

Illustration: Numerische Ungenauigkeit wirkt sich wirklich aus.

Punkte als Liste von Paaren:

```
rawdata = [
    (0,0),(0,1),(0.5,2),(3,-0.2),(3.5,-0.4),
    (3,3),(-2,1.5),(-4,3.5),(-5,0.5),(-1,-2)
]
```

Wir machen draus eine Liste von numpy-Punkten zum Testen:

```
testlist = [npp(p) for p in rawdata]
```

In einem Jupyter-Notebook können wir mit der folgenden Zeile die Laufzeit
empirisch ermitteln:

```
get_ipython().run_line_magic('timeit', 'convex_hull_slow(testlist)')
```

Man erhält ein Ergebnis von folgender Form:

24.9 milliseconds +/- 131 microseconds per loop

(mean +/- std. dev. of 7 runs, 10 loops each)

Die Berücksichtigung numerischer Ungenauigkeiten ist wirklich notwendig:

```
g = PlaneLine(testlist[4],testlist[5])
```

Hier sollte "theoretisch" NULL herauskommen - tut es aber nicht, wegen
numerischer Ungenauigkeit!

```
g.eval_equation(testlist[5])
```

Und tatsächlich funktioniert unser Algorithmus für die test-Liste nicht,
wenn epsilon=0:

```
convex_hull_slow(testlist)
```

liefert [4, 5, 7, 8, 9], aber

```
convex_hull_slow(testlist, epsilon=0)
```

liefert [9, 4]