Code Snippets

B.1 Masspoint spring forces

```
  forces(debugMode) {
    //spring energy
    let springDir = p5.Vector.sub(this.origin, this.pos);
    let springForce = springDir.mult(this.stiffness);
    springForce.setMag(min(1, springForce.mag()));
    this.acc.add(springForce);

    // damping force
    let dampDir = this.vel.copy().normalize();
    let dampForce = dampDir.mult(-this.damping);
    dampForce.mult(pow(this.vel.mag(), 1.0)); //up this from 1 for
stronger damping
    this.acc.add(dampForce);

    //debug force
    if (mouseIsPressed && debugMode) {
      this.acc.add(createVector(1, 0));
    }
  }
```

B.2 Self-Intersection check

```
  //this checks whether the sequence a, b, c makes a ccw turn
  ccw(a, b, c) {
    return (c.y - a.y) * (b.x - a.x) > (b.y - a.y) * (c.x - a.x);
  }

  //this checks intersection by telling whether p1 and p2 are on opposite sides of the
line p3-p4, and same for p3 and p4 vs line p1-p2
  segmentsIntersect(p1, p2, p3, p4) {
    return (this.ccw(p1, p3, p4) != this.ccw(p2, p3, p4)) &&
           (this.ccw(p1, p2, p3) != this.ccw(p1, p2, p4));
  }

  //this just checks every non-adjacent pair of line segments for intersections, and
nudges by difference in midpoints if there are intersections
  fixSelfIntersecting(pushStrength, maxIterations) {
    let n = this.ptPos.length;

    for (let iters=0; iters<maxIterations; iters++) {
      let changed = false;
      for (let i=0; i<n; i++) {
        //a1 to a2 is the first line segment
        let a1 = this.ptPos[i];
        let a2 = this.ptPos[(i + 1) % n];

        for (let j=i+2; j<n+i-1; j++) {
          //b1 to b2 is the second line segment
          let b1 = this.ptPos[j%n];
          let b2 = this.ptPos[(j + 1)%n];

          if (this.segmentsIntersect(a1, a2, b1, b2)) {
            //ok theres an intersection, fix by pushing apart midpts
```

B.3 Volume calculator

```
getVolume() {
  this.updatePositions();

  let volume = 0;
  let j = this.ptInd.length-1;
  for (let i=0; i<this.ptPos.length; i++) {
    let pos1 = this.ptPos[i];
    let pos2 = this.ptPos[j];

    volume += (pos2.x + pos1.x) * (pos2.y - pos1.y);
    j = i; //j always trails i by 1
  }
  return round(abs(volume/2), 2);
}
```

B.4 IO Parsing for regions

```
function loadRegions() {
  for(let i=0; i<regionIndices.length; i++) {
    let indexStrings =
regionIndices[i].match(/\d+/g);
    let indexInts = [];
    for (let j=0; j<indexStrings.length; j++) {
      indexInts.push(int(indexStrings[j]));
    }

    let cStr = centers[i].match(/^(-?\d+\.?\d+)\s+(-?
\d+\.?\d+)\s*$/);
    let centerPos = createVector(float(cStr[1]),
float(cStr[2]));

    regions.push(new Region(color('grey'),
indexInts, centerPos));
  }
}
```