EigenCalculator:
An eigenvalue and basis calculator

Group 10 (BSCS 2-1)

Buenaventura, Harvey
Feolino, Melvin
James, Crystal
Sangilan, Isaiah
Valeza, Reinwald Marone

Date: January 17, 2025

System Overview

The EigenCalculator is a web-based mathematical application developed to compute and visualize eigenvalues, eigenspaces, and related matrix properties for square matrices of order n≤5. The system is intended to support learning and analysis in Linear Algebra by combining symbolic computation, numerical approximation, and geometric visualization.

The application is implemented using the Streamlit framework, allowing interactive user input and real-time result rendering within a browser environment.The system follows a modular architecture composed of three main components: the backend computation module, the visualization module, and the frontend user interface. This separation of components ensures maintainability, scalability, and clarity in system operation. Data flows sequentially from user input to computation and finally to visualization and presentation.

Backend Computation

User-defined matrix entries are collected through the frontend and converted into a SymPy matrix object for symbolic processing. The backend computes the determinant of the matrix to determine its basic algebraic properties. A symbolic variable is introduced to compute the characteristic polynomial using the determinant expression det(A−λI).

```
import streamlit as st
import math
from sympy import Matrix, symbols, N, eye, factor, Rational
import streamlit.components.v1 as components

# ========================================
# PART 0: DATA STRUCTURE
# ========================================
class EigenObject:
    def __init__(self, eigenvalue, basis, multiplicity):
        # 1. Handle Eigenvalue
        val_n = N(eigenvalue)
        if val_n.is_real:
            self.eigenvalue = float(val_n)
            self.is_complex = False
        else:
            # Keep as string if complex to prevent float conversion errors
            self.eigenvalue = str(val_n).replace('I', 'i')
            self.is_complex = True

        self.multiplicity = multiplicity

        # 2. Handle Basis Vectors
        self.basis = []
        for vec in basis:
            formatted_vec = []
            for v in vec:
                v_n = N(v)
                if v_n.is_real:
                    # Round real numbers for clean display
                    formatted_vec.append(round(float(v_n), 4))
                else:
                    # Keep complex numbers as strings
                    formatted_vec.append(str(v_n).replace('I', 'i'))
            self.basis.append(formatted_vec)

    def to_dict(self):
        return {
            "eigenvalue": self.eigenvalue,
            "multiplicity": self.multiplicity,
            "basis": self.basis,
            "is_complex": self.is_complex
        }
```

The polynomial is also factorized to clearly identify the eigenvalues. Eigenvalues and their algebraic multiplicities are extracted using SymPy’s eigenvalue routines. For each eigenvalue λ, the corresponding eigenspace is computed by determining the null space of the matrix A−λI.

```
# ==========================================
# PART 1: BACKEND LOGIC
# ==========================================
def calculate_eigen_data(matrix_input):
    try:
        # Force exact arithmetic (Rational) to ensure precision with zeros
        exact_matrix_data = []
        for row in matrix_input:
            exact_row = [Rational(str(val)) for val in row]
            exact_matrix_data.append(exact_row)

        matrix = Matrix(exact_matrix_data)
        n = matrix.shape[0]

        lam = symbols('λ')
        char_poly = matrix.charpoly(lam)
        factored_poly = factor(char_poly.as_expr())

        eigenvals_dict = matrix.eigenvals()
        eigen_objects = []
        spectrum = []
        eigenspaces = []

        for lam_val, multiplicity in eigenvals_dict.items():
            # Calculate Nullspace
            eigen_matrix = matrix - lam_val * eye(n)
            nullsp = eigen_matrix.nullspace()

            # Create Object instance
            eig_obj = EigenObject(lam_val, nullsp, multiplicity)
            eigen_objects.append(eig_obj)

            # Add to spectrum list
            spectrum.append(eig_obj.eigenvalue)

            eigenspaces.append(eig_obj.to_dict())
```

The resulting null space vectors form a basis for the eigenspace. All computed values are converted into numerical form where necessary and stored in a structured format for frontend use.

```
return {
            "success": True,
            "determinant": str(N(matrix.det())),
            "characteristic_polynomial": str(char_poly.as_expr()),
            "factored_polynomial": str(factored_poly),
            "eigen_objects": [eo.to_dict() for eo in eigen_objects],
            "spectrum": spectrum,
            "eigenspaces": eigenspaces,
            "matrix_disp": str(matrix)
        }
```

Visualization Module

The visualization module is responsible for generating graphical representations of eigenvectors and eigenspaces. Images are dynamically created using Scalable Vector Graphics (SVG), which allows precise control over geometric elements and ensures resolution-independent rendering within the browser.

```
def generate_eigen_svg(data):
    WIDTH, HEIGHT = 400, 400
    PADDING = 40
    AXIS_COLOR = "#555"
    GRID_COLOR = "#eee"
    LABEL_FONT_SIZE = 14
    DASH_ARRAY = "10,5"
    COLOR_EIGENVALUE = "#3B5BDB"
    COLOR_EIGENBASIS = "#FFA500"
    UNIT_CIRCLE_COLOR = "#aaa"

    def project(vector, scale, origin_x, origin_y):
        # Only project vectors of dimension >= 2
        if len(vector) < 2: return origin_x, origin_y

        try:
            x, y = float(vector[0]), float(vector[1])
            return origin_x + x * scale, origin_y - y * scale
        except:
            return origin_x, origin_y

    def normalize(vector):
        try:
            vec_floats = [float(x) for x in vector]
            norm = math.sqrt(sum(c**2 for c in vec_floats))
            return [c / norm for c in vec_floats] if norm != 0 else vec_floats
        except:
            return vector

    def dashed_arrow(x1, y1, x2, y2, color, dashed=True):
        dash_attr = f'stroke-dasharray="{DASH_ARRAY}"' if dashed else ''
        return f'<line x1="{x1}" y1="{y1}" x2="{x2}" y2="{y2}" stroke="{color}" stroke-width="2"
marker-end="url(#{color.replace("#","")}_arrow)" {dash_attr} />'

    def draw_grid(grid_spacing, width, height):
        lines = ""
        for x in range(0, width + 1, int(grid_spacing)):
            lines += f'<line x1="{x}" y1="0" x2="{x}" y2="{height}" stroke="{GRID_COLOR}" stroke-width="1"/>\n'
        for y in range(0, height + 1, int(grid_spacing)):
            lines += f'<line x1="0" y1="{y}" x2="{width}" y2="{y}" stroke="{GRID_COLOR}" stroke-width="1"/>\n'
        return lines

    def draw_unit_circle(scale, origin_x, origin_y):
        return f'<circle cx="{origin_x}" cy="{origin_y}" r="{scale}" fill="none" stroke="{UNIT_CIRCLE_COLOR}"
stroke-width="2" stroke-dasharray="5,5"/>'
```

A two-dimensional Cartesian coordinate system is constructed, consisting of horizontal and vertical axes, evenly spaced grid lines, and a unit circle centered at the origin. Eigenvectors are normalized and scaled to fit within the fixed visualization area while preserving their direction and relative orientation. Each eigenvector is projected from mathematical coordinates to screen coordinates using linear transformations.

```
    SCALE = (min(WIDTH, HEIGHT)/2 - PADDING)
    ORIGIN_X, ORIGIN_Y = WIDTH // 2, HEIGHT // 2
    GRID_SPACING = SCALE / 2

    vectors_svg = ""
    labels_svg = ""

    has_real_vectors = False

    for obj in data.get("eigen_objects", []):
        # Skip visualization for complex eigenvalues
        if obj.get("is_complex"):
            continue

        eigenvalue = obj["eigenvalue"]
        basis_list = obj.get("basis", [])
        if not basis_list: continue
        try:
            test_val = float(basis_list[0][0])
            has_real_vectors = True
        except:
            continue

        def signed_vector(v):
            return [c * (-1 if eigenvalue < 0 else 1) for c in v]

        first_vec = normalize(signed_vector(basis_list[0]))

        for idx, v in enumerate(basis_list, start=1):
            vec_norm = normalize(signed_vector(v))
            x2, y2 = project(vec_norm, SCALE, ORIGIN_X, ORIGIN_Y)
            vectors_svg += dashed_arrow(ORIGIN_X, ORIGIN_Y, x2, y2, COLOR_EIGENBASIS)

        x2, y2 = project(first_vec, SCALE, ORIGIN_X, ORIGIN_Y)
        vectors_svg += dashed_arrow(ORIGIN_X, ORIGIN_Y, x2, y2, COLOR_EIGENVALUE, dashed=False)
        labels_svg += f'<text x="{x2}" y="{y2 - 10}" font-size="{LABEL_FONT_SIZE}" fill="{COLOR_EIGENVALUE}"
font-weight="bold" text-anchor="middle">λ={eigenvalue:.1f}</text>\n'

    if not has_real_vectors:
        return None
```

Arrows are drawn from the origin to represent eigenvectors, with distinct colors and line styles used to differentiate principal eigenvectors from other eigenspace basis vectors. Text labels indicating eigenvalue magnitudes are placed near the corresponding vectors to enhance interpretability. This image generation process provides users with an intuitive geometric understanding of matrix behavior.

```
arrow_defs = f'''
    <defs>
        <marker id="{COLOR_EIGENBASIS.replace("#","")}_arrow" markerWidth="10" markerHeight="10" refX="0"
refY="3" orient="auto" markerUnits="strokeWidth">
            <path d="M0,0 L0,6 L9,3 z" fill="{COLOR_EIGENBASIS}" />
        </marker>
        <marker id="{COLOR_EIGENVALUE.replace("#","")}_arrow" markerWidth="10" markerHeight="10" refX="0"
refY="3" orient="auto" markerUnits="strokeWidth">
            <path d="M0,0 L0,6 L9,3 z" fill="{COLOR_EIGENVALUE}" />
        </marker>
    </defs>
    '''

    return f'''<svg width="{WIDTH}" height="{HEIGHT}" viewBox="0 0 {WIDTH} {HEIGHT}"
xmlns="http://www.w3.org/2000/svg" style="background-color:white; border-radius:12px;">
      {arrow_defs}
      {draw_grid(GRID_SPACING, WIDTH, HEIGHT)}
      <line x1="0" y1="{ORIGIN_Y}" x2="{WIDTH}" y2="{ORIGIN_Y}" stroke="{AXIS_COLOR}" stroke-width="2"/>
      <line x1="{ORIGIN_X}" y1="0" x2="{ORIGIN_X}" y2="{HEIGHT}" stroke="{AXIS_COLOR}" stroke-width="2"/>
      {draw_unit_circle(SCALE, ORIGIN_X, ORIGIN_Y)}
      {vectors_svg}
      {labels_svg}
    </svg>'''
```

Frontend User Interface

The frontend user interface is developed using Streamlit and serves as the primary interaction point for the user. Users select the matrix size using a slider and enter matrix elements through dynamically generated numerical input fields. Upon initiating the computation, the interface displays a loading indicator while backend calculations are performed.

```
st.set_page_config(page_title="Eigen Calculator", layout="wide", page_icon="📐")

# Custom CSS for styling
st.markdown("""
<style>
/* Global Reset */
.stApp {
    background-color: #F4F6F9;
    color: #212529 !important;
}

/* Hide Unwanted Elements */
.stDeployButton, [data-testid="stToolbar"], [data-testid="stHeader"] {
    display: none !important;
}
section[data-testid="stSidebar"] {
    display: none !important;
}

/* Inputs */
.stNumberInput input {
    background-color: white !important;
    color: black !important;
    border: 1px solid #ddd;
    border-radius: 8px;
}

/* Buttons */
.stButton button {
    background-color: #3B5BDB;
    color: white !important;
    border: none;
    border-radius: 8px;
}

/* Expanders */
div[data-testid="stExpander"] {
    background-color: white !important;
    border: 1px solid #ddd !important;
    border-radius: 8px;
    box-shadow: 0 2px 6px rgba(0,0,0,0.05);
}
div[data-testid="stExpander"] summary {
    color: #212529 !important;
    font-weight: 600;
}
div[data-testid="stExpander"] summary:hover {
    color: #3B5BDB !important;
}
div[data-testid="stExpanderDetails"] {
    color: #212529 !important;
}

/* Tab Styling */
div[data-baseweb="tab-list"] {
    background-color: #212529 !important;
    border-radius: 8px;
    padding: 8px;
    gap: 8px;
}
div[data-baseweb="tab"] {
    color: white !important;
    background-color: transparent !important;
}
div[data-baseweb="tab"] p {
    color: white !important;
}
div[data-baseweb="tab"][aria-selected="true"] {
    background-color: #3B5BDB !important;
    border-radius: 6px;
}
div[data-baseweb="tab"][aria-selected="true"] p {
    color: white !important;
    font-weight: bold;
}

/* Result Card */
.result-card {
    background-color: white;
    padding: 25px;
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.08);
    border-left: 5px solid #3B5BDB;
}
.metric-value {
    font-size: 2rem;
    font-weight: bold;
    color: #3B5BDB;
}
</style>
""", unsafe_allow_html=True)

st.title("Eigen Calculator")
st.markdown("Enter an $n \\times n$ matrix (where $n \leq 5$) to calculate eigenvalues and eigenspace bases.")

col_input, col_result = st.columns([1, 1], gap="large")

with col_input:
    st.markdown("### Matrix Configuration")

    with st.container(border=True):
        n_size = st.slider("Matrix Size (n)", min_value=2, max_value=5, value=2)

        st.markdown(f"**Enter Matrix Elements ({n_size}x{n_size})**")

        matrix_input = []
        cols = st.columns(n_size)
        for i in range(n_size):
            row = []
            for j in range(n_size):
                with cols[j]:
                    val = st.number_input(f"a[{i+1},{j+1}]", value=0.0, step=1.0, key=f"{i}_{j}",
label_visibility="collapsed")
                    row.append(val)
            matrix_input.append(row)

        st.markdown("<br>", unsafe_allow_html=True)
        calc_btn = st.button("Calculate")

        st.markdown("""
        <div style="text-align: center; margin-top: 20px;">
            <p style="color: #666; font-size: 0.9rem; margin-bottom: 10px;">Need help solving? Click the links
below!</p>
            <div style="display: flex; flex-direction: column; gap: 10px;">
                <a href="https://drive.google.com/file/d/1VeY_l_g50bxMhHB8ovb0i9tR87He6hWW/view?usp=sharing"
target="_blank" style="text-decoration: none; color: #3B5BDB; font-weight: bold; font-size: 1.05rem;">
                    📖 Read the Manual Guide
                </a>
                <a href="https://drive.google.com/file/d/1MRF4rxETLYiJbUcTmzUKrDOdIgHJCvT2/view?usp=sharing"
target="_blank" style="text-decoration: none; color: #3B5BDB; font-weight: bold; font-size: 1.05rem;">
                    🧮 Read the Calculator Guide
                </a>
            </div>
        </div>
        """, unsafe_allow_html=True)
```

Once computation is complete, results are displayed in an organized layout that includes the determinant, characteristic polynomial, and factorized polynomial. Tab-based navigation is used to separate spectral analysis and eigenspace information.

```
with col_result:
    if calc_btn:
        with st.spinner("Crunching the numbers..."):
            result = calculate_eigen_data(matrix_input)

            if result["success"]:
                st.markdown(f"""
                <div class="result-card">
                    <div style="color:#666; font-size:0.9rem; text-transform:uppercase;">Determinant</div>
                    <div class="metric-value">{result['determinant']}</div>
                    <hr style="border-top: 1px solid #eee;">
                    <div style="color:#666; font-size:0.9rem; text-transform:uppercase;">Characteristic
Polynomial</div>
                    <div style="font-family:monospace; color:#333;">{result['characteristic_polynomial']}</div>
                    <div style="font-family:monospace; color:#555; font-size:0.9em;">Factored:
{result['factored_polynomial']}</div>
                </div>
                """, unsafe_allow_html=True)

                st.markdown("### Analysis")

                tab1, tab2 = st.tabs(["Spectrum", "Eigenspaces"])

                with tab1:
                    st.write("The set of eigenvalues (spectrum):")
                    clean_spectrum = str(result['spectrum']).replace("'", "")
                    st.latex(f"\\sigma(A) = {clean_spectrum}")

                    svg_html = generate_eigen_svg(result)
                    if svg_html:
                        st.markdown("**2D Projection of Real Eigenvectors:**")
                        components.html(f"<div style='display:flex; justify-content:center;'>{svg_html}</div>",
height=420)
                    else:
                        st.info("No real eigenvectors to graph (Complex or Zero vectors).")

                with tab2:
                    for idx, space in enumerate(result['eigenspaces']):
                        eig_label = space['eigenvalue']
                        if isinstance(eig_label, float):
                            eig_label = f"{eig_label:.4f}"

                        with st.expander(
                                f"λ = {eig_label} (Mult: {space['multiplicity']})",
                                expanded=False
                            ):
                            st.write(f"**Multiplicity:** {space['multiplicity']}")

                            if space['basis']:
                                vectors_text = ", ".join([str(vec) for vec in space['basis']])
                                st.write(f"**Basis Vectors:** {vectors_text}")
                            else:
                                st.write("**Basis Vectors:** None (Zero Nullspace)")

            else:
                st.error(f"Computation Error: {result['error']}")
    else:
        st.markdown("""
        <div style="background-color:white; padding:40px; border-radius:12px; text-align:center; color:#888;
border:1px solid #ddd;">
            <h3>Waiting for Input</h3>
            <p>Set your matrix dimensions and values on the left, then hit Calculate.</p>
        </div>
        """, unsafe_allow_html=True)
```

The generated SVG image is embedded directly into the interface, allowing users to visually examine eigenvectors alongside numerical and symbolic results. Expandable panels are used to present eigenspace bases in a structured and uncluttered manner. Custom styling enhances readability and overall user experience.

LIMITATION AND ASSUMPTIONS

1. Matrix Size: Only supports square matrices from 2×2 to 5×5. Sympy slows down processing matrixes especially at 10x100 matrixes.
2. Numeric Input: Matrix elements must be real numbers; no symbolic or complex entries.
3. Visualization: Eigenvectors are projected onto 2D; higher-dimensional vectors may lose accuracy.
4. Scaling: Displayed vectors are normalized and scaled for visualization only.
5. Multiplicity: Shows available basis vectors; defective matrices may have fewer geometric multiplicities.
6. Errors: Invalid input or singular matrices produce a generic computation error.
7. Frontend: Layout best viewed on standard desktop screens; may not scale well on mobile.
8. Numerical Accuracy: Close or repeated eigenvalues may have minor inaccuracies in eigenvectors or eigenvalues.

FUTURE IMPROVEMENTS

1. Support Larger Matrices: Extend input to handle matrices larger than 5×5 efficiently.
2. Complex & Symbolic Numbers: Allow complex or symbolic entries for advanced computations.
3. 3D/Interactive Visualization: Add 3D plots or interactive vector visualization for higher-dimensional eigenvectors.
4. Enhanced Error Handling: Provide specific feedback for singular or invalid matrices.
5. Mobile-Friendly UI: Optimize layout for small screens and responsive devices.
6. Export Options: Allow users to download eigenvalues, eigenvectors, and plots.

CREDITS

Eigenvalue and Eigenspace Calculator was developed using:

- Python – for the core computation and logic.
- SymPy – for symbolic mathematics, matrix operations, eigenvalues, and eigenvectors.
- Streamlit – for building the interactive web interface.
- SVG & HTML – for custom 2D visualization of eigenvectors and basis vectors.
- CSS Styling – for enhanced UI design and responsive layout along with Streamlit.

TEAM CONTRIBUTIONS:

Reinwald Marone Valeza & Crystal Jane James – Backend development, computation logic and project documentation.

Harvey Buenaventura & Melvin Feolino – Frontend design, connecting backend logic to the interface, and user interaction implementation.

Isaiah Sangilan – Mathematical guide, eigenvalue and eigenvector theory support.

Matrix Configuration

Matrix Size (n)

2

2

DETERMINANT

45.00

CHARACTERISTIC POLYNOMIAL

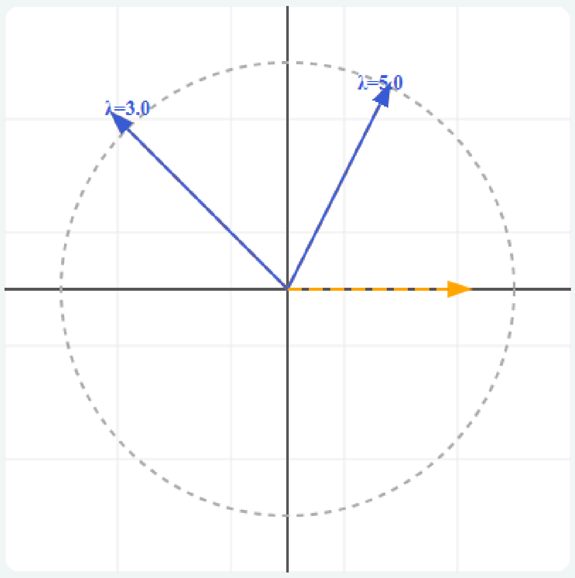
```
λ**3 - 11*λ**2 + 39*λ - 45
Factored: (λ - 5)*(λ - 3)**2
```

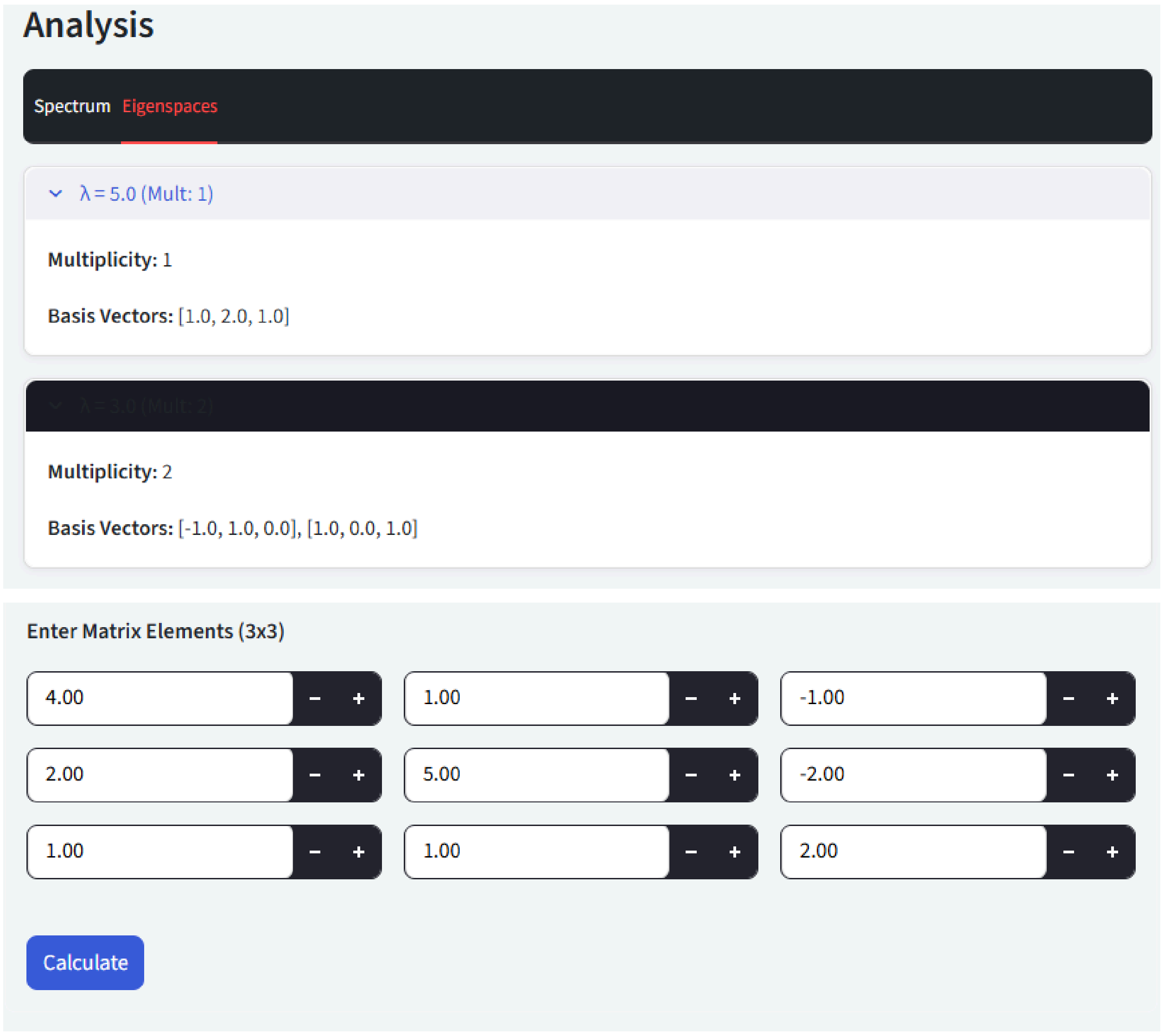
Spectrum Eigenspaces

The set of eigenvalues (spectrum):

$$\sigma(A) = [5.0, 3.0, 3.0]$$

2D Projection of Eigenvectors:





Analysis

λ = 5.0 (Mult: 1)

Multiplicity: 1

Basis Vectors: [1.0, 2.0, 1.0]

λ = 3.0 (Mult: 2)

Multiplicity: 2

Basis Vectors: [-1.0, 1.0, 0.0], [1.0, 0.0, 1.0]

Enter Matrix Elements (3x3)

4.00	1.00	-1.00
2.00	5.00	-2.00
1.00	1.00	2.00