My Python Prototype

```
import numpy as np

# transition matrix: risk flows from column to row
# column-stochastic: each column sums to 1
# example: column 0 means step 1 sends 50% risk to step 2, 50% to step 3
matrix = np.array([[0.0, 0.8, 0.3],
[0.5, 0.0, 0.7],
[0.5, 0.2, 0.0]])

def find_critical_risk(m):
# find eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(m)

# get the principal eigenvector (eigenvalue closest to 1)
idx = np.argmax(eigenvalues.real)
principal_eigenvector = eigenvectors[:, idx].real

# normalize to sum to 1 (interpret as risk distribution)
risk_scores = principal_eigenvector / principal_eigenvector.sum()

return risk_scores

# final_scores shows where risk accumulates at equilibrium
final_scores = find_critical_risk(matrix)
print("risk equilibrium scores:", final_scores)
print("most critical step:", np.argmax(final_scores))
```

I set up each surgical step as a node in a graph. Risk flows between steps through the transition matrix. I use the principal eigenvector to find where risk piles up when everything settles. Those high-risk steps are where I think a robotic system could help the most.