



## GPT-4 at vttoth

### Conversation:

Can you write a JavaScript subroutine that plots a numerical function? The subroutine should take the necessary parameters including the function to be plotted, and return an SVG.

Here's a simple JavaScript subroutine that plots a numerical function and returns an SVG. This example uses the D3.js library, so make sure to include it in your HTML file.

```
function plotFunction(func, xMin, xMax, yMin, yMax, width, height) {
  // Create an SVG element
  const svg = d3.create("svg")
    .attr("width", width)
    .attr("height", height);

  // Create scales for the x and y axes
  const xScale = d3.scaleLinear()
    .domain([xMin, xMax])
    .range([0, width]);
  const yScale = d3.scaleLinear()
    .domain([yMin, yMax])
    .range([height, 0]);

  // Create the x and y axes
  const xAxis = d3.axisBottom(xScale);
  const yAxis = d3.axisLeft(yScale);

  // Add the axes to the SVG
  svg.append("g")
    .attr("transform", `translate(0, ${height})`)
    .call(xAxis);
  svg.append("g")
    .call(yAxis);

  // Create a line generator
  const line = d3.line()
    .x(d => xScale(d.x))
    .y(d => yScale(d.y));

  // Generate data points for the function
  const data = d3.range(xMin, xMax, 0.01).map(x => ({ x, y: func(x) }));

  // Add the function plot to the SVG
  svg.append("path")
    .datum(data)
    .attr("fill", "none")
    .attr("stroke", "steelblue")
    .attr("stroke-width", 1.5)
    .attr("d", line);

  return svg.node();
}
```

To use this function, you need to provide a numerical function, the x and y axis limits, and the width and height of the SVG. For example:

```
const myFunc = x => Math.sin(x);
const svgNode = plotFunction(myFunc, -2 * Math.PI, 2 * Math.PI, -1, 1, 800, 400);
document.body.appendChild(svgNode);
```

This will plot the sine function between  $-2\pi$  and  $2\pi$  and append the resulting SVG to the body of the HTML document.

## Can we avoid using third-party libraries and still keep it simple?

Yes, you can create a simple JavaScript function to plot a numerical function and return an SVG without using third-party libraries. Here's an example:

```
function plotFunction(func, xMin, xMax, yMin, yMax, width, height) {
  // Create an SVG element
  const svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
  svg.setAttribute("width", width);
  svg.setAttribute("height", height);

  // Create scales for the x and y axes
  const xScale = x => (x - xMin) / (xMax - xMin) * width;
  const yScale = y => height - (y - yMin) / (yMax - yMin) * height;

  // Generate data points for the function
  const data = Array.from({ length: width }, (_, i) => {
    const x = xMin + i * (xMax - xMin) / width;
    return { x, y: func(x) };
  });

  // Add the function plot to the SVG
  const pathData = data.map((d, i) => {
    const x = xScale(d.x);
    const y = yScale(d.y);
    return (i === 0 ? "M" : "L") + x + "," + y;
  }).join(" ");

  const path = document.createElementNS("http://www.w3.org/2000/svg", "path");
  path.setAttribute("d", pathData);
  path.setAttribute("fill", "none");
  path.setAttribute("stroke", "steelblue");
  path.setAttribute("stroke-width", 1.5);
  svg.appendChild(path);

  return svg;
}
```

To use this function, you need to provide a numerical function, the x and y axis limits, and the width and height of the SVG. For example:

```
const myFunc = x => Math.sin(x);
const svgNode = plotFunction(myFunc, -2 * Math.PI, 2 * Math.PI, -1, 1, 800, 400);
document.body.appendChild(svgNode);
```

This will plot the sine function between  $-2\pi$  and  $2\pi$  and append the resulting SVG to the body of the HTML document.

## Wonderful, I love it. You do realize why I prefer to avoid 3rd party code, right?

Yes, I understand why you might prefer to avoid third-party code. There are several reasons for this preference:

1. **Dependency management:** Using third-party libraries can introduce additional dependencies, which may require updates and maintenance over time.
2. **Size and performance:** Third-party libraries can increase the size of your codebase, potentially affecting the performance and load times of your application.
3. **Security:** Relying on external code can introduce potential security risks if the library has vulnerabilities or if it is compromised.
4. **Control and customization:** By writing your own code, you have more control over the implementation and can easily customize it to fit your specific needs.

By creating a solution without third-party libraries, you can address these concerns and have a more lightweight, self-contained, and secure implementation.

## Going back to the plotting function you just created for me. How much trouble would it be to add the ability to plot several functions, for instance by passing to it optionally an array of functions as opposed to a single function?

It's not much trouble to modify the `plotFunction` to accept an array of functions. You can update the function to loop through the array of functions and plot each one. Here's an updated version of the `plotFunction`:

```
function plotFunction(functions, xmin, xmax, ymin, ymax, width, height) {
  if (!Array.isArray(functions)) {
    functions = [functions];
  }

  // Create an SVG element
  const svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
  svg.setAttribute("width", width);
  svg.setAttribute("height", height);

  // Create scales for the x and y axes
  const xScale = x => (x - xmin) / (xmax - xmin) * width;
  const yScale = y => height - (y - ymin) / (ymax - ymin) * height;

  functions.forEach(func => {
    // Generate data points for the function
    const data = Array.from({ length: width }, (_, i) => {
      const x = xmin + i * (xmax - xmin) / width;
      return { x, y: func(x) };
    });

    // Add the function plot to the SVG
    const pathData = data.map((d, i) => {
      const x = xScale(d.x);
      const y = yScale(d.y);
```

```

    return (i === 0 ? "M" : "L") + x + "," + y;
  }).join(" ");

  const path = document.createElementNS("http://www.w3.org/2000/svg", "path");
  path.setAttribute("d", pathData);
  path.setAttribute("fill", "none");
  path.setAttribute("stroke", "steelblue");
  path.setAttribute("stroke-width", 1.5);
  svg.appendChild(path);
});

return svg;
}

```

Now you can pass an array of functions to the `plotFunction`:

```

const myFunc1 = x => Math.sin(x);
const myFunc2 = x => Math.cos(x);
const svgNode = plotFunction([myFunc1, myFunc2], -2 * Math.PI, 2 * Math.PI, -1, 1,
800, 400);
document.body.appendChild(svgNode);

```

This will plot both the sine and cosine functions between and and append the resulting SVG to the body of the HTML document.

**Amazing, I like it very much. Works out of the box. Allow me to up the ante. Can we "spruce up" this function with options to draw axes and axis labels, regular or boxed, and also pass options to the subroutine to change the appearance of the plot (color, type of line) of each function?**

Certainly! I've updated the `plotFunction` to include options for drawing axes, axis labels, and customizing the appearance of each function plot. Here's the updated version:

```

function plotFunction(functions, xMin, xMax, yMin, yMax, width, height, options = {})
{
  if (!Array.isArray(functions)) {
    functions = [functions];
  }

  // Default options
  const defaultOptions = {
    drawAxes: true,
    axisLabels: { x: 'x', y: 'y' },
    plotStyles: functions.map(() => ({ color: 'steelblue', lineWidth: 1.5, lineStyle:
'solid' })),
  };

  // Merge user options with default options
  options = { ...defaultOptions, ...options };

  // Create an SVG element
  const svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
  svg.setAttribute("width", width);
  svg.setAttribute("height", height);

  // Create scales for the x and y axes

```

```

const xScale = x => (x - xMin) / (xMax - xMin) * width;
const yScale = y => height - (y - yMin) / (yMax - yMin) * height;

if (options.drawAxes) {
  // Draw x-axis
  const xAxis = document.createElementNS("http://www.w3.org/2000/svg", "line");
  xAxis.setAttribute("x1", xScale(xMin));
  xAxis.setAttribute("y1", yScale(0));
  xAxis.setAttribute("x2", xScale(xMax));
  xAxis.setAttribute("y2", yScale(0));
  xAxis.setAttribute("stroke", "black");
  xAxis.setAttribute("stroke-width", 1);
  svg.appendChild(xAxis);

  // Draw y-axis
  const yAxis = document.createElementNS("http://www.w3.org/2000/svg", "line");
  yAxis.setAttribute("x1", xScale(0));
  yAxis.setAttribute("y1", yScale(yMin));
  yAxis.setAttribute("x2", xScale(0));
  yAxis.setAttribute("y2", yScale(yMax));
  yAxis.setAttribute("stroke", "black");
  yAxis.setAttribute("stroke-width", 1);
  svg.appendChild(yAxis);

  // Add axis labels
  const xLabel = document.createElementNS("http://www.w3.org/2000/svg", "text");
  xLabel.textContent = options.axisLabels.x;
  xLabel.setAttribute("x", xScale(xMax) - 10);
  xLabel.setAttribute("y", yScale(0) - 5);
  svg.appendChild(xLabel);

  const yLabel = document.createElementNS("http://www.w3.org/2000/svg", "text");
  yLabel.textContent = options.axisLabels.y;
  yLabel.setAttribute("x", xScale(0) + 5);
  yLabel.setAttribute("y", yScale(yMax) + 10);
  svg.appendChild(yLabel);
}

functions.forEach((func, index) => {
  // Generate data points for the function
  const data = Array.from({ length: width }, (_, i) => {
    const x = xMin + i * (xMax - xMin) / width;
    return { x, y: func(x) };
  });

  // Add the function plot to the SVG
  const pathData = data.map((d, i) => {
    const x = xScale(d.x);
    const y = yScale(d.y);
    return (i === 0 ? "M" : "L") + x + "," + y;
  }).join(" ");

  const path = document.createElementNS("http://www.w3.org/2000/svg", "path");
  path.setAttribute("d", pathData);
  path.setAttribute("fill", "none");
  path.setAttribute("stroke", options.plotStyles[index].color);
  path.setAttribute("stroke-width", options.plotStyles[index].linewidth);
  path.setAttribute("stroke-dasharray", options.plotStyles[index].lineStyle ===
'dashed' ? '4, 4' : '');
  svg.appendChild(path);
});

```

```

    return svg;
}

```

Now you can pass additional options to the `plotFunction`:

```

const myFunc1 = x => Math.sin(x);
const myFunc2 = x => Math.cos(x);
const options = {
  drawAxes: true,
  axisLabels: { x: 'x', y: 'y' },
  plotStyles: [
    { color: 'steelblue', lineWidth: 1.5, lineStyle: 'solid' },
    { color: 'red', lineWidth: 1.5, lineStyle: 'dashed' },
  ],
};
const svgNode = plotFunction([myFunc1, myFunc2], -2 * Math.PI, 2 * Math.PI, -1, 1,
800, 400, options);
document.body.appendChild(svgNode);

```

This will plot both the sine and cosine functions between and with axes, axis labels, and customized appearance for each function plot. The sine function will be in steel blue with a solid line, while the cosine function will be in red with a dashed line.

**Very impressive, thank you. One more request: Can we add tickmarks and tick labels? Oh, and optionally change the background color and axis (also label/tickmark) color?**

Sure! I've updated the `plotFunction` to include options for tickmarks, tick labels, and customizing the background and axis colors. Here's the updated version:

```

function plotFunction(functions, xMin, xMax, yMin, yMax, width, height, options = {})
{
  if (!Array.isArray(functions)) {
    functions = [functions];
  }

  // Default options
  const defaultOptions = {
    drawAxes: true,
    axisLabels: { x: 'x', y: 'y' },
    plotStyles: functions.map(() => ({ color: 'steelblue', lineWidth: 1.5, lineStyle:
'solid' })),
    tickCount: { x: 5, y: 5 },
    backgroundColor: 'white',
    axisColor: 'black',
  };

  // Merge user options with default options
  options = { ...defaultOptions, ...options };

  // Create an SVG element
  const svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");
  svg.setAttribute("width", width);
  svg.setAttribute("height", height);

  // Set background color
  svg.style.backgroundColor = options.backgroundColor;

```

```

// Create scales for the x and y axes
const xScale = x => (x - xMin) / (xMax - xMin) * width;
const yScale = y => height - (y - yMin) / (yMax - yMin) * height;

if (options.drawAxes) {
  // Draw x-axis
  const xAxis = document.createElementNS("http://www.w3.org/2000/svg", "line");
  xAxis.setAttribute("x1", xScale(xMin));
  xAxis.setAttribute("y1", yScale(0));
  xAxis.setAttribute("x2", xScale(xMax));
  xAxis.setAttribute("y2", yScale(0));
  xAxis.setAttribute("stroke", options.axisColor);
  xAxis.setAttribute("stroke-width", 1);
  svg.appendChild(xAxis);

  // Draw y-axis
  const yAxis = document.createElementNS("http://www.w3.org/2000/svg", "line");
  yAxis.setAttribute("x1", xScale(0));
  yAxis.setAttribute("y1", yScale(yMin));
  yAxis.setAttribute("x2", xScale(0));
  yAxis.setAttribute("y2", yScale(yMax));
  yAxis.setAttribute("stroke", options.axisColor);
  yAxis.setAttribute("stroke-width", 1);
  svg.appendChild(yAxis);

  // Add axis labels
  const xLabel = document.createElementNS("http://www.w3.org/2000/svg", "text");
  xLabel.textContent = options.axisLabels.x;
  xLabel.setAttribute("x", xScale(xMax) - 10);
  xLabel.setAttribute("y", yScale(0) - 5);
  xLabel.style.fill = options.axisColor;
  svg.appendChild(xLabel);

  const yLabel = document.createElementNS("http://www.w3.org/2000/svg", "text");
  yLabel.textContent = options.axisLabels.y;
  yLabel.setAttribute("x", xScale(0) + 5);
  yLabel.setAttribute("y", yScale(yMax) + 10);
  yLabel.style.fill = options.axisColor;
  svg.appendChild(yLabel);

  // Add tickmarks and tick labels
  for (let i = 1; i < options.tickCount.x; i++) {
    const xPos = xMin + i * (xMax - xMin) / options.tickCount.x;
    const tick = document.createElementNS("http://www.w3.org/2000/svg", "line");
    tick.setAttribute("x1", xScale(xPos));
    tick.setAttribute("y1", yScale(0) - 5);
    tick.setAttribute("x2", xScale(xPos));
    tick.setAttribute("y2", yScale(0) + 5);
    tick.setAttribute("stroke", options.axisColor);
    tick.setAttribute("stroke-width", 1);
    svg.appendChild(tick);

    const tickLabel = document.createElementNS("http://www.w3.org/2000/svg",
"text");
    tickLabel.textContent = xPos.toFixed(2);
    tickLabel.setAttribute("x", xScale(xPos) - 10);
    tickLabel.setAttribute("y", yScale(0) + 20);
    tickLabel.style.fill = options.axisColor;
    svg.appendChild(tickLabel);
  }

  for (let i = 1; i < options.tickCount.y; i++) {
    const yPos = yMin + i * (yMax - yMin) / options.tickCount.y;
    const tick = document.createElementNS("http://www.w3.org/2000/svg", "line");
    tick.setAttribute("x1", xScale(0) - 5);

```

```

    tick.setAttribute("y1", yScale(yPos));
    tick.setAttribute("x2", xScale(0) + 5);
    tick.setAttribute("y2", yScale(yPos));
    tick.setAttribute("stroke", options.axisColor);
    tick.setAttribute("stroke-width", 1);
    svg.appendChild(tick);

    const tickLabel = document.createElementNS("http://www.w3.org/2000/svg",
"text");
    tickLabel.textContent = yPos.toFixed(2);
    tickLabel.setAttribute("x", xScale(0) - 30);
    tickLabel.setAttribute("y", yScale(yPos) + 5);
    tickLabel.style.fill = options.axisColor;
    svg.appendChild(tickLabel);
  }
}

functions.forEach((func, index) => {
  // Generate data points for the function
  const data = Array.from({ length: width }, (_, i) => {
    const x = xMin + i * (xMax - xMin) / width;
    return { x, y: func(x) };
  });

  // Add the function plot to the SVG
  const pathData = data.map((d, i) => {
    const x = xScale(d.x);
    const y = yScale(d.y);
    return (i === 0 ? "M" : "L") + x + "," + y;
  }).join(" ");

  const path = document.createElementNS("http://www.w3.org/2000/svg", "path");
  path.setAttribute("d", pathData);
  path.setAttribute("fill", "none");
  path.setAttribute("stroke", options.plotStyles[index].color);
  path.setAttribute("stroke-width", options.plotStyles[index].lineWidth);
  path.setAttribute("stroke-dasharray", options.plotStyles[index].lineStyle ===
'dashed' ? '4, 4' : '');
  svg.appendChild(path);
});

return svg;
}

```

Now you can pass additional options to the `plotFunction` to customize the appearance of the axes, tickmarks, and tick labels:

```

const myFunc1 = x => Math.sin(x);
const myFunc2 = x => Math.cos(x);
const options = {
  drawAxes: true,
  axisLabels: { x: 'x', y: 'y' },
  plotStyles: [
    { color: 'steelblue', lineWidth: 1.5, lineStyle: 'solid' },
    { color: 'red', lineWidth: 1.5, lineStyle: 'dashed' },
  ],
  tickCount: { x: 5, y: 5 },
  backgroundColor: 'lightgray',
  axisColor: 'black',
};
const svgNode = plotFunction([myFunc1, myFunc2], -2 * Math.PI, 2 * Math.PI, -1, 1,

```



```
800, 400, options);
document.body.appendChild(svgNode);
```

This will plot both the sine and cosine functions between  $-2\pi$  and  $2\pi$  with a light gray background, black axes, tickmarks, and tick labels, and customized appearance for each function plot. The sine function will be in steel blue with a solid line, while the cosine function will be in red with a dashed line.

## Would you happen to recall how I can calculate the Bessel J0 function in JavaScript without relying on external libraries?

Yes, you can calculate the Bessel J0 function in JavaScript without relying on external libraries using a numerical method like the series expansion. Here's an implementation of the Bessel J0 function using the series expansion:

```
function besselJ0(x) {
  const maxIterations = 100;
  const epsilon = 1e-15;
  let sum = 0;
  let term;
  let m = 0;

  do {
    term = Math.pow(-1, m) * Math.pow(x / 2, 2 * m) / (factorial(m) * factorial(m));
    sum += term;
    m++;
  } while (Math.abs(term) > epsilon && m < maxIterations);

  return sum;
}

function factorial(n) {
  if (n === 0) return 1;
  let result = 1;
  for (let i = 1; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

You can use the `besselJ0` function to calculate the Bessel J0 function for a given value of  $x$ :

```
const x = 2;
const result = besselJ0(x);
console.log(`Bessel J0(${x}) = ${result}`);
```

Please note that this implementation uses a series expansion, which may not be the most efficient or accurate method for all values of  $x$ . However, it should work well for small to moderate values of  $x$ .

Love it, works, but I note that it becomes unstable when the absolute value of the argument is greater than ~30. Would it not make sense to switch over to one of the sensible large value approximations before it happens?

You're right, the series expansion method can become unstable for large values of the argument. In such cases, it's better to use an asymptotic approximation for the Bessel  $J_0$  function. One common approximation is based on the Bessel function's asymptotic behavior for large arguments, which can be expressed as:

$$J_0(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \frac{\pi}{4}\right)$$

Here's an updated version of the `besselJ0` function that switches to the asymptotic approximation for large values of  $x$ :

```
function besselJ0(x) {
  const maxIterations = 100;
  const epsilon = 1e-15;
  const largeXThreshold = 30;

  if (Math.abs(x) > largeXThreshold) {
    // Asymptotic approximation for large x
    return Math.sqrt(2 / (Math.PI * x)) * Math.cos(x - Math.PI / 4);
  } else {
    // Series expansion for small to moderate x
    let sum = 0;
    let term;
    let m = 0;

    do {
      term = Math.pow(-1, m) * Math.pow(x / 2, 2 * m) / (factorial(m) * factorial(m));
      sum += term;
      m++;
    } while (Math.abs(term) > epsilon && m < maxIterations);

    return sum;
  }
}

function factorial(n) {
  if (n === 0) return 1;
  let result = 1;
  for (let i = 1; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

Now the `besselJ0` function should provide more stable results for large values of  $x$ .