CpPGFplots in Org Mode

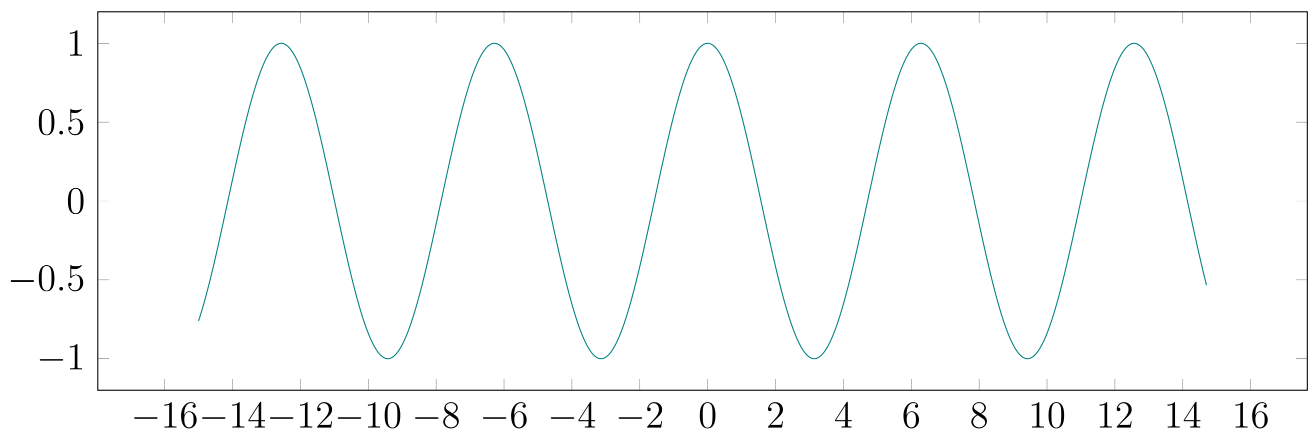
alexpaniman

CpPGFplots is a library that aims to make plotting: consistent and easy in C++. It uses pgfplots backend to produce images, wrapping it's syntax with a powerful programming language and choosing defaults that produce good-looking graphs as frequently as possible (and tries to be smart about it when possible).

Showcase

To plot a simple $cos(x)$ you just need to use `function` and specify range. Library will automatically determine necessary amount of samples and output latex code you can embed in your article (only requirement for your latex is to use `pgfplots`, but that's obvious).

```
function(cos(x), { -15, 15 });
```



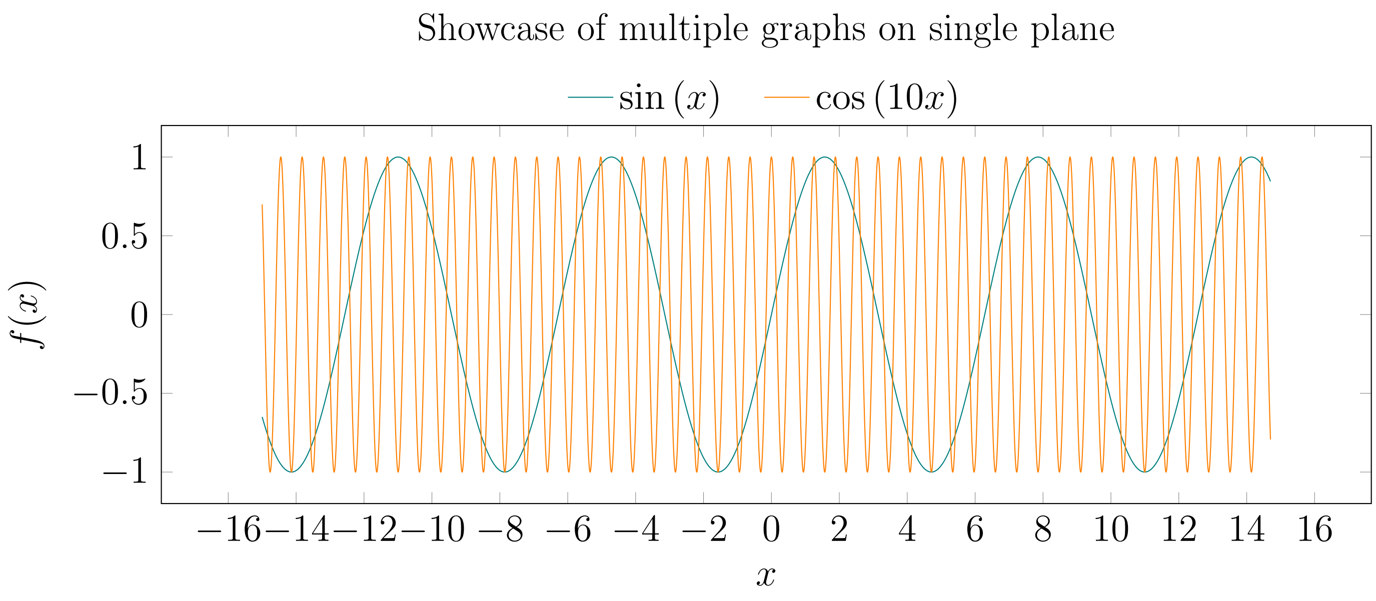
A little more elaborate example showcases multiple graphs on single plane, labels and legend:

```
auto plot_sin = function(sin(x),    { -15, 15 });
plot_sin.name = "$\\sin \\left ( x \\right )$";

auto plot_cos = function(cos(10*x), { -15, 15 });
plot_cos.name = "$\\cos \\left ( 10 x \\right )$";

plotting_plane plane = plot_sin + plot_cos;
plane.name = "Showcase of multiple graphs on single plane";

auto &[x_axis, y_axis] = plane.axes;
x_axis.label = "$x$";
y_axis.label = "$f(x)$";
```

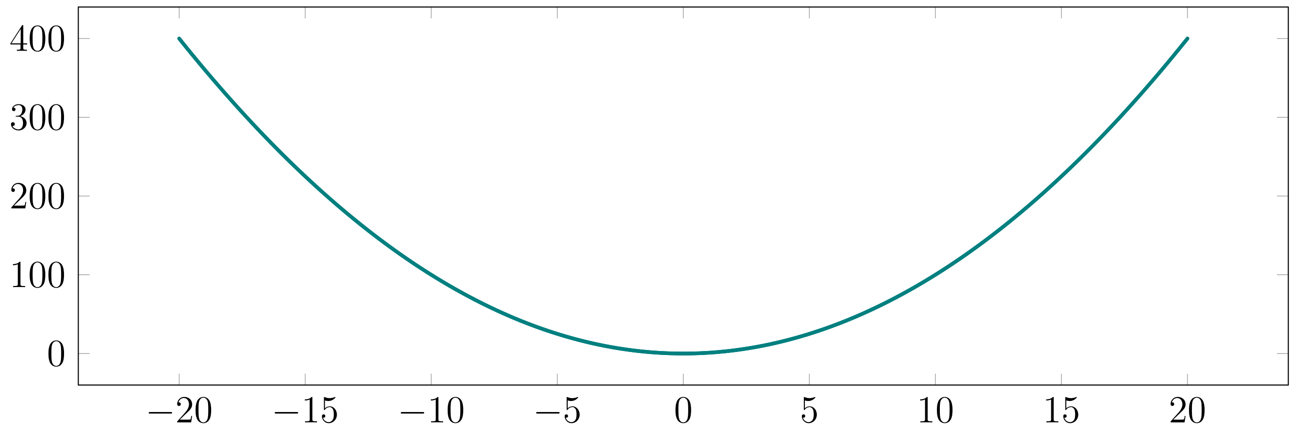


There are, of course, a large number of parameters you can tweak to get different results, let's, for example, see sampled points in on this graph:

```
auto parabola = function(x*x, { - 20, 20 });
parabola.mark_size = 0.5; // Enable marks

plotting_plane plane("This is a parabola ($x^2$)");
plane + parabola;
```

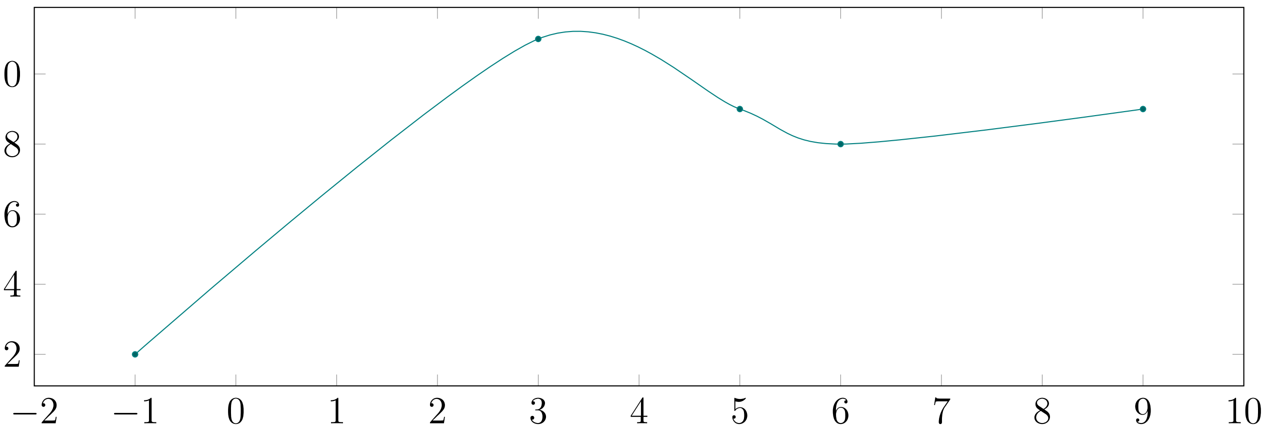
This is a parabola (x^2)



This graph showcases plotting from points and a new interpolation mode (before that we only built plots with segments), here it's interpolated smoothly, hence the name.

```
auto current_plot = points(
    { -1,  2 },
    {  3, 11 },
    {  5,  9 },
    {  6,  8 },
    {  9,  9 }
);

current_plot.interpolate = SMOOTH;
```



And, having all of C++ around, you can do lots of things you wouldn't be able to do in pure pgfplots (or, it would be harder, at lease). Let's, for example, build a progression of fibonacci numbers:

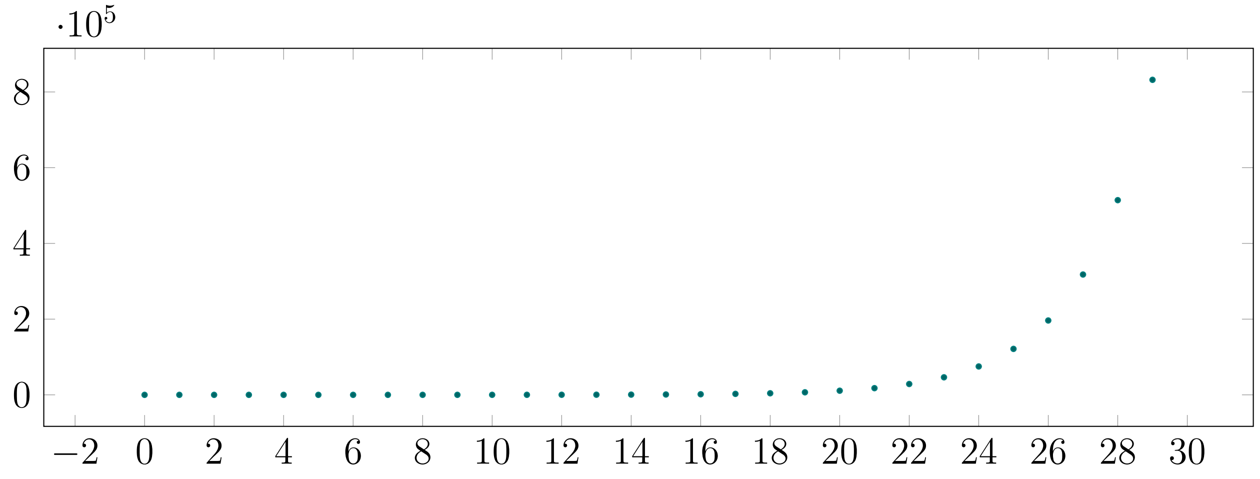
```
std::vector<vec2> points;

auto fib = [](auto &&rec, int x) {
    if (x == 0 || x == 1)
        return 1;

    return rec(rec, x - 1) + rec(rec, x - 2);
};

for (int i = 0; i < 30; ++ i)
    points.emplace_back(i, fib(fib, i));

plot current_plot { points };
current_plot.interpolate = NONE;
```


$\cdot 10^5$