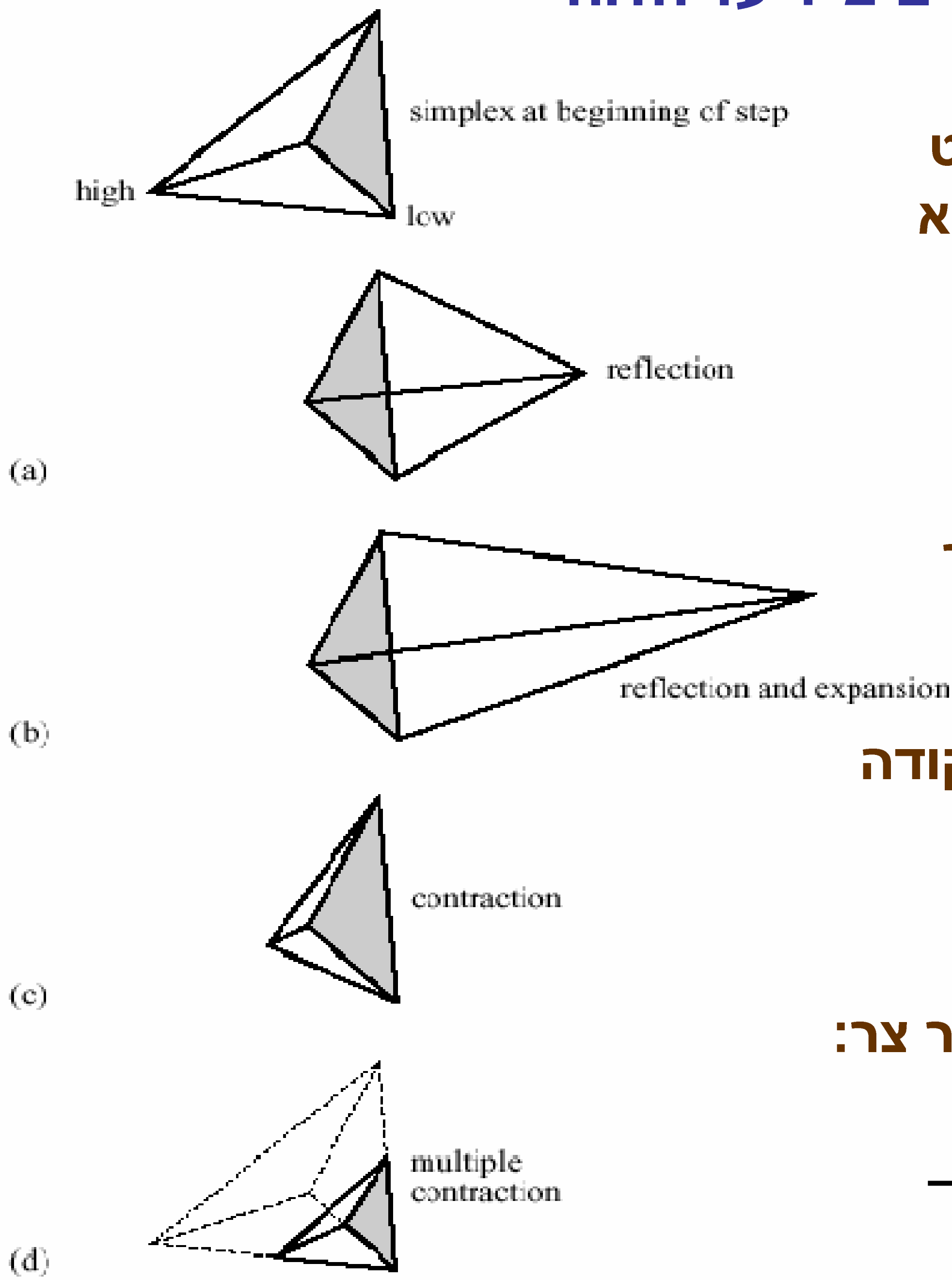
מציאת מינימום

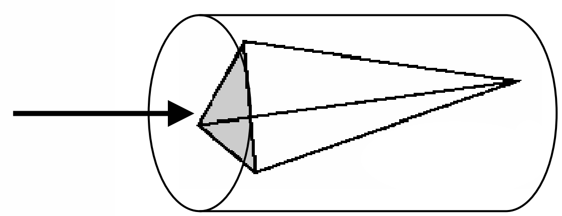
קודם: שקפים 1-2 על הלוח



<u>שיטת Simplex:</u> מנסים לקדם את הסימפלקס לכיוון המינימום. פשוט מחפשים בכל מיני כיוונים, עד שלא מוצאים נקודה שבה f יותר נמוכה ממה שכבר מצאנו.

<u>סוגי צעדים:</u>
השתקפות (reflection) לכיוון הפוך מהנקודה הגבוהה.
אז גם התפשטות (expansion).
התכווצות (contraction) לכיוון הנקודה הנמוכה, בעזרת נקודה אחת או N נקודות בסימפלקס.

למשל, לפעמים הירידה היא באזור צר:



מציאת מינימום

void amoeba(float **p, float y[], int ndim, float ftol, float (*funk)(float []), int *nfunk)

Multidimensional minimization of the function funk(x) where x[1..ndim] is a vector in ndim dimensions, by the downhill simplex method of Nelder and Mead. The matrix p[1..ndim+1] [1..ndim] is input. Its ndim+1 rows are ndim-dimensional vectors which are the vertices of the starting simplex. Also input is the vector y[1..ndim+1], whose components must be preinitialized to the values of funk evaluated at the ndim+1 vertices (rows) of p; and ftol the fractional convergence tolerance to be achieved in the function value (n.b.!). On output, p and y will have been reset to ndim+1 new points all within ftol of a minimum function value, and nfunk gives the number of function evaluations taken.

למשל (N=2) :

```
float funk1(float x[])
{
  return  x[1]*x[1]+(x[2]-2)*(x[2]-2);
}

p[1][1]=0;
p[1][2]=0;
p[2][1]=1;
p[2][2]=0;
p[3][1]=0;
p[3][2]=1;
```

וצריך לחשב גם את y (שקף הבא)

מציאת מינימום

נראה שתי שיטות לחשב את y[1] (וכדומה אפשר גם את y[2] ו- y[3]).

הדרך הפשוטה (בעזרת ווקטור x):

```
x[1]=p[1][1];
x[2]=p[1][2];
y[1]=funk1(x);
```

הדרך הישירה (בעזרת שליחת המצביע המתאים):

```
y[1]=funk1(&p[1][0]);
```

זה עובד, כי הפונקציה funk1 משתמשת רק ב- x[1] ו- x[2]. בשפת ה- C, x[1] זה בעצם *(x+1) , ז"א: הוסף אחד למקום ש- x מצביע אליו, ותראה מה יש שם. אז אם x זה &p[1][0] , אז x[1] זה p[1][1] ו- x[2] זה p[1][2].

<u>הערה כללית:</u> אחרי התכנסות לנקודה P_0 חדשה, כדאי לנסות התחלות מחדש:

$$\mathbf{P}_{\mathrm{i}} = \mathbf{P}_0 + \lambda \mathbf{e}_{\mathrm{i}}$$

ולהריץ שוב (ושוב) עד שאין שיפור. אפשר גם להקטין את λ בכל פעם.

קודם: שקפים 3-7 על הלוח

שיטות של כיוונים מצומדים Conjugate direction methods

u ו-v הם ווקטורים מצומדים אם:

$$0 = \mathbf{u} \cdot \delta(\nabla f) = \mathbf{u} \cdot \mathbf{A} \cdot \mathbf{v}$$

void powell(float p[], float **xi, int n, float ftol, int *iter, float *fret, float (*func)(float []))

Minimization of a function func of n variables. Input consists of an initial starting point p[1..n]; an initial matrix xi[1..n][1..n], whose columns contain the initial set of directions (usually the n unit vectors); and ftol, the fractional tolerance in the function value such that failure to decrease by more than this amount on one iteration signals doneness. On output, p is set to the best point found, xi is the then-current direction set, fret is the returned function value at p, and iter is the number of iterations taken. The routine linmin is used.

brent

למשל (N=2) :

```
p[1]=0;
p[2]=0;
xi[1][1]=1;
xi[2][1]=0;
xi[1][2]=0;
xi[2][2]=1;
```

כמו קודם, כדאי לנסות התחלות מחדש.

מציאת מינימום : סיכום

הבעיה: למצוא את המינימום של f(x) או של f(x,y,z,w) .

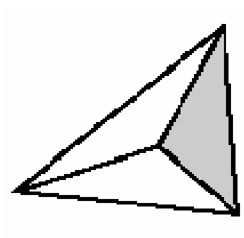
במימד 1: תוחמים בעזרת 3 נקודות.

שיטה 1: שיטת חיתוך הזהב: חלוקה לא שווה. **golden**

התכנסות ליניארית: $\epsilon_{n+1} = \epsilon_n \times 0.61803$

שיטה 2: שיטת ברנט: אינטרפולציה פרבולית **brent**

רב-מימדי:



שיטה 3: שיטת סימפלקס/אמבה

amoeba

שיטה 4: שיטות של כיוונים מצומדים (מינימיזציות חד-מימדיות)

powell

$$0 = \mathbf{u} \cdot \delta(\nabla f) = \mathbf{u} \cdot \mathbf{A} \cdot \mathbf{v}$$