```
open List
exception Not_a_matrix
exception Dimension_mismatch
exception No_solutions
exception Infinite_solutions

let matvalid mat =
    let l = length (hd mat) in
    for_all (fun a -> if length a = l then true else false) mat;;

let checkDimension mat b =
    (matvalid mat) && (length mat = length b);;

let rec swap_helper mat i j r =
        if r = length mat then []
        else if r = i then nth mat j::swap_helper mat i j (r+1)
        else if r = j  then nth mat i::swap_helper mat i j (r+1)
        else nth mat r::swap_helper mat i j (r+1)

let swap mat i j = swap_helper mat i j 0;;

let rec mult_helper mat i c r =
        match mat with
        [] -> []
        |hd::tl -> if i = r then (map (fun x -> x *. c) hd)::tl
                           else hd::(mult_helper tl i c (r+1));;

let mult mat i c = mult_helper mat i c 0;;

let rec add_rows_helper mat i j r =
        if r = length mat then []
        else if r = i then (map2 (fun x y -> x +. y) (nth mat i) (nth mat
j))::add_rows_helper mat i j (r+1)
        else nth mat r::add_rows_helper mat i j (r+1);;

let addRows mat i j = add_rows_helper mat i j 0;;

let epsilon = 10. ** (-10.)

let float_zero x = abs_float(x) <= epsilon

let rec dimension l =
        match l with
        | [] -> 0
        | hd::tl -> 1 + (dimension tl)

let rec firstNonZeroIndex l i =
        match l with
        | [] -> max_int
        | hd::tl -> if (not (float_zero hd)) then i
                    else (firstNonZeroIndex tl (i+1))

let rec firstNonZeroColumn mat =
        match mat with
        | [] -> max_int
        | hd::tl ->  let z = (firstNonZeroIndex hd 0) in
        let k = (firstNonZeroColumn tl) in
                if z < k then z else k

let rec hasNonZeroIndex l i =
        if (i == 0) then let k = (hd l) in (not (float_zero k))
        else hasNonZeroIndex (tl l) (i-1)

let rec firstRowWithLead mat i rowNum=
        match mat with
        | [] -> max_int
        | hd::tl -> if (hasNonZeroIndex hd i) then rowNum
                    else (firstRowWithLead tl i (rowNum+1))
```

```
let rec normalizeH mat c j =
        if (j == 0) then mat
        else
                let mat0c = (nth (nth mat 0) c) in
                let matjc = (nth (nth mat j) c) in
                let x = ((-1. *. matjc)/. mat0c) in
                let z = (mult mat 0 x) in
                let f = (addRows z j 0) in
                normalizeH ((hd mat)::(tl f)) c (j-1)

let normalize mat c =
        match mat with
        | [] -> []
        | row1::rest -> normalizeH mat c ((length mat)-1)

let rec rowEchelon mat =
        match mat with
        | [] -> []
        | _ ->
        let c = firstNonZeroColumn mat in
                if c == max_int then mat
                else
                        let r = firstRowWithLead mat c 0 in
                        let mat1 = swap mat 0 r in
                        let mat2 = normalize mat1 c in
                        (hd mat2)::(rowEchelon (tl mat2))

let rec numSolutionsH mat c maxCol =
        match mat with
        | [] -> if ((c+1) == maxCol) then 1 else max_int
        | hd::tl -> let z = (firstNonZeroIndex hd 0) in
                    if (c == z)
                    then (numSolutionsH tl (c+1) maxCol)
                    else if ((length hd) == (z + 1)) then 0
                    else max_int

let numSolutions mat = numSolutionsH (rowEchelon mat) 0 (dimension (hd mat))

let rec mySum l =
        match l with
        | [] -> 0.
        | h::t -> h +. (mySum t)

let rec solveEqnH a row currSol currSum =
        match currSol with
        | [] -> ((mySum row) +. currSum)/. a
        | h::t -> (solveEqnH a (tl row) t (h *. (hd row) *. (-1.) +. currSum))

let rec solveEqn row currSol =
        let z = hd row in
        if (not (float_zero z)) then solveEqnH (hd row) (tl row) currSol 0.
        else solveEqn (tl row) currSol


let rec solveRowEchelonH mat currSol =
        match mat with
        | [] -> currSol
        | row1::rest -> solveRowEchelonH (rest) ((solveEqn row1 currSol)::currSol)

let solveRowEchelon mat =
        let rmat = rev mat in
        solveRowEchelonH rmat []

let rec createMat mat b =
        match mat with
        | [] -> []
        | h::t -> (h @ [(hd b)])::(createMat t (tl b))
```

```
let solve mat b =
        if (checkDimension mat b) == false then raise Dimension_mismatch
        else
                let mat1 = createMat mat b in
                let n = numSolutions mat1 in
                if n == 0 then raise No_solutions
                else if n == max_int then raise Infinite_solutions
                else solveRowEchelon (rowEchelon mat1)
```