Code Documentation: More_Zeeman

UMEDCTA Repository

December 3, 2025

Contents

1 More_Zeeman/index_unified.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>More Machine - Zeeman Catastrophe & Cantorian Diagonalization</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }
        body {
            font-family: 'Helvetica Neue', Arial, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: #333;
            min-height: 100vh;
            display: flex;
            flex-direction: column;
            align-items: center;
            padding: 20px;
        }
        header {
            text-align: center;
            margin-bottom: 30px;
            color: white;
        }
        header h1 {
            font-size: 2.5em;
            font-weight: 300;
            margin-bottom: 10px;
        }
        header p {
            font-size: 1.2em;
            opacity: 0.9;
        }
        #main-content {
            display: grid;
            grid-template-columns: 650px 550px;
            gap: 30px;
            max-width: 1300px;
            width: 100%;
            justify-content: center;
        }
        .panel {
            background: white;
            border-radius: 15px;
            padding: 25px;
            box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
        }
        #zeeman-panel {
            grid-row: 1 / 3;
        }
        #matrix-panel {
            grid-column: 2;
            grid-row: 1;
        }
        #acoustic-panel {
            grid-column: 2;
            grid-row: 2;
        }
        .panel h2 {
```

```
            font-size: 1.5em;
            font-weight: 500;
            margin-bottom: 20px;
            color: #667eea;
        }
        canvas {
            border: 2px solid #e0e0e0;
            border-radius: 10px;
            display: block;
            cursor: crosshair;
        }
        .controls {
            margin-top: 20px;
            padding-top: 20px;
            border-top: 1px solid #e0e0e0;
        }
        .control-group {
            margin-bottom: 15px;
        }
        .control-group label {
            display: block;
            font-weight: 500;
            margin-bottom: 8px;
            color: #555;
        }
        .control-group input[type="range"] {
            width: 100%;
            height: 6px;
            border-radius: 3px;
            background: #e0e0e0;
            outline: none;
            -webkit-appearance: none;
        }
        .control-group input[type="range"]::-webkit-slider-thumb {
            -webkit-appearance: none;
            appearance: none;
            width: 18px;
            height: 18px;
            border-radius: 50%;
            background: #667eea;
            cursor: pointer;
        }
        .control-group input[type="range"]::-moz-range-thumb {
            width: 18px;
            height: 18px;
            border-radius: 50%;
            background: #667eea;
            cursor: pointer;
            border: none;
        }
        .value-display {
            display: inline-block;
            min-width: 50px;
            text-align: right;
            font-weight: 500;
            color: #667eea;
        }
        #zeeman-status {
            padding: 10px;
            background: #f5f5f5;
            border-radius: 8px;
            text-align: center;
            font-weight: 500;
        }
        #status-value {
```

```
            color: #667eea;
            font-weight: 700;
        }
        #matrix-grid {
            display: grid;
            gap: 2px;
            justify-content: center;
            margin-top: 20px;
        }
        .matrix-cell {
            width: 50px;
            height: 50px;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 24px;
            font-weight: bold;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
            border-radius: 8px;
            transition: all 0.3s ease;
        }
        .matrix-cell:hover {
            transform: scale(1.05);
        }
        svg {
            border: 2px solid #e0e0e0;
            border-radius: 10px;
            display: block;
        }
    </style>
</head>
<body>
    <header>
        <h1>The More Machine</h1>
        <p>Zeeman Catastrophe Machine + Cantorian Diagonalization</p>
    </header>
    <div id="main-content">
        <div class="panel" id="zeeman-panel">
            <h2>The Thinker (Zeeman Machine)</h2>
            <canvas id="zeeman-canvas" width="600" height="600"></canvas>
            <div class="controls">
                <div class="control-group">
                    <label>
                        Spring Constant (k): <span class="value-display" id="k-value">2.0</span>
                    </label>
                    <input type="range" id="spring-k" min="0.5" max="5" value="2" step="0.1">
                </div>
                <div class="control-group">
                    <label>
                        Natural Length (L₀): <span class="value-display" id="l0-value">100</span>
                    </label>
                    <input type="range" id="natural-length" min="50" max="200" value="100" step="5">
                </div>
                <div class="control-group">
                    <label>
                        Time Speed: <span class="value-display" id="time-value">1.0</span>
                    </label>
                    <input type="range" id="time-speed" min="0.1" max="3" value="1" step="0.1">
                </div>
                <div id="zeeman-status">State: <span id="status-value">Stable</span></div>
            </div>
        </div>
        <div class="panel" id="matrix-panel">
            <h2>The Memory (More Machine)</h2>
```

```
            <p style="margin-bottom: 15px; color: #666;">
                Each catastrophe is a memorable moment. The matrix reads the wheel's state (M or W) and grows via Cantorian diagonalization.
            </p>
            <div id="matrix-grid"></div>
        </div>
        <div class="panel" id="acoustic-panel">
            <h2>The Sound of Time (Acoustic Metaphor)</h2>
            <p style="margin-bottom: 15px; color: #666;">
                Tension and release visualized as air compression. The piston responds to the wheel's angular velocity, creating pressure waves through particles.
            </p>
            <svg id="acoustic-svg" width="500" height="250"></svg>
        </div>
    </div>
    <script src="script_unified.js"></script>
</body>
</html>

```

2 More_Zeeman/script_unified.js

```
document.addEventListener('DOMContentLoaded', () => {

    // --- MATRIX AUTOMATON CLASS ---
    class MatrixAutomaton {
        constructor() {
            this.gridContainer = document.getElementById('matrix-grid');
            this.matrix = [['M']]; // Start with single M
            this.maxSize = 8;
            this.render();
        }

        // Cantorian diagonalization
        grow(finalCharacter) {
            let baseMatrix = this.matrix;
            let newSize = this.matrix.length + 1;

            // Ripple effect when at max size
            if (this.matrix.length >= this.maxSize) {
                newSize = this.maxSize;
                const shifted = [];
                for (let i = 1; i < this.matrix.length; i++) {
                    shifted.push(this.matrix[i].slice(1));
                }
                baseMatrix = shifted;
            }

            const n = baseMatrix.length;
            const newMatrix = Array(newSize).fill(null).map(() => Array(newSize).fill(null));

            // Copy base matrix
            for (let i = 0; i < n; i++) {
                for (let j = 0; j < n; j++) {
                    newMatrix[i][j] = baseMatrix[i][j];
                }
            }

            // Get diagonal and negate it
            const diagonal = [];
            for (let i = 0; i < n; i++) {
                diagonal.push(baseMatrix[i][i]);
            }
            const negatedDiagonal = diagonal.map(char => (char === 'M' ? 'W' : 'M'));

            // Fill last row and column with negated diagonal
            for (let i = 0; i < n; i++) {
                newMatrix[n][i] = negatedDiagonal[i];
                newMatrix[i][n] = negatedDiagonal[i];
            }

            // Fill corner with new character from Zeeman machine
            newMatrix[n][n] = finalCharacter;

            this.matrix = newMatrix;
            this.render();
        }

        render() {
            this.gridContainer.innerHTML = '';
            const size = this.matrix.length;
            this.gridContainer.style.gridTemplateColumns = `repeat(${size}, 50px)`;

            for (let i = 0; i < size; i++) {
```

```
                for (let j = 0; j < size; j++) {
                    const cell = document.createElement('div');
                    cell.classList.add('matrix-cell');
                    cell.textContent = this.matrix[i][j] || '';
                    this.gridContainer.appendChild(cell);
                }
            }
        }
    }

    // --- ZEEMAN CATASTROPHE MACHINE CLASS (WITH PROPER PHYSICS) ---
    class ZeemanMachine {
        constructor(matrixAutomaton) {
            this.canvas = document.getElementById('zeeman-canvas');
            this.ctx = this.canvas.getContext('2d');
            this.automaton = matrixAutomaton;

            // DOM Elements
            this.statusEl = document.getElementById('status-value');
            this.timeSpeedSlider = document.getElementById('time-speed');
            this.springKSlider = document.getElementById('spring-k');
            this.naturalLengthSlider = document.getElementById('natural-length');
            this.kValueDisplay = document.getElementById('k-value');
            this.l0ValueDisplay = document.getElementById('l0-value');
            this.timeValueDisplay = document.getElementById('time-value');

            // Canvas parameters
            this.width = this.canvas.width;
            this.height = this.canvas.height;
            this.wheelRadius = 60;
            this.attachmentRadius = 50; // R
            this.wheelCenter = { x: this.width / 2, y: this.height / 2 };
            this.fixedAnchor = { x: this.width / 2, y: 50 };

            // Physics parameters (user-controllable)
            this.springK = 2.0; // Spring constant
            this.L0 = 100; // Natural length
            this.timeSpeed = 1.0;

            // Physics state
            this.angle = Math.PI / 2; // Start pointing down ('M')
            this.previousAngle = this.angle;

            // Interaction state
            this.isDragging = false;
            this.controlPoint = { x: this.width / 2 + 100, y: this.height / 2 + 100 };

            // Machine state
            this.isStable = true;
            this.currentStability = 1.0;

            this.bindEvents();
            this.loop();
        }

        bindEvents() {
            // Control point dragging
            this.canvas.addEventListener('mousedown', e => {
                const rect = this.canvas.getBoundingClientRect();
                const mx = e.clientX - rect.left;
                const my = e.clientY - rect.top;
                const dist = Math.sqrt(Math.pow(mx - this.controlPoint.x, 2) + Math.pow(my -
                ↪  this.controlPoint.y, 2));
                if (dist < 15) {
                    this.isDragging = true;
```

```
            }
        });

        window.addEventListener('mousemove', e => {
            if (this.isDragging) {
                const rect = this.canvas.getBoundingClientRect();
                this.controlPoint.x = Math.max(0, Math.min(this.width, e.clientX - rect.left));
                this.controlPoint.y = Math.max(0, Math.min(this.height, e.clientY - rect.top));
            }
        });

        window.addEventListener('mouseup', () => {
            this.isDragging = false;
        });

        // Parameter sliders
        this.timeSpeedSlider.addEventListener('input', e => {
            this.timeSpeed = parseFloat(e.target.value);
            this.timeValueDisplay.textContent = this.timeSpeed.toFixed(1);
        });

        this.springKSlider.addEventListener('input', e => {
            this.springK = parseFloat(e.target.value);
            this.kValueDisplay.textContent = this.springK.toFixed(1);
        });

        this.naturalLengthSlider.addEventListener('input', e => {
            this.L0 = parseFloat(e.target.value);
            this.l0ValueDisplay.textContent = this.L0.toFixed(0);
        });
    }

    // Get attachment point on wheel rim
    getAttachmentPoint(angle) {
        return {
            x: this.wheelCenter.x + this.attachmentRadius * Math.cos(angle),
            y: this.wheelCenter.y + this.attachmentRadius * Math.sin(angle)
        };
    }

    // Calculate potential energy using Hooke's Law: E = 0.5*k*(L-L0)^2
    calculateEnergy(angle) {
        const P = this.getAttachmentPoint(angle);

        // Calculate current lengths
        const L1 = Math.sqrt(Math.pow(P.x - this.fixedAnchor.x, 2) + Math.pow(P.y -
        ↪  this.fixedAnchor.y, 2));
        const L2 = Math.sqrt(Math.pow(P.x - this.controlPoint.x, 2) + Math.pow(P.y -
        ↪  this.controlPoint.y, 2));

        // Energy only if stretched beyond L0
        const E1 = (L1 > this.L0) ? 0.5 * this.springK * Math.pow(L1 - this.L0, 2) : 0;
        const E2 = (L2 > this.L0) ? 0.5 * this.springK * Math.pow(L2 - this.L0, 2) : 0;

        return E1 + E2;
    }

    // Calculate gradient (torque) - dE/dAngle
    calculateGradient(angle) {
        const R = this.attachmentRadius;

        // Coordinates relative to wheel center
        const Fx = this.fixedAnchor.x - this.wheelCenter.x;
        const Fy = this.fixedAnchor.y - this.wheelCenter.y;
        const Cx = this.controlPoint.x - this.wheelCenter.x;
```

```
        const Cy = this.controlPoint.y - this.wheelCenter.y;

        // Attachment point relative to center
        const Px = R * Math.cos(angle);
        const Py = R * Math.sin(angle);

        // Calculate lengths
        const L1 = Math.sqrt(Math.pow(Px - Fx, 2) + Math.pow(Py - Fy, 2));
        const L2 = Math.sqrt(Math.pow(Px - Cx, 2) + Math.pow(Py - Cy, 2));

        let Term1 = 0;
        if (L1 > this.L0 && L1 > 0.001) {
            Term1 = this.springK * (1 - this.L0 / L1) * (Fx * Math.sin(angle) - Fy *
            ↪  Math.cos(angle));
        }

        let Term2 = 0;
        if (L2 > this.L0 && L2 > 0.001) {
            Term2 = this.springK * (1 - this.L0 / L2) * (Cx * Math.sin(angle) - Cy *
            ↪  Math.cos(angle));
        }

        return R * (Term1 + Term2);
    }

    // Estimate stability (second derivative) using central differences
    estimateStability(angle) {
        const dA = 0.01;
        const gradientPlus = this.calculateGradient(angle + dA);
        const gradientMinus = this.calculateGradient(angle - dA);
        return (gradientPlus - gradientMinus) / (2 * dA);
    }

    // Update disc state using gradient descent (finds local minimum - hysteresis)
    updateState() {
        let newAngle = this.angle;
        const learningRate = 0.001;
        const maxIterations = 250;
        const convergenceThreshold = 0.05;
        let stability = 0;

        // Gradient descent loop
        for (let i = 0; i < maxIterations; i++) {
            const gradient = this.calculateGradient(newAngle);

            if (Math.abs(gradient) < convergenceThreshold) {
                stability = this.estimateStability(newAngle);
                if (stability > 0) break; // Found stable minimum
            }

            let step = learningRate * gradient;
            const maxStep = 0.1;
            step = Math.max(-maxStep, Math.min(maxStep, step));

            newAngle -= step;
            newAngle = (newAngle + 4 * Math.PI) % (2 * Math.PI);
        }

        if (stability <= 0) {
            stability = this.estimateStability(newAngle);
        }

        // Detect catastrophe (snap)
        let angleDiff = newAngle - this.angle;
        while (angleDiff > Math.PI) angleDiff -= 2 * Math.PI;
```

```
253             while (angleDiff < -Math.PI) angleDiff += 2 * Math.PI;
254
255             let snapped = Math.abs(angleDiff) > 0.6;
256
257             this.angle = newAngle;
258             this.currentStability = stability / 5000; // Normalize for visualization
259             this.currentStability = Math.max(0, Math.min(1, this.currentStability));
260
261             return snapped;
262         }
263
264         update() {
265             const wasStable = this.isStable;
266             const snapped = this.updateState();
267
268             // Determine current output character
269             const currentOutput = (this.angle > Math.PI / 2 && this.angle < 3 * Math.PI / 2) ? 'W' :
                ↪  'M';
270
271             // Check stability
272             this.isStable = this.currentStability > 0.3;
273
274             if (this.isStable) {
275                 this.statusEl.textContent = 'Stable - ' + currentOutput;
276                 // Trigger matrix growth on catastrophe
277                 if (snapped && !wasStable) {
278                     this.automaton.grow(currentOutput);
279                 }
280             } else {
281                 this.statusEl.textContent = 'Superimposed (Indeterminate)';
282             }
283         }
284
285         draw() {
286             this.ctx.clearRect(0, 0, this.width, this.height);
287
288             const P = this.getAttachmentPoint(this.angle);
289
290             // Calculate lengths for visualization
291             const L1 = Math.sqrt(Math.pow(P.x - this.fixedAnchor.x, 2) + Math.pow(P.y -
                ↪  this.fixedAnchor.y, 2));
292             const L2 = Math.sqrt(Math.pow(P.x - this.controlPoint.x, 2) + Math.pow(P.y -
                ↪  this.controlPoint.y, 2));
293
294             // Draw elastic bands
295             this.ctx.lineWidth = 3;
296
297             // Band to fixed anchor
298             this.ctx.beginPath();
299             this.ctx.moveTo(this.fixedAnchor.x, this.fixedAnchor.y);
300             this.ctx.lineTo(P.x, P.y);
301             if (L1 <= this.L0) {
302                 this.ctx.setLineDash([5, 5]); // Dashed if slack
303                 this.ctx.strokeStyle = '#999';
304             } else {
305                 this.ctx.setLineDash([]);
306                 this.ctx.strokeStyle = '#c0392b'; // Red
307             }
308             this.ctx.stroke();
309
310             // Band to control point
311             this.ctx.beginPath();
312             this.ctx.moveTo(this.controlPoint.x, this.controlPoint.y);
313             this.ctx.lineTo(P.x, P.y);
314             if (L2 <= this.L0) {
```

```
                this.ctx.setLineDash([5, 5]); // Dashed if slack
                this.ctx.strokeStyle = '#999';
            } else {
                this.ctx.setLineDash([]);
                this.ctx.strokeStyle = '#2980b9'; // Blue
            }
            this.ctx.stroke();
            this.ctx.setLineDash([]);

            // Draw fixed anchor
            this.ctx.beginPath();
            this.ctx.arc(this.fixedAnchor.x, this.fixedAnchor.y, 8, 0, 2 * Math.PI);
            this.ctx.fillStyle = '#c0392b';
            this.ctx.fill();

            // Draw wheel
            this.ctx.beginPath();
            this.ctx.arc(this.wheelCenter.x, this.wheelCenter.y, this.wheelRadius, 0, 2 * Math.PI);
            this.ctx.fillStyle = '#ecf0f1';
            this.ctx.fill();
            this.ctx.strokeStyle = '#7f8c8d';
            this.ctx.lineWidth = 3;
            this.ctx.stroke();

            // Draw 'M' character rotating with wheel
            this.ctx.save();
            this.ctx.font = 'bold 60px sans-serif';
            this.ctx.fillStyle = this.isStable ? '#2c3e50' : 'rgba(44, 62, 80, 0.3)';
            this.ctx.textAlign = 'center';
            this.ctx.textBaseline = 'middle';
            this.ctx.translate(this.wheelCenter.x, this.wheelCenter.y);
            this.ctx.rotate(this.angle + Math.PI / 2); // +90° to make M upright

            // Apply blur if unstable
            if (!this.isStable) {
                this.ctx.filter = `blur(${(1 - this.currentStability) * 5}px)`;
            }

            this.ctx.fillText('M', 0, 0);
            this.ctx.restore();

            // Draw attachment point
            this.ctx.beginPath();
            this.ctx.arc(P.x, P.y, 6, 0, 2 * Math.PI);
            this.ctx.fillStyle = '#555';
            this.ctx.fill();

            // Draw control point
            this.ctx.beginPath();
            this.ctx.arc(this.controlPoint.x, this.controlPoint.y, 12, 0, 2 * Math.PI);
            this.ctx.fillStyle = '#2980b9';
            this.ctx.fill();
            this.ctx.strokeStyle = this.isDragging ? '#fff' : '#333';
            this.ctx.lineWidth = 2;
            this.ctx.stroke();
        }

        loop() {
            this.update();
            this.draw();
            requestAnimationFrame(() => this.loop());
        }
    }

    // --- ACOUSTIC VISUALIZATION CLASS ---
```

```
380     class AcousticVisualization {
381         constructor(zeemanMachine) {
382             this.zeeman = zeemanMachine;
383             this.svg = document.getElementById('acoustic-svg');
384             this.svgNS = "http://www.w3.org/2000/svg";
385 
386             // Parameters
387             this.width = 500;
388             this.height = 300;
389             this.numParticles = 200;
390             this.particleRadius = 2;
391             this.speakerWidth = 40;
392             this.speakerHeight = 150;
393             this.speakerX = 60;
394             this.speakerY = (this.height - this.speakerHeight) / 2;
395             this.chamberEndX = this.width - 20;
396 
397             // Wave propagation
398             this.waveSpeed = 6;
399             this.historySeconds = 1.5;
400             this.fps = 60;
401             this.history = new Array(Math.ceil(this.fps * this.historySeconds)).fill(0);
402             this.historyIndex = 0;
403             this.maxDisplacement = 30;
404             this.sensitivityScale = 15;
405             this.smoothedDisplacement = 0;
406             this.smoothingFactor = 0.2;
407 
408             this.particles = [];
409             this.pistonRect = null;
410 
411             this.init();
412         }
413 
414         init() {
415             this.svg.innerHTML = '';
416             this.particles = [];
417 
418             // Draw walls
419             const drawWall = (y) => {
420                 const wall = document.createElementNS(this.svgNS, 'line');
421                 wall.setAttribute('x1', this.speakerX);
422                 wall.setAttribute('y1', y);
423                 wall.setAttribute('x2', this.chamberEndX);
424                 wall.setAttribute('y2', y);
425                 wall.setAttribute('stroke', '#999');
426                 wall.setAttribute('stroke-width', '2');
427                 this.svg.appendChild(wall);
428             };
429             drawWall(this.speakerY);
430             drawWall(this.speakerY + this.speakerHeight);
431 
432             // Draw piston
433             this.pistonRect = document.createElementNS(this.svgNS, 'rect');
434             this.pistonRect.setAttribute('x', this.speakerX - this.speakerWidth);
435             this.pistonRect.setAttribute('y', this.speakerY);
436             this.pistonRect.setAttribute('width', this.speakerWidth);
437             this.pistonRect.setAttribute('height', this.speakerHeight);
438             this.pistonRect.setAttribute('fill', '#aaa');
439             this.pistonRect.setAttribute('stroke', '#666');
440             this.pistonRect.setAttribute('stroke-width', '2');
441             this.svg.appendChild(this.pistonRect);
442 
443             // Create air particles in a grid
444             const particleGroup = document.createElementNS(this.svgNS, 'g');
```

```
            this.svg.appendChild(particleGroup);

            const startX = this.speakerX + 5;
            const endX = this.chamberEndX - 5;
            const startY = this.speakerY + 5;
            const endY = this.speakerY + this.speakerHeight - 5;

            const area = (endX - startX) * (endY - startY);
            const density = this.numParticles / area;
            const spacing = Math.sqrt(1 / density);

            const numX = Math.floor((endX - startX) / spacing);
            const numY = Math.floor((endY - startY) / spacing);

            for (let i = 0; i <= numX; i++) {
                for (let j = 0; j <= numY; j++) {
                    if (this.particles.length >= this.numParticles) break;

                    const baseX = startX + i * spacing + (Math.random() - 0.5) * spacing * 0.4;
                    const baseY = startY + j * spacing + (Math.random() - 0.5) * spacing * 0.4;

                    const circle = document.createElementNS(this.svgNS, 'circle');
                    circle.setAttribute('cx', baseX.toFixed(1));
                    circle.setAttribute('cy', baseY.toFixed(1));
                    circle.setAttribute('r', this.particleRadius);
                    circle.setAttribute('fill', '#667eea');
                    circle.setAttribute('opacity', '0.7');
                    particleGroup.appendChild(circle);

                    this.particles.push({ element: circle, baseX, baseY });
                }
            }
        }

        animate() {
            // Calculate angular velocity (change in angle)
            let dTheta = this.zeeman.angle - this.zeeman.previousAngle;
            while (dTheta > Math.PI) dTheta -= 2 * Math.PI;
            while (dTheta < -Math.PI) dTheta += 2 * Math.PI;

            // Target displacement based on velocity
            const targetDisplacement = this.maxDisplacement * Math.tanh(dTheta * this.sensitivityScale);

            // Smooth the displacement
            this.smoothedDisplacement += (targetDisplacement - this.smoothedDisplacement) *
            ↪  this.smoothingFactor;

            // Store in history for wave propagation
            this.history[this.historyIndex] = this.smoothedDisplacement;
            this.historyIndex = (this.historyIndex + 1) % this.history.length;

            // Move piston
            this.pistonRect.setAttribute('x', (this.speakerX - this.speakerWidth +
            ↪  this.smoothedDisplacement).toFixed(2));

            // Calculate piston face position
            const pistonFaceX = this.speakerX + this.smoothedDisplacement;

            // Update particle positions (wave propagation)
            for (const p of this.particles) {
                const distance = p.baseX - this.speakerX;
                const timeDelay = distance / this.waveSpeed;
                let lookBackIndex = Math.round(this.historyIndex - 1 - timeDelay);
                lookBackIndex = (lookBackIndex % this.history.length + this.history.length) %
                ↪  this.history.length;
```

```
507
508                 const historicalDisplacement = this.history[lookBackIndex] || 0;
509                 const damping = Math.exp(-0.003 * distance);
510                 const currentX = p.baseX + historicalDisplacement * damping;
511
512                 // Clamp to stay within chamber
513                 const clampedX = Math.max(
514                     pistonFaceX + this.particleRadius,
515                     Math.min(this.chamberEndX - this.particleRadius, currentX)
516                 );
517                 p.element.setAttribute('cx', clampedX.toFixed(2));
518             }
519
520             // Update previousAngle for next frame
521             this.zeeman.previousAngle = this.zeeman.angle;
522         }
523     }
524
525     // --- INITIALIZATION ---
526     const matrixAutomaton = new MatrixAutomaton();
527     const zeemanMachine = new ZeemanMachine(matrixAutomaton);
528     const acousticViz = new AcousticVisualization(zeemanMachine);
529
530     // Animation loop for acoustic visualization
531     function animateAll() {
532         acousticViz.animate();
533         requestAnimationFrame(animateAll);
534     }
535     animateAll();
536 });
537
```