1 Dot product with side's left normal

```
(%i1) load("vect");
(%o1) /Applications/Maxima.app/Contents/Resources/maxima/share/maxima/5.36.1/share/vector/vect.mac
```

```
(%i2) x: matrix([1,1]);
```
(%o2) $\begin{bmatrix} 1 & 1 \end{bmatrix}$

```
(%i3) y: matrix([1,1]);
```
(%o3) $\begin{bmatrix} 1 & 1 \end{bmatrix}$

```
(%i4) x . y;
(%o4) 2
```

```
(%i5) y: matrix([-1,1]);
```
(%o5) $\begin{bmatrix} -1 & 1 \end{bmatrix}$

```
(%i6) x.y;
(%o6) 0
```

```
(%i7) x: matrix([x1,x2]);
```
(%o7) $\begin{bmatrix} x1 & x2 \end{bmatrix}$

```
(%i8) y: matrix([y1,y2]);
```
(%o8) $\begin{bmatrix} y1 & y2 \end{bmatrix}$

```
(%i9) x.y;
```
(%o9) $x2\ y2 + x1\ y1$

Compute "left normal" vector to a line segment pt1 --> pt2.
The vector is actually (-deltaY, deltaX).

```
(%i10) deltaY: y2-y1; deltaX: x2-x1;
```
(%o10) $y2 - y1$
(%o11) $x2 - x1$

```
(%i12) leftNormal: matrix([-deltaY,deltaX]);
```
(%o12) $\begin{bmatrix} y1 - y2 & x2 - x1 \end{bmatrix}$

Now compute a vector from pt1 (start of line segment) to ptt = (xt,yt) (the point under test).
This is done by simple (vector) subtraction.

```
(%i13) testVector: matrix([xt-x1,yt-y1]);
```
(%o13) $\begin{bmatrix} xt - x1 & yt - y1 \end{bmatrix}$

Now we let Maxima simplify the dot product.

```
(%i14) leftNormal . testVector;
```
(%o14) $(x2 - x1)(yt - y1) + (xt - x1)(y1 - y2)$

...which is obvious, but I had made the mistake of trying to expand the above expression
by hand and then simplify after cancelling some opposite-sign terms out and factoring.
I think that was doomed to failure.

We see that the totologic blog post has the signs reversed (presumably because
he had the y-axis reversed for typical computer screens).

2 Distance from point to side (segment)

We're going to find an expression for d^2, the distance d of the test point from the triangle side, squared.

```
(%i25) kill (x,y); p1: matrix([x1,y1]); p2: matrix([x2,y2]); p: matrix([x,y]);
```
(%o25) *done*

(%o26) $\begin{bmatrix} x1 & y1 \end{bmatrix}$

(%o27) $\begin{bmatrix} x2 & y2 \end{bmatrix}$

(%o28) $\begin{bmatrix} x & y \end{bmatrix}$

p1p2 \`dot\` pp1 == |p1p2| * |pp1| * cos(theta, the angle between the vectors)

```
(%i31) p1p2: p2-p1;
```
(%o31) $\begin{bmatrix} x2-x1 & y2-y1 \end{bmatrix}$

```
(%i32) pp1: p - p1;
```
(%o32) $\begin{bmatrix} x-x1 & y-y1 \end{bmatrix}$

(Inconsistent naming, but so what.
p1p2 is the vector FROM p1 TO p2.
pp1 is the vector FROM p1 TO p.)

```
(%i34) p1p2 . pp1;
```
(%o34) $(y-y1)(y2-y1)+(x-x1)(x2-x1)$

```
(%i47) pp1;
```
(%o47) $\begin{bmatrix} x-x1 & y-y1 \end{bmatrix}$

```
(%i51) pp1[1][1];
```
(%o51) $x-x1$

Length^2 of pp1, which is the hypotenuse of a right triangle

```
(%i53) hypSqrd: pp1[1][1]^2 + pp1[1][2]^2;
```
(%o53) $(y-y1)^2+(x-x1)^2$

One leg of right triangle, the projection of pp1 onto p1p2, has length squared = (p1p2 . pp1)^2 / |p1p2|^2

```
(%i95) p1p2Sqrd: p1p2[1][1]^2 + p1p2[1][2]^2;
```
(%o95) $(y2-y1)^2+(x2-x1)^2$

```
(%i96) projSqrd: (pp1 . p1p2)^2 / p1p2Sqrd;
```
(%o96) $$\frac{((y-y1)(y2-y1)+(x-x1)(x2-x1))^2}{(y2-y1)^2+(x2-x1)^2}$$

Length, d, of remaining leg of triangle is unknown, BUT Pythagoras tells us
|pp1|^2 - (pp1 . p1p2)^2 / |p1p2|^2 = d^2.

```
(%i97) dSqrd: hypSqrd - projSqrd;
```
(%o97) $$-\frac{((y-y1)(y2-y1)+(x-x1)(x2-x1))^2}{(y2-y1)^2+(x2-x1)^2}+(y-y1)^2+(x-x1)^2$$

If we set up a simple test of a vertical line of length 2 (from (0,-1) to (0,1)), and a test point on the x-axis (y=0) being brought to the y-axis (x -> 0), then the distance of the test point from the line segment should be exactly x. And it is (don't forget we're calculating distance SQUARED):

```
(%i98) ev(dSqrd, x1=0, y1=-1, x2=0, y2=1, y=0);
```
(%o98) x^2

```
(%i104) ev(dSqrd, x1=0, y1=-1, x2=0, y2=1);
```
(%o104) x^2

If our test point is at (1,y) and we're projecting onto a vertical line on the y-axis, it doesn't matter what our x value is, because only the y value determines what the projection will be. And if y is larger than the length of our "triangle side" p1p2, then so be it, the projection will also be large.

```
(%i112) ev( projSqrd, x1=0, y1=0, x2=0, y2=2, x=1);
```

(%o112) y^2

```
(%i111) ev( projSqrd, x1=0, y1=0, x2=0, y2=2, x=1, y=10);
```

(%o111) 100

So, we have to check if the projection is larger than |p1p2|, and, if so, we know that the test vector pp1 is actually LONGER than the triangle side, so we truncate the testing against epsilon^2 by simply computing the distance from the test point p to p2, the other ("far") end of the triangle side.

That's not described in this document, but it is in the code.