Optimization using GNU Octave

Technical Report · April 2015

DOI: 10.13140/RG.2.1.4586.7043

1 author:

M. P. Gururajan
Indian Institute of Technology Bombay
55 PUBLICATIONS **237** CITATIONS

SEE PROFILE

Optimization using GNU Octave

M. P. Gururajan

January-May, 2015

`GNU Octave` for optimization

`GNU Octave` is a high-level interactive language; it is a free software and is used primarily for numerical computations and plotting.

In this tutorial, we show examples of optimization problems solved using `GNU Octave`. The problems solved here are taken from the optimization textbook of S S Rao [1] (unless stated otherwise).

1 Linear programming

The function `glpk` is used to solve Linear Programming (LP) problems of the type

min $\mathbf{c}^T\mathbf{x}$

subject to the linear constraints

$\mathbf{Ax} = \mathbf{b}$ where $x \geq 0$ and its variations, namely, maximize instead of minimize, and inequality constraints or mixed constraints instead of equality constraints.

We now show the octave scripts that use the function `glpk` to solve some LP problems.

1.1 Problem 1

This is the solved problem 3.4 in [1].

Maximize $F = x_1 + 2x_2 + x_3$

subject to

$2x_1 + x_2 - x_3 \leq 2$

$-2x_1 + x_2 - 5x_3 \geq -6$

$4x_1 + x_2 + x_3 \leq 6$

$x_i \geq 0$, $i = 1, 2, 3$

The following script solves the problem.

```
# This is the octave script to solve the (solved) problem 3.4 of S S Rao.
```

```
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [1,2,1]';
# Matrix of constraint coefficients
A = [2,1,-1; -2,1,-5;4,1,1];
# The right hand side of constraints
b = [2,-6,6]';
# Lower bound for the design variables
lb = [0,0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: U and L indicate inequalities with upper  and lower bound
# respectively
ctype = "ULU"
# Variable type: C is for continuous
vartype = "CCC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = -1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution
```

```
xmin

fmin

status

extra
```

This problem has an optimal solution and the solution returned by `GNU Octave` is the same as that obtained using calculation by hand.

```
...

OPTIMAL LP SOLUTION FOUND

...

xmin =

   0
   4
   2

fmin =  10
```

1.2 Problem 2

This is the solved problem 3.5 in [1].

Minimize $f = -3x_1 - 2x_2$

subject to

$2x_1 - x_2 \leq 1$

$3x_1 - 2x_2 \leq 6$

$x_i \geq 0$, $i = 1, 2$

The following script solves the problem.

```
# This is the octave script to solve the (solved) problem 3.5 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [-3,-2]';
# Matrix of constraint coefficients
A = [1,-1; 3,-2];
# The right hand side of constraints
b = [1,6]';
# Lower bound for the design variables
lb = [0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: U indicates that these are inequalities with upper bound
ctype = "UU"
# Variable type: C is for continuous
vartype = "CC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = 1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;

# The third part: Optimization
```

```
# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

This problem has an unbounded solution and `GNU Octave` returns the same information.

```
...
LP HAS UNBOUNDED PRIMAL SOLUTION
...
```

1.3 Problem 3

This is the solved problem 3.6 in [1].

Minimize $f = -40x_1 - 100x_2$

subject to

$10x_1 + 5x_2 \leq 2500$

$4x_1 + 10x_2 \leq 2000$

$2x_1 + 3x_2 \leq 900$

$x_i \geq 0$, $i = 1, 2$

The following script solves the problem.

```
# This is the octave script to solve the (solved) problem 3.6 of S S Rao.
# Preamble
```

```
clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [-40,-100]';
# Matrix of constraint coefficients
A = [10,5; 4,10; 2,3];
# The right hand side of constraints
b = [2500,2000,900]';
# Lower bound for the design variables
lb = [0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: U indicates that these are inequalities with upper bound
ctype = "UUU"
# Variable type: C is for continuous
vartype = "CC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = 1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin
```

```
fmin

status

extra
```

As shown in [1], this problem has infinite number of solutions. However, `GNU Octave` only gives one of the solutions and does not indicate that there are infinite number of solutions.

```
OPTIMAL LP SOLUTION FOUND
ctype = UUU
xmin =

     0
   200

fmin = -20000
status = 0
```

1.4 Two-phase primal simplex method

Even though it had not been mentioned explicitly, `gplk`, when invoked, carries out two-phase primal simplex method. As an example, consider the solved example 3.7 in [1].

Minimize $f = 2x_1 + 3x_2 + 2x_3 - x_4 + x_5$

subject to the constraints

$3x_1 - 3x_2 + 4x_3 + 2x_4 - x_5 = 0$

$x_1 + x_2 + x_3 + 3x_4 + x_5 = 2$

$x_i \geq 0$, $i = 1, 2, 3, 4, 5$

The following script utilises two phase primal simplex method to solve the same.

```
# This is the octave script to solve the (solved) problem 3.7 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [2,3,2,-1,1]';
# Matrix of constraint coefficients
A = [3,-3,4,2,-1; 1,1,1,3,1];
# The right hand side of constraints
b = [0,2]';
# Lower bound for the design variables
lb = [0,0,0,0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: S indicates equality constraint
ctype = "SS"
# Variable type: C is for continuous
vartype = "CCCCC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = 1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;
# The method to be used; 1 stands for two phase primal simplex (and
# is the default)
param.dual = 1;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
```

```
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

The solution returned by `GNU Octave` is the same as that obtained using calculation by hand.

```
...
OPTIMAL LP SOLUTION FOUND
ctype = SS
xmin =

   0.00000
   0.00000
   0.00000
   0.40000
   0.80000

fmin =  0.40000
status = 0
...
```

1.5 Dual simplex method

Consider the following solved problem (4.1) in [1].

Minimize $f = 20x_1 + 16x_2$

subject to

$x_1 \geq 2.5$

$x_2 \geq 6$

$2x_1 + x_2 \geq 17$

$x_1 + x_2 \geq 12$

$x_i \geq 0,\ i = 1, 2$

This problem can be solved using the dual simplex method as follows.

```
# This is the octave script to solve the (solved) problem 4.3 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [20,16]';
# Matrix of constraint coefficients
A = [1,0; 0,1; 2,1; 1,1];
# The right hand side of constraints
b = [2.5,6,17,12]';
# Lower bound for the design variables
lb = [0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: S indicates equality constraint
ctype = "LLLL"
# Variable type: C is for continuous
vartype = "CC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = 1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
```

```
# The limit on number of simplex iterations
param.itlim = 10;
# The simplex method to be used; 3 stands for two phase dual
# simplex. It is also possible to use 2; then
# two phase dual simplex is used; if it fails,
# the program goes back to dual phase
# primal simplex
param.dual = 3;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

The script returns the correct answer:

```
...
OPTIMAL LP SOLUTION FOUND
ctype = LLLL
xmin =

   5
   7

fmin =  212
status = 0
...
```

2 Integer programming

Consider the following problem:

Maximize $f(\mathbf{X}) = 5x_1 + 6x_2 + 3x_3 + 2x_4 + 8x_5$

subject to the following constraints:

$4x_1 + 8x_2 + 2x_3 + 5x_4 + 3x_5 \leq 2000$

$9x_1 + 7x_2 + 4x_3 + 3x_4 + 8x_5 \leq 2500$

$x_i \geq 0$, $i = 1, 2, 3, 4, 5$

Here, it is known that the variables x_i represent an article of type i; the cost function is the cargo load consisting of the 5 different types of articles which needs to be maximized – see the solved problem 1.7 from [1]. Hence, in this optimization problem, the variables are to be constrained to be integers. The following script does that:

```
# This is the octave script to solve the (solved) problem 1.7 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [5,6,3,2,8]';
# Matrix of constraint coefficients
A = [4,8,2,5,3; 9,7,4,3,8];
# The right hand side of constraints
b = [2000,2500]';
# Lower bound for the design variables
lb = [0,0,0,0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: U indicates inequality constraint with upper bound
ctype = "UU"
# Variable type: I is for integer type
vartype = "IIIII";
```

```
# Sense: if 1, it is minimization; if -1, it is maximization
s = -1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;
# The simplex method to be used; 3 stands for two phase dual
# simplex. It is also possible to use 2; then
# two phase dual simplex is used; if it fails,
# the program goes back to dual phase
# primal simplex
param.dual = 1;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

There is a optimum solution for this LP problem and `GNU Octave` returns the solution:

```
...
INTEGER OPTIMAL SOLUTION FOUND
ctype = UU
```

```
xmin =

     0
     0
     1
     0
   312

fmin =  2499
...
```

3 Interior point method

Consider the following optimization problem:

Minimize $f = 2x_1 + x_2 - x_3$

subject to

$x_2 - x_3 = 0$

$x_1 + x_2 + x_3 = 1$

$x_i \geq 0$, $i = 1, 2, 3$

This problem can be solved using Karmakar's interior point method – see the solved problem 4.13 of [1]. Here is the `GNU Octave` script to solve this porblem:

```
# This is the octave script to solve the (solved) problem 4.13 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [2,1,-1]';
# Matrix of constraint coefficients
```

```
A = [0,1,-1; 1,1,1];
# The right hand side of constraints
b = [0,1]';
# Lower bound for the design variables
lb = [0,0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: S indicates equality constraint
ctype = "SS"
# Variable type: C is for continuous type
vartype = "CCC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = 1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 100;
# The default lpsolver is the revised simplex method (=1); however,
# for the interior point method, this variable takes a value of 2.
param.lpsolver = 2;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

The solution obtained using the given script is as follows:

```
...
Guessing initial point...
Optimization begins...
  0: obj =   5.500000000e+00; rpi =  3.6e+00; rdi =  7.5e-01; gap =  7.4e-01
  1: obj =   7.582591964e-01; rpi =  3.6e-01; rdi =  7.5e-02; gap =  6.9e-01
  2: obj =   1.324533094e-01; rpi =  3.6e-02; rdi =  7.5e-03; gap =  2.0e-01
  3: obj =   1.360289259e-02; rpi =  3.6e-03; rdi =  7.7e-04; gap =  2.2e-02
  4: obj =   1.360646460e-03; rpi =  3.6e-04; rdi =  7.7e-05; gap =  2.2e-03
  5: obj =   1.360650025e-04; rpi =  3.6e-05; rdi =  7.7e-06; gap =  2.2e-04
  6: obj =   1.360650061e-05; rpi =  3.6e-06; rdi =  7.7e-07; gap =  2.2e-05
  7: obj =   1.360650061e-06; rpi =  3.6e-07; rdi =  7.7e-08; gap =  2.2e-06
  8: obj =   1.360650061e-07; rpi =  3.6e-08; rdi =  7.7e-09; gap =  2.2e-07
  9: obj =   1.360650061e-08; rpi =  3.6e-09; rdi =  7.7e-10; gap =  2.2e-08
 10: obj =   1.360650061e-09; rpi =  3.6e-10; rdi =  7.7e-11; gap =  2.2e-09
OPTIMAL SOLUTION FOUND
ctype = SS
xmin =

   6.8033e-10
   5.0000e-01
   5.0000e-01

fmin =    1.3607e-09
...
```

4 Quadratic programming

Consider the following optimization problem: Minimize $f = -4x_1 + x_1^2 - 2x_1x_2 + 2x_2^2$

subject to

$2x_1 + x_2 \leq 6$

$x_1 - 4x_2 \leq 0$

$x_1, x_2 \geq 0$

This is a Quadratic Programming (QP) problem – see the solved problem 4.14 of [1].

This problem can be stated in the following fashion:

Minimize $f = 0.5x' \star H \star x + x' \star q$

subject to $A \star x \geq b$ and $x \geq 0$.

In this problem, this gives, $H = (2, -2; -2, 4)$; $q = (-4, 0)$; $A = (2, 1; 1, -4)$; and $b = (6, 0)$.

In `GNU Octave`, the command for solving the quadratic programming problem is `qp` and it solves the following problem:

Minimize $0.5x^T Hx + x^T q$

subject to

$Ax = b$

$lb \leq x \leq ub$

$A_{lb} \leq A_{in}x \leq A_{ub}$

Thus the given QP can be solved using the following `GNU Octave` script.

```
# This is the octave script to solve the (solved) problem 4.14 of S S Rao.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
H = [2,-2; -2, 4];
q = [-4,0]';
# Constraints
A = [];
b = [];
# Matrix of constraint coefficients
A_in = [2,1; 1,-4];
# The right hand side of constraints
A_ub = [6,0]';
```

```
A_lb = [];
# Lower bound for the design variables
lb = [0,0]';
# Upper bound for the design variables
ub = [];
# Initial point
x0 =[];

# The second part: Optimization

# qp function is the one that solves the Quadratic Programming (QP) problem.
[xmin, fmin, info, lambda] = qp(x0,H,q,A,b,lb,ub,A_lb,A_in,A_ub)
```

The solution returned by the script is as follows:

```
xmin =

   2.4615
   1.0769

fmin = -6.7692
info =

  scalar structure containing the fields:

    solveiter =  6
    info = 0

lambda =

   0.00000
   0.00000
   0.61538
   0.00000
```

5 Nonlinear programming: one-dimensional minimization methods

There are many numerical methods of finding the zeros of a given non-linear polynomial. In this section, we give some GNU Octave scripts that implement elimination, interpolation and direct root methods.

5.1 Elimination methods

Consider the function $f(\lambda) = \lambda(\lambda - 1.5)$. We want to find the $\lambda^\star$ which minimizes the function $f(\lambda)$. Here are the scripts that will solve the problem using unrestricted search, unrestricted search with accelerated step size, exhaustive search, dichotomous search, and interval halving method. All these are elimination methods.

5.1.1 Unrestricted search

```
function y = f(x)
y = x*(x-1.5);
endfunction

x = input ("Give the starting point: ");
ss = input ("Give the initial step-size: ");

if(f(x+ss) > f(x))
ss = -ss;
endif

while (f(x)-f(x+ss) > 1e-8)
x = x+ss;
endwhile

disp("The minimum is bracketed between: "), disp(x-ss)
disp("and: "), disp(x+ss)
disp("The f(x) values at these two points are: "),disp(f(x-ss))
disp("and: "),disp(f(x+ss))
```

```
disp("The final step-size is: "),disp(ss)
```

5.1.2 Unrestricted search with accelerated step size

```
function y = f(x)
y = x*(x-1.5);
endfunction

x = input ("Give the starting point: ");
ss = input ("Give the initial step-size: ");
if(f(x+ss) > f(x))
ss = -ss;
endif
while (f(x)-f(x+ss) > 1e-8)
x = x+ss;
ss = 2*ss;
endwhile
disp("The minimum is bracketed between: "), disp(x-0.5*ss)
disp("and: "), disp(x+ss)
disp("The f(x) values at these two points are: "),disp(f(x-0.5*ss))
disp("and: "),disp(f(x+ss))
disp("The final step-size is: "),disp(ss)
```

5.1.3 Exhaustive search

```
function y = f(x)
y = x*(x-1.5);
endfunction

x0 = input ("Give the starting point: ");
x1 = input ("Give the final point: ");
L = x1-x0;
epsilon = input ("Give the required accuracy: ");

n = (2*L/epsilon) -1.;
ss = L/n;
```

```
x(1) = x0;
F(1) = f(x0);
for i=1:n
x(i+1) = x0+i*ss;
endfor

for i=1:n
F(i+1) = f(x(i+1));
endfor

for i=1:n
if ( (F(i+1) - F(i)) > 0)
disp("The interval lies between "),disp(x(i))
disp("and "),disp(x(i+1))
disp("The function values are "),disp(f(x(i)))
disp("and "),disp(f(x(i+1)))
break
endif
endfor

plot(x,F,"s")
```

5.1.4 Dichotomous search

```
function y = f(x)
y = x*(x-1.5);
endfunction

x0 = input ("Give the starting point: ");
x1 = input ("Give the final point: ");
L = x1-x0;
epsilon = input ("Give the required accuracy: ");
delta = 0.01*epsilon;
a = delta/L;

n = int8(2*log2((1-a)/(2.*epsilon-a)));
if(mod(n,2)==1) n = n+1;
```

```
else n;
endif

disp("The number of iterations: "),disp(n)

for i=1:n
L = x1 - x0;
a = x0+0.5*L - 0.5*delta;
b = x0+0.5*L + 0.5*delta;
if (f(a) < f(b))
x1 = a;
else
x0 = a;
endif
endfor

disp("The optimum point: "),disp(0.5*(x0+x1))
disp("The function value at optimum point: ")
disp(0.5*(f(x0)+f(x1)))
```

5.1.5 Interval halving method

```
function y = f(x)
y = x*(x-1.5);
endfunction

x0 = input ("Give the starting point: ");
x1 = input ("Give the final point: ");
epsilon = input ("Give the required accuracy: ");
a = 2*epsilon;
L = x1-x0;

n = int8(2*(1.0+log2((L/a))));
if(mod(n,2)==0) n = n+1;
else n;
endif
```

```
disp("The number of iterations: "),disp(n)

for i=1:n
L = x1-x0;
a = x0+0.25*L;
b = x0+0.5*L;
c = x0+0.75*L;
if ( (f(c)>f(b)) && (f(b)>f(a)))
x1 = b;
elseif( (f(c)<f(b)) &&  (f(b)<f(a)))
x0 = b;
else
x0 = a;
x1 = c;
endif
endfor

disp("The optimum point: "),disp(0.5*(x0+x1))
disp("The function value at optimum point: ")
disp(0.5*(f(x0)+f(x1)))
```

5.1.6 `fminbnd` and golden section method

To find a minimum point of a univariate function.

The GNU Octave function `fminbnd` internally uses a Golden Section search strategy and is used to find the minimum point of a univariate function by restricting the search to an interval bound by, say a and b. The syntax for calling this function is as follows: `fminbnd (fun, a, b, options)`. Here is an example.

```
f = inline( "x*(x-1.5)")
fminbnd(f,0,1,optimset("TolX",0.1))
```

5.2 Interpolation methods

In interpolation methods, the function for which minima is to be found is approximated by quadratic or cubic polynomials near the minima in such a way that the minima of the approximating polynomial coincide with those of the function.

5.2.1 Quadratic interpolation method

Here is the script that identifies the zeros of the function $f(\lambda) = \lambda^5 - 5\lambda^3 - 20\lambda + 5$ using the quadratic interpolation method. Note that the script also plots the approximating function as the iterations proceed (for the sake of understanding the method better).

```
function y = f(x)
y = x^5 - 5* x^3 - 20*x + 5;
endfunction

clf

t = input ("Initial step size: ");
eps0 = input ("Accuracy: ");

A = 0;
fA = f(0);
f1 = f(t);
f2 = f1;

clf
alpha = 1.0:0.01:3.0;
hold on

while(f2 == f1)
if(f1 > fA)
fC = f1;
C = t;
fB = f(0.5*t);
```

```
B = 0.5*t;
lambdastar = (4*fB-3*fA-fC)*0.5*t/(4*fB-2*fC-2*fA);
elseif(f1 <= fA)
fB = f1;
B = t;
f2 = f(2*t);
if(f2 > f1)
fC = f2;
C = 2*t;
lambdastar = (4*fB-3*fA-fC)*t/(4*fB-2*fC-2*fA);
else
f1=f2;
t = 2*t;
endif
endif
endwhile

a = fA;
b = (4*fB-3*fA-fC)/(2*t);
c = (fC+fA-2*fB)/(2*t*t);

h = a + b*lambdastar + c*lambdastar*lambdastar;
fstar = f(lambdastar);

epsilon = abs((h-fstar)/fstar)

X = [A,B,C];
Y = [fA,fB,fC];
p = polyfit(X,Y,2);
YY = polyval(p,alpha);
plot(alpha,YY,'rs')

while(epsilon > eps0)

if(lambdastar > B)
if(fstar < fB)
A = B;
B = lambdastar;
```

```
C = C;
else
A = A;
B = B;
C = lambdastar;
endif
else
if(fstar < fB)
A = A;
B = lambdastar;
C = B;
else
A = lambdastar;
B = B;
C = C;
endif
endif

fA = f(A);
fB = f(B);
fC = f(C);

lambdastar = (fA*(B^2-C^2) + fB*(C^2-A^2) + fC*(A^2-B^2))/(2*(fA*(B-C)+fB*(C-A)+fC

a = fA;
b = (4*fB-3*fA-fC)/(2*t);
c = (fC+fA-2*fB)/(2*t*t);

h = a + b*lambdastar + c*lambdastar*lambdastar;
fstar = f(lambdastar);

epsilon = abs((h-fstar)/fstar)

X = [A,B,C];
Y = [fA,fB,fC];
p = polyfit(X,Y,2);
YY = polyval(p,alpha);
#plot(alpha,YY,'go')
```

```
endwhile

disp("The lambda star is: "),disp(lambdastar)

plot(alpha,alpha.^5 - 5.* alpha.^3 - 20.*alpha .+ 5)
X = [A,B,C];
Y = [fA,fB,fC];
p = polyfit(X,Y,2);
YY = polyval(p,alpha);
plot(alpha,YY,'k*')
```

5.3 Direct root methods

Direct root methods try to get the minima of the given function using the fact that the necessary condition for minima is that the first derivative is zero at the point. Hence, by finding the zeros of the first derivative, one can obtain the minima point. In this section, we show the GNU Octave implementations of three direct root methods, namely Newton(-Raphson), quasi-Newton and secant methods to find the minimum of the function

$$0.65 - \frac{0.75}{(1+\lambda^2)} - 0.65\lambda \tan^{-1}\left(\frac{1}{\lambda}\right)$$

5.3.1 Newton(-Raphson) method

```
function y = g(x)
y = 1.5*x/(1+x^2)^2 + 0.65*x/(1+x^2)-0.65*atan(1/x);
endfunction

function z = gp(x)
z = (2.8-3.2*x^2)/(1+x^2)^3;
endfunction

x0 = input ("Give the starting point: ");
epsilon = input("Give the accuracy: ");
```

```
i=0
while(abs(g(x0)) > epsilon)
x1 = x0 - (g(x0)/gp(x0));
x0=x1;
i=i+1
endwhile

disp("The solution is: "),disp(x0)
```

5.3.2 Quasi-Newton method

```
function y = f(x)
y = 0.65 - 0.75/(1+x^2)-0.65*x*atan(1/x);
endfunction

x0 = input ("Give the starting point: ");
dx = input ("Give the step size: ");
epsilon = input("Give the accuracy: ");

deps = 1000.0;
i=0
while(deps > epsilon)
f1 = f(x0);
fp = f(x0+dx);
fm = f(x0-dx);
x1 = x0 - (dx*(fp-fm))/(2.*(fp-2*f1+fm));
x0=x1;
deps = abs((fp-fm)/(2*dx));
i=i+1
endwhile

disp("The solution is: "),disp(x0)
```

5.3.3 Secant method

```
function y = g(x)
y = 1.5*x/(1+x^2)^2 + 0.65*x/(1+x^2)-0.65*atan(1/x);
```

```
endfunction

t0 = input ("Give the initial step length: ");
epsilon = input("Give the accuracy: ");

L1 = 0+eps;
A = L1;
fpA = g(A);
i=1;
if(g(t0) < 0)
L1 = t0;
A = L1;
fpA = g(A);
t0= 2*t0;
endif
B = t0;
fpB = g(B);
i = 1;
while(abs(g(L1)) >= epsilon)
L1 = A - fpA*(B-A)/(fpB-fpA);
if(g(L1) >= 0)
B = L1;
fpB = g(B);
else
A = L1;
fpA = g(L1);
endif
i=i+1;
endwhile

disp("The number of iterations: "),disp(i)
disp("The solution is: "),disp(L1)
```

6 Nonlinear programming: Unconstrained optimization techniques

Unconstrained minimization techniques are of two types: direct search methods (which do not involve derivatives) and descent methods (which involve gradients (first-order methods) and gradients and Hessians (second-order methods)).

6.1 Direct search method

6.1.1 Random walk method

```
more on

x1(1) = input("Give the first component of initial point :");
x1(2) = input("Give the second component of initial point:");
lambda = input("Give the step length:");
minlam = input("Give the minimum allowable step length:");
N = input("Maximum number of iterations:");

i=1;
f1 = f(x1);

while(lambda > minlam)
while(i<=N)
u = 2.*(0.5-rand(size(x1)));
if(norm(u) > 1) u = zeros(size(x1));
else
u = u./norm(u);
endif
x = x1+lambda*u;
f2 = f(x);
if(f2>f1) i = i+1;
else
x1=x;
f1 = f(x);
```

```
i=1;
endif
endwhile
lambda = 0.5*lambda;
i=1;
endwhile

display("The optimum point is:"),disp(x1)
display("The optimum value is:"),disp(f(x1))

fminsearch(@(x) x(1)-x(2)+ 2.*x(1)^2+ 2*x(1)*x(2) + x(2)^2 ,[-2,2])
fminunc(@(x) x(1)-x(2)+ 2.*x(1)^2+ 2*x(1)*x(2) + x(2)^2 ,[-2,2])
```

6.1.2 Univariate method

```
more on

function y = f(x,y)
y = x-y+2*x*x+2*x*y+y*y;
endfunction

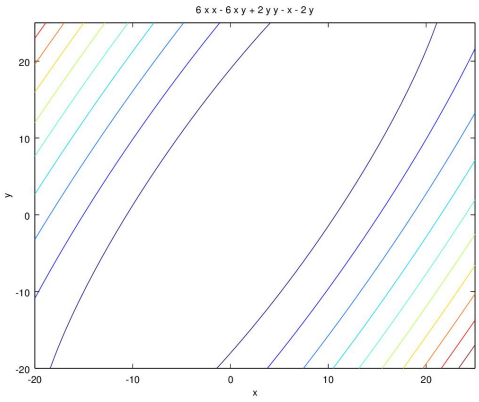
ss = input ("Give the initial step-size: ");
epsilon = input ("Give the required accuracy: ");
a = ss;
b = ss;
x = 0;
y = 0;
x1 = x+ ss;
y1 = y+ ss;
i=0;
while(abs(f(x,y) - f(x1,y1)) > epsilon && i < 10000)
if(f(x+ss,y) > f(x,y))
ss = -ss;
endif
while (f(x,y)-f(x+ss,y) > 1.e-2*epsilon)
x = x+ss;
ss = 2*ss;
```

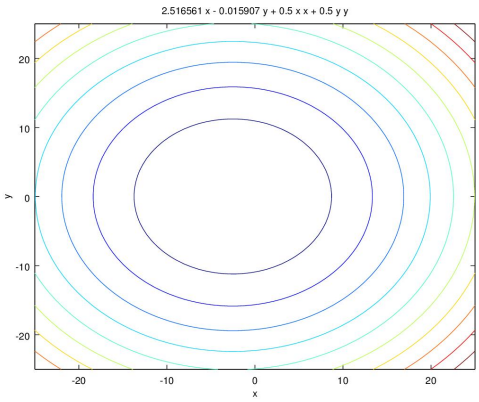
```
endwhile
if(f(x,y+a) > f(x,y))
a = -a;
endif
while (f(x,y)-f(x,y+a) > 1e-2*epsilon)
y = y+a;
a = 2*a;
endwhile
i = i+2;
ss = b;
a=b;
x1 = x + 0.5*b;
y1 = y + 0.5*b;
endwhile
disp("Number of iterations:"),disp(i)
disp("Optimum point:"),disp(x)
disp("and:"),disp(y)
disp("Minima:"),disp(f(x,y))
```

6.2 Scaling and contour plots

6.2.1 Script for scaling

```
A = [12,-6; -6,4];
B = [-1;-2];
[V,lambda] = eig(A);
Ab = V'*A*V;
S = zeros(size(Ab))
a = size(Ab);
for i=1:a(1)
S(i,i) = 1./sqrt(Ab(i,i));
endfor
T = V*S;
B'*T
0.5*T'*A*T
```





6.2.2 Script for plotting contours

```
f = @(x,y) 2.516561.*x-0.015907.*y+0.5.*x.*x+0.5.*y.*y;
ezcontour(f,[-25 25 -25 25])
print -djpg ScaledContour.jpg
g = @(x,y) 6.*x.*x-6.*x.*y+2.*y.*y-x-2.*y;
ezcontour(g,[-20 25 -20 25])
print -djpg UnScaledContour.jpg
```

The resultant contour plots are shown in Fig. 6.2.2 and Fig. 6.2.2.

6.3 Gradient methods

6.3.1 Cauchy method

```
function y = f(x)
y = x(1)-x(2) + 2*x(1)^2 + 2*x(1)*x(2)+x(2)^2;
endfunction

function yp = fp(x)
yp(1) = 1.0 +2*x(2) + 4*x(1);
yp(2) = -1.0 + 2*x(1) + 2*x(2);
endfunction

more on

x0(1) = input("Give the starting point: first component:");
x0(2) = input("Give the starting point: second component:");

i=1;

while(norm(fp(x0)) > 1.e-6)

S = -fp(x0);

ss = 0.01;
x1 = x0+ss*S;

while(ss > 1.e-12)
while( (f(x0) - f(x1)) > 0)
x0 = x1;
x1 = x0+ss*S;
endwhile
x0 = x0-ss*S;
ss = 0.5*ss;
endwhile
x0=x1;
i=i+1;
endwhile
```

```
disp("Number of iterations:"),disp(i)
disp("Solution:"),disp(x0)
disp("Function value:"),disp(f(x0))
disp("Norm of fprime:"),disp(norm(fp(x0)))
```

6.3.2 Fletcher-Reeves method

```
function y = f(x)
y = x(1)-x(2) + 2*x(1)^2 + 2*x(1)*x(2)+x(2)^2;
endfunction

function yp = fp(x)
yp(1) = 1.0 +2*x(2) + 4*x(1);
yp(2) = -1.0 + 2*x(1) + 2*x(2);
endfunction

more on

x0(1) = input("Give the starting point: first component:");
x0(2) = input("Give the starting point: second component:");

i=1;

S0 = -fp(x0);
nfp0 = norm(S0);

ss = 0.01;
x1 = x0+ss*S0;

while(ss > 1.e-12)
while( (f(x0) - f(x1)) > 0)
x0 = x1;
x1 = x0+ss*S0;
endwhile
x0 = x0-ss*S0;
ss = 0.5*ss;
```

```
endwhile
x0 = x1;


nfp1 = norm(fp(x0));
S1 = -fp(x1) + nfp1^2*S0/nfp0^2;

while(norm(fp(x0)) > 1.e-6)
S0 = S1;
nfp0 = norm(S0);

ss = 0.01;
x1 = x0+ss*S1;

while(ss > 1.e-12)
while( (f(x0) - f(x1)) > 0)
x0 = x1;
x1 = x0+ss*S1;
endwhile
x0 = x0-ss*S1;
ss = 0.5*ss;
endwhile
x0=x1;
nfp1 = norm(fp(x0));
S1 = -fp(x1) + nfp1^2*S0/nfp0^2;
i=i+1;
endwhile

disp("Number of iterations:"),disp(i)
disp("Solution:"),disp(x0)
disp("Function value:"),disp(f(x0))
disp("Norm of fprime:"),disp(norm(fp(x0)))
```

6.3.3 Newton's method

```
function y = f(x)
y = x(1,1)-x(2,1) + 2*x(1,1)^2 + 2*x(1,1)*x(2,1)+x(2,1)^2;
```

```
endfunction

function yp = fp(x)
yp(1,1) = 1.0 +2*x(2,1) + 4*x(1,1);
yp(2,1) = -1.0 + 2*x(1,1) + 2*x(2,1);
endfunction

function J = fpp(x)
J(1,1) = 4;
J(1,2) = 2;
J(2,1) = 2;
J(2,2) = 2;
endfunction

more on

x0(1,1) = input("Give the starting point: first component:");
x0(2,1) = input("Give the starting point: second component:");

i=1;
JJ = inv(fpp(x0))

while(norm(fp(x0)) > 1.e-7)

S = -JJ*fp(x0);

ss = 0.01;
x1 = x0+ss*S;

while(ss > 1.e-12)
while( (f(x0) - f(x1)) > 0)
x0 = x1;
x1 = x0+ss*S;
endwhile
x0 = x0-ss*S;
ss = 0.5*ss;
endwhile
x0=x1;
```

```
i=i+1;
endwhile

disp("Number of iterations:"),disp(i)
disp("Solution:"),disp(x0)
disp("Function value:"),disp(f(x0))
disp("Norm of fprime:"),disp(norm(fp(x0)))
```

6.3.4 Marquardt method

```
function y = f(x)
y = x(1,1)-x(2,1) + 2*x(1,1)^2 + 2*x(1,1)*x(2,1)+x(2,1)^2;
endfunction

function yp = fp(x)
yp(1,1) = 1.0 +2*x(2,1) + 4*x(1,1);
yp(2,1) = -1.0 + 2*x(1,1) + 2*x(2,1);
endfunction

function J = fpp(x)
J(1,1) = 4;
J(1,2) = 2;
J(2,1) = 2;
J(2,2) = 2;
endfunction

more on

x0(1,1) = input("Give the starting point: first component:");
x0(2,1) = input("Give the starting point: second component:");

i=1;
alpha1 = 10000;
c1 = 0.25;
c2 = 2.0;
epsilon = 1.0e-12;
```

```
JJ = fpp(x0);

f0 = f(x0);

while(norm(fp(x0)) > epsilon)

x1 = x0 - inv(JJ+alpha1*eye(size(JJ)))*fp(x0);

f1 = f(x1);

if(f1 < f0)
alpha1 = c1*alpha1;
i=i+1;
else
alpha1 = c2*alpha1;
i=i+1;
endif

x0 = x1;
f0 = f(x0);

endwhile

disp("Number of iterations:"),disp(i)
disp("Solution:"),disp(x0)
disp("Function value:"),disp(f(x0))
disp("Norm of fprime:"),disp(norm(fp(x0)))
```

6.4 `fminsearch` and `fminunc`

The GNU octave commands `fminsearch` and `fminunc` can also be used for nonlinear optimization. While `fminunc` uses gradient search (BFGS), `fminsearch` uses Nelder-Mead algorithm.

7 Nonlinear programming: constrained optimization

Sequential quadratic programming can be used to solve nonlinear programming problems with constraints. Here is an example:

```
function r = g (x)
r = [ -0.6/x(1) - 0.3464/x(2) + 0.1;
    -6.0+x(1);
    -7.0+x(2) ];
endfunction

function obj = f (x)
obj = 0.1*x(1) + 0.05773*x(2);
endfunction

x0 = [-1.0;1.0];

[x, obj, info, iter, nf, lambda] ...
= sqp (x0, @f, [], @g,0,100,1000,1.e-12)
```

The above script solves the following optimization problem (solved problem 7.5 of S S Rao [1]:

Minimize $f(\mathbf{X}) = 0.1x_1 + 0.05773x_2$

subject to

$g_1(\mathbf{X}) = \frac{0.6}{x_1} + \frac{0.3464}{x_2} - 0.1 \leq 0$

$g_2(\mathbf{X}) = 6 - x_1 \leq 0$

$g_3(\mathbf{X}) = 7 - x_2 \leq 0$

8 Exercises

1. Write `GNU Octave` scripts to solve the following LP problem.

   Maximize $f = -2x_1 - x_2 + 5x_3$

subject to

$x_1 - 2x_2 + x_3 \leq 8$

$3x_1 - 2x_2 \geq -18$

$2x_1 + x_2 - 2x_3 \leq -4$

$x_i \geq 0,\ i = 1, 2, 3$

2. Write `GNU Octave` scripts to solve the following LP problem.

   Maximize $f = x + 2y$

   subject to

   $x - y \geq -8$

   $5x - y \geq 0$

   $x + y \geq 8$

   $-x + 6y \geq 12$

   $5x + 2y \leq 68$

   $x \leq 10$

   $x \geq 0,\ y \geq 0$

3. Write a GNU Octave script to take (a) the interval and (b) the required number of iterations as inputs and identify the interval of uncertainty using the Fibanocci method. The flowchart is given in p. 245 of S S Rao [1].

4. Write a GNU Octave script to take the initial point $\mathbf{X}$, the direction $\mathbf{S}$ and initial step size t_0 as inputs and implement the cubic interpolation method of identifying the minimizing $\tilde{\lambda}$ for a given function (Example 5.11; p. 263 of S S Rao [1]). The flowchart is given in p. 262 of S S Rao [1].

5. Consider the excess free energy in a regular solution model.

$$\Delta G = \Omega x(1-x) + RT\left[x\log(x) + (1-x)\log(1-x)\right]$$

   Let us non-dimensionalise:

$$\frac{\Delta G}{RT} = \frac{\Omega}{RT}x(1-x) + \left[x\log(x) + (1-x)\log(1-x)\right]$$

Let us assume some $\alpha = \Omega/RT$. α is an inverse temperature.

Let us call the non-dimensionalised free energy $\Delta G' = \Delta G/RT$

Plot $\Delta G'$ for three different α: namely 2.0, 3.0 4.0 and 5.0

From these plots, it is clear that at high temperatures ($\alpha < 2$) there is only one minimum for $\Delta G'^{)}$. However, for lower temperatures ($\alpha > 2$), there are two minima – one to the left of composition 0.5 and the other to the right. At any given temperature, if we identify these compositions that minimize the free energy, then the plot of the composition as a function of (normalised) temperature ($1/alpha$) is the phase diagram. Write a GNU Octave script for obtaining such a phase diagram.

6. Implement Powells method using GNU Octave. The flow-chart for the same is given on p.305 of S S Rao [1].

7. Implement DFP method using the algorithm given in pp. 333-334 of S S Rao [1].

9 Solutions to selected problems

1.
```
# This is the octave script to solve the (exercise) problem 3.1 of S S Rao.
# Note that the exercise asks only to state the LP problem in standard form.
# However, for doing the computations in Octave standard form is not
# necessary.
# Preamble

clear all
clf

# The first part: definition of the problem

# Cost coefficients
c = [-2,-1,5]';
# Matrix of constraint coefficients
A = [1,-2,1; 3,-2,0;2,1,-2];
# The right hand side of constraints
b = [8,-18,-4]';
# Lower bound for the design variables
lb = [0,0,0]';
# Upper bound for the design variables
ub = [];
# Constraint type: U and L indicate inequalities with upper and lower bound
# respectively
ctype = "ULU"
# Variable type: C is for continuous
vartype = "CCC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = -1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;
```

```
# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

2. ```
   # This is the octave script to solve the (exercise) problem 3.16 of S S Rao.
   # Preamble

   clear all
   clf

   # The first part: definition of the problem

   # Cost coefficients
   c = [1,2]';
   # Matrix of constraint coefficients
   A = [1,-1; 5,-1; 1,1;-1,6;5,2;1,0];
   # The right hand side of constraints
   b = [-8,0,8,12,68,10]';
   # Lower bound for the design variables
   lb = [0,0]';
   # Upper bound for the design variables
   ub = [];
   # Constraint type: U and L indicate inequalities with upper and lower bound
   # respectively
   ctype = "LLLLUU"
   # Variable type: C is for continuous
   ```
   ```

```
vartype = "CC";
# Sense: if 1, it is minimization; if -1, it is maximization
s = -1;

# The second part: parameters for computation

# Level of messages ourput by solver. 3 is full output. Default is 1.
param.msglev = 3;
# The limit on number of simplex iterations
param.itlim = 10;

# The third part: Optimization

# glpk function is the one that solves the Linear Programming (LP) problem.
[xmin, fmin, status, extra] = glpk(c,A,b,lb,ub,ctype,vartype,s,param);

# The fourth part: Output the solution

xmin

fmin

status

extra
```

3.
```
function y = G(x,a)
y = a.*x.*(1.-x) .+ x.*log(x) + (1.-x).*log(1.-x);
endfunction

clf
hold on;
x=0.0001:0.0001:0.9999;
a = 1.0;
for i=1:4
a = a+1;
plot(x,G(x,a));
```

```
endfor

print -djpg FreeEnergy.jpg

clf

a(1) = 2.0;

for i=1:400

c(i) = fminbnd(@(x) G(x,a(i)),0.0001,0.4999,optimset('TolX',1.e-12));
d(i) = fminbnd(@(x) G(x,a(i)),0.5001,0.9999,optimset('TolX',1.e-12));

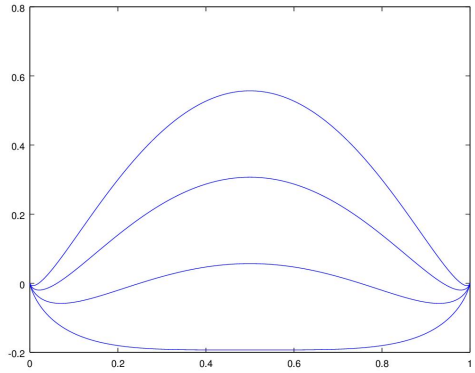
a(i+1) = a(i) + 0.01;
endfor

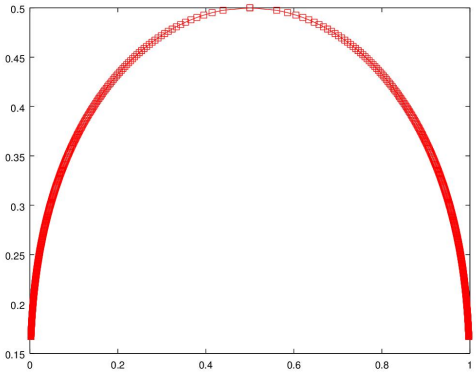
for i=1:400
b(i) = a(i);
endfor

plot(c,1./b,'rs-')
hold on
plot(d,1./b,'rs-')

print -djpg RegSolPhDia.jpg
```

The figures generated by the script are as shown in Fig. 3 and 3.


0.8
0.6
0.4
0.2
0
-0.2
0
0.2
0.4
0.6
0.8
1


0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0
0.2
0.4
0.6
0.8
1

References

[1] Engineering optimization: Theory and practice, S S Rao, New Age International Publishers, Third enlarged edition, 2013.