Algorithm B.1 The Simplex algorithm.

Note that one extra column is assumed to have been allocated, for x_{m+n}.

```
struct simplex_t {
    int       m;            /* Constraints. */
    int       n;            /* Decision variables. */
    int       var[n+m];     /* 0..n − 1 are nonbasic. */
    double    a[m][n+1];    /* A. */
    double    b[m];         /* b. */
    double    x[n+1];       /* x. */
    double    c[n];         /* c. */
    double    y;            /* y. */
}
```

```
procedure init(s, m, n, a, b, x, y, var)
begin
    int i,k
    *s = (m,n,a,b,x,y,var) // assign each attribute
    if s.var = null then
        s.var = new int [m+n+1]
        for (i = 0; i < m+n; i = i + 1)
            s.var[i] = i
    for (k = 0, i = 1; i < m; i = i + 1)
        if b[i] < b[k] then
            k = i
    return k
end
```

```
function select_nonbasic(s)
begin
    int i
    for (i = 0; i < s.n; i = i + 1)
        if s.c[i] > ϵ then
            return i
    return -1
end
```

```
procedure prepare(s, k)
begin
    int m = s.m
    int n = s.n
    // make room for x_{m+n} at s.var[n] by moving s.var[n..n+m-1] one
    // step to the right.
    for (i = m + n; i > n; i = i − 1)
        s.var[i] = s.var[i − 1]
    s.var[n] = m + n
    // add x_{m+n} to each constraint
    n = n + 1
    for (i = 0; i > m; i = i + 1)
        s.a[i][n − 1] ← −1
    s.x = new double [m + n]
    s.c = new double [n]
    s.c[n − 1] = −1
    s.n = n
    pivot(s, k, n-1)
end
```

```
function initial(s, m, n, a, b, x, y, var)
begin
    int i,j,k
    double w
    k = init(s, m, n, a, b, x, y, var)
    if b[k] ≥ 0 then
        return 1 // feasible
    prepare(s,k)
    n = s.n
    s.y = xsimplex(m, n, s.a, s.b, s.c, s.x, 0, s.var,1)
    for (i = 0; i < m+n; i = i + 1) {
        if s.var[i] = m+n then
            if |s.x[i]| > ϵ then
                delete s.x
                delete s.c
                return 0 // infeasible
    }

    // The rest of this function is on the next page.
```

```
    if i ≥ n then
        // x_{n+m} is basic. find good nonbasic.
        for (j = k = 0; k < n; k = k + 1)
            if |s.a[i−n][k]| > |s.a[i−n][j]| then
                j = k
        pivot(s,i-n,j)
        i = j
    if i < n-1 then
        // x_{n+m} is nonbasic and not last. swap columns i and n-1
        k = s.var[i]; s.var[i] = s.var[n-1]; s.var[n-1] = k
        for (k = 0; k < m; k = k + 1)
            w = s.a[k][n-1]; s.a[k][n-1] = s.a[k][i]; s.a[k][i] = w
    else
        // x_{n+m} is nonbasic and last. forget it.
    delete s.c
    s.c = c
    s.y = y
    for (k = n-1; k < n+m-1; k = k + 1)
        s.var[k] = s.var[k+1]
    n = s.n = s.n - 1
    t = new double [n]
    for (k = 0; k < n; k = k + 1) {
        for (j = 0; j < n; j = j + 1)
            if k = s.var[j] then
                // x_k is nonbasic. add c_k
                t[j] = t[j] + s.c[k]
                goto next_k
        // x_k is basic.
        for (j = 0; j < m; j = j + 1)
            if s.var[n+j] = k then
                // x_k is at row j
                break
        s.y = s.y + s.c[k] * s.b[j]
        for (i = 0; i < n; i = i + 1)
            t[i] = t[i] - s.c[k] * s.a[j][i]
    next_k:;
    }
    for (i = 0; i < n; i = i + 1)
        s.c[i] = t[i]
    delete t and s.x
    return 1
end
```

```
function pivot(s, row, col)
begin
    auto a = s.a
    auto b = s.b
    auto c = s.c
    int m = s.m
    int n = s.n
    int i,j,t
    t = s.var[col]
    s.var[col] = s.var[n+row]
    s.var[n+row] = t
    s.y = s.y + c[col] * b[row] / a[row][col]
    for (i = 0; i < n; i = i + 1)
        if i ≠ col then
            c[i] = c[i] - c[col] * a[row][i] / a[row][col]
    c[col] = - c[col] / a[row][col]
    for (i = 0; i < m; i = i + 1)
        if i ≠ row then
            b[i] = b[i] - a[i][col] * b[row] / a[row][col]
    for (i = 0; i < m; i = i + 1)
        if i ≠ row then
            for (j = 0; j < n; j = j + 1)
                if j ≠ col then
                    a[i][j] = a[i][j] - a[i][col] * a[row][j] / a[row][col]
    for (i = 0; i < m; i = i + 1)
        if i ≠ row then
            a[i][col] = -a[i][col] / a[row][col]
    for (i = 0; i < n; i = i + 1)
        if i ≠ col then
            a[row][i] = a[row][i] / a[row][col]
    b[row] = b[row] / a[row][col]
    a[row][col] = 1 / a[row][col]
end
```

```
function xsimplex(m, n, a, b, c, x, y, var, h)
begin
    simplex_t s
    if !initial(&s, m, n, a, b, c, x, y, var) then
        delete s.var
        return NaN // not a number
    while (col ← select_nonbasic(&s)) ≥ 0) {
        row ← -1
        for (i = 0; i < m; i = i + 1)
            if a[i][col] > ϵ  and
                (row < 0  or b[i] / a[i][col] < b[row] / a[row][col]) then
                row = i
        if row < 0 then
            delete s.var
            return ∞ // unbounded
        pivot(s,row, col)
    }
    if h = 0 then
        for (i = 0; i < n; i = i + 1)
            if s.var[i] < n then
                x[s.var[i]] = 0
        for (i = 0; i < m; i = i + 1)
            if s.var[n+i] < n then
                x[s.var[n+i]] = s.b[i]
        delete s.var
    else
        for (i = 0; i < n; i = i + 1)
            x[s.var[i]] = 0
        for (i = n; i < n+m; i = i + 1)
            x[s.var[i]] = s.b[i-n]
    return s.y
end
```

```
function simplex(m, n, a, b, c, x, y)
begin
    return xsimplex(m,n,a,b,c,x,y,null,0)
end
```