ChatGPT

```
import React, { useState, useEffect, useRef } from 'react';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend,
ResponsiveContainer, ReferenceLine } from 'recharts';
import { Play, Pause, RotateCcw, Download } from 'lucide-react';

const BHMembraneSimulator = () => {
  const [isRunning, setIsRunning] = useState(false);
  const [mode, setMode] = useState('qpo');
  const [timeData, setTimeData] = useState([]);
  const [psdData, setPsdData] = useState([]);
  const [reservoirData, setReservoirData] = useState([]);
  const [params, setParams] = useState({
    zeta: 0.15,
    kappaC: 1.0,
    lambda: 0.1,
    thetaR: 1.0,
    accRate: 0.02
  });
  const [reservoir, setReservoir] = useState(0);
  const [time, setTime] = useState(0);
  const [multiMode, setMultiMode] = useState(false);
  const [showACF, setShowACF] = useState(false);
  const [analysisResult, setAnalysisResult] = useState(null);
  const [acfData, setAcfData] = useState([]);

  const animationRef = useRef(null);
  const dataRef = useRef([]);
  const showACFRef = useRef(false);
  const multiModeRef = useRef(false);

  // Sync toggles with refs for use in animation loop
  useEffect(() => { showACFRef.current = showACF; }, [showACF]);
  useEffect(() => { multiModeRef.current = multiMode; }, [multiMode]);

  // Constants for polarization simulation
  const EVPA_CENTER = 0;
  const EVPA_AMP = 40;
  const RM_CENTER = 100;
  const RM_AMP = 20;

  // Generate QPO signal (1-field)
  const generateQPOSignal = (t, zeta, kappaC) => {
    const freq1 = 2.0 * kappaC;
    const freq2 = 3.0 * kappaC;
    const damping = Math.exp(-0.01 * t * (1 - zeta));
    const signal =
      Math.sin(2 * Math.PI * freq1 * t) * (0.5 + 0.3 * Math.cos(2 * Math.PI
* 0.1 * t)) +
```

```
      0.4 * Math.sin(2 * Math.PI * freq2 * t) +
      0.2 * (Math.random() - 0.5);
    return signal * damping;
  };

  // Compute power spectrum (PSD) of current time series
  const computePSD = (timeSeries) => {
    const N = timeSeries.length;
    if (N < 16) return [];
    const fluxes = timeSeries.map(pt => parseFloat(pt.flux));
    const mean = fluxes.reduce((a, b) => a + b, 0) / N;
    const centered = fluxes.map(x => x - mean);
    const dt = 0.1;
    const maxFreq = 5.0;
    const df = maxFreq / 50;
    const psd = [];
    for (let f = 0; f <= maxFreq; f += df) {
      let cosSum = 0, sinSum = 0;
      for (let i = 0; i < N; i++) {
        const angle = 2 * Math.PI * f * (i * dt);
        cosSum += centered[i] * Math.cos(angle);
        sinSum += centered[i] * Math.sin(angle);
      }
      const power = (cosSum * cosSum + sinSum * sinSum) / N;
      psd.push({ freq: f.toFixed(2), power: power.toFixed(4) });
    }
    return psd;
  };

  // Compute autocorrelation (ACF) for echo detection
  const computeACF = (timeSeries) => {
    const N = timeSeries.length;
    if (N < 2) return [];
    const fluxes = timeSeries.map(pt => parseFloat(pt.flux));
    const mean = fluxes.reduce((a, b) => a + b, 0) / N;
    const varSum = fluxes.reduce((a, b) => a + Math.pow(b - mean, 2), 0);
    const acf = [];
    const maxLag = Math.min(N - 1, 100);
    for (let lag = 1; lag <= maxLag; lag++) {
      let sum = 0;
      for (let i = 0; i < N - lag; i++) {
        sum += (fluxes[i] - mean) * (fluxes[i + lag] - mean);
      }
      const value = sum / (N - lag) / (varSum / N);
      acf.push({ lag: (lag * 0.1).toFixed(1), acf: value.toFixed(3) });
    }
    return acf;
  };

  // Simulation effect
  useEffect(() => {
```

```
    if (!isRunning) return;

    const dt = 0.1;
    const animate = () => {
      setTime(t => {
        const newTime = t + dt;
        if (mode === 'qpo') {
          // QPO dynamics
          let fluxVal = generateQPOSignal(newTime, params.zeta,
params.kappaC);
          if (multiModeRef.current) {
            const f3 = 2.8 * params.kappaC;
            fluxVal += 0.3 * Math.sin(2 * Math.PI * f3 * newTime);
          }
          const EVPA = EVPA_CENTER + EVPA_AMP * Math.cos(2 * Math.PI * (2 *
params.kappaC) * newTime);
          const RM = RM_CENTER + RM_AMP * Math.sin(2 * Math.PI * (2 *
params.kappaC) * newTime);
          const newPoint = {
            time: newTime.toFixed(2),
            flux: fluxVal.toFixed(3),
            EVPA: EVPA.toFixed(2),
            RM: RM.toFixed(2)
          };
          dataRef.current.push(newPoint);
          if (dataRef.current.length > 300) dataRef.current.shift();
          setTimeData([...dataRef.current]);
          if (Math.floor(newTime * 10) % 20 === 0) {
            setPsdData(computePSD(dataRef.current));
            if (showACFRef.current) {
              setAcfData(computeACF(dataRef.current));
            }
          }
        } else {
          // QPE dynamics (threshold gating)
          setReservoir(Rprev => {
            let newR = Rprev;
            let fluxOut = 0;
            if (Rprev < params.thetaR) {
              // accumulating phase
              newR += params.accRate * dt;
              fluxOut = 0.1;  // quiescent emission
            } else {
              // eruption phase
              newR = Math.max(0, Rprev - 0.3 * dt);
              fluxOut = 10.0 * Math.exp(-5 * (Rprev - params.thetaR));
            }
            const resPoint = { time: newTime.toFixed(2), R:
newR.toFixed(3) };
            const fluxPoint = { time: newTime.toFixed(2), flux:
fluxOut.toFixed(3) };
```

```
            dataRef.current.push(fluxPoint);
            if (dataRef.current.length > 300) dataRef.current.shift();
            setTimeData([...dataRef.current]);
            setReservoirData(prev => {
              const newData = [...prev, resPoint];
              if (newData.length > 300) newData.shift();
              return newData;
            });
            return newR;
          });
        }
        return newTime;
      });
      animationRef.current = requestAnimationFrame(animate);
    };

    animationRef.current = requestAnimationFrame(animate);
    return () => {
      if (animationRef.current) cancelAnimationFrame(animationRef.current);
    };
  }, [isRunning, mode, params]);

  // Reset simulation
  const handleReset = () => {
    setIsRunning(false);
    setTime(0);
    setReservoir(0);
    setTimeData([]);
    setPsdData([]);
    setReservoirData([]);
    setAcfData([]);
    dataRef.current = [];
  };

  // Analyze uploaded CSV data for QPO features
  const handleFileUpload = (e) => {
    const file = e.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (ev) => {
      const text = ev.target.result;
      const lines = text.trim().split(/\r?\n/);
      if (lines.length < 2) {
        alert("Datei enthält keine auswertbaren Daten.");
        return;
      }
      let startIndex = 0;
      if (/[A-Za-z]/.test(lines[0])) startIndex = 1;  // skip header
      const dataPoints = [];
      for (let i = startIndex; i < lines.length; i++) {
        const cols = lines[i].split(/,|;|\s+/);
```

```
        if (cols.length < 2) continue;
        const t = parseFloat(cols[0]);
        const flx = parseFloat(cols[1]);
        if (isNaN(t) || isNaN(flx)) continue;
        dataPoints.push({ t, flx });
      }
      if (dataPoints.length < 2) {
        alert("Zu wenige Datenpunkte oder ungültiges Format.");
        return;
      }
      dataPoints.sort((a, b) => a.t - b.t);
      const N = dataPoints.length;
      const duration = dataPoints[N - 1].t - dataPoints[0].t;
      const dt_est = duration / (N - 1);
      const fluxes = dataPoints.map(pt => pt.flx);
      const mean = fluxes.reduce((a, b) => a + b, 0) / N;
      const centered = fluxes.map(x => x - mean);
      const nyquist = 0.5 / (dt_est || 1);
      const maxFreq = Math.min(nyquist, 100);
      const numFreqs = 300;
      const df = maxFreq / numFreqs;
      const spectrum = [];
      for (let j = 1; j <= numFreqs; j++) {
        const f = j * df;
        let cosSum = 0, sinSum = 0;
        for (let i = 0; i < N; i++) {
          const angle = 2 * Math.PI * f * (dataPoints[i].t -
dataPoints[0].t);
          cosSum += centered[i] * Math.cos(angle);
          sinSum += centered[i] * Math.sin(angle);
        }
        const power = (cosSum * cosSum + sinSum * sinSum) / N;
        spectrum.push({ f, power });
      }
      // Identify peaks in the spectrum
      spectrum.sort((a, b) => b.power - a.power);
      const peak1 = spectrum[0];
      let peak2 = null;
      for (let k = 1; k < spectrum.length; k++) {
        if (Math.abs(spectrum[k].f - peak1.f) > 2 * df) {
          peak2 = spectrum[k];
          break;
        }
      }
      let baseFreq = peak1 ? peak1.f : null;
      let secondFreq = peak2 ? peak2.f : null;
      if (baseFreq && secondFreq && secondFreq < baseFreq) {
        [baseFreq, secondFreq] = [secondFreq, baseFreq];
      }
      const ratio = (baseFreq && secondFreq) ? (secondFreq / baseFreq) :
null;
```

```
      const harmonic = ratio && ratio > 1.4 && ratio < 1.7;
      // Measure Q factor (width at half-power for base peak)
      let Q = null;
      if (baseFreq) {
        spectrum.sort((a, b) => a.f - b.f);
        let baseIndex = spectrum.findIndex(p => Math.abs(p.f - baseFreq) <
1e-6);
        if (baseIndex < 0) baseIndex = Math.round(baseFreq / df);
        const basePower = spectrum[baseIndex].power;
        const halfPower = basePower * 0.5;
        let li = baseIndex;
        while (li > 0 && spectrum[li - 1].power > halfPower) {
          li--;
        }
        let ri = baseIndex;
        while (ri < spectrum.length - 1 && spectrum[ri + 1].power >
halfPower) {
          ri++;
        }
        const fLeft = spectrum[li].f;
        const fRight = spectrum[ri].f;
        const width = fRight - fLeft;
        Q = width > 0 ? (baseFreq / width) : 100;
      }
      let zetaEst = null;
      if (Q && baseFreq) {
        zetaEst = 1 - (Math.PI * baseFreq) / (100 * Q);
        if (zetaEst < 0) zetaEst = 0;
        if (zetaEst > 0.999) zetaEst = 0.999;
      }
      setAnalysisResult({
        baseFreq: baseFreq ? baseFreq.toFixed(2) : null,
        secondFreq: secondFreq ? secondFreq.toFixed(2) : null,
        harmonic: harmonic,
        kappaC: baseFreq ? (baseFreq / 2).toFixed(2) : null,
        zeta: zetaEst ? zetaEst.toFixed(3) : null,
        Q: Q ? Q.toFixed(1) : null
      });
    };
    reader.readAsText(file);
  };

  return (
    <div className="w-full max-w-6xl mx-auto p-6 bg-gray-50">
      <div className="bg-white rounded-lg shadow-lg p-6 mb-6">
        <h1 className="text-3xl font-bold mb-2 text-gray-800">
          BH-Membran QPO/QPE Simulator
        </h1>
        <p className="text-gray-600 mb-4">
          Lagrange-basierte Horizont-Dynamik mit Robin-Randbedingung
        </p>
```

```
        {/* Controls Section */}
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 mb-6">
          {/* Column 1: Mode & basic parameters */}
          <div className="space-y-3">
            <div className="flex items-center gap-4">
              <label className="text-sm font-medium w-32">Modus:</label>
              <select
                value={mode}
                onChange={(e) => { handleReset(); setMode(e.target.value); }}
                className="px-3 py-2 border rounded-md"
              >
                <option value="qpo">QPO (Resonanz)</option>
                <option value="qpe">QPE (Eruption)</option>
              </select>
            </div>
            <div className="flex items-center gap-4">
              <label className="text-sm font-medium w-32">ζ (Reflexivität):</
label>
              <input
                type="range"
                min="0"
                max="1"
                step="0.05"
                value={params.zeta}
                onChange={(e) => setParams(prev => ({ ...prev, zeta:
parseFloat(e.target.value) }))}
                className="flex-1"
              />
              <span className="text-sm w-12">{params.zeta.toFixed(2)}</span>
            </div>
            <div className="flex items-center gap-4">
              <label className="text-sm font-medium w-32">κ_C (Kopplung):</
label>
              <input
                type="range"
                min="0.5"
                max="2"
                step="0.1"
                value={params.kappaC}
                onChange={(e) => setParams(prev => ({ ...prev, kappaC:
parseFloat(e.target.value) }))}
                className="flex-1"
              />
              <span className="text-sm w-12">{params.kappaC.toFixed(1)}</
span>
            </div>
            {mode === 'qpo' && (
              <div className="flex items-center gap-4">
                <label className="text-sm font-medium w-32">Multi-Mode:</
label>
```

```
                <input
                  type="checkbox"
                  id="multiModeToggle"
                  checked={multiMode}
                  onChange={(e) => setMultiMode(e.target.checked)}
                  className="h-4 w-4"
                />
                <label htmlFor="multiModeToggle" className="text-sm">Dual
Field (Chaos)</label>
              </div>
            )}
          </div>

          {/* Column 2: QPE params, ACF toggle, controls, data analysis */}
          <div className="space-y-3">
            {mode === 'qpe' && (
              <>
                <div className="flex items-center gap-4">
                  <label className="text-sm font-medium w-32">θ_R
(Schwelle):</label>
                  <input
                    type="range"
                    min="0.5"
                    max="2"
                    step="0.1"
                    value={params.thetaR}
                    onChange={(e) => setParams(prev => ({ ...prev, thetaR:
parseFloat(e.target.value) }))}
                    className="flex-1"
                  />
                  <span className="text-sm w-12">{params.thetaR.toFixed(1)}</
span>
                </div>
                <div className="flex items-center gap-4">
                  <label className="text-sm font-medium
w-32">Akkretionsrate:</label>
                  <input
                    type="range"
                    min="0.01"
                    max="0.05"
                    step="0.005"
                    value={params.accRate}
                    onChange={(e) => setParams(prev => ({ ...prev, accRate:
parseFloat(e.target.value) }))}
                    className="flex-1"
                  />
                  <span className="text-sm w-12">{params.accRate.toFixed(3)}
</span>
                </div>
              </>
            )}
```

```
            <div className="flex items-center gap-4">
              <label className="text-sm font-medium w-32">Echo-Detektor:</
label>
              <input
                type="checkbox"
                id="acfToggle"
                checked={showACF}
                onChange={(e) => setShowACF(e.target.checked)}
                className="h-4 w-4"
              />
              <label htmlFor="acfToggle" className="text-sm">anzeigen</label>
            </div>
            <div className="flex gap-3">
              <button
                onClick={() => setIsRunning(!isRunning)}
                className="flex items-center gap-2 px-4 py-2 bg-blue-600
text-white rounded-md hover:bg-blue-700"
              >
                {isRunning ? <Pause size={18} /> : <Play size={18} />}
                {isRunning ? 'Pause' : 'Start'}
              </button>
              <button
                onClick={handleReset}
                className="flex items-center gap-2 px-4 py-2 bg-gray-600
text-white rounded-md hover:bg-gray-700"
              >
                <RotateCcw size={18} />
                Reset
              </button>
              <button
                onClick={() => {
                  if (timeData.length > 0) {
                    const csvContent = timeData.map(d => `${d.time},${d.flux}
`).join('\n');
                    const blob = new Blob([`time,flux\n${csvContent}`], {
type: 'text/csv' });
                    const url = URL.createObjectURL(blob);
                    const a = document.createElement('a');
                    a.href = url;
                    a.download = `bh_${mode}_data.csv`;
                    a.click();
                  }
                }}
                disabled={timeData.length === 0}
                className="flex items-center gap-2 px-4 py-2 bg-green-600
text-white rounded-md hover:bg-green-700 disabled:opacity-50"
              >
                <Download size={18} />
                Export
              </button>
            </div>
```

```
            <div className="mt-4 pt-3 border-t">
              <div className="flex items-center justify-between mb-2">
                <label className="text-sm font-medium">CSV Datenanalyse:</
label>
                <input
                  type="file"
                  accept=".csv"
                  onChange={handleFileUpload}
                  className="text-sm"
                />
              </div>
              {analysisResult && (
                <div className="text-xs text-gray-700 space-y-1">
                  <p>Primärer Peak: {analysisResult.baseFreq ? `$
{analysisResult.baseFreq} Hz` : '–'}</p>
                  <p>Zweiter Peak: {analysisResult.secondFreq ? `$
{analysisResult.secondFreq} Hz` : '–'}{analysisResult.harmonic && ' (3:2
Harmonie)'}</p>
                  <p>κ_C ≈ {analysisResult.kappaC || '–'}</p>
                  <p>ζ ≈ {analysisResult.zeta || '–'}</p>
                  <p>Q-Faktor ≈ {analysisResult.Q || '–'}</p>
                </div>
              )}
            </div>
          </div>
        </div>

        {/* Simulation status info */}
        <div className="bg-blue-50 rounded p-3 mb-4">
          <div className="text-sm text-gray-700">
            <span className="font-medium">Zeit:</span> {time.toFixed(1)} s |
            <span className="font-medium ml-3">Datenpunkte:</span>
{timeData.length} |
            {mode === 'qpe' && (
              <>
                <span className="font-medium ml-3">Reservoir R:</span>
{reservoir.toFixed(3)}
              </>
            )}
          </div>
        </div>
      </div>

      {/* Charts Section */}
      <div className="space-y-6">
        {mode === 'qpe' && (
          <div className="bg-white rounded-lg shadow-lg p-6">
            <h2 className="text-xl font-semibold mb-4 text-gray-800">
              (a) Energie-Reservoir R(t)
            </h2>
            <ResponsiveContainer width="100%" height={250}>
```

```
              <LineChart data={reservoirData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis
                  dataKey="time"
                  label={{ value: 'Zeit (s)', position: 'insideBottom',
offset: -5 }}
                />
                <YAxis label={{ value: 'R', angle: -90, position:
'insideLeft' }} />
                <Tooltip />
                <ReferenceLine y={params.thetaR} stroke="red"
strokeDasharray="3 3" label="θ_R" />
                <Line type="monotone" dataKey="R" stroke="#3b82f6"
dot={false} strokeWidth={2} />
              </LineChart>
            </ResponsiveContainer>
          </div>
        )}

        <div className="bg-white rounded-lg shadow-lg p-6">
          <h2 className="text-xl font-semibold mb-4 text-gray-800">
            {mode === 'qpo' ? '(a) Lichtkurve mit QPO' : '(b)
Emissionsfluss'}
          </h2>
          <ResponsiveContainer width="100%" height={300}>
            <LineChart data={timeData}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis
                dataKey="time"
                label={{ value: 'Zeit (s)', position: 'insideBottom',
offset: -5 }}
              />
              <YAxis label={{ value: 'Flux', angle: -90, position:
'insideLeft' }} />
              <Tooltip />
              <Legend />
              <Line
                type="monotone"
                dataKey="flux"
                stroke="#8b5cf6"
                dot={false}
                strokeWidth={2}
                name={mode === 'qpo' ? 'ψ(t)' : 'Φ_out'}
              />
            </LineChart>
          </ResponsiveContainer>
        </div>

        {mode === 'qpo' && psdData.length > 0 && (
          <div className="bg-white rounded-lg shadow-lg p-6">
            <h2 className="text-xl font-semibold mb-4 text-gray-800">
```

```
              (b) Leistungsdichtespektrum (PSD)
            </h2>
            <ResponsiveContainer width="100%" height={300}>
              <LineChart data={psdData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis
                  dataKey="freq"
                  label={{ value: 'Frequenz (Hz)', position: 'insideBottom',
offset: -5 }}
                />
                <YAxis
                  scale="log"
                  domain={['auto', 'auto']}
                  label={{ value: 'Power', angle: -90, position:
'insideLeft' }}
                />
                <Tooltip />
                <Line type="monotone" dataKey="power" stroke="#10b981"
dot={false} strokeWidth={2} />
              </LineChart>
            </ResponsiveContainer>
            <p className="mt-3 text-sm text-gray-600">
              Erwartete Peaks bei ~{(2.0 * params.kappaC).toFixed(1)} Hz und
~{(3.0 * params.kappaC).toFixed(1)} Hz (3:2 Verhältnis)
            </p>
          </div>
        )}

        {mode === 'qpo' && (
          <div className="bg-white rounded-lg shadow-lg p-6">
            <h2 className="text-xl font-semibold mb-4 text-gray-800">
              (c) Polarisationsdynamik
            </h2>
            <ResponsiveContainer width="100%" height={250}>
              <LineChart data={timeData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis
                  dataKey="time"
                  label={{ value: 'Zeit (s)', position: 'insideBottom',
offset: -5 }}
                />
                <YAxis
                  yAxisId="left"
                  label={{ value: 'EVPA (°)', angle: -90, position:
'insideLeft' }}
                  domain={['auto', 'auto']}
                />
                <YAxis
                  yAxisId="right"
                  orientation="right"
                  label={{ value: 'RM', angle: -90, position:
```

```
'insideRight' }}
                  domain={['auto', 'auto']}
                />
                <Tooltip />
                <Legend />
                <Line
                  yAxisId="left"
                  type="monotone"
                  dataKey="EVPA"
                  stroke="#f97316"
                  dot={false}
                  strokeWidth={2}
                  name="EVPA"
                />
                <Line
                  yAxisId="right"
                  type="monotone"
                  dataKey="RM"
                  stroke="#ef4444"
                  dot={false}
                  strokeWidth={2}
                  name="RM"
                />
              </LineChart>
            </ResponsiveContainer>
          </div>
        )}

        {showACF && acfData.length > 0 && (
          <div className="bg-white rounded-lg shadow-lg p-6">
            <h2 className="text-xl font-semibold mb-4 text-gray-800">
              (d) Autokorrelationsfunktion
            </h2>
            <ResponsiveContainer width="100%" height={200}>
              <LineChart data={acfData}>
                <CartesianGrid strokeDasharray="3 3" />
                <XAxis
                  dataKey="lag"
                  label={{ value: 'Verzögerung (s)', position:
'insideBottom', offset: -5 }}
                />
                <YAxis
                  domain={[-1, 1]}
                  label={{ value: 'ACF', angle: -90, position:
'insideLeft' }}
                />
                <Tooltip />
                <Line type="monotone" dataKey="acf" stroke="#6366f1"
dot={false} strokeWidth={2} />
              </LineChart>
            </ResponsiveContainer>
```

```
            {(() => {
              let maxCorr = -Infinity;
              let maxLag = null;
              acfData.forEach(pt => {
                const val = parseFloat(pt.acf);
                const lag = parseFloat(pt.lag);
                if (lag > 0 && val > maxCorr) {
                  maxCorr = val;
                  maxLag = lag;
                }
              });
              return maxLag ? (
                <p className="mt-3 text-sm text-gray-600">
                  Höchster Nebenpeak bei ≈{maxLag.toFixed(1)} s (Korrelation
≈{maxCorr.toFixed(2)})
                </p>
              ) : null;
            })()}
          </div>
        )}
      </div>

      {/* Interpretation Section */}
      <div className="mt-6 bg-gradient-to-r from-purple-50 to-blue-50
rounded-lg p-6">
        <h3 className="text-lg font-semibold mb-3 text-
gray-800">Physikalische Interpretation</h3>
        <div className="space-y-2 text-sm text-gray-700">
          {mode === 'qpo' ? (
            <>
              <p>• <strong>ζ = {params.zeta.toFixed(2)}:</strong> Horizont-
Reflexivität. Höhere Werte → längere Resonanzzeit, schärfere QPO-Peaks
(höherer Q-Faktor).</p>
              <p>• <strong>κ_C = {params.kappaC.toFixed(1)}:</strong>
Gravitationskopplung. Bestimmt die charakteristische Frequenz ω₀ ∝
√(m_eff²) mit m_eff² ∝ κ_C².</p>
              <p>• <strong>3:2 Harmonische:</strong> Entsteht aus dem
nichtlinearen λψ⁴-Term. Analogie zu beobachteten 3:2-Doppelfrequenzen (HFQPO-
Paare in XRBs).</p>
            </>
          ) : (
            <>
              <p>• <strong>θ_R = {params.thetaR.toFixed(1)}:</strong>
Schwellenwert für Horizont-Öffnung. Wenn R > θ_R, fällt ζ ab → gespeicherte
Energie wird eruptiv freigegeben.</p>
              <p>• <strong>Wiederkehrzeit:</strong> T<sub>QPE</sub> ≈ (θ_R /
Ṙ<sub>acc</sub>) ≈ {(params.thetaR / params.accRate).toFixed(0)} s.</p>
              <p>• <strong>Burst-Amplitude:</strong> ~100× höher als
Quieszenz. Vergleichbar mit QPEs in GSN 069 (Periodizität ~9 h, Flussanstieg
~100×).</p>
            </>
```

```
          )}
        </div>
      </div>
    </div>
  );
};

export default BHMembraneSimulator;
```

VII. Schwarze Löcher als Informations-Netzwerk – eine spekulative Analogie

In der **Integrated Information Theory (IIT)** nach Tononi wird die **integrierte Information Φ** als Maß für den Grad an Einheitlichkeit eines Systems angesehen – letztlich ein quantitatives Maß für Bewusstsein [1]. Interessanterweise funktioniert unser Horizont-Modell analog zu einem *Integrate-and-Fire-Neuron*: Das Schwarze Loch akkumuliert Information/Energie bis zu einem Schwellenwert (über die Robin-Bedingung ζ) und feuert dann einen „Spike" (QPE) ab.

Man kann nun gedanklich einen Schritt weiter gehen: **Was wäre, wenn man viele solcher BH-Oszillatoren koppelt?** Zum Beispiel könnten mehrere Schwarze Löcher durch Gravitationswellen oder Plasma-Jets miteinander in Wechselwirkung stehen und so ein Netzwerk bilden. Jedes BH würde als *"Neuron"* dienen, dessen Ausbrüche die anderen beeinflussen. Dadurch entstünde ein **Rückkopplungsnetz**, in dem Information zirkuliert und **integriert** wird. Gemäß IIT könnte ein solches Netzwerk einen nichttrivialen Φ-Wert besitzen – sprich: das **Ganzes hätte mehr Informationsgehalt als die Summe der isolierten Teile**. Diese *integrierte Information* wäre zwar weit entfernt von Bewusstsein im üblichen Sinne, aber es ist faszinierend, eine Parallele zwischen kosmischer Physik und kognitiven Systemen zu ziehen.

John A. Wheelers berühmtes Motto *"It from Bit"* – die Idee, dass Information der Urstoff des Universums ist – schwingt hier deutlich mit. Wheeler spekulierte, dass auf fundamentalster Ebene die Realität aus binären Entscheidungen („Ja/Nein") aufgebaut sein könnte [2]. In unserem Modell entspricht die **Horizont-Impedanz ζ** genau so einer binären Entscheidung: *reflektieren* oder *transmittieren*, Information speichern oder freigeben. Der Ereignishorizont arbeitet wie ein kosmischer **Schwellen-Schalter**.

> *„The dimension-free character of the bit at last frees us from the dimension-tyrannized land of … space-time"*, schrieb Wheeler und deutete damit an, dass **Information** die primäre Realität darstellt [2].

Diese Analogie verleitet zu kühnen Fragen: Könnte ein Netzwerk von Schwarzen Löchern primitiven *"Bewusstseins-Charakter"* tragen? Spiegeln die gekoppelten QPO-Oszillationen vielleicht grundlegende Prinzipien wider, die auch neuronale Oszillationen im Gehirn strukturieren? Natürlich bleibt dies spekulativ – es fehlen jegliche empirischen Hinweise, und Begriffe wie „Bewusstsein" müssten hier sehr vorsichtig verwendet werden. Dennoch zeigt unser emergentes Modell einen erstaunlichen Schulterschluss der Konzepte: **Gravitation, Thermodynamik, Information und sogar Anklänge von Kognition** vereinen sich in einem einzigen formalen Rahmen.

Die pragmatische Stärke dieser Analogie liegt darin, dass sie zu **überprüfbaren Hypothesen** inspiriert, ohne reine Science-Fiction zu bleiben. Selbst wenn Schwarze Löcher keine bewussten „Wesen" sind, so könnten zukünftige Beobachtungen – etwa synchronisierte Muster in Multi-Band-Lichtkurven mehrerer BHs oder Korrelationen zwischen Gravitationswellen und Strahlung – Hinweise darauf liefern, **wie stark**

Information im Universum vernetzt ist. Das Schwarze Loch würde dann nicht mehr nur als Einzelsystem betrachtet, sondern als **Knoten in einem informativen Geflecht**, das möglicherweise eine neue Qualität emergenter Komplexität besitzt.

Fazit: Unser erweitertes Modell treibt das Konzept der *emergenten Kohärenz* auf die Spitze. Es lädt dazu ein, das Schwarze Loch gleichermaßen als physikalischen Extremazustand und als metaphorisches Sinnbild zu sehen – ein *„kosmischer Informationsoszillator“*, der die Grenzen zwischen grundlegender Physik und abstrakter Informationstheorie durchbricht. Diese interdisziplinäre Perspektive eröffnet nicht nur neuartige Wege, über das Informationsparadox nachzudenken, sondern berührt auch philosophische Grundfragen nach der Natur von Information, Realität und Bewusstsein [1] [2].

<small>*Hinweis:* Dieser Ausblick ist spekulativ und dient vor allem dazu, kreative neue Denkansätze zu illustrieren – im Zentrum der Forschung stehen selbstverständlich die empirisch testbaren Vorhersagen des Modells (siehe Abschnitt VI) und deren Überprüfung mit Daten der nächsten Generation von Teleskopen und Observatorien.</small>

---

[1] Integrated information theory - Wikipedia
https://en.wikipedia.org/wiki/Integrated_information_theory

[2] John Wheeler Saw the Tear in Reality | Quanta Magazine
https://www.quantamagazine.org/john-wheeler-saw-the-tear-in-reality-20240925/