Simulador HTML e gerador de códigos por IA como plataforma prática e compartilhável para gráficos interativos ao ensino-aprendizagem e à pesquisa científica

José Maurício Schneedorf Ferreira da Silva

10.47247/GS/6063.109.0.8

Dados Internacionais de Catalogação na Publicação (CIP)

x Robótica, Inovação e Educação: conexões para o futuro. Guilherme Sumariva (organizador) – Santo André: V&V Editora, 2026.
250 p. : 14 x 21 cm

Inclui bibliografia
ISBN 978-65-6063-109-0
DOI 10.47247/GS/6063.109.0

1. Um. 2. Dois. I. Três. II. Título.

CDD x

Elaborado por Maurício Amormino Júnior – CRB6/2422

V&V Editora

Santo André, São Paulo – Brasil
Tel./Whatsapp: (11) 94019-0635 E-mail: contato@vveditora.com
vveditora.com

Pensamento computacional e ensino básico

O *pensamento computacional (PC)* é definido como uma habilidade cognitiva direcionada à solução de um problema dividindo-o em partes, identificando padrões e soluções possíveis, e selecionando a melhor dessas em função de critérios pré-definidos (Melo *et al.,* 2023). Não está necessariamente relacionado ao uso de computador (*PC desplugado*), exibindo-se em áreas outrora díspares, como artes, ciência, tecnologia, engenharia, e matemática, e hoje harmoniosamente rearranjadas sobre a sigla anglo-saxônica *STEAM*. Não obstante, a inserção de uma linguagem de programação à solução de uma situação-problema adere-se com mais facilidade, tanto pela ideia central que lhe é revestida, como pela iniciativa do poder público. Desse último, elenca-se a prsença do *PC* no ensino básico desde 2022 junto ao complemento da Base Nacional Comum Curricular (BNCC) para Ciências da Computação, bem como das Diretrizes Curriculares Nacionais para a Formação Inicial de Professores para a Educação Básica, e do *eixo de inclusão digital* constante da Política Nacional de Educação Digital - PNED). Junto às ideias principais para inserção do *PC* e de *fenômenos digitais* nesse nível de ensino, agregam-se diversas ferramentas homônimas para sua aplicação, tais como Geogebra, MIT Scratch, e PhET Colorado, e MIT App Inventor.

Pensamento computacional e simulações

Em contrapartida, o *PC plugado*, ainda que natural da cadeira de Ciências da Computação, apresenta-se menos preponderante em outras áreas, e com menor apelo estatal à avaliação e normatização junto à instituições de ensino superior (Melo *et al.,* 2023). Entre os motivos prováveis dessa carência, enumeram-se a falta de clareza sobre seu uso em ambiente educacional, de mecanismos de avaliação sobre o tema (Lyon e Magana, 2020), falta de capacitação técnica de instrutor e aluno, desmotivação inerente ao aprendizado de uma linguagem de programação, e a desconexão para sua aplicação no mundo real. Seguindo nessa linha, Terroso e Pinto (2022) acrescentam a natureza própria de uma linguagem de programação (conceitos abstratos, sintaxe complexa e *non-human readable*), e a conotação negativa que encerra a programação para não programadores.

Para contornar esse distanciamento do *PC* que envolve uma linguagem de programação ao nível superior, os autores mencionam algumas ações mediadas por computador, como o *design* e desenvolvimento de jogos, elaboração de ferramentas físicas programáveis, aprendizagem baseada em projetos (PBL), e gamificação. Mas há várias outras, incluindo o uso de placas microcontroladoras do mundo *geek* (*Arduino, ESP32*; linguagens C/C++ e Python) para interação com o mundo real por sensores, bem como o emprego de *simulações por computador.*

A *aprendizagem baseada em simulações* com computador (do inglês, s*imulation-based learning, SBL*) remonta há quase 6 décadas, quando McKenney e Dill (1966) reportaram o uso de jogos educacionais em gerenciamento de negócios para um curso de MBA da Universidade de Harvard. A *SBL* pode ser compreendida como o *"uso de programas que podem representar visualmente fenômenos reais ou hipotéticos, e permitir que os utilizadores interajam com os programas"* (Krisnadi *et al.*, 2022). Numa revisão não tão recente, Hallinger e Wang (2020) reportaram o estado da arte da *SBL* a partir de 2.812 artigos indexados pela base *Scopus* entre 1965-2018. Os autores concluiram que essa metodologia ativa distingue-se evolucionariamente de outras, como aprendizagem baseada em casos (*CBL*), em problemas (*PBL*), e em projetos (*PBL*), por internalizar novas tecnologias digitais em rápido crescimento. Ainda que o trabalho seja anterior à emergência de saúde global de 2019, não há como discordar dos autores, frente à *big data*, realidade aumentada e virtual, computação quântica, laboratórios virtuais, e inteligência artificial generativa (IA) dos anos recentes.

Três características importantes parecem agregar valor educacional à motivação de aprendizes no uso de *SBL* (Krisnadi *et al.*, 2022):

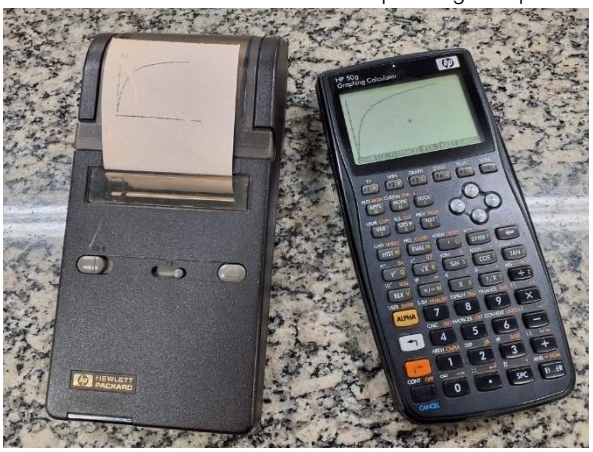
1. Interatividade: ação de usuário e "feedback" do sistema;
2. Exploração paramétrica: visualização do fenômeno sob variação em parâmetro(s);
3. Reporte: discussão do fenômeno à luz da simulação realizada.

Ainda que não substituam a experiência do mundo real, as simulações possuem algumas qualidades interessantes na

comparação com o ensino tradicional. Não requerem infraestutura laboratorial, o que implica em dispensa de área, equipe, e de equipamentos, esses por vezes sofisticados e onerosos. Além disso, são realizadas em tempo ínfimo, quando não instantaneamente, e com feedback de igual monta. Permitem o ajuste de parâmetros para visualização imediata de um resultado distinto da proposta original (exploração ou manipulação paramétrica), e facultam a realização de atividades não replicáveis em condições normais, fenômenos não observáveis em laboratório (Farmonov, 2023), ou mesmo inexistentes. Exemplos não faltam, como no mundo submicroscópico, nanotecnologia, materiais radioativos, estudos astronômicos, estudos quânticos, validações termocinéticas, ou big crunch (colapso astrofísico do universo). Além disso, tendem a multiplicar-se junto à novas tecnologias digitais dos últimos anos, como a realidade virtual, IA, e computadores quânticos, associados ou não. Retomando o trabalho sessentista de McKenney e Dill, experiências singulares a se obter "com o aperfeiçoamento das técnicas e linguagens de simulação e com as expansões tanto na capacidade dos computadores quanto na flexibilidade dos arranjos pelos quais os alunos podem interagir com a máquina".

Em relação a essas experiências, a simulação mediada por algoritmos já fora utilizada em situações críticas variadas, e mesmo anteriores à popularização maciça de computadores pessoais, como por calculadoras programáveis HP-41C da Hewlett-Packard em nove missões do ônibus espacial estadunidense na década de 1980. Em relação a essas "tecnologias do passado" aplicadas na simulação de fenômenos naturais, permite-se o autor uma saudosa reverência junto a Figura 1.

Figura 1: Calculadora gráfica Hewlett-Packard HP50G e impressora com sensor infravermelho 82240B. A tela gráfica e sua impressão apresentam uma simulação para dados de isoterma de adsorção de Langmuir e ajuste não linear de Gauss-Newton a um complexo ligante-proteína.



Fonte: Autor.

Entre os tipos de *SBL* com potencial para melhoria do ensino como um todo, vale mencionar os *programas para gráficos*. Inerentes à pesquisa científica, esses programas despontam no ensino superior de forma complementar às atividades formais em sala de aula e outros ambientes educacionais (Kučerka *et al.,* 2024). Programas gráficos tendem a desenvolver no aluno habilidades (meta)cognitivas que incluem a imaginação espacial, pensamento analítico e de solução de problemas, além de prepará-lo ao mercado de trabalho, como em engenharia, *design*, e pesquisa científica. Kučera *et al.* (2024) classificam os programas gráficos em 3 tipos:

> # Taxonomia de programas para gráficos (Kucerka et al., 2024)
>
> 1. Editores gráficos (ex: Photshop, Gimp, Inkscape, Canva);
>
> 2. Softwares CAD ("Computer-Aided Design") - desenho técnico, simulação, análise, modelos 2D e 3D (ex: AutoCAD, SolidWorks, Inventor);
>
> 3. Softwares de Simulação (ex: Simulink, Tinkercad, Ansys);

4. Softwares de matemática (ex: R, Matlab, Octave, Maple, Wolfram Mathematica, Python, Jupyter Notebooks, Maxima).

Entre esses tipos, os programas de matemática figuram de modo ascertivo ao ensino superior, ao permitir a elaboração de gráficos 2D e 3D de dados para funções matemáticas e análise de dados. Além disso, tais programas podem envolver álgebra computacional (CAS - Computer Algebra System), permitindo computação numérica ou simbólica para cálculo integral, diferencial, e matricial. De modo geral, também consolidam uma linguagem própria para programação, cálculos, gráficos, visualização de dados, e simulação. Alguns são pagos (Matlab, Mathematica, Maple), outros de uso aberto (R, Python, Octave, Maxima). Alguns permitem interatividade e feedback (R, Python, Mathematica), outros não (Maxima, Octave). Uns permitem gráficos e paineis interativos (dashboards; R Python), outros não (demais). Alguns permitem acesso por computação em nuvem (R, Mathematica, Python), e todos são alternativamente instaláveis em mídia física.

Por outro lado, esses programas elencados (e tantos mais) exigem memória física e de acesso aleatório (RAM) de bom desempenho, possuem janelas múltiplas e abas para funcionalidades excessivas ao usuário comum, gráficos por vezes estáticos e sem interatividade, limitação de uso em ambientes virtuais de aprendizagem, uma linguagem de programação específica por linhas de comandos, compartilhamento limitado de arquivos para quem não possui o software, e saída gráfica em tela separada, entre outros. Como um todo, essas características fornecem condições àsperas para uma curva de aprendizagem dos programas, e que seja voltada à aplicações ao ensino e à pesquisa, forçando o aprendiz a uma sobrecarga informacional (carga cognitiva; Sigolo e Casarin, 2024). Em suma, soluções que exigem aculturação razoável para uma aplicação desejada, inibindo seu uso imediato e compartilhamento.

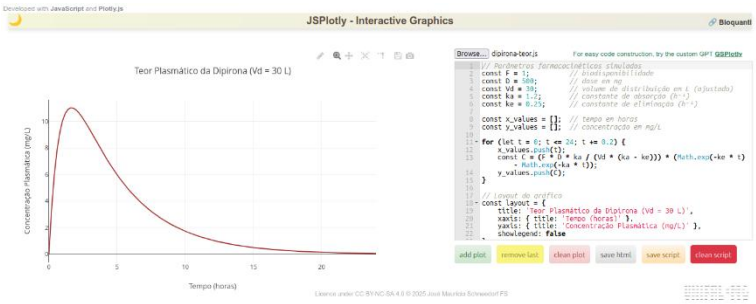
Nessa direção, este trabalho apresenta uma solução tecnológica para a produção de gráficos interativos ao ensino e à pesquisa, e cujos códigos são facultativamente produzidos por engenharia de prompts junto a assistente de IA generativa.

JSPlotly & GSPlotly: simulador gráfico e gerador de códigos por IA

JSPlotly é um aplicativo de livre distribuição e edição (licença CC BY-NC 4.0) para visualização de dados por gráficos interativos em browser. Foi desenvolvido em JavaScript para um contêiner de HTML dividido entre ecrã gráfico e editor de códigos, e personalizado com folhas de estilo em cascata (CSS). Possui apenas uma janela (Figura 2) e utiliza JavaScript, uma linguagem de programação moderna para interatividade na web, bem como bibliotecas correlatas, como Plotly.js para a confecção dos gráficos interativos. Mediante essa limitação para seu uso pleno ao ensino e pesquisa, buscou-se desenvolver um GPT personalizado (OpenAI, 2021), GSPlotly, para conversão de linguagem natural em códigos prontos para uso no JSPlotly. Dessa forma, o gerador permite ao aprendiz sobrelevar o foco diretamente ao conteúdo ou tema desejado, trocando diálogos com o assistente de IA por engenharia de prompts, e sem qualquer necessidade para conhecimento prévio em programação. Assim, e distinto das tecnologicas elecadas acima, a plataforma intenciona convergir o interesse principal para seu uso no conteúdo (tema), e não na forma (código).

Não obstante, e ainda que não necessite de conhecimento em linguagem JavaScript, o código gerado pelo GSPlotly e executado por cópia/cola no JSPlotly abre um caminho virtualmente infinito para sua aprendizagem, frente à sua modificação para novas condições que envolvam o fenômeno em estudo (exploração paramétrica ou comandos sintáticos). Dessa forma, a plataforma oportuniza desde a simples reprodução do gráfico, como a visualização de novo gráfico por modificação pontual de seu código e feedback imediato, e mesmo a reconstrução do código para outro interesse do usuário, características que coadunam para um ensino reprodutível (Dogucu, 2022). Ao mesmo tempo, a solução naturalmente acaba por instigar no aprendiz a leitura e interpretação pontuais ou progressivas da própria linguagem de programação envolvida.

Figura 2: Janela única do *JSPlotly*, exibindo o editor de códigos e seu gráfico interativo resultante. O gráfico ilustra uma curva farmacocinética de concentração plasmática em função do tempo para uma dose oral de dipirona, e produzida a partir de modelo monocompartimental com absorção de 1a. ordem para o fármaco.



Fonte: Autor.

Preferiu-se neste texto apresentar algumas características potenciais da plataforma para ensino e pesquisa ao nível superior. Contudo, para apreciar detalhes de seu uso, vídeo de apresentação, download do aplicativo, bem como exemplos ao ensino básico, como para o superior em Bioquímica, convida-se o leitor a visitar o website Bioquanti. O site constitui uma iniciativa voltada à valorização do ensino reprodutível (Dogucu, 2024) no país, uma metodologia ativa que utiliza o letramento em programação para a produção de documentos dinâmicos e objetos didáticos para aprendizagem de conteúdos curriculares (ou… códigos para conteúdos). Uma descrição detalhada para uso da plataforma em ensino e aprendizagem pode ser observada neste texto (Schneedorf, 2025). Complementarmente, um manuscrito para aplicação da plataforma ao ensino básico encontra-se em submissão junto a esta mesma casa editorial.

Percebe-se pela Figura 2 que o JSPlotly apresenta uma janela única e sem abas para suas funcionalidades. Além disso, seus comandos principais são executados com apenas 6 botões no editor de códigos: adição de gráfico (Add plot), remoção do último gráfico (Remove plot), limpeza do ecrã (Clean plot), armazenamento do gráfico como um arquivo HTML interativo (Save HTML), armazenamento do código (Save script), e limpeza do editor (Clean script). Além disso, os ícones da barra superior permitem ações

interativas. Da direita para a esquerda, de troca da cor da curva gerada, arquivamento do gráfico como PNG (rasterizado, para web) ou SVG (vectorizado, para publicação técnico-científica), demarcação de coordenadas gráficas (Toggle Spike Lines), auto-escalonamento, translação, ampliação/redução, bem como personalização do gráfico em editor do desenvolvedor da biblioteca, Plotly Chart Studio. A área gráfica também possui algumas ações interativas de mouse, como hover (passagem e informação de pontos), zoom (botão de rolagem do mouse), deslocamento de eixos, e rotulagem personalizada para título de gráfico e eixos. Adicionalmente, pela natureza da biblioteca *Plotly.js*, é possível adicionar *pop ups* informativos (*tooltip*) deslizadores (*slider*), menu suspenso, caixas de checagem (*checkbox*), e botões.

Pelo treinamento junto ao assistente de IA, o *GSPlotly* elabora um texto descritivo da situação-problema, a(s) equação(ções) envolvida(s), o código-fonte para cópia/cola no *JSPlotly*, e sugestões de melhoria. Dessa forma, a plataforma pode ser utilizada de modos distintos. Basicamente, pede-se um gráfico no *GSPlotly* (ex: *"me forneça um gráfico para formação de peróxido de hidrogênio no tempo, e em função de teores variados de água e oxigênio ajustáveis no código"*) e cola-se o código *JavaScript* para *"rodar"* no *JSPlotly* (*Add plot*). Por logística reversa, também é possível descobrir detalhes sobre determinada situação-problema e equação envolvida a partir de um código compartilhado. Para isso, basta arguir o assistente de IA após transcrever-lhe o código.

Adicionalmente, pela natureza própria da plataforma *JSPlotly/GSPlotly*, a solução tecnológica naturalmente promove uma *inversão logística do pensamento computacional*, o qual define uma sequência para abstração, análise (decomposição e reconhecimento), e criação de algoritmo. Com auxílio do *GSPlotly*, primeiro é gerado um algoritmo que faculta sua decodificação por *lógica de programação* (Oliveira *et al.,* 2025) (análise), seguindo-se a interpretação pelo simulador *JSPlotly* para simulações com exploração paramétrica e sobreposição de curvas, e ao final oportunizando ao aprendiz uma apropriação do tema envolvido.

4 Exemplos de uso ao ensino superior

Para ilustrar o potencial de uso da solução tecnológica ao ensino e pesquisa, seguem dois exemplos pertinentes à simulação e análise de dados simultâneas a situações-problema. Outros exemplos estão presentes no site Bioquanti.

Determinação de energia de ativação por ajuste linear de dados

O exemplo pretende ilustrar o potencial da tecnologia para análise de dados definidos pelo usuário, pela determinação da energia de ativação que acompanha uma reação, e cuja constante de velocidade varia com a temperatura. Para isso, é conduzida uma *regressão linear por mínimos quadrados*, e cuja relação de Arrhenius é apresentada na Equação 1 (Jensen, 1985), seguindo-se o código gerado no *GSPlotly*, e o gráfico resultante no *JSPlotlty* (Figura 3):

$$ln(k_{\exp}) = -\frac{E_a}{R} \cdot \left(\frac{1}{T}\right) + ln(A) + \varepsilon \qquad \text{(Equação 1)}$$

Onde:

k_{exp} = constante de velocidade medida no experimento;

A = fator pré-exponencial da equação (frequência de colisão efetiva);

T = temperatura absoluta (K);

R = constante geral dos gases, 8,314 J*(mol*K)$^{-1}$;

Ea = energia de ativação para a barreira energética da reação; (J/mol)

ϵ = ruído aleatório das medidas experimentais

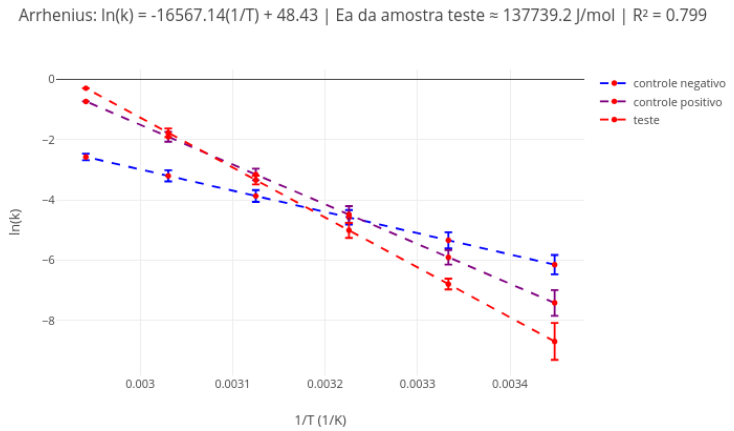
```
// Constante dos gases   "Declaração de constantes e variáveis"
const R = 8.314; // J/mol·K

// Dados experimentais
const T = [290, 300, 310, 320, 330, 340]; // Temperaturas (K)
const k = [0.000024, 0.0171, 0.0077, 0.0304, 0.0590, 0.999];
 // Constantes de velocidade (1/s)
```

```
// Transformações para gráfico de Arrhenius
const x_values = T.map(temp => 1 / temp);          // 1/T
const y_values = k.map(valor => Math.log(valor));  // ln(k)

// Geração de erros aleatórios simulados (±5%)
const y_error = y_values.map(y => 0.05 * Math.abs(y) * (0.8 + 0.4
* Math.random())); // Varia entre 4% e 6%

// Flags  "Sinalizadores para apresentação alternativa de pontos e
de linha de regressão"
const mostrarPontos = true;
const mostrarReta = true;

// Ajuste linear  "Regressão de dados a modelo linear (sem uso de
biblioteca externa)"
const n = x_values.length;
const sumX = x_values.reduce((a, b) => a + b, 0);
const sumY = y_values.reduce((a, b) => a + b, 0);
const sumXY = x_values.reduce((acc, xi, i) => acc + xi *
y_values[i], 0);
const sumXX = x_values.reduce((acc, xi) => acc + xi * xi, 0);

const slope = (n * sumXY - sumX * sumY) / (n * sumXX - sumX *
sumX);
const intercept = (sumY - slope * sumX) / n;
const y_fit = x_values.map(x => slope * x + intercept);

// R²  "Cálculo do coeficiente de determinação"
const mediaY = sumY / n;
const ss_tot = y_values.reduce((acc, yi) => acc + Math.pow(yi -
mediaY, 2), 0);
const ss_res = y_values.reduce((acc, yi, i) => acc + Math.pow(yi -
y_fit[i], 2), 0);
const R2 = 1 - ss_res / ss_tot;

// Energia de ativação
const Ea = -slope * R; // J/mol

// Layout
const layout = {
```

```
    title: `Arrhenius: ln(k) = ${slope.toFixed(2)}(1/T) +
${intercept.toFixed(2)} | Ea ≈ ${Ea.toFixed(1)} J/mol | R² =
${R2.toFixed(3)}`,
    xaxis: { title: "1/T (1/K)" },
    yaxis: { title: "ln(k)" }
};

// Trace com barras de erro
let trace;
if (mostrarPontos && !mostrarReta) {
    trace = {
        x: x_values,
        y: y_values,
        mode: "markers",
        type: "scatter",
        name: "ln(k)",
        marker: { size: 6 },
        error_y: {
            type: "data",
            array: y_error,
            visible: true
        }
    };
}
if (mostrarReta) {
    trace = {
        x: x_values,
        y: y_fit,
        mode: "lines+markers",
        type: "scatter",
        name: "Ajuste Linear",
        line: { width: 2, dash: "dash", color: "blue" },
        marker: { size: 6, color: "red" },
        error_y: {
            type: "data",
            array: y_error,
            visible: true
        }
    };
}
```

```
// Retorno final
return { x_values, y_values: mostrarReta ? y_fit : y_values, layout,
trace };
```

Figura 3: Determinação de energia de ativação de reação por ajuste linear de dados com ruído experimental.



Fonte: Autor.

A Figura 3 foi produzida por *manipulação paramétrica*, intercalando-se os vetores de dados da variável dependente (*y_values*), seguido da execução do gráfico para cada um (*Add plot*). Complementarmente, para a visualização do ajuste linear, foi conduzida a visualização também alternativa de pontos e de linha de ajuste correlata, por meio dos sinalizadores contidos no código (*Flags*), e utilizando-se a *opção booleana true/false* (ex: *mostrarPontos = true; mostrarReta = false*).

Observe pela Figura 3 que o exemplo possibilita, em paralelo à aprendizagem implícita em programação e à manipulação paramétrica para dados e parâmetros, uma apropriação mais significativa e de espectro transdisciplinar para conceitos técnico-científicos abordados junto ao fenômeno, como físico-química, planejamento experimental, estatística, e programação.

Interação ligante-proteína e ajuste não linear dos dados

Para este segundo exemplo, pretendeu-se ilustrar uma curva de saturação entre um ligante (ex: fármaco) e uma proteína, tal como observado na Figura 1, e comumente definida pela relação hiperbólica de Langmuir (Equação 2) que segue (Dixon, 1974):

$$y = \frac{n \cdot L}{K_d + L} \qquad \text{(Equação 2)}$$

Onde:

y = fração de saturação do ligante na macromolécula;

n = saturação máxima de ligange na macromolécula;

Kd = constante de equilíbrio de dissociação ligante-proteína

L = teor de ligante livre da solução

Nesse caso, a simulação da interação ligante-proteína pode ser idealizada junto ao *GSPlotly* para produzir o código que segue, e exibida no *JSPlotly* tal como representado na Figura 4.

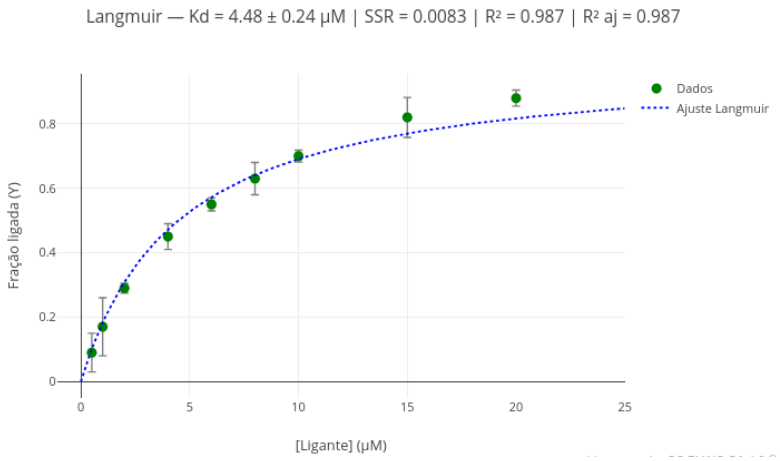
```
// Dados experimentais-  "Atribuição de variáveis por criação de vetores (arrays)"
const x_dados = [0.5, 1, 2, 4, 6, 8, 10, 15, 20];
const y_dados = [0.09, 0.17, 0.29, 0.45, 0.55, 0.63, 0.70, 0.82, 0.88];

// Erros experimentais manuais
const y_error = [0.06, 0.09, 0.015, 0.04, 0.02, 0.05, 0.018, 0.062, 0.025];

// Flags de exibição
const mostrarPontos = false;
const mostrarCurva = true;

// Função de Langmuir
function langmuir(Kd, x) {        "Definição da expressão matemática"
    return x.map(xi => xi / (Kd + xi));
}

// Ajuste via mínimos quadrados (busca direta)
```

```
let melhor_Kd = 0;
let menorErro = Infinity;
let y_ajuste = [];

for (let Kd = 0.1; Kd <= 20; Kd += 0.01) {     "Estrutura de
repetição"
    const y_modelo = langmuir(Kd, x_dados);
    const erro = y_dados.reduce((acc, y, i) => acc + Math.pow(y -
y_modelo[i], 2), 0);
    if (erro < menorErro) {
        menorErro = erro;
        melhor_Kd = Kd;
        y_ajuste = y_modelo;
    }
}

// Estatísticas básicas          "Cálculos estatísticos"
const n = x_dados.length;
const p = 1;
const mediaY = y_dados.reduce((a, b) => a + b, 0) / n;

const SSR = y_dados.reduce((acc, y, i) => acc + Math.pow(y -
y_ajuste[i], 2), 0);
const SST = y_dados.reduce((acc, y) => acc + Math.pow(y -
mediaY, 2), 0);

const R2 = 1 - SSR / SST;
const R2_ajustado = 1 - ((SSR / (n - p)) / (SST / (n - 1)));

// Derivadas parciais de Y em relação a Kd
const derivadas = x_dados.map(L => -L / Math.pow(melhor_Kd +
L, 2));
const soma_derivadas2 = derivadas.reduce((acc, d) => acc + d *
d, 0);

// Erro padrão de Kd
const s2 = SSR / (n - p);
const erro_Kd = Math.sqrt(s2 / soma_derivadas2);

// Curva ajustada
const x_fit = [], y_fit = [];
```

```
for (let x = 0; x <= 25; x += 0.2) {
    x_fit.push(x);
    y_fit.push(x / (melhor_Kd + x));
}

// Layout com tudo   "Etiquetas de texto"
const layout = {
    title: `Langmuir — K_n = ${melhor_Kd.toFixed(2)} ±
${erro_Kd.toFixed(2)} µM | SSR = ${SSR.toFixed(4)} | R² =
${R2.toFixed(3)} | R² aj = ${R2_ajustado.toFixed(3)}`,
    xaxis: { title: "[Ligante] (µM)" },
    yaxis: { title: "Fração ligada (Y)" }
};

// Trace condicional
let trace;
if (mostrarPontos) {
    trace = {
        x: x_dados,
        y: y_dados,
        mode: "markers",
        type: "scatter",
        name: "Dados",
        marker: { size: 10, color: "green" },
        error_y: {
            type: "data",
            array: y_error,
            visible: true,
            color: "gray",
            thickness: 1.5
        }
    };
}

if (mostrarCurva) {
    trace = {
        x: x_fit,
        y: y_fit,
        mode: "lines",
        type: "scatter",
```

```
        name: "Ajuste Langmuir",
        line: { color: "blue", width: 2, dash: "dot" }
    };
}

// Retorno final     "Retorno de valores"
return {
    x_values: trace.x,
    y_values: trace.y,
    layout,
    trace
};
```

Figura 4: Determinação de constante de equilíbrio de dissociação para formação de complexo ligante-proteína por ajuste não linear aos dados.



Fonte: Autor.

Sob o ponto de vista *técnico-científico*, observe que a Figura 4 apresenta dados, barras de erro experimentais, curva de ajuste não linear (*fitting*), bem como os resultados estatísticos do *fitting* por mínimos quadrados (constante de equilíbrio de dissociação *Kd* e erro-padrão da estimativa, soma dos quadrados dos resíduos *SSR*, coeficiente de determinação R^2, e coeficiente de determinação corrigido para os graus de liberdade da regressão R^2-*adj*).

Sob o ponto de vista do *letramento em programação*, note que o código gerado pelo assistente de IA permite ao aprendiz todo o raciocínio para sua tradução a uma *lógica de programação* que envolve entradas, saídas, cálculos e laço iterativo (calma, não foi de primeira, não !), e permitindo seu acompanhamento e edição em paralelo ao *feedback* gráfico.

Sob o ponto de vista *socioconstrutivista*, perceba que, distinto da imagem da Figura 1, o objeto didático agora produzido possui vantagens que elencam um conjunto significativo de ações interativas ao aprendiz, permitindo uma exploração paramétrica capaz de conduzi-lo a alfabetização gráfica e a uma resignificação do aprendizado para o tema abordado. Além disso, também facultando o armazenamento do objeto como arquivo HTML interativo, para seu uso posterior, como para compartilhamento entre aprendizes e instrutores, e em qualquer ambiente digital, incluindo AVAs e redes sociais.

Finalmente, salienta-se que o código-fonte do *JSPlotly* inclui, além de *Plotly.js* para construção do gráficos, um conjunto de *bibliotecas para computação numérica e simbólica*, a saber: *Math.min.js* (álgebra simbólica, cálculo numérico e vetorial, manipulação de unidades físicas), *Numeric.min.js* (álgebra linear, métodos numéricos, equações diferenciais ordinárias), *Simple-statistics_min.js* (estatística descritiva, regressão linear, média móvel, correlação, histogramas, quartis, desvio-padrão), *ml-regression.min.js* (regressão linear e polinomial, modelos de aprendizado supervisionado), e *p5.min.js* (visualização criativa, animações matemáticas, e manipulação de vetores e curvas). Ainda assim, e mesmo que esse conjunto permita um considerável espectro de recursos matemáticos, estatísticos, e visuais, basta utilizar um editor de notas simples, para incluir no código-fonte da aplicação outra biblioteca *JavaScript* desejada.

Conclusão

Este texto apresenta *JSPlotly* como um simulador para funções matemáticas e análise de dados, bem como *GSPlotly* como um GPT personalizado para a criação de códigos *JavaScript* utilizados pelo simulador. Como um todo, a plataforma oferece um espectro ímpar de oportunidades para o ensino e a pesquisa científica em nível

básico e superior, esse reforçado no texto, e frente a várias funcionalidades da solução tecnológica. Entre essas, a produção de códigos e de gráficos interativos ao ensino e aprendizagem e pesquisa científica para diversas áreas do conhecimento e graus de aprofundamento, bem como a facilitação para geração de códigos por assistente de IA, e que são exibidos em paralelo aos gráficos resultantes. Dessa forma, a tecnologia pretende conciliar o pensamento computacional e o letramento em programação convergidos diretamente para conteúdos curriculares e pesquisa, e com irrestrita capacidade para personalização e compartilhamento de sua tríade de códigos, gráficos e plataforma.

A elaboração deste capítulo

A formatação de texto (*RMarkdown*), inserção e referenciação cruzada de figuras, hiperlinks, o gerenciamento bibliográfico, e a conversão do *documento dinâmico* final para compilação a um modelo de arquivo *DOCX* previamente configurado, foram exclusivamente conduzidos com a linguagem de programação *R* (versão 4.3.3, fev/2024) em ambiente de desenvolvimento integrado *RStudio* (versão 2024.09.1 Build 394), e pacote *quarto* (versão 1.4.4) como sistema de publicação científica.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Referências

DIXON, H. B. Curves of ligand binding. The use of hyperbolic functions for expressing titration curves. Biochemical Journal, [*s. l.*], vol. 137, n.º 3, p. 443–447, 1974.

DOGUCU, M. Reproducibility in the Classroom. Annual Review of Statistics and Its Application, [*s. l.*], vol. 12, 2024.

DOGUCU, M.; ÇETINKAYA-RUNDEL, M. Tools and recommendations for reproducible teaching. **Journal of Statistics and Data Science Education**, [*s. l.*], vol. 30, n.º 3, p. 251–260, 2022.

FARMONOV, U. ADVANTAGES OF USING VIRTUAL LABORATORIES IN TEACHING OF PHYSICS. **Mental Enlightenment Scientific-Methodological Journal**, [*s. l.*], vol. 4, n.º 6, p. 70–74, 2023.

FROST, V. S. Teaching the Basic Concepts of Communications Systems Using Interactive Graphics and Calculations. *Em*:, 2021. **2021 ASEE Midwest Section Conference**. [*S. l.: s. n.*], 2021.

HALLINGER, P.; WANG, R. The evolution of simulation-based learning across the disciplines, 1965–2018: A science map of the literature. **Simulation & Gaming**, [*s. l.*], vol. 51, n.º 1, p. 9–32, 2020.

JENSEN, Finn. Activation energies and the Arrhenius equation. **Quality and Reliability Engineering International**, v. 1, n. 1, p. 13-17, 1985.

KRISNADI, D. *et al.* Computer simulation development for simulation-based learning in high-school Physics. [*s. l.*], 2022.

KUČERKA, M.; ŽÁČOK, L.; BERNÁT, M. The Use of Graphic Programs in the Educational Process of Technical Subjects at Secondary Schools and Universities. [*s. l.*], 2024.

LIEBIG, P. *et al.* Interactive, browser-based graphics to visualize complex data in education of biomedical sciences for veterinary students. **Medical science educator**, [*s. l.*], vol. 32, n.º 6, p. 1323–1335, 2022.

LYON, J. A.; J. MAGANA, A. Computational thinking in higher education: A review of the literature. **Computer Applications in Engineering Education**, [*s. l.*], vol. 28, n.º 5, p. 1174–1189, 2020.

MCKENNEY, J. L.; DILL, W. R. Influences on learning in simulation games. **American Behavioral Scientist**, [*s. l.*], vol. 10, n.º 2, p. 28–32, 1966.

MELO, N. M. T. de; CAMPOS, L. S.; ARAUJO, E. C. de. Uma visão sobre Pensamento Computacional no Ensino Superior do Brasil: características e desafios. *Em*: UMA VISÃO SOBRE PENSAMENTO COMPUTACIONAL NO ENSINO SUPERIOR DO BRASIL, 2023. **Simpósio Brasileiro de Informática na Educação (SBIE)**. [*S. l.*]: SBC, 2023. p. 1465–1476.

OLIVEIRA JUNIOR, E. R. de; BORTOLI, L. Â. de; CASTAMAN, A. S. Confraria da Lógica: uma organização educacional para potencializar o pensamento computacional entre estudantes da Educação Profissional e Tecnológica. **Caderno Pedagógico**, [*s. l.*], vol. 22, n.º 1, p. e13548–e13548, 2025.

OPENAI. **ChatGPT documentation**. https://platform.openai.com/docs/guides/embeddings, 2021.X

SIGOLO, B. de O. O.; CASARIN, H. de C. S. Contribuições da teoria da carga cognitiva para compreensão da sobrecarga informacional: uma revisão de literatura. **RDBCI: Revista Digital de Biblioteconomia e Ciência da Informação**, [*s. l.*], vol. 22, p. e024027, 2024.

TERROSO, T.; PINTO, M. Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education. *Em*: PROGRAMMING FOR NON-PROGRAMMERS, 2022. **Third International Computer Programming Education Conference (ICPEC 2022)**. [*S. l.*]: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. p. 13–11.