STAT 513/413: Lecture 19
Optimization

(Fundamental)

So, what is it all that root finding good for?

The most frequent application of all the methods for finding roots are are not that much for roots themselves - but for finding a minimizer (or maximizer) of some function

$$F(x) \looparrowright \min_x !$$

Sometimes they speak just about "finding minimum or maximum" - but if we adopt more precise language, *minimum* (or *maximum*) is the value

$$\min_x f(x) \qquad (\text{or } \max_x f(x))$$

while *minimizer* (or *maximizer*) is an x that attains it: it is an x_0 such that

$$f(x_0) = \min_x f(x) \qquad (\text{or } f(x_0) = \max_x f(x))$$

Of course, in even a more precise world we can ask whether such x_0 exists at all - that is whether max or min exists at all (instead of sup and inf, which exist always, even if sometimes not finite)

Nonetheless, one has to keep in mind that the most precise formalism is often not adopted

Maximization vs minimization

We could well focus just on minimization (or maximization): maximization (or minimization) can be done by considering $-F$ instead

$$F(x) \looparrowright \max_x ! \quad \text{is equivalent to} \quad -F(x) \looparrowright \min_x !$$

Optimization vs equation solving

If we are able to evaluate the derivative $f(x)$ of $F(x)$, then the minimization of F may be replaced by solving the equation

$$f(x) = 0$$

(Here, "derivative" is meant in an abstract sense; in the multidimensional case, the equation above is actually a system of equations)

Of course, it is not the same thing: by solving this equation we may, in general, obtain only *local* minimum; or even not a minimum, but a maximum, or something else

(The same is actually true for numerical equation solving methods: they do not guarantee to find all zeros, but just some zero - everything depends on the starting point of the iterations.)

Differentiability implies continuity; if a function is not differentiable, but still continuous, there are algorithms that may work; if a function is not even continuous (at least in some sense), the prospects of finding the minimizer are not big

General deliberations

So, this is how it is: the majority of optimization methods out there do not guarantee to find a *global*, but only *local* optima

Sometimes we are fine with those; and sometimes we hope that a good initial point will give us the right result. And sometimes we base our hope on a specific character of the function: if we know somehow that a local minimum found would be a global one, then we are fine

We are not actually very happy when there are several (and distant) different minimizers giving the same global minimum (even if in such a case every local minimum is a global minimum) - as we are typically not after the value of the global minimum itself, but rather a global minimizer

However, we could accept this situation if all minimizers are somewhat close: such functions are called *unimodal* (to be made precise later). Every local minimum of such a function is a global one - but not all functions with this property are unimodal

If a function has only one global (and thus local) minimum we call it *strictly unimodal*

Convex functions

A prominent example of unimodal and strictly unimodal functions are convex functions. A function F is called *convex*, if for any two points z_1 and z_2 all values of F on the segment connecting z_1 and z_2 lie under the line connecting $F(z_1)$ and $F(z_2)$: for any z_1, z_2,

$$F(\alpha_1 z_1 + \alpha_2 z_2) \leqslant \alpha_1 F(z_1) + \alpha_2 F(z_2) \qquad \alpha_1, \alpha_2 \geqslant 0,\ \alpha_1 + \alpha_2 = 1$$

If sharp inequality $<$ holds true for every z_1, z_2 and every $\alpha_1, \alpha_2 > 0$, then F is *strictly* convex

Important examples: x^2 (strictly), $|x|$; also all functions $ax + b$, of course - the latter are the only ones with the property that both f and $-f$ are convex

A function F is called (strictly) *concave*, if F is (strictly) convex

For one-dimensional functions ($x \in \mathbb{R}$), convexity can be verified in terms of derivatives as follows: if f' is nondecreasing, then f is convex; if f'' is nonnegative, then f is convex. Strict convexity of f follows from f' being increasing or f'' strictly positive

The definition and the criteria are analogous in the multidimensional case

Some properties of convex functions

The concept readily generalizes into multidimensional case: F is convex, if for any $z_1 = (x_1, y_1)^\mathsf{T}$, $z_2 = (x_2, y_2)^\mathsf{T}$, and $\alpha_1, \alpha_2 \geqslant 0$, $\alpha_1 + \alpha_2 = 1$

$$\mathrm{F}\left(\alpha_1 \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \alpha_2 \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}\right) \leqslant \alpha_1 \mathrm{F}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \alpha_2 \mathrm{F}\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$$

(Strict convexity defined in the completely analogous fashion)

Quadratic functions of the type $x^\mathsf{T} A x$ (possibly also $+Bx + c$) are convex whenever A is nonnegative definite

Convex functions satisfy the property that every local minimum of a convex function is its global minimum. They are also unimodal.

Convex functions have a few other properties that distinguish them among other classes of unimodal functions - in particular, with respect to composition and other aspects

Convexity is a very important property in optimization

So, what does it mean "unimodal"

For instance, it can be this: function F is unimodal if for every c the set $\{\mathrm{x} : \mathrm{F}(\mathrm{x}) \leqslant \mathrm{c}\}$ is convex

(The empty set is convex, by the way)

Then, every convex function is unimodal - but not every unimodal function is convex

Unimodal functions, as defined above, are also called *quasiconvex*. A more general definition of unimodality replaces "convex" in the definition above by "connected"

- but this only in case you know what it means;

otherwise beyond the scope of this course

Global optimization

If we are not happy with a local minimum - because we do not have any guarantees it is a global minimum, and really need the latter - then we have to use a *global* optimization method

Such methods have been proposed (an example is the *simulated annealing* method, available in R), but typically are not that efficient, not that elaborated, and do not give complete, but merely *some* guarantees - thus in practice we often resort to some strategies pursuing local minima

One-dimensional case again trivial - but fun

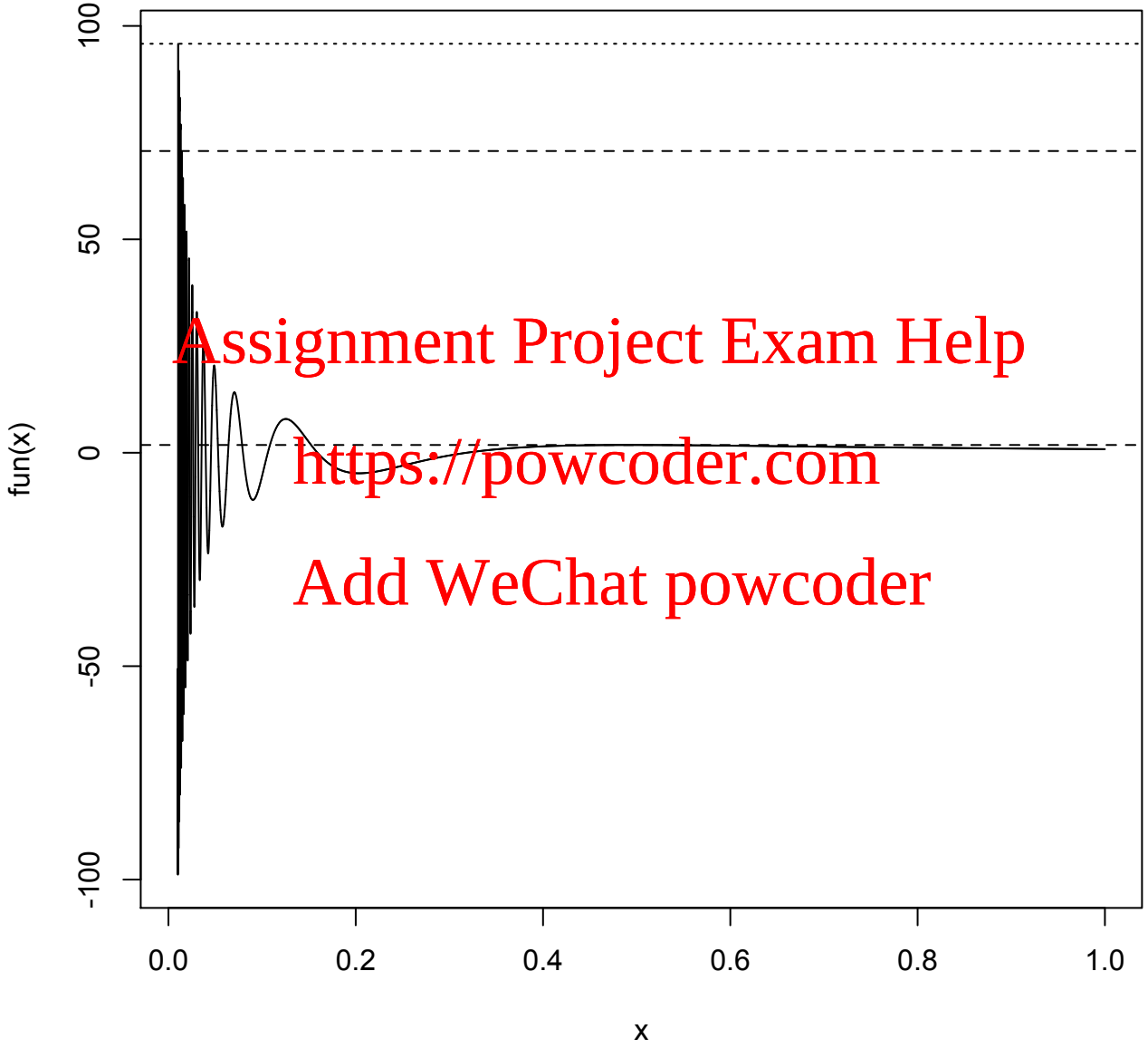
The multimodal (= multiple local maxima) case brings nothing new

The best global method (and feasible in dimension one): direct search

Direct search vs `optimize`

```
> fun = function(x) (1/x)*sin(1/x)
> x=seq(0.01,1,len=100000)
> plot(x,fun(x),type="l")
> x[which(fun(x) == max(fun(x)))]
[1] 0.0104356
> fun(x[which(fun(x) == max(fun(x)))])
[1] 95.8233
> abline(h=fun(x[which(fun(x) == max(fun(x)))]),lty=3)
> obj=optimize(fun,lower=0.01,upper=1,maximum=TRUE,tol=.Machine$double.eps)
> obj
$maximum
[1] 0.4929125
$objective
[1] 1.819706
> abline(h=obj$objective,lty=2)
> obj=optimize(fun,lower=0.01,upper=0.02,maximum=TRUE,tol=.Machine$double.eps)
> obj
$maximum
[1] 0.01414428
$objective
[1] 70.69291
> abline(h=obj$objective,lty=2)
```

A picture



Search methods for unimodal functions

How to make the direct search more optimal? It all comes to how many times we have to evaluate the function: the fewer times, the more optimal the method is

Fibonacci numbers, F_n: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$F_1 = F_2 = 1$; otherwise, $F_n = F_{n-1} + F_{n-2}$

Discrete search: unimodal function f on $1, 2, \ldots, k$

i	1	2	3	4	5	6	7
$f(i)$	0.6377	0.4196	0.1546	0.1010	0.3274	0.5693	0.6803

1	2	3	4	5	6	7	8	9	10	11	12
0.57	0.54	0.49	0.35	0.21	0.16	0.23	0.37	0.50	0.28	0.58	0.59

Fibonacci search

Allows for finding the optimum in least possible number of evaluations

General rule: we have always $k = F_n - 1$ for some n (if not, we padd on either end with some numbers larger than every other one already involved)

We evaluate at the points F_{n-2} and F_{n-1}:

- if $f(F_{n-2}) < f(F_{n-1})$, we discard all $F_{n-1}, F_{n-1}+1, \ldots, F_n - 1$
- if $f(F_{n-2}) > f(F_{n-1})$, then we discard $1, 2, \ldots F_{n-2}$

(If $f(F_{n-2}) = f(F_{n-1})$, then we may discard any one of the above.)

After figuring out the result, we have $F_{n-1} - 1$ points left, and we repeat the procedure with F_{n-2} and F_{n-3}; finding the minimum needs (at most) $n - 1$ evaluations

Golden section search

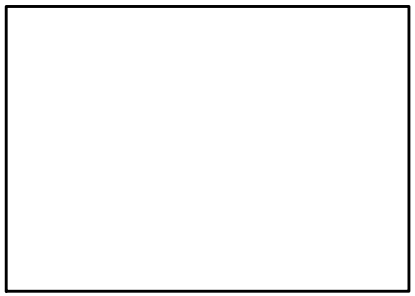
For continuous problems, Fibonacci search is impractical, as we need to know the number of evaluations that give different values in advance. We choose instead to divide the search interval - which after rescaling we can assume to be $[0, 1]$ - at locations γ^2 and γ, where

$$\gamma = \frac{\sqrt{5}-1}{2} = \lim_{n\to\infty} \frac{F_n}{F_{n-1}} = 0.618034$$

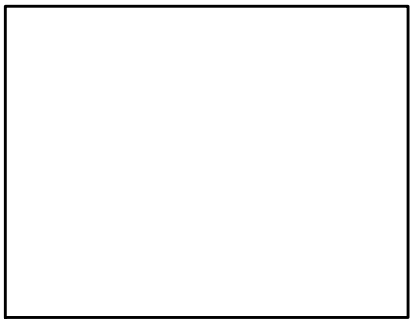
is so-called golden ratio

In R: `optimize()` - golden ratio search + parabolic interpolation

golden



a4



letter

Multidimensional case: simplex method

In R as Nelder-Mead method; not to be confused with simplex method in linear programming

Simplex: segment in dim 1, triangle in dim 2, tetrahedron in dim 3, ...

If d is the dimension, it has $d+1$ vertices; we suppose that

$$f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_d) \leqslant f(x_{d+1})$$

(in fact, we rather like to have $<$)

We take the centroid $x_1, x_2, \ldots, x_d$ - which is their mean $\bar{x}$

and reflect x_{d+1} about $\bar{x}$, obtaining $x_r = 2\bar{x} - x_{d+1}$; we can do it scaled, in which case

$$x_r = \bar{x} + \alpha(\bar{x} - x_{d+1}) \qquad \text{(typically } \alpha = 1)$$

(in fact, we rather like to have $<$)

and then we have four possibilities...

Four possibilities - before repeating all again

1. if $f(x_1) \leqslant f(x_r) < f(x_d)$, we *accept* x_r in place of x_{d+1}, reorder points and repeat

2. if $f(x_r) < f(x_1)$, we *expand* in the direction of x_r; try new x_e

$$x_e = \bar{x} + \gamma(\bar{x} - x_{d+1}) \qquad \text{(typically } \gamma = 2\text{)}$$

if $f(x_e) < f(x_r)$, accept x_e, reorder points and repeat

if $f(x_e) \geqslant f(x_r)$, accept x_r, reorder points and repeat

(Note: we do the expansion only once at a time)

3. if $f(x_r) \geqslant f(x_d)$, then we *contract*: compute

$$x_c = \bar{x} + \beta(\bar{x} - x_{d+1}) \qquad \text{(typically } \beta = 0.5\text{)}$$

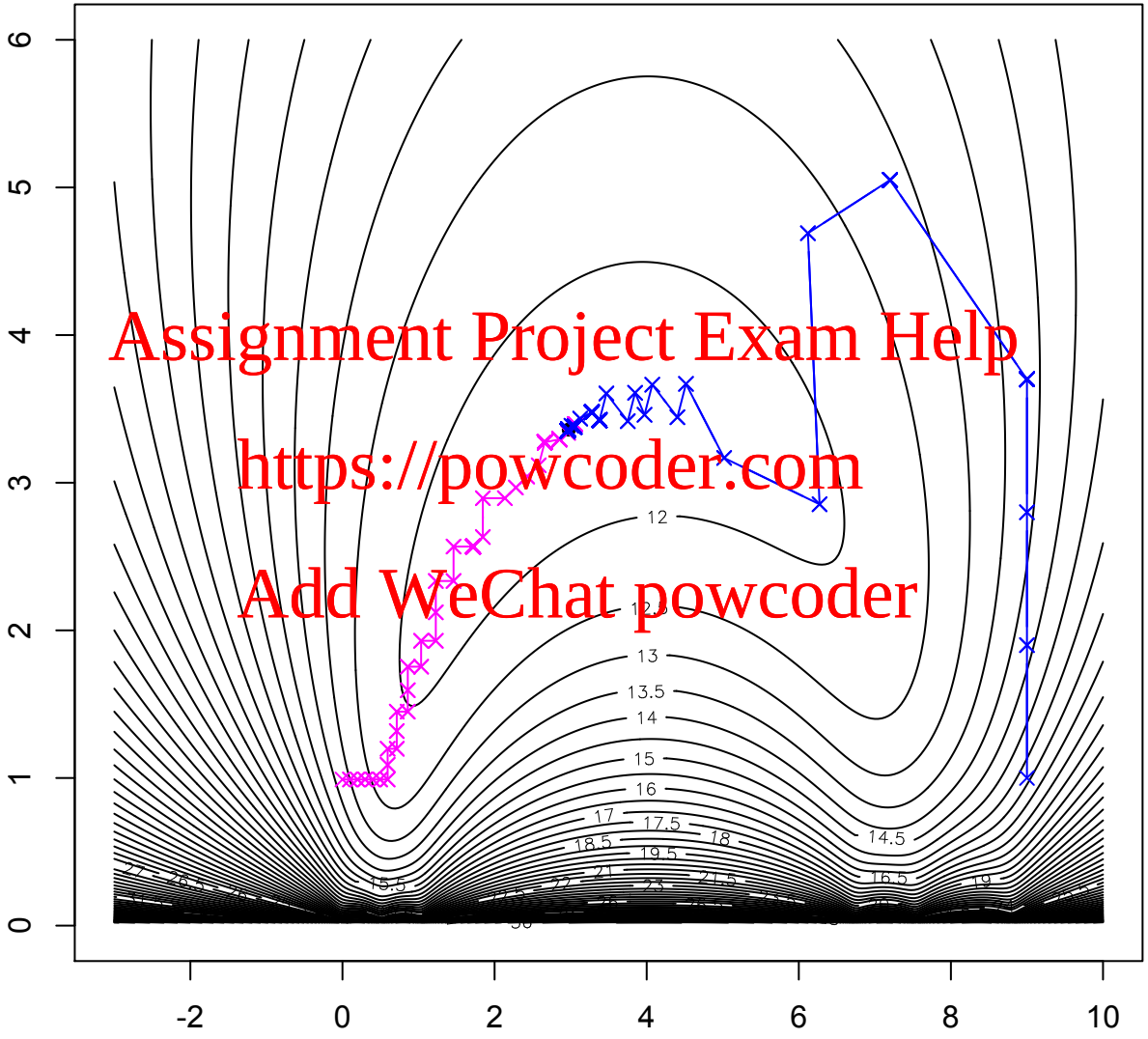
if $f(x_c) < f(x_{d+1})$, then we accept x_c in place of x_{d+1}, reorder points and repeat

4. if $f(x_c) \geqslant f(x_{d+1})$, then we *shrink*: compute d new vertices, keeping x_1 in place and taking, for $i = 2, 3, \ldots, d+1$

$$x_i = x_1 + \delta(x_i - x_1) \qquad \text{(typically } \delta = 0.5\text{)}$$

Picturing it

A demo



Another dubious method: coordinate descent

Minimize at one coordinate at time: first in x_1, then in x_2, then in x_3 ... finally in x_d, and when done, start again: first in x_1, then in x_2, ...

Doesn't work! Well... unless the objective function is differentiable, then it may

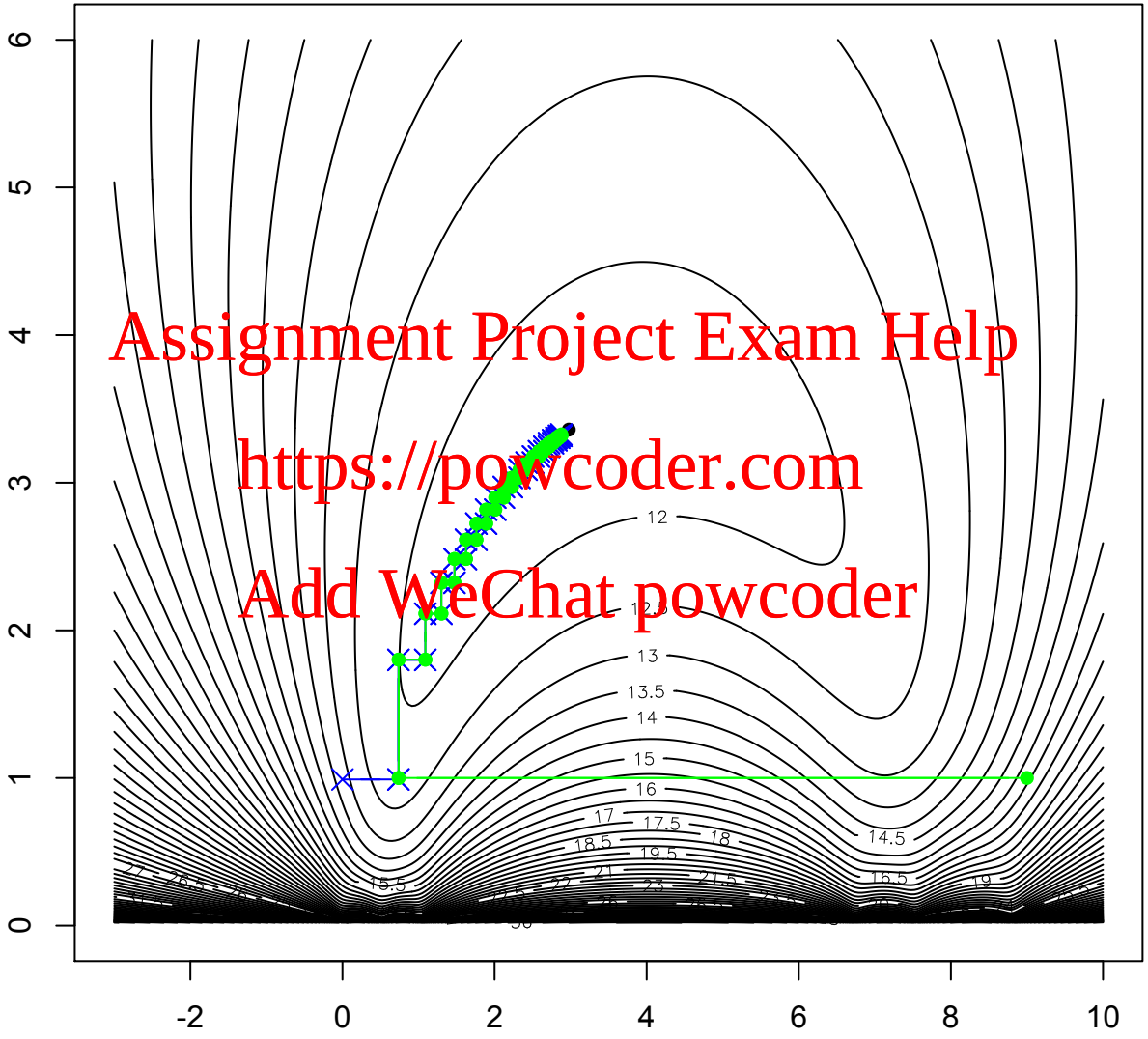
(For nondifferentiable function of two variables one can find a counterexample showing that the method gets stuck far from the optimum.)

But, it was a breakthrough for minimizing functions, in β, like

$$\sum_{i=1}^{p}(y_i - x_{i1}\beta_1 - \cdots - x_{id}\beta_d)^2 + \lambda \sum_{j=1}^{d} |\beta_j| \qquad (\lambda > 0)$$

so-called LASSO, for p and q large

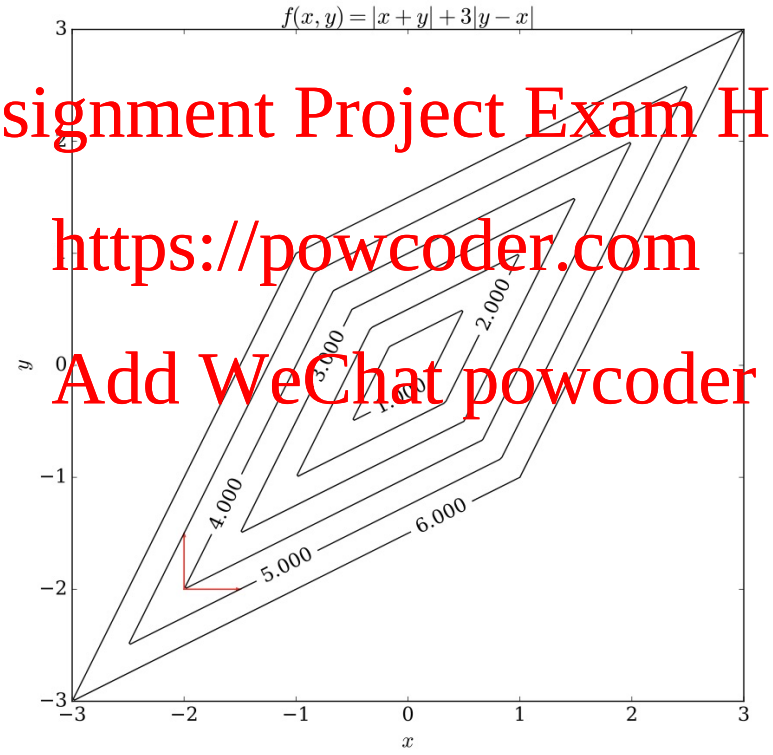
Another demo



A counterexample

Limitations [edit]

Coordinate descent has two problems. One of them is having a non-smooth multivariable function. The following picture shows that coordinate descent iteration may get stuck at a non-stationary point if the level curves of a function are not smooth. Suppose that the algorithm is at the point (-2, -2); then there are two axis-aligned directions it can consider for taking a step, indicated by the red arrows. However, every step along these two directions will increase the objective function's value (assuming a minimization problem), so the algorithm will not take any step, even though both steps together would bring the algorithm closer to the optimum. While this example shows that coordinate descent is not necessarily convergent to the optimum, it is possible to show formal convergence under reasonable conditions.[3]



A demonstration?

```
[1] -2 -3                          [1] -2 -3
[1] -3.00001  -3.00000             [1] -3 -3
[1] -3.000010 -3.000019            [1] -3 -3
[1] -3.000029 -3.000019            [1] -3 -3
[1] -3.000029 -3.000038            [1] -3 -3
[1] -3.000048 -3.000038            [1] -3 -3
[1] -3.000048 -3.000057            [1] -3 -3
[1] -3.000066 -3.000057            [1] -3 -3
[1] -3.000066 -3.000076            [1] -3 -3
[1] -3.000085 -3.000076            [1] -3 -3
[1] -3.000085 -3.000095            [1] -3 -3
[1] -3.000104 -3.000095            [1] -3 -3
[1] -3.000104 -3.000113            [1] -3 -3
[1] -3.000123 -3.000113            [1] -3 -3
[1] -3.000123 -3.000132            [1] -3 -3
[1] -3.000141 -3.000132            [1] -3 -3
[1] -3.000141 -3.000151            [1] -3 -3
[1] -3.000160 -3.000151            [1] -3 -3
[1] -3.000160 -3.000169            [1] -3 -3
...
```

A bit of code

```
fun = function(x,y) abs(x+y) + 3*abs(x-y)
x = -2; y=-3
print(c(x,y))

tst = seq(-5,5,len=1000001)

for (k in 1:100)
{
  ## x = optimize(function(z) fun(z,y), interval=c(-5,5))$minimum
  ## print(c(x,y))
  ## y = optimize(function(z) fun(x,z), interval=c(-5,5))$minimum
  ## print(c(x,y))
  fx = fun(tst,y)
  x = tst[which(fx == min(fx))]
  print(c(x,y))
  fy = fun(x,tst)
  y = tst[which(fy == min(fy))]
  print(c(x,y))
}
```

Some afterword

"In earlier years, many optimization methods were simple enough so that it was appropriate for someone with a problem to consult a research paper, or even develop a special-purpose method, and then to write a computer program to execute steps of the method. This approach is no longer feasible today, for several reasons.

...

These developments mean that the typical person who wishes to solve an optimization problem would not (and, in our view, should not) start from scratch to devise his own optimization method or write his own implementation. Rather, he should be able to use selected routines from high-quality mathematical software libraries. However, this does not mean that the user should remain in complete ignorance of the nature of the algorithms that he will use, or the important features of optimization software."

(Gill, Murray, and Wright, 1981)