cos - cos) + cos;
    else
        cos = .225 * (cos * cos - cos) + cos;
}
```

The URL: http://lab.polygonal.de/?p=205

**Notice** you ***only*** use **arithmetic opertions** !!!

Program it and see how **accurate** it is....