Question 1

a)

The problem is a binary integer problem. The x_i represent an indicator variable for the presence of the ith crew member. The objective function to solve is:

$min \sum_{i=1}^{n} x_i Salary_i$

```r
salary = c(8000, 5000, 4000, 10000, 9000, 5000, 3000, 6000, 4000, 5000)
```

Constraints:

$\sum_{i=1}^{n} x_i Fishing_i \geq 15$

$\sum_{i=1}^{n} x_i Sailing_i \geq 15$

$\sum_{i=1}^{n} x_i Navigation_i \geq 15$

$\sum_{i=1}^{n} x_i Cooking_i \geq 3$

$x_1, x_2, ..., x_n \geq 0$

```r
fishing = c(2, 1, 1, 5, 4, 3, 1, 2, 4, 2)
navigation = c(5, 4, 4, 2, 3, 2, 4, 2, 2, 1)
sailing = c(3, 3, 2, 5, 3, 3, 1, 3, 2, 4)
cooking = c(1,1,1,0,0,0,1,1,0,0)

constraints = c(15,15,15,3)
constraints.dir = c(">=",">=",">=",">=")
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3

```r
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

The minimum achieveable cost for the voyage is: $ 31000

The optimal crew is therefore [Benjamin, Clara, Dave, Francois, Gordon, Idi] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	1	1	0	1	1	0	1	0

b)

Dave (x_4) and Ed (x_5) cannot be together on the voyage therefore we add the additional constraint:

$x_4 + x_5 \leq 1$

```r
daveOrEd = c(0,0,0,1,1,0,0,0,0,0)
constraints = c(15,15,15,3,1)
constraints.dir = c(">=",">=",">=",">=","<=")
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3
daveOrEd	0	0	0	1	1	0	0	0	0	0	<=	1

```r
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking, daveOrEd),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

The minimum achieveable cost for the voyage is: \$ 31000

The optimal crew is therefore [Benjamin, Clara, Dave, Francois, Gordon, Idi] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	1	1	0	1	1	0	1	0

c)

Dave (x_4) and Gordon (x_7) cannot be together on the voyage therefore we add the additional constraint:

$x_4 + x_7 \leq 1$

```r
daveOrGordon = c(0,0,0,1,0,0,1,0,0,0)
constraints = c(15,15,15,3,1)
constraints.dir = c(">=",">=",">=",">=","<=")
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3
daveOrGordon	0	0	0	1	0	0	1	0	0	0	<=	1

```
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking, daveOrGordon),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

The minimum achieveable cost for the voyage is: $ 32000

The optimal crew is therefore [Benjamin, Ed, Francois, Gordon, Harriet, Idi] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	0	0	1	1	1	1	1	0

d)

In order to exclude Idi (x_9) from the analysis, we can simply assign him a very large penalty salary thereby ensuring he never is a candidate for the optimal solution. Another option would have been to remove his entries altogther from the initial feasible solution but that would require re-formulating the problem

```
salary = c(8000, 5000, 4000, 10000, 9000, 5000, 3000, 6000, 10000000, 5000)
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3

```
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

The minimum achieveable cost for the voyage is: $ 36000

The optimal crew is therefore [Benjamin, Clara, Dave, Ed, Francois, Gordon] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	1	1	1	1	1	0	0	0

e)

By manually searching across the problem space, we come to the conclusion that given the baseline optimal solution, $11200 is the lowest amount we can pay Dave whereby he is not included in any optimal solution. Therefore to include him in any optimal solution we can pay him anywhere from $10000 up to $11199.99

The following analysis shows that at $11200 Dave is omitted from the optimal solution:

```r
salary = c(8000, 5000, 4000, 11200, 9000, 5000, 3000, 6000, 4000, 5000)

fishing = c(2, 1, 1, 5, 4, 3, 1, 2, 4, 2)
navigation = c(5, 4, 4, 2, 3, 2, 4, 2, 2, 1)
sailing = c(3, 3, 2, 5, 3, 3, 1, 3, 2, 4)
cooking = c(1,1,1,0,0,0,1,1,0,0)

constraints = c(15,15,15,3)
constraints.dir = c(">=",">=",">=",">=")
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3

```r
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

The minimum achieveable cost for the voyage is: $ 32000

The optimal crew is therefore [Benjamin, Ed, Francois, Gordon, Harriet, Idi] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	0	0	1	1	1	1	1	0

f)

By removing the salary restriction, and making the optimisation problem purely a crew-minimisation problem, the new optimisation function simply looks like this:

$min \sum_{i=1}^{n} x_i$

With a new salary vector like so:

```
salary = rep(1,10)
```

The constraints remain the same:

```
fishing = c(2, 1, 1, 5, 4, 3, 1, 2, 4, 2)
navigation = c(5, 4, 4, 2, 3, 2, 4, 2, 2, 1)
sailing = c(3, 3, 2, 5, 3, 3, 1, 3, 2, 4)
cooking = c(1,1,1,0,0,0,1,1,0,0)

constraints = c(15,15,15,3)
constraints.dir = c(">=",">=",">=",">=")
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	constraints.dir	constraints
fishing	2	1	1	5	4	3	1	2	4	2	>=	15
sailing	3	3	2	5	3	3	1	3	2	4	>=	15
navigation	5	4	4	2	3	2	4	2	2	1	>=	15
cooking	1	1	1	0	0	0	1	1	0	0	>=	3

```
lp.object = lp (direction = "min",
                objective.in = salary,
                const.mat = rbind(fishing, sailing, navigation, cooking),
                const.dir = constraints.dir,
                const.rhs = constraints,
                all.bin=TRUE)
```

An optimal crew is therefore [Angela, Benjamin, Dave, Ed, Harriet, Idi] given by the solution vector:

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
1	1	0	1	1	0	0	1	1	0

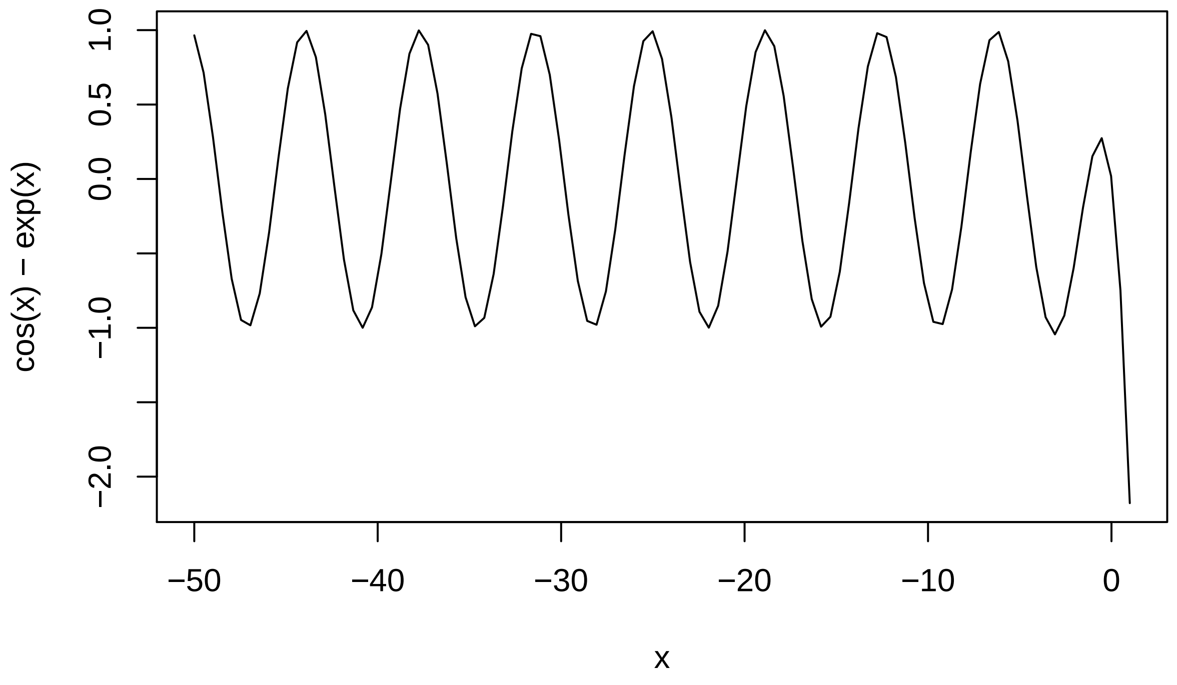
Therefore the minimum crew size is 6

Question 2

The Function

The function to optimise is a univariate unconstrained NLP:

$$\max f(x) = cos(x) - e^x$$

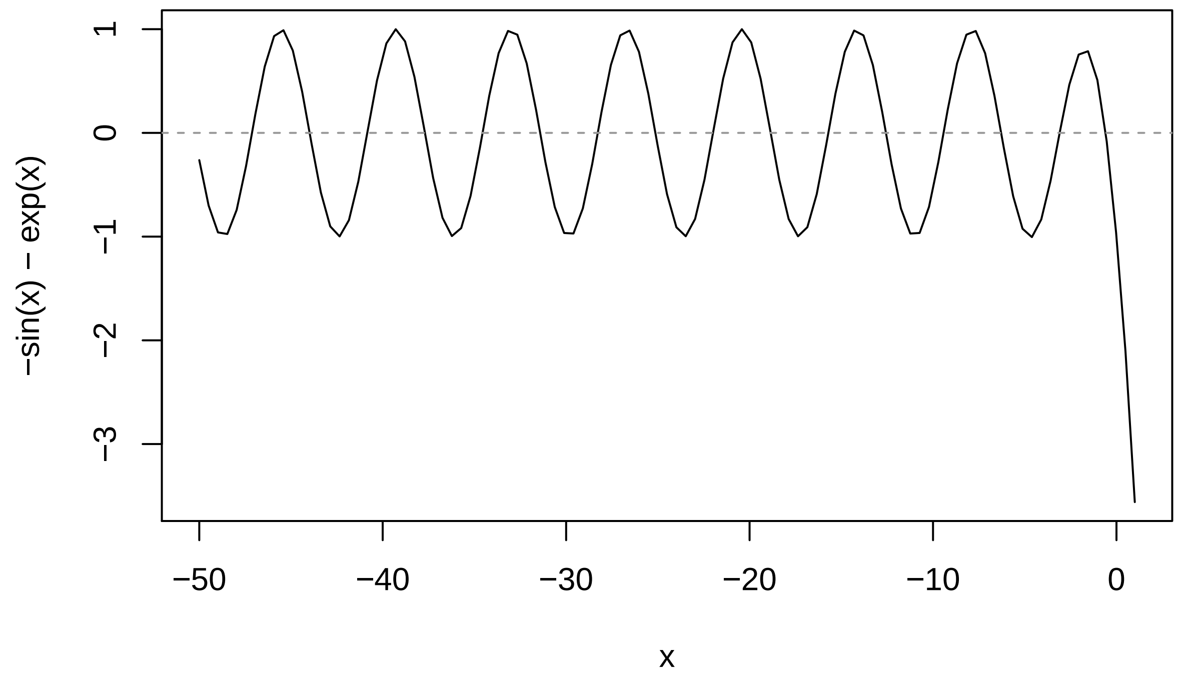


This is a non-trivial function to maximise as it is not unimodal and we are not given any constraints within which to find any maxima.

Analytic Approach

The first approach we take with a problem like this is to evaluate the first derivative and find the points at which it equals zero:

$$f'(x) = -sin(x) - e^x$$



As can be seen the local maxima can be identified at the when the line intersects with zero, namely all the multiples of π. This is not particularly useful however as we appear to have infinitely many local maxima as x tends to $-\infty$

It is not obvious from the graph, however analytically we can see that with every period of $cos(x)$ the function gets very slightly closer to 1 due to the $-e^x$ term. However because $-e^x$ only ever becomes vanishingly small but never actually reaches zero, $cos(x) - e^x$ too will never reach one. With every period of $cos(x)$ we reach a new (very slightly) larger local maxima. Therefore there are an infinite number of local maxima.

Numerical Approaches

None of the numerical approaches outlined below were able to find any global optima due to the somewhat pathological nature of the objective function. They are only described in this section to outline their limitations for this particular example.

Newton Raphson Because both first and second derivatives of the objective function can be evaluated, we look at the Newton-Raphson point approximation method. As can be seen however this approach will only ever find a local extremum, and more worryingly, for a 'badly behaved' function such as this one, can end up finding a minimum instead of a maximum completely depending on the starting location. The plot below shows that due to the arbitrary step size, even when starting close to a local maxima, the algorithm jumps over the maxima, and proceeds to head to a minima instead.

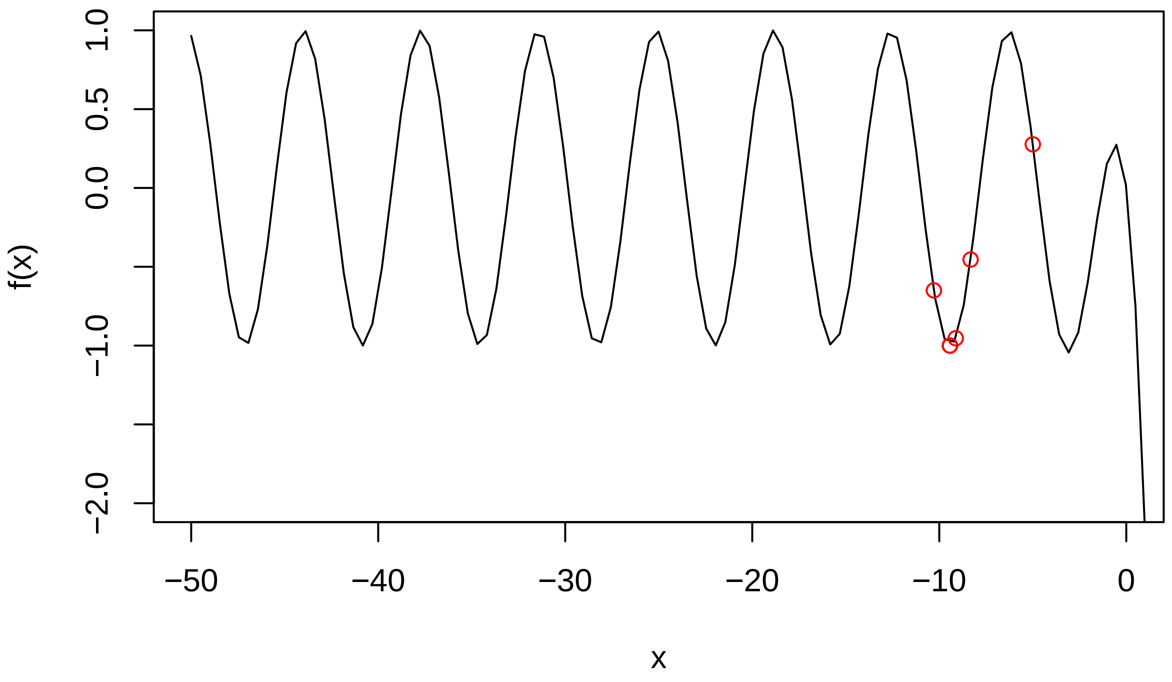
$$f''(x) = -cos(x) - e^x$$

```r
fprimeprime <- function (x) {
  -cos(x) - exp(x)
}
```

```r
x = c(-5,rep(NA,6))
fval = rep(NA,7)
fprimeval = rep(NA,7)
```

```
fprimeprimeval = rep(NA,7)

for(i in 1:6){
  fval[i]=f(x[i])
  fprimeval[i]=fprime(x[i])
  fprimeprimeval[i]=fprimeprime(x[i])
  x[i+1]=x[i]-fprimeval[i]/fprimeprimeval[i]
}
```

x	fval	fprimeval	fprimeprimeval
-5.000000	0.2769242	-0.9656622	-0.2904001
-8.325282	-0.4542872	0.8907365	0.4538026
-10.288110	-0.6499430	-0.7600461	0.6498749
-9.118583	-0.9535970	0.3013232	0.9533778
-9.434641	-1.0000313	-0.0099433	0.9998714
-9.424697	-1.0000807	0.0000003	0.9999193
-9.424697	NA	NA	NA



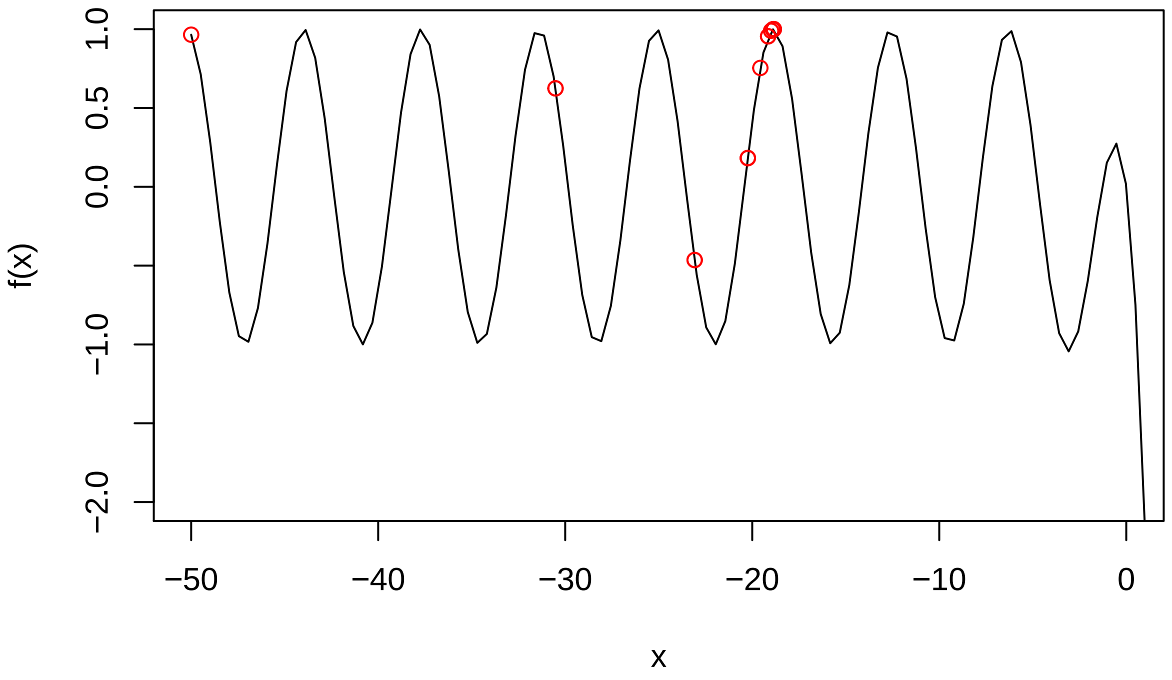
Line Search Again, because this method should not be used with more than one stationary point, the technique can be discarded immediately.

Golden Section Search Golden Section again, is not appropriate for this problem but to show that it could have been useful if the problem was unimodal we can see the following output does end up converging to a local maxima. However due to the step size and the multi-modal nature of this problem, the choice of maxima that it eventually settles on does appear to be fairly arbitrary.

```
n=1
a=0;
```

```
a[1]=-50
b=0;
b[1]=1
phi=0.5*(sqrt(5)-1)
L=0;
L[1]=a[1]+phi*phi*(b[1]-a[1])
U=0;
U[1]=b[1]-(L[1]-a[1])
tolerance=0.001
while(abs(b[n]-a[n])>tolerance) {
  if(f(L[n])>=f(U[n])){
    a[n+1]=a[n]
    b[n+1]=U[n]
    L[n+1]=a[n]+U[n]-L[n]
    U[n+1]=L[n]
    }
  else{
    a[n+1]=L[n]
    b[n+1]=b[n]
    L[n+1]=U[n]
    U[n+1]=b[n]-U[n]+L[n]
    }
  n=n+1
  }
width=b-a
```

a	b	L	U	width
-50.00000	1.00000	-30.51973	-18.48027	51.0000000
-30.51973	1.00000	-18.48027	-11.03947	31.5197334
-30.51973	-11.03947	-23.07893	-18.48027	19.4802666
-23.07893	-11.03947	-18.48027	-15.63813	12.0394669
-23.07893	-15.63813	-20.23680	-18.48027	7.4407997
-20.23680	-15.63813	-18.48027	-17.39467	4.5986671
-20.23680	-17.39467	-19.15120	-18.48027	2.8421326
-20.23680	-18.48027	-19.56586	-19.15120	1.7565345
-19.56586	-18.48027	-19.15120	-18.89493	1.0855980
-19.15120	-18.48027	-18.89493	-18.73654	0.6709365
-19.15120	-18.73654	-18.99282	-18.89493	0.4146616
-18.99282	-18.73654	-18.89493	-18.83443	0.2562749
-18.89493	-18.73654	-18.83443	-18.79704	0.1583866
-18.89493	-18.79704	-18.85754	-18.83443	0.0978883
-18.89493	-18.83443	-18.87182	-18.85754	0.0604983
-18.87182	-18.83443	-18.85754	-18.84871	0.0373900
-18.85754	-18.83443	-18.84871	-18.84326	0.0231083
-18.85754	-18.84326	-18.85208	-18.84871	0.0142817
-18.85208	-18.84326	-18.84871	-18.84663	0.0088266
-18.85208	-18.84663	-18.85000	-18.84871	0.0054551
-18.85208	-18.84871	-18.85080	-18.85000	0.0033715
-18.85080	-18.84871	-18.85000	-18.84951	0.0020837
-18.85000	-18.84871	-18.84951	-18.84920	0.0012878
-18.85000	-18.84920	-18.84970	-18.84951	0.0007959



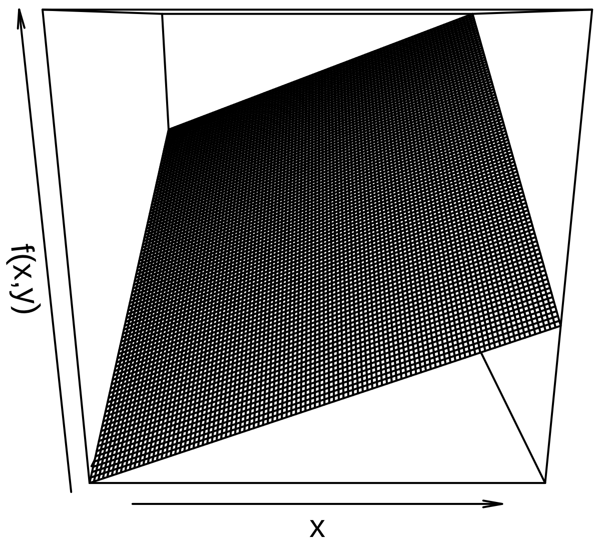
In conclusion, none of the numerical techniques were appropriate for this problem. If we did have a constrained optimisation problem, we could have used Golden Section search to first narrow down the search space, and then Newton-Raphson to home in on the final extreme point.

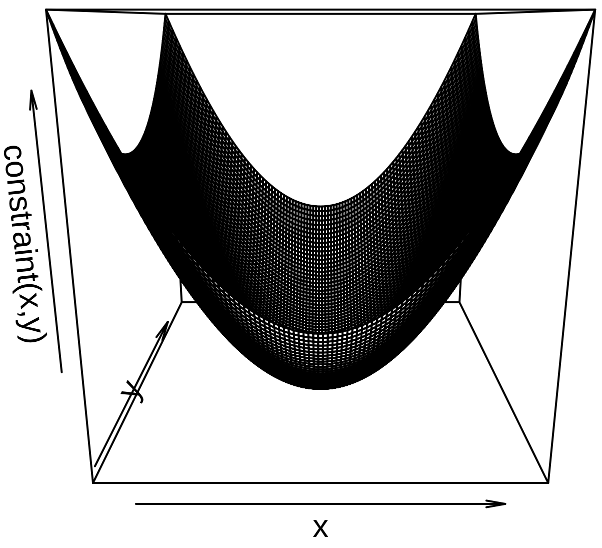
Question 3

Objective Function

The function to optimise is a convex function with non-linear and linear (box) constraints:

$$\max\ 3x + 5y$$
$$\text{st}\ 9x^2 + 5y^2 \leq k$$
$$0 \leq x \leq 4$$
$$y \geq 2$$





Kuhn-Tucker Conditions

By reformulating the NLP we can derive the Kuhn-Tucker conditions:

$$\min\ -3x - 5y$$
$$\text{st}\ 9x^2 + 5y^2 - k \leq 0$$
$$x - 4 \leq 0$$
$$-x \leq 0$$
$$-y - 2 \leq 0$$

Leads to:

i. $L = -3x - 5y + u_1(9x^2 + 5y^2 - k) + u_2(x - 4) + u_3(-x) + u_4(-y - 2)$

ii. $-3 + 18u_1 x + u_2 - u_3 = 0$

iii. $-5 + 10u_1 y - u_4 = 0$

iv. $u_1(9x^2 + 5y^2 - k) = 0$

v. $u_2(x - 4) = 0$

vi. $u_3(-x) = 0$

vii. $u_4(-y-2) = 0$

viii. $9x^2 + 5y^2 \leq k$

ix. $0 \leq x \leq 4$

x. $y \geq 2$

xi. $u_1 \geq 0, u_2 \geq 0, u_3 \geq 0, u_4 \geq 0,$

Because of the large number of cases that this approach requires we investigate, the analytical approach is too cumbersome for this example, and instead we turn to a numerical approach.

Optimization Routine

The standard R package `optim` does not support constrained non-linear optimization, hence why we have to use a different package. After much investigation the only package that would be able to handle this problem is NLopt, for which we can utilize the R port of this project, `nloptr`[1]

Within the `nloptr` package we must choose an underlying algorithm. The package provides a vast number of algorithms, however the only one that supports arbitrary nonlinear inequality and equality constraints is COBYLA (Constrained Optimization BY Linear Approximations)[2].

The set up is straightfoward; all we have to provide are our objective function, constraint function, and whatever linear (box) constraints we require. A basic optimization routine looks like so:

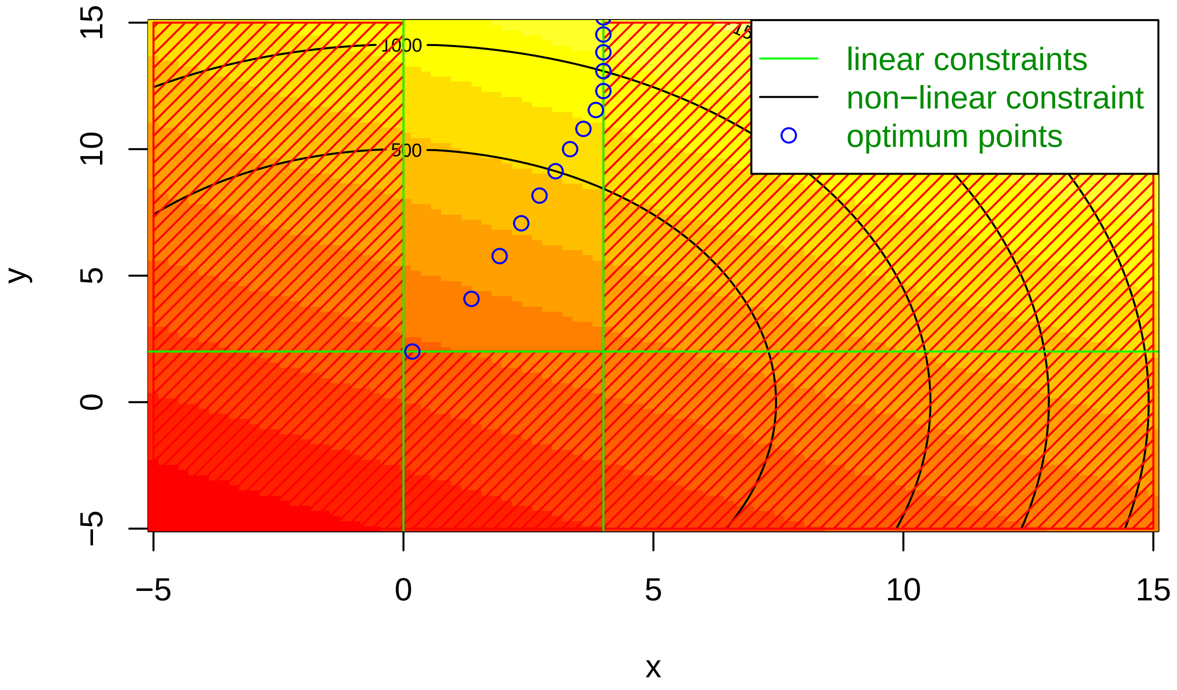
```r
library(nloptr)

#objective function
eval_f0 <- function (x, k) {
  return (-(3*x[1] + 5*x[2])) #to minimize we take the negative
}

#non-linear constraint function
eval_g0 <- function( x, k ) {
  return (9 * x[1]^2 + 5 * x[2]^2 - k)
}

#linear box constraints
xLower <- 0
xUpper <- 4
yLower <- 2
yUpper <- Inf

#values of the k parameter to test against
k <- seq(1, 10)

runOptim <- function (k) {
  nloptr( x0=c(xUpper, xUpper),
            eval_f=eval_f0,
```

[1] http://cran.r-project.org/web/packages/nloptr/vignettes/nloptr.pdf

[2] http://ab-initio.mit.edu/wiki/index.php/NLopt__Algorithms#COBYLA__.28Constrained__Optimization__BY__Linear__Approximations.29

```
            lb = c(xLower,yLower),
            ub = c(xUpper,yUpper),
            eval_g_ineq = eval_g0,
            opts = list("algorithm"="NLOPT_LN_COBYLA",
                        "xtol_rel"=1.0e-4,
                        "print_level" = 0),
            k = k )
}
```

For our specific purposes however it is slightly more complicated as we want to find the optimum across all of k. My approach to solving this problem was to see how the optimizer performed at extreme values of k and then narrowed the problem space down. This led me to the following results:



As can be seen, the maximum value of the objective function is achieved as k is maximised:

k interval	x	y	f(x,y)
0:100	0.17829	2.00015	10.5356
100:200	1.36076	4.08252	24.4949
200:300	1.92445	5.77354	34.64102
300:400	2.35698	7.07109	42.42641
400:500	2.72176	8.1649	48.98979
500:600	3.04284	9.12875	54.77226
600:700	3.33341	9.99996	60
700:800	3.6004	10.80124	64.80741
800:900	3.84913	11.54693	69.28203
900:1000	4	12.29634	73.4817
1000:1100	3.99999	13.08434	77.42167
1100:1200	4	13.82751	81.13754
1200:1300	4	14.53272	84.66361
1300:1400	3.99997	15.20527	88.02625
1400:1500	4	15.84929	91.24645