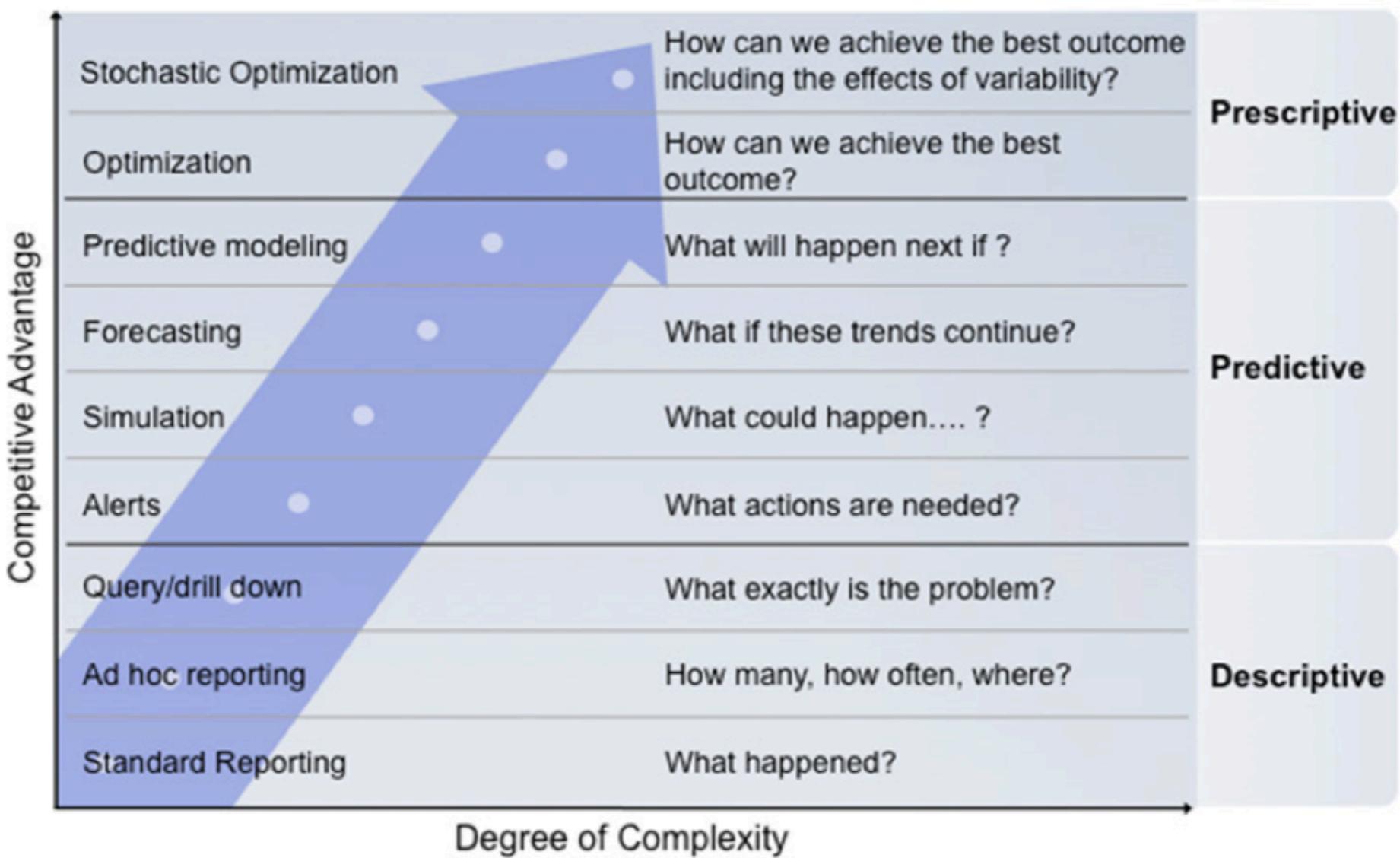


Highlights about Stochastic Optimization

- Context
- Simple examples
- Simple examples in R
- CSC801-002 syllabus

Optimization -- at the very top of *** Big Data Analytics ***



Source:

Thomas H. Davenport and Jeanne G. Harris in **Competing on Analytics: The New Science of Winning, (Harvard Business School Press, 2007)**

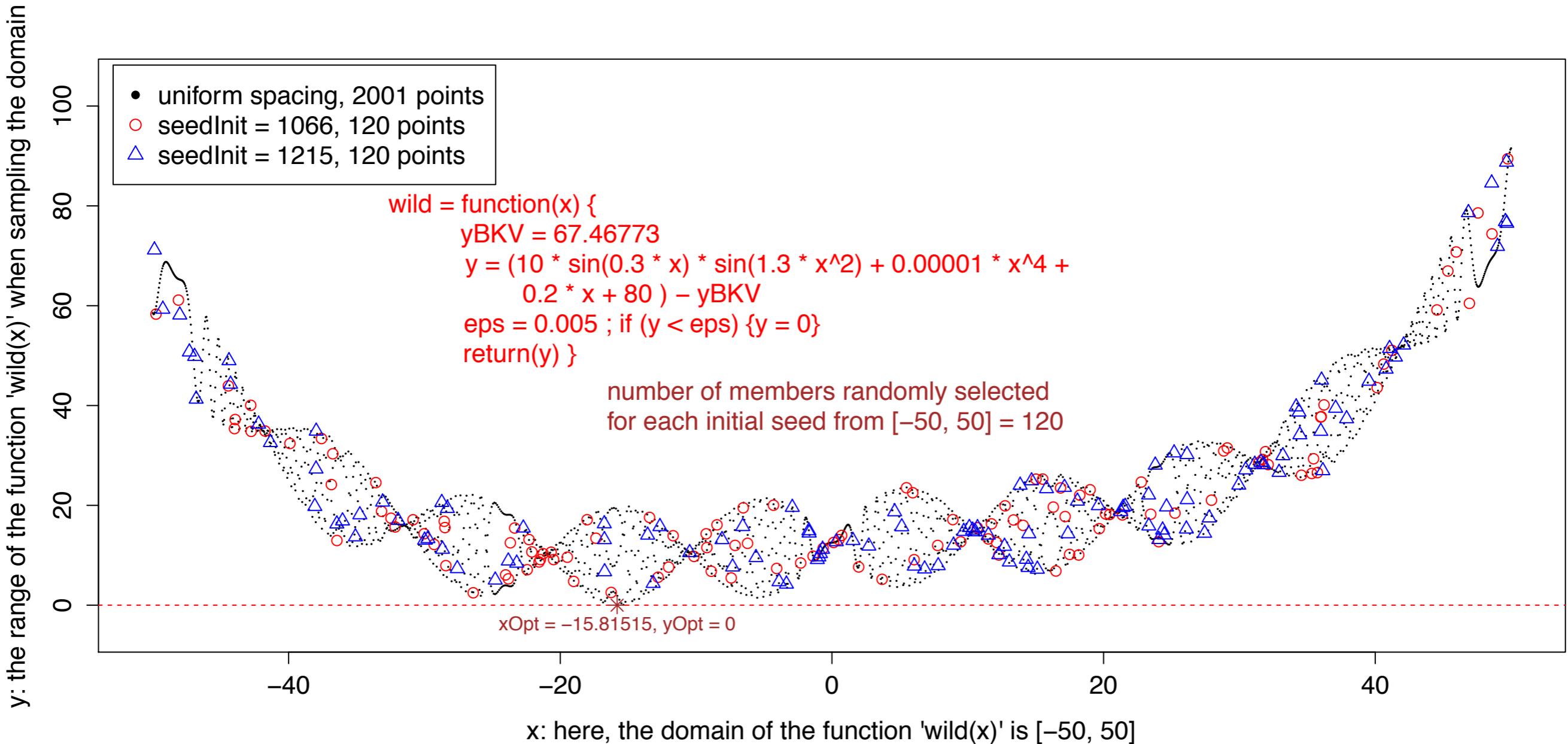
The function 'wild(x)' as part of the R-package DEoptim distribution!

Message:

finding the single optimum solution with 7 significant digits cannot be left to chance alone,

xOpt = -15.81515, yOpt = 0

-- it would take too many random samples and too much runtime!

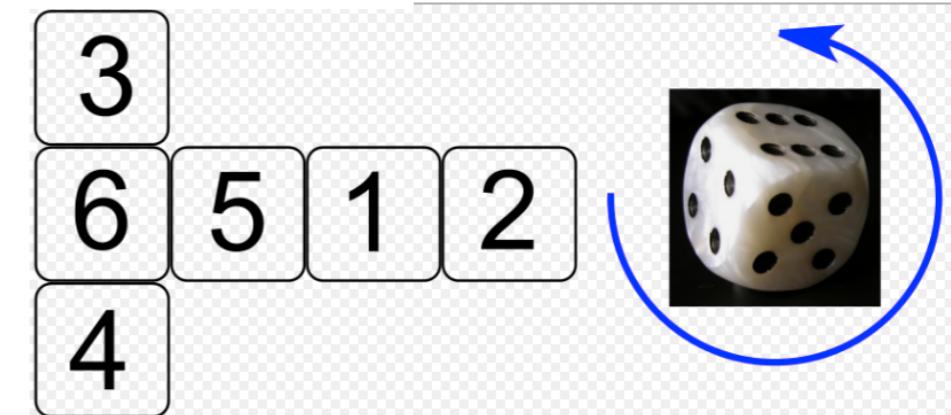


The starting point: <https://en.wikipedia.org/wiki/Dice>

coin = 2-sided dice: the game stops on the trial that “hits” the target value of head!



6-sided dice: the game stops on the trial that “hits” the target value of 6!



60-sided dice: the game stops on the trial that “hits” the target value of 60!

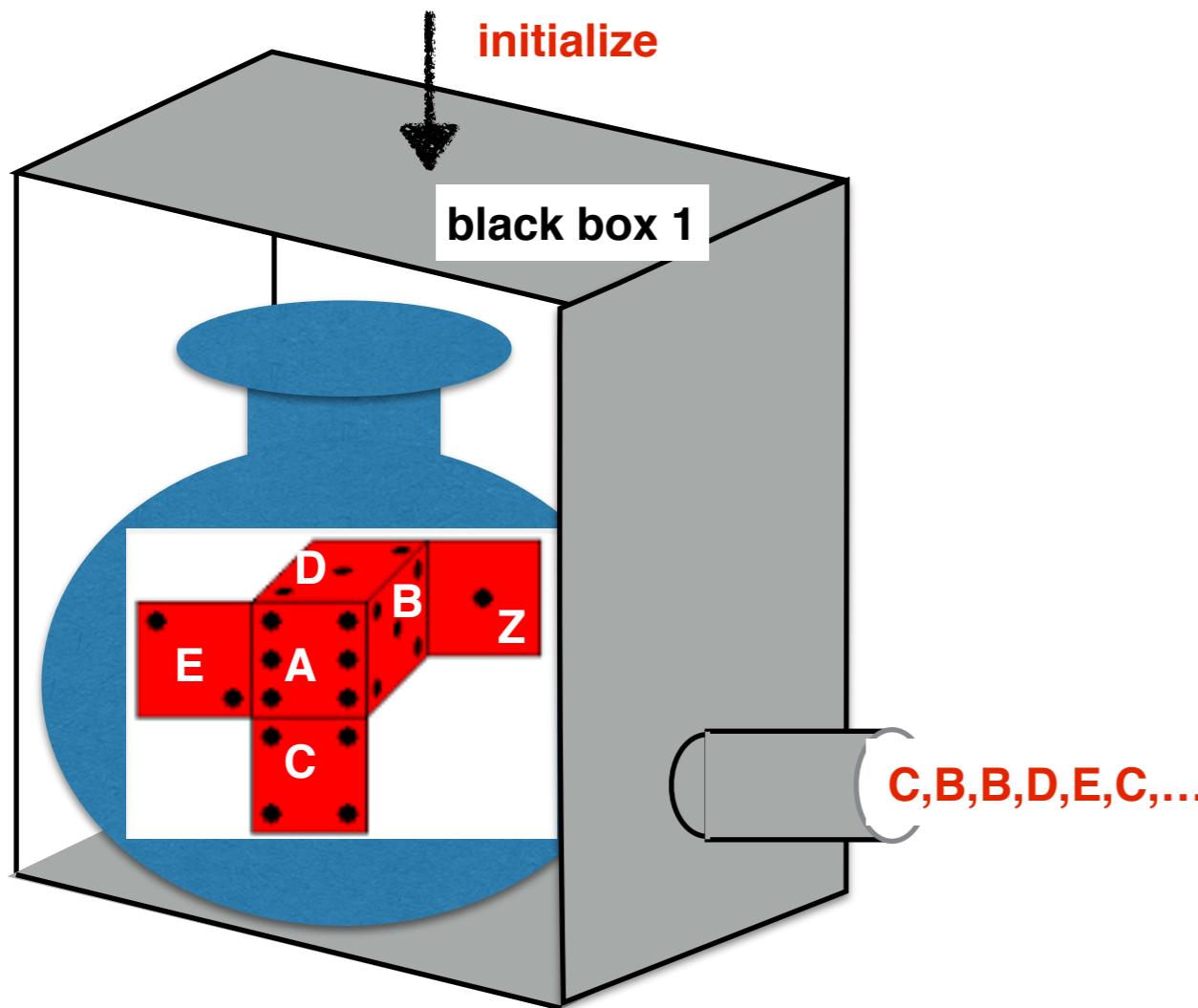
Let X denote the random variable “trial”. When throwing a dice, each trial that does not return a target value is called a “failure”. We stop the experiment on the trial that returns the target value, i.e. we record the trial that achieves the FIRST success. In this context, given that the probability of each success is p , the number of trials required to obtain the FIRST success has geometric distribution.

$$\Pr(X = k) = (1 - p)^{k-1} p$$

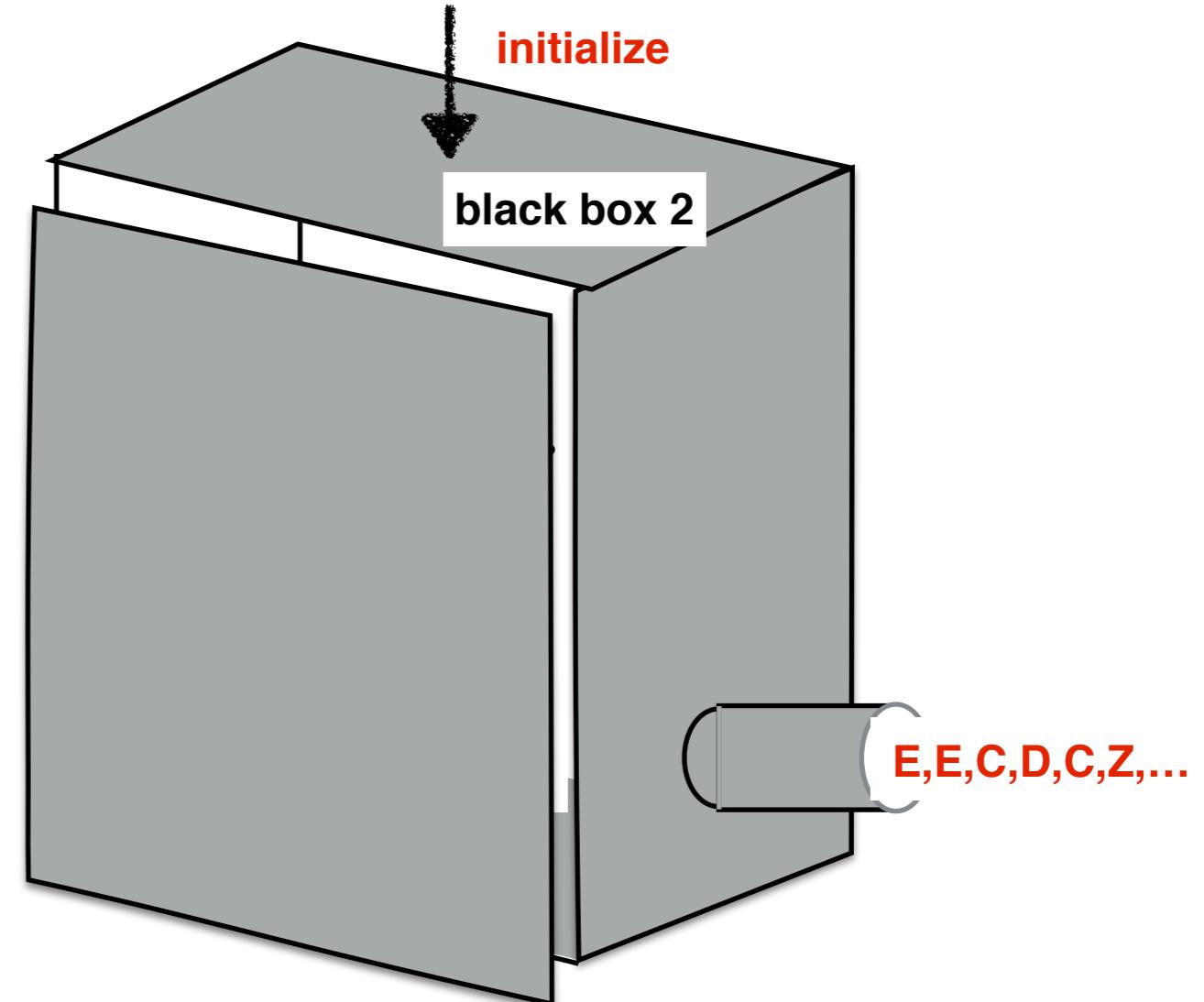
The **expected value** of a geometrically distributed random variable X is $1/p$ and the **variance** is $(1 - p)/p^2$



Are sequences from these two 'black boxes' equivalent? (1)

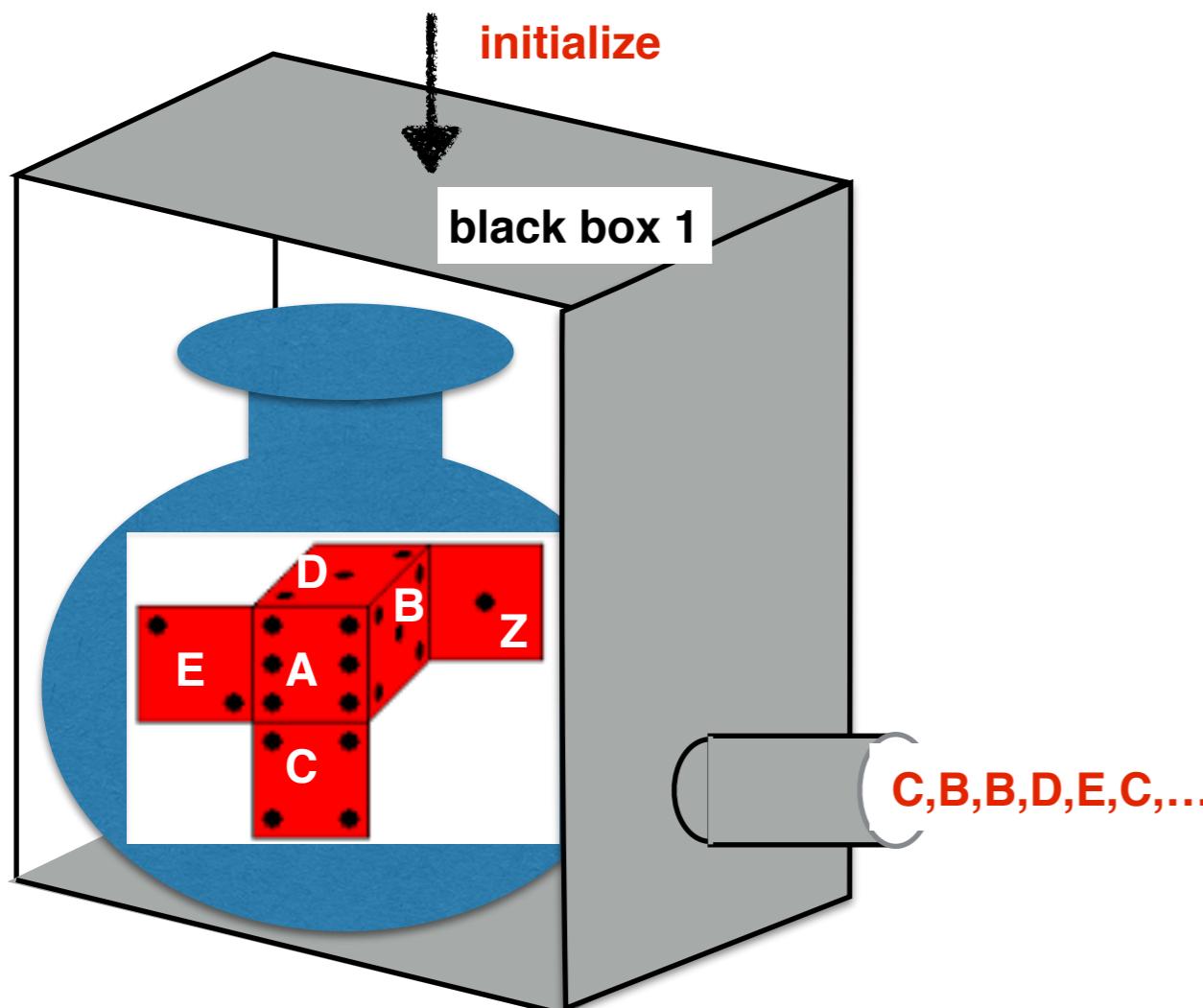


An urn-dwelling dwarf
is throwing 6-sided dice

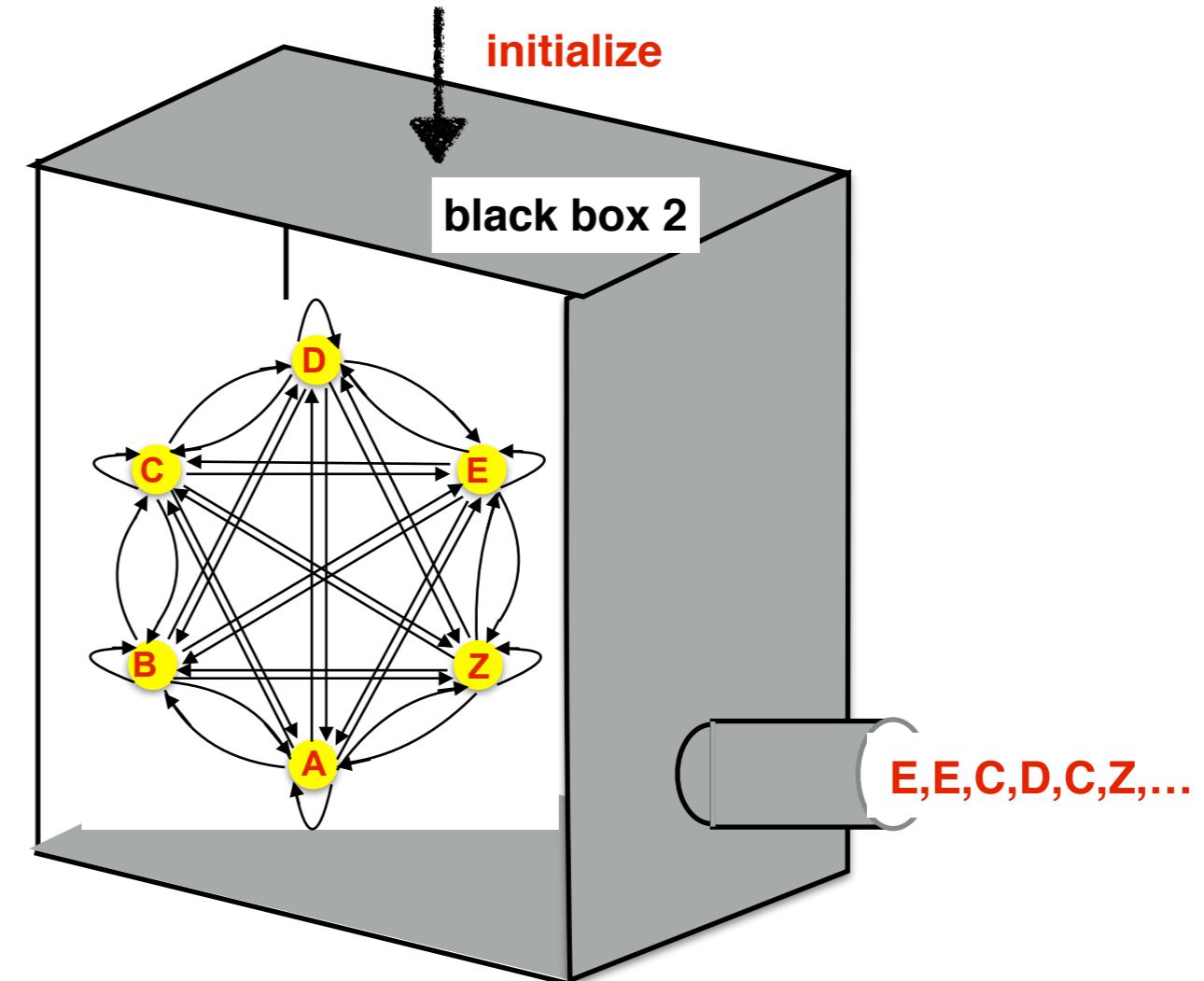


Sequences produced by this box could be
equivalent to sequences from the box on the left ...
What model produces these sequences??

Are sequences from these two 'black boxes' equivalent? (2)



An urn-dwelling dwarf
is throwing 6-sided dice

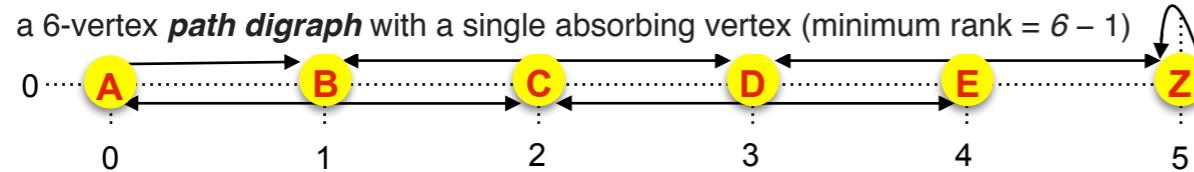


Here we have a case of random walk on a
complete graph, defined on $6 \times 6 = 36$ edges!

In other words, this is a case of an ergodic
Markov chain since it is possible to go from
every state to every state.

In this box, the dwarf is walking, i.e.
following the edges from vertex-to-vertex,
and can reach every state.

Evaluating a version of drunkard's walk (on a path graph)



file = drunk-0006-1-plateau.adjm

A	B	C	D	E	Z
0	1	0	0	0	0
1	0	1	0	0	0
0	1	0	1	0	0
0	0	1	0	1	0
0	0	0	1	0	1
0	0	0	0	0	1

graph adjacency matrix

file = drunk-0006-1-plateau.stpm

A	B	C	D	E	Z
0.0	1.0	0.0000	0.0	0.0	0.0
0.5000	0.0	0.5000	0.0	0.0	0.0
0.0	0.5000	0.0	0.5000	0.0	0.0
0.0	0.0	0.5000	0.0	0.5000	0.0
0.0	0.0	0.0	0.5000	0.0	0.5000
0.0	0.0	0.0	0.0	1.0	

state transition probability matrix

Remember:

coin == two-sided dice

A = pub

Z = home (and an absorbing state)

If in A, he must always take a step to B

If in not in A, he flips a fair coin to decide on the left/right step.

Once he reaches Z, he cannot leave home.

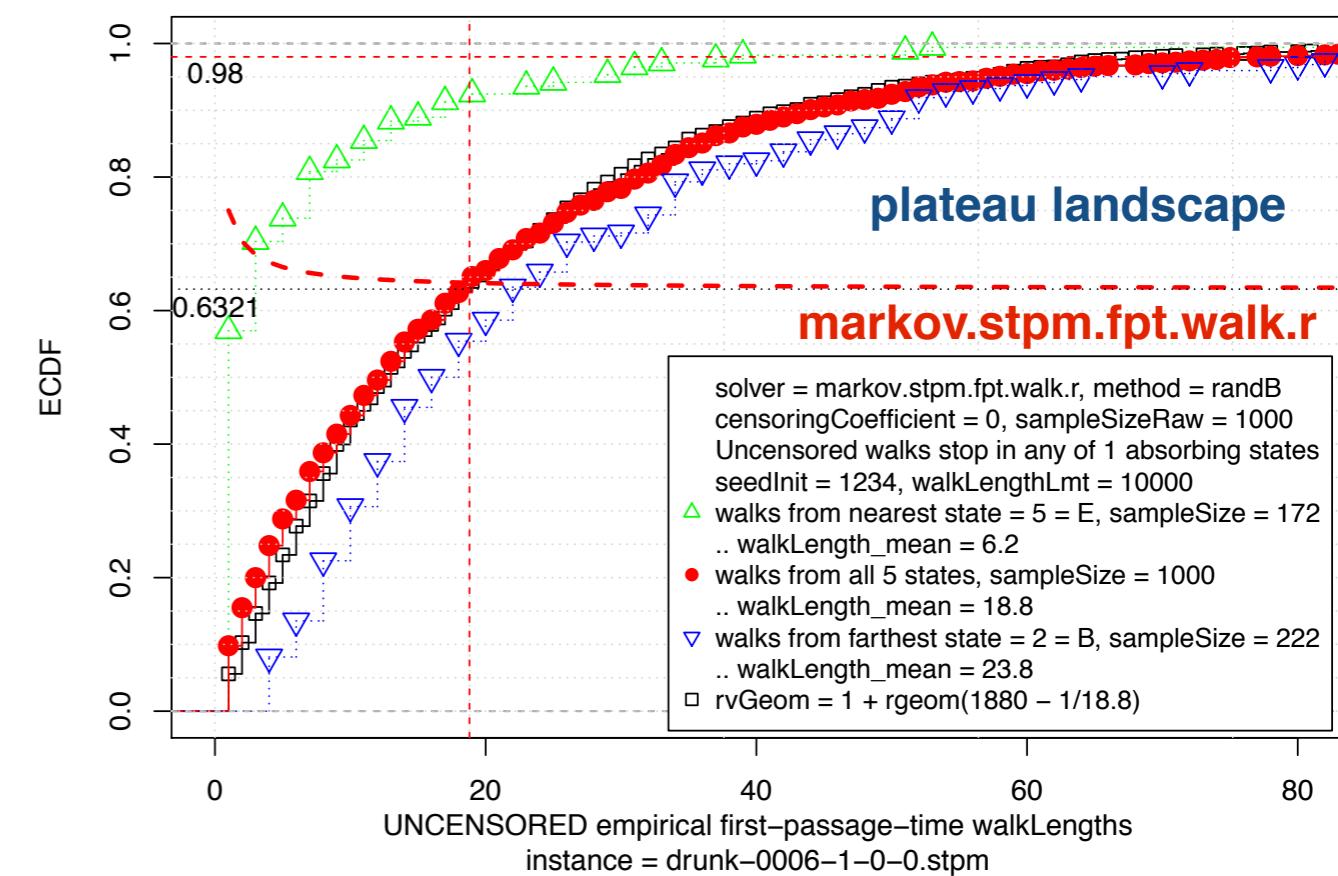
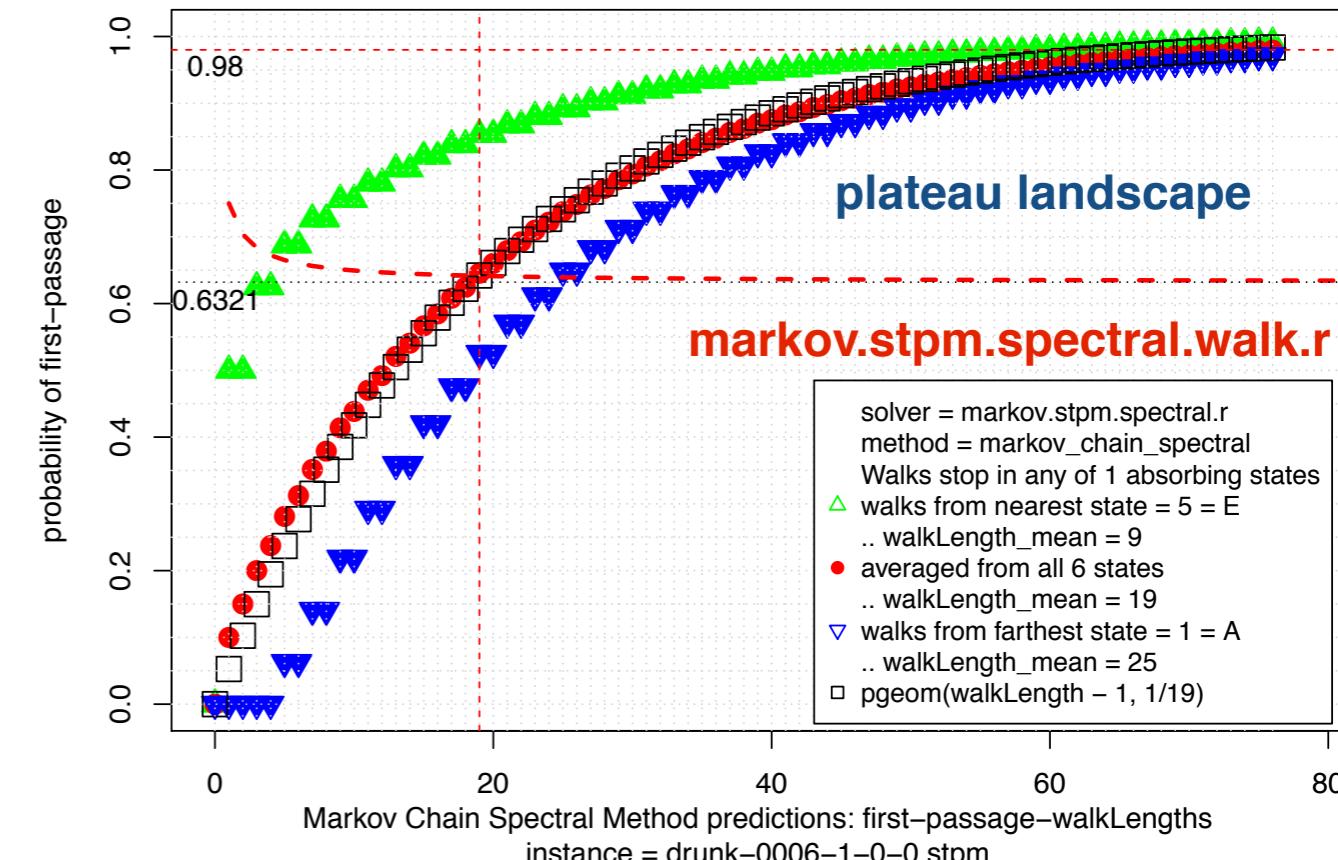
Questions about this path graph:

how many steps to reach Z

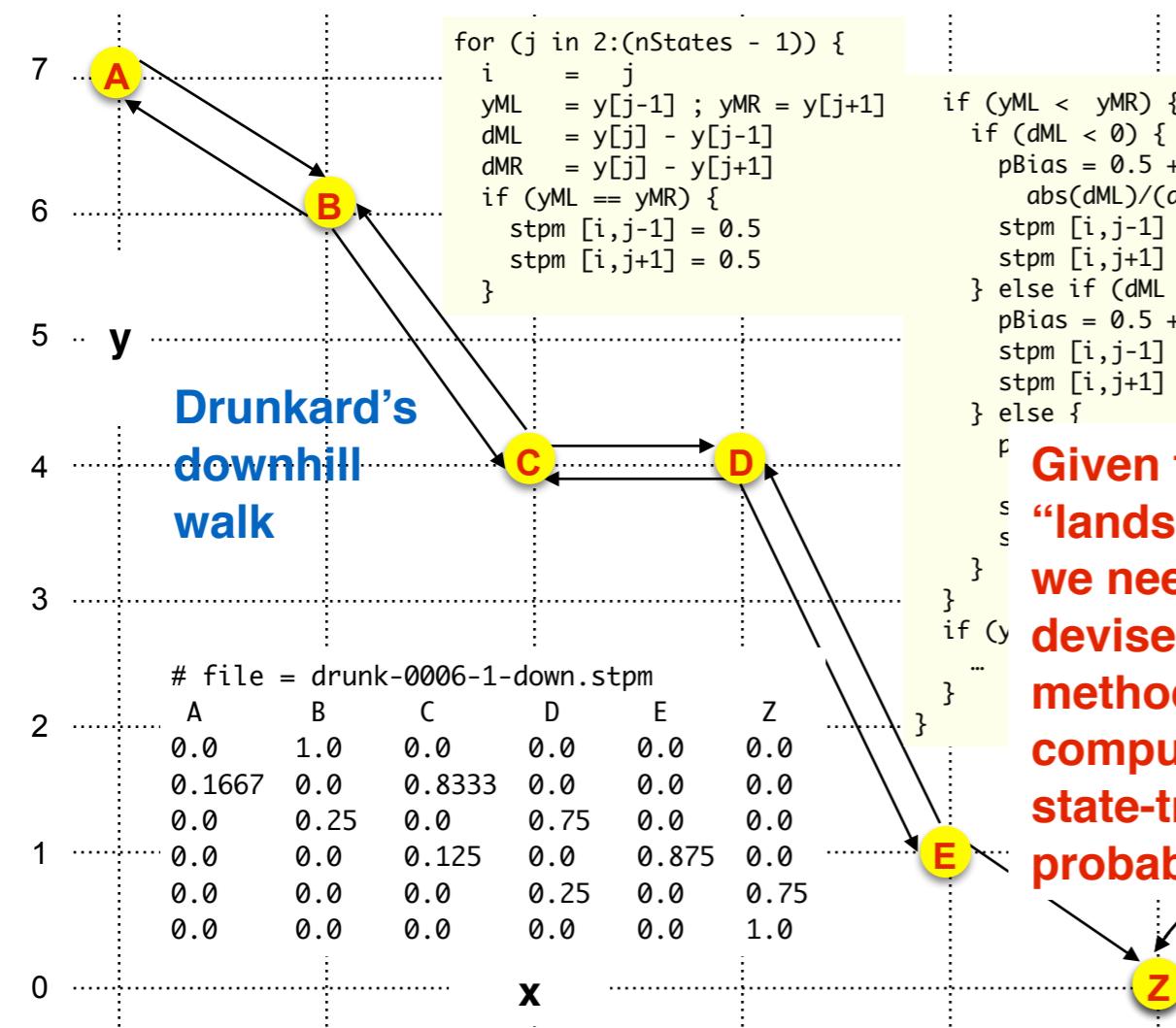
-- when vertices are on a plateau landscape?

-- when vertices are on a down-hill landscape?

-- when vertices are on a down-hill/up-hill landscape?



8

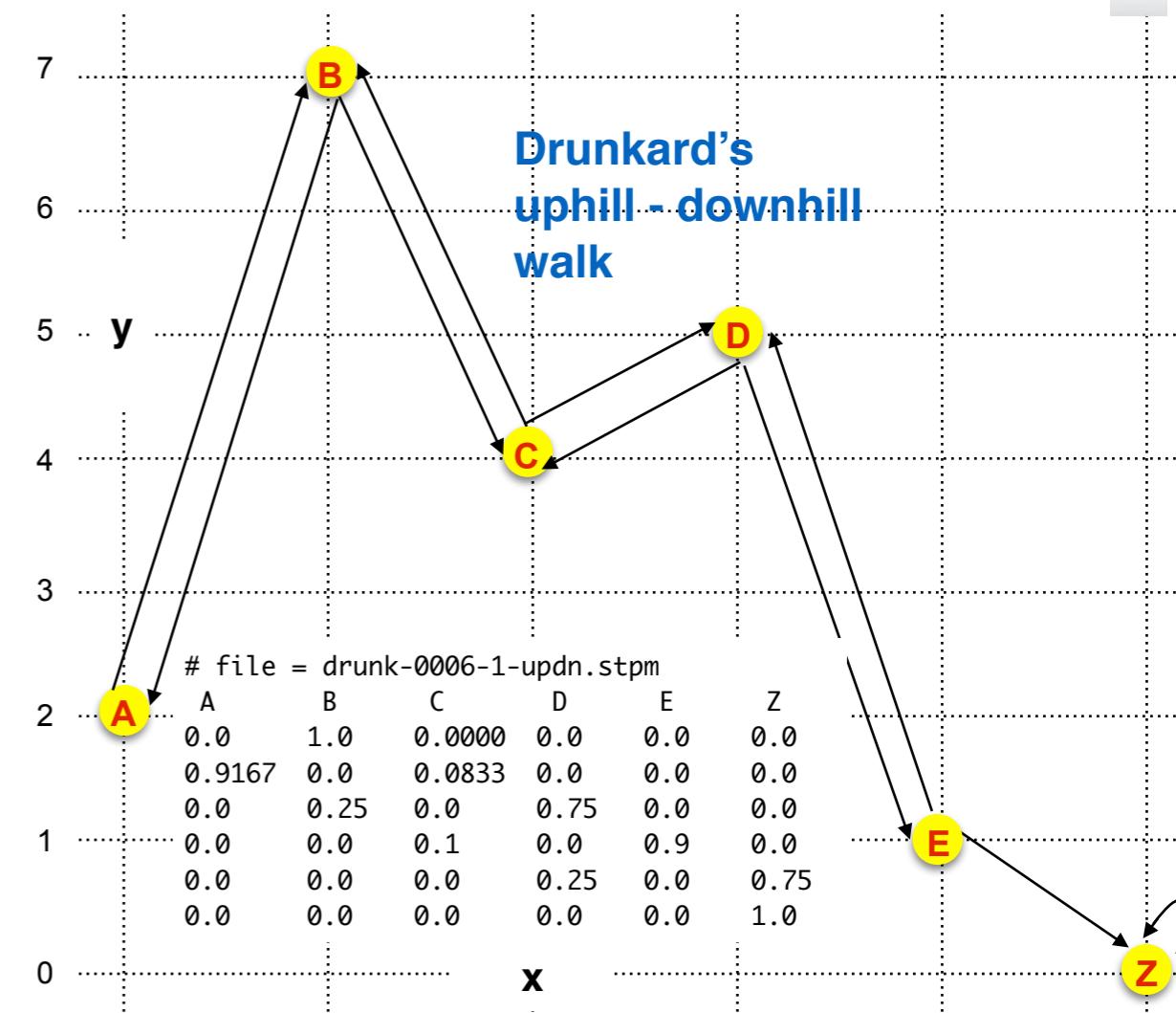
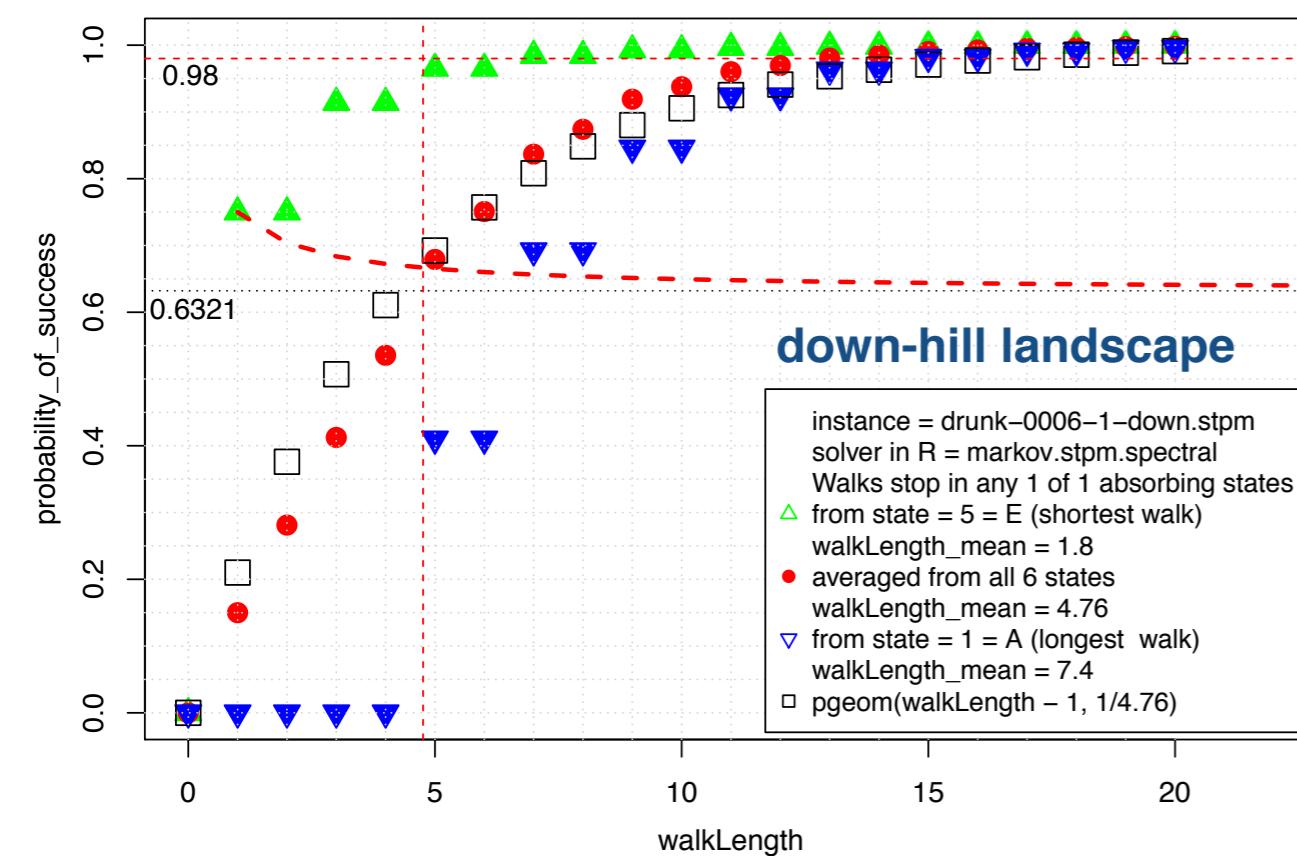


```

f (yML < yMR) {
  if (dML < 0) {
    pBias = 0.5 + 0.5*
      abs(dML)/(abs(dML) + 1)
    stpm [i,j-1] = pBias
    stpm [i,j+1] = 1 - pBias
  } else if (dML == 0) {
    pBias = 0.5 + 0.5/(1+1)
    stpm [i,j-1] = pBias
    stpm [i,j+1] = 1 - pBias
  } else {

```

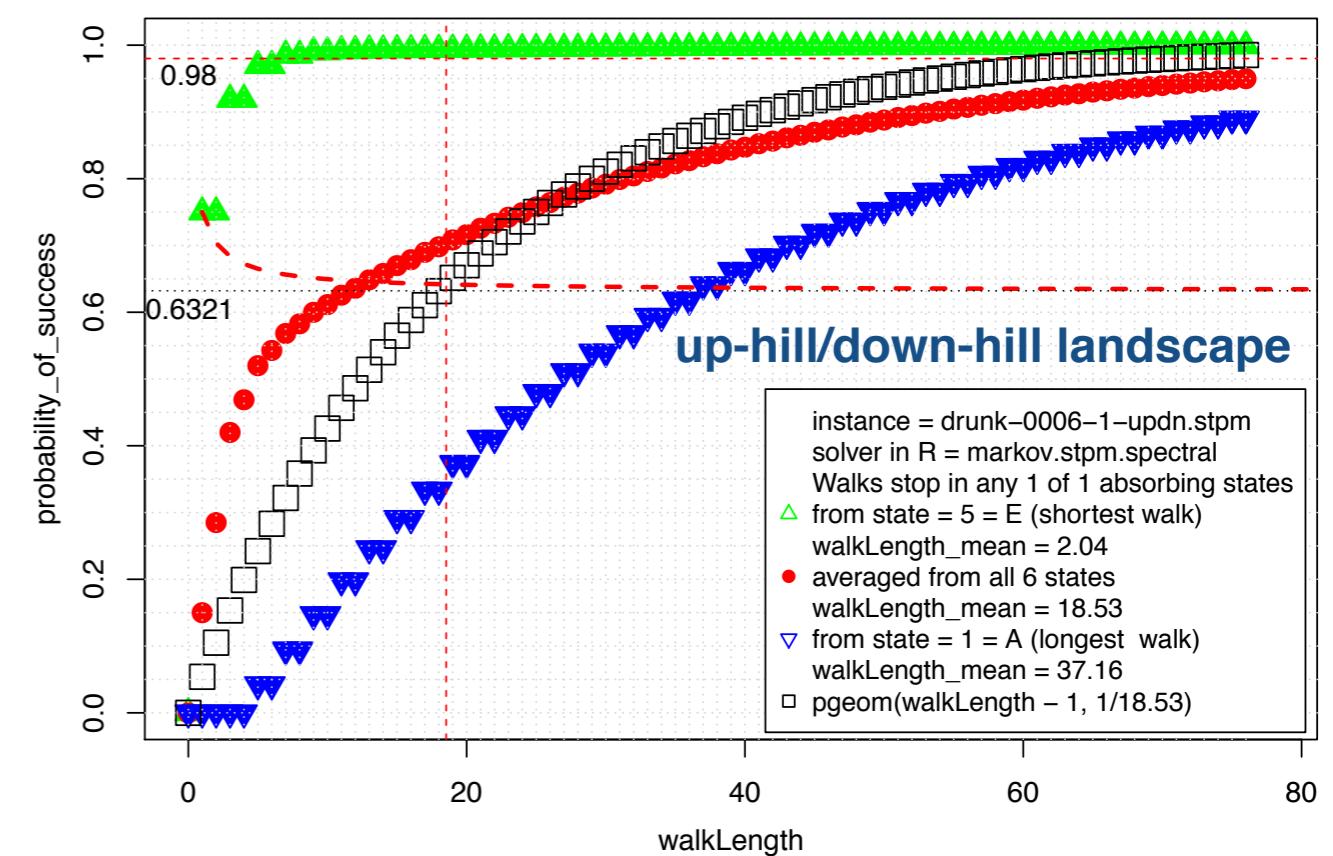
Given the “landscape”, we need to devise a method to compute state-transition probabilities!!



```

# file = drunk-0006-1-updn.stp
2 ..... A B C D
      0.0 1.0 0.0000 0.0
      0.9167 0.0 0.0833 0.0
      0.0 0.25 0.0 0.75
1 ..... 0.0 0.0 0.1 0.0
      0.0 0.0 0.0 0.25
      0.0 0.0 0.0 0.0

```



Encapsulating DEoptim for statistical experiment

R-code

```

DEoptimExp = function(funcName, nDim, upperLmt1, NP, itermax,
                      valueTarget=0, epsilon= 0.005, seedInit=-1,
                      sampleSize=10, method="DEoptim") {
  ...
  for (sampleId in seq_len(sampleSize)) {
    set.seed(seed)
    out = DEoptim(funcName, lower=lowerLmt, upper=upperLmt,
                  DEoptim.control(NP=NP, itermax=itermax, trace=F,
                  VTR=valueTarget))
    isCensored = FALSE
    if (out$optim$iter >= itermax && out$optim$bestval >= valueTarget) {
      isCensored = TRUE ; numCensored = numCensored + 1
    }
    if (!isCensored) {
      solutionCoords = c(solutionCoords, out$optim$bestmem)
      iterations = c(iterations, out$optim$iter)
    }
    row = c(sampleId, seed, nDim, out$optim$bestval,
            out$optim$nfeval, out$optim$iter, as.logical(isCensored))
    #write(row, file=fileExp, ncolumns=numCols, append=TRUE, sep="\t")
    write(row, file=fileExp, ncolumns=7, append=TRUE, sep="\t")
    # new random seed for the next sample
    seed = round(9999999*runif(1))
  }
  numSolutions = length(solutionCoords)/nDim
  solutionCoords = unique(solutionCoords)
  numSolutionsUniq = length(solutionCoords)/nDim
  cat("\n*****",
      "\n          date =", date,
      "\n          solver =", "DEoptim",
      "\n          strategy =", "default",
      ...
      "\n  numSolutionsUniq =", numSolutionsUniq,
      "\n  solutionCoordMin =", min(solutionCoords),
      "\n  solutionCoordMax =", max(solutionCoords),
      "\n          delta =", max(solutionCoords) - min(solutionCoords),
      "\n  iterations_median =", round(median(iterations),4),
      "\n  iterations_mean =", round(mean(iterations),4),
      "\n  iterations_stdev =", round(sd(iterations),4),
      "\n  iterations_cVar =", round(sd(iterations)/mean(iterations),4),
      "\n  iterations_min =", round(min(iterations),4),
      "\n  iterations_max =", round(max(iterations),4),
      "\n  solutionCoords =", solutionCoords, "\n\n")
  ...
}

```

commands/results under R-shell

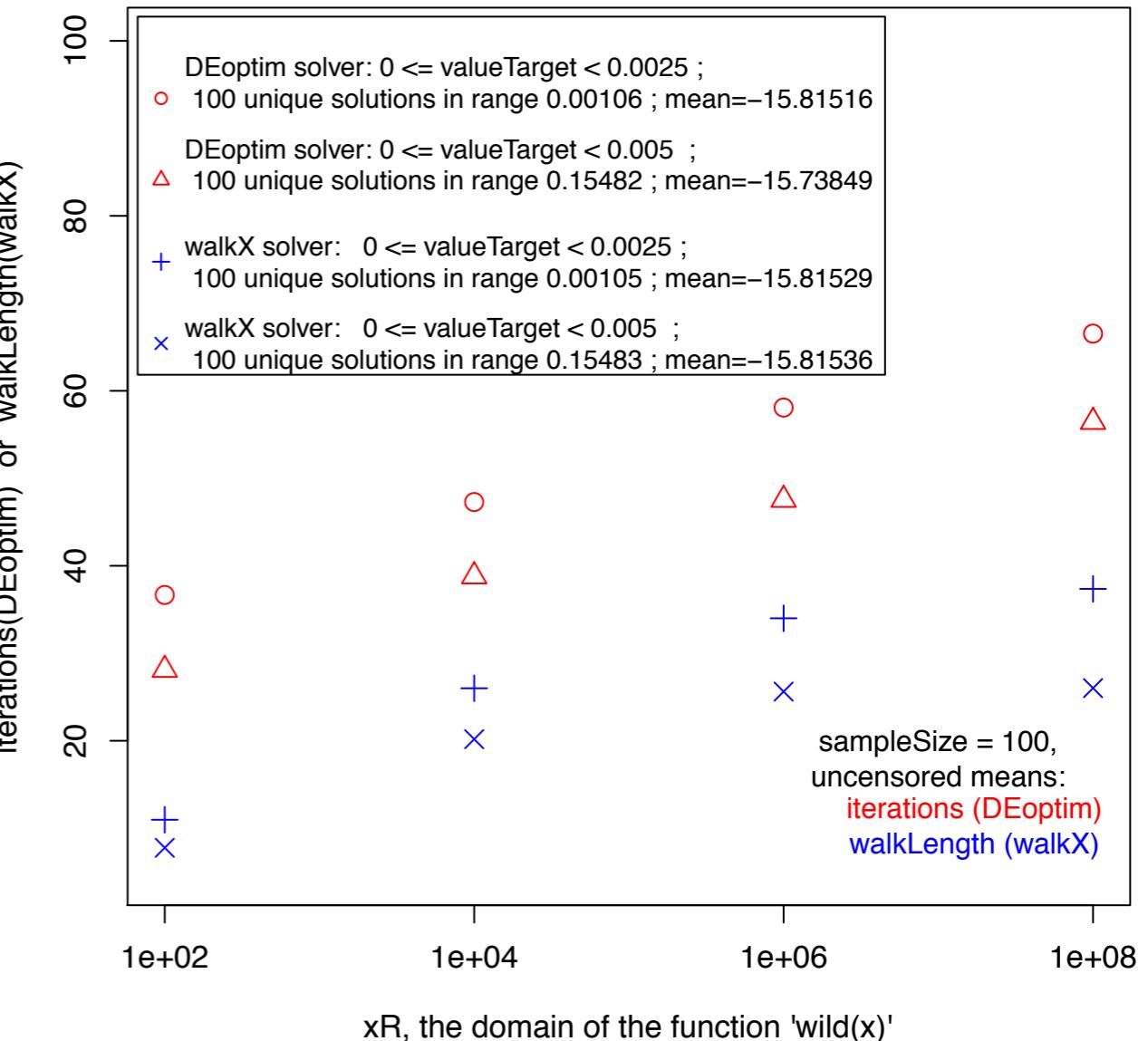
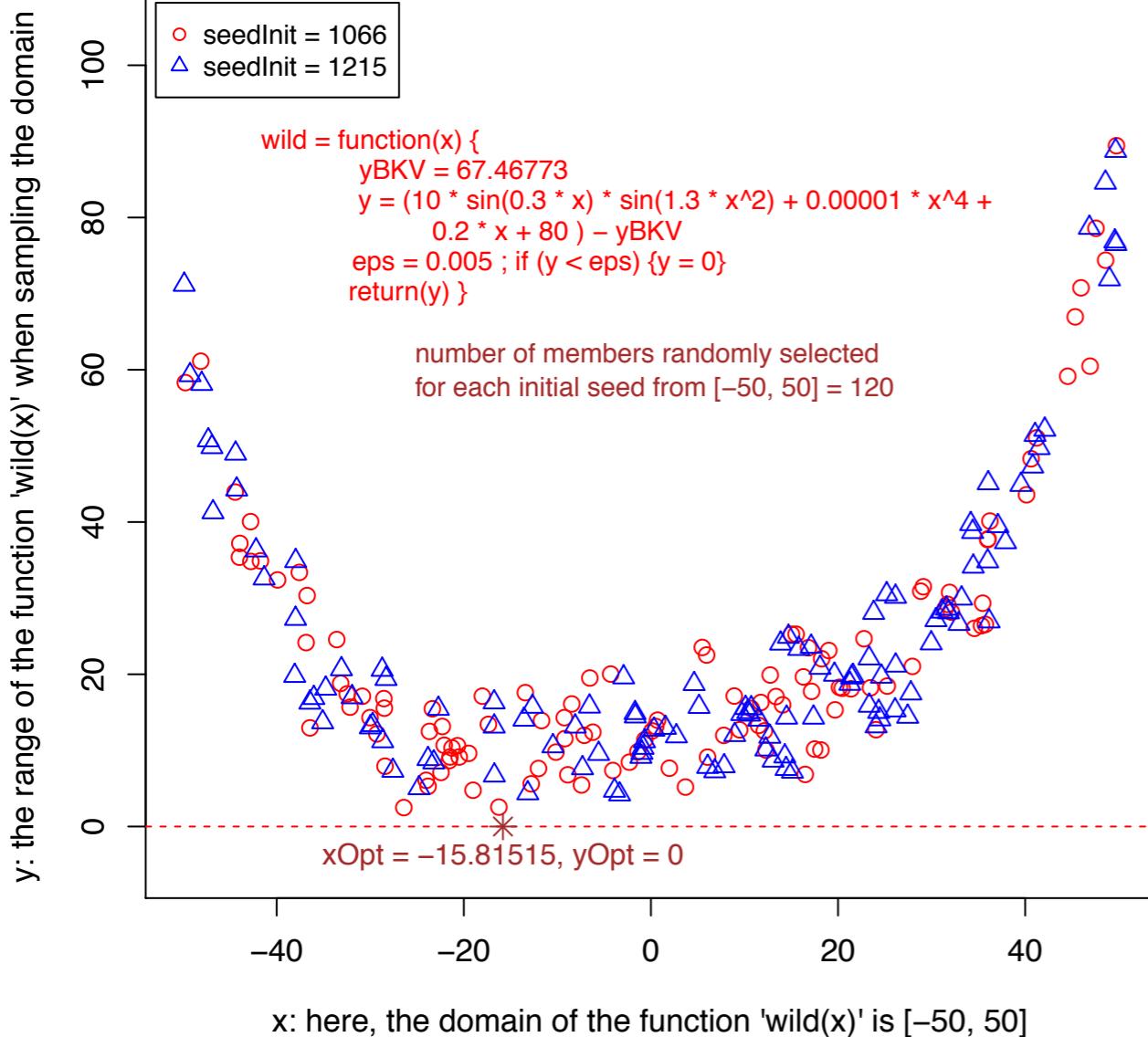
```

[1] "Fri Nov 10 12:23:00 2017"
> getwd()
[1] "/Users/brglez/Desktop/_DesktopWork/_prop2NSF-2017/_r-nsf2017"
>
> source("DEoptimExp-wild.r"); DEoptimExp("wild", nDim=1,
epsilon=0.005, upperLmt=50, NP=120, itermax=10000, valueTarget=0,
seedInit=1215, sampleSize=10)
...
*****
          date = Fri Nov 10 12:26:02 2017
          solver = DEoptim
          strategy = default
          funcName = wild
          nDim = 1
          epsilon = 0.005
          lowerLmt = -50
          upperLmt = 50
          iterationsLmt = 10000
          numPop = 120
          valueTarget = 0
          seedInit = 1215
          sampleSize = 100
          numCensored = 0
          numSolutions = 100
          numSolutionsUniq = 100
          solutionCoordMin = -15.81569
          solutionCoordMax = -15.81462
          delta = 0.001066469
          iterations_median = 37
          iterations_mean = 36.67
          iterations_stdev = 14.8916
          iterations_cVar = 0.4061
          iterations_min = 6
          iterations_max = 67
          cntProbe_mean = 4400.4
          solutionCoords = -15.8148 -15.81484 -15.8147 -15.81526
                           -15.81502 -15.8155 -15.81554 -15.8154
...
          -15.81524 -15.81567 -15.81501 -15.81486

```

*in the follow-up course, you will know how
create and package a similar solver!*

Plotting results of the experiment: DEoptim-vs-walkX



Needed:
comparisons of function evaluation counts
(currently, DEoptim reports these counts incorrectly)

R-functions considers by walkDice and walkXYZ ... (1)

```

1 markov.fpt.stpm <- function(instance, P, isSingle,
2                               stateInit, stateTarget, walkLmt,
3                               method, seedInit) {
4   colNames = colnames(P)
5
6   walkSeq <- (function(
7     stateInit, stateTarget, P, walkLmt, method) {
8     nStates = nrow(P) ; walkSeqVec = stateInit
9     isCensored = TRUE ; isTrapped = FALSE ;
10    i = stateInit ; allZeros = integer(nStates)
11
12   for (t in seq_len(walkLmt)) {
13     pivot.sel <- (function(nStates, P, i, method) {
14       row = P[i,] ; col = P[,i]
15       switch(
16         method,
17         randB={ ;# next-state from stpm
18           iNext = sample(nStates, size=1, prob=row)
19         },
20         met0={ ;# next-state under Metropolis
21           iNext = sample(nStates, 1, prob=row)
22           p_ij = P[i,iNext] ; p_ji = P[iNext,i]
23           q = p_ij/p_ji      ; u = runif(1)
24           if (q <= u) { iNext = i }
25         },
26         max1={ ;# next-state under max1 method
27           iNext = which.is.max(row)
28           row[iNext] = 0 ;# remove 1 edge
29         },
30       )
31
32     vCurr = diagVec[i]
33
34     if (isTrapped) {
35       if (vCurr > valueMax) {
36         i = stateTarget
37         break
38       }
39     }
40
41     if (isCensored) {
42       if (vCurr > valueMax) {
43         i = stateTarget
44         break
45       }
46     }
47
48     if (vCurr < valueThres) {
49       if (vNext > vCurr) {
50         if ((vNext - vCurr) > valueThres) {
51           i = stateTarget
52           break
53         }
54       }
55     }
56
57     if (vCurr < valueThres) {
58       if (vNext > vCurr) {
59         if ((vNext - vCurr) > valueThres) {
60           i = stateTarget
61           break
62         }
63       }
64     }
65
66     if (vCurr < valueThres) {
67       if (vNext > vCurr) {
68         if ((vNext - vCurr) > valueThres) {
69           i = stateTarget
70           break
71         }
72       }
73     }
74
75     if (vCurr < valueThres) {
76       if (vNext > vCurr) {
77         if ((vNext - vCurr) > valueThres) {
78           i = stateTarget
79           break
80         }
81       }
82     }
83
84     if (vCurr < valueThres) {
85       if (vNext > vCurr) {
86         if ((vNext - vCurr) > valueThres) {
87           i = stateTarget
88           break
89         }
90       }
91     }
92
93     if (vCurr < valueThres) {
94       if (vNext > vCurr) {
95         if ((vNext - vCurr) > valueThres) {
96           i = stateTarget
97           break
98         }
99       }
100    }
101
102    if (vCurr < valueThres) {
103      if (vNext > vCurr) {
104        if ((vNext - vCurr) > valueThres) {
105          i = stateTarget
106          break
107        }
108      }
109    }
110
111    if (vCurr < valueThres) {
112      if (vNext > vCurr) {
113        if ((vNext - vCurr) > valueThres) {
114          i = stateTarget
115          break
116        }
117      }
118    }
119
120    if (vCurr < valueThres) {
121      if (vNext > vCurr) {
122        if ((vNext - vCurr) > valueThres) {
123          i = stateTarget
124          break
125        }
126      }
127    }
128
129    if (vCurr < valueThres) {
130      if (vNext > vCurr) {
131        if ((vNext - vCurr) > valueThres) {
132          i = stateTarget
133          break
134        }
135      }
136    }
137
138    if (vCurr < valueThres) {
139      if (vNext > vCurr) {
140        if ((vNext - vCurr) > valueThres) {
141          i = stateTarget
142          break
143        }
144      }
145    }
146
147    if (vCurr < valueThres) {
148      if (vNext > vCurr) {
149        if ((vNext - vCurr) > valueThres) {
150          i = stateTarget
151          break
152        }
153      }
154    }
155
156    if (vCurr < valueThres) {
157      if (vNext > vCurr) {
158        if ((vNext - vCurr) > valueThres) {
159          i = stateTarget
160          break
161        }
162      }
163    }
164
165    if (vCurr < valueThres) {
166      if (vNext > vCurr) {
167        if ((vNext - vCurr) > valueThres) {
168          i = stateTarget
169          break
170        }
171      }
172    }
173
174    if (vCurr < valueThres) {
175      if (vNext > vCurr) {
176        if ((vNext - vCurr) > valueThres) {
177          i = stateTarget
178          break
179        }
180      }
181    }
182
183    if (vCurr < valueThres) {
184      if (vNext > vCurr) {
185        if ((vNext - vCurr) > valueThres) {
186          i = stateTarget
187          break
188        }
189      }
190    }
191
192    if (vCurr < valueThres) {
193      if (vNext > vCurr) {
194        if ((vNext - vCurr) > valueThres) {
195          i = stateTarget
196          break
197        }
198      }
199    }
200
201    if (vCurr < valueThres) {
202      if (vNext > vCurr) {
203        if ((vNext - vCurr) > valueThres) {
204          i = stateTarget
205          break
206        }
207      }
208    }
209
210    if (vCurr < valueThres) {
211      if (vNext > vCurr) {
212        if ((vNext - vCurr) > valueThres) {
213          i = stateTarget
214          break
215        }
216      }
217    }
218
219    if (vCurr < valueThres) {
220      if (vNext > vCurr) {
221        if ((vNext - vCurr) > valueThres) {
222          i = stateTarget
223          break
224        }
225      }
226    }
227
228    if (vCurr < valueThres) {
229      if (vNext > vCurr) {
230        if ((vNext - vCurr) > valueThres) {
231          i = stateTarget
232          break
233        }
234      }
235    }
236
237    if (vCurr < valueThres) {
238      if (vNext > vCurr) {
239        if ((vNext - vCurr) > valueThres) {
240          i = stateTarget
241          break
242        }
243      }
244    }
245
246    if (vCurr < valueThres) {
247      if (vNext > vCurr) {
248        if ((vNext - vCurr) > valueThres) {
249          i = stateTarget
250          break
251        }
252      }
253    }
254
255    if (vCurr < valueThres) {
256      if (vNext > vCurr) {
257        if ((vNext - vCurr) > valueThres) {
258          i = stateTarget
259          break
260        }
261      }
262    }
263
264    if (vCurr < valueThres) {
265      if (vNext > vCurr) {
266        if ((vNext - vCurr) > valueThres) {
267          i = stateTarget
268          break
269        }
270      }
271    }
272
273    if (vCurr < valueThres) {
274      if (vNext > vCurr) {
275        if ((vNext - vCurr) > valueThres) {
276          i = stateTarget
277          break
278        }
279      }
280    }
281
282    if (vCurr < valueThres) {
283      if (vNext > vCurr) {
284        if ((vNext - vCurr) > valueThres) {
285          i = stateTarget
286          break
287        }
288      }
289    }
290
291    if (vCurr < valueThres) {
292      if (vNext > vCurr) {
293        if ((vNext - vCurr) > valueThres) {
294          i = stateTarget
295          break
296        }
297      }
298    }
299
300    if (vCurr < valueThres) {
301      if (vNext > vCurr) {
302        if ((vNext - vCurr) > valueThres) {
303          i = stateTarget
304          break
305        }
306      }
307    }
308
309    if (vCurr < valueThres) {
310      if (vNext > vCurr) {
311        if ((vNext - vCurr) > valueThres) {
312          i = stateTarget
313          break
314        }
315      }
316    }
317
318    if (vCurr < valueThres) {
319      if (vNext > vCurr) {
320        if ((vNext - vCurr) > valueThres) {
321          i = stateTarget
322          break
323        }
324      }
325    }
326
327    if (vCurr < valueThres) {
328      if (vNext > vCurr) {
329        if ((vNext - vCurr) > valueThres) {
330          i = stateTarget
331          break
332        }
333      }
334    }
335
336    if (vCurr < valueThres) {
337      if (vNext > vCurr) {
338        if ((vNext - vCurr) > valueThres) {
339          i = stateTarget
340          break
341        }
342      }
343    }
344
345    if (vCurr < valueThres) {
346      if (vNext > vCurr) {
347        if ((vNext - vCurr) > valueThres) {
348          i = stateTarget
349          break
350        }
351      }
352    }
353
354    if (vCurr < valueThres) {
355      if (vNext > vCurr) {
356        if ((vNext - vCurr) > valueThres) {
357          i = stateTarget
358          break
359        }
360      }
361    }
362
363    if (vCurr < valueThres) {
364      if (vNext > vCurr) {
365        if ((vNext - vCurr) > valueThres) {
366          i = stateTarget
367          break
368        }
369      }
370    }
371
372    if (vCurr < valueThres) {
373      if (vNext > vCurr) {
374        if ((vNext - vCurr) > valueThres) {
375          i = stateTarget
376          break
377        }
378      }
379    }
380
381    if (vCurr < valueThres) {
382      if (vNext > vCurr) {
383        if ((vNext - vCurr) > valueThres) {
384          i = stateTarget
385          break
386        }
387      }
388    }
389
390    if (vCurr < valueThres) {
391      if (vNext > vCurr) {
392        if ((vNext - vCurr) > valueThres) {
393          i = stateTarget
394          break
395        }
396      }
397    }
398
399    if (vCurr < valueThres) {
400      if (vNext > vCurr) {
401        if ((vNext - vCurr) > valueThres) {
402          i = stateTarget
403          break
404        }
405      }
406    }
407
408    if (vCurr < valueThres) {
409      if (vNext > vCurr) {
410        if ((vNext - vCurr) > valueThres) {
411          i = stateTarget
412          break
413        }
414      }
415    }
416
417    if (vCurr < valueThres) {
418      if (vNext > vCurr) {
419        if ((vNext - vCurr) > valueThres) {
420          i = stateTarget
421          break
422        }
423      }
424    }
425
426    if (vCurr < valueThres) {
427      if (vNext > vCurr) {
428        if ((vNext - vCurr) > valueThres) {
429          i = stateTarget
430          break
431        }
432      }
433    }
434
435    if (vCurr < valueThres) {
436      if (vNext > vCurr) {
437        if ((vNext - vCurr) > valueThres) {
438          i = stateTarget
439          break
440        }
441      }
442    }
443
444    if (vCurr < valueThres) {
445      if (vNext > vCurr) {
446        if ((vNext - vCurr) > valueThres) {
447          i = stateTarget
448          break
449        }
450      }
451    }
452
453    if (vCurr < valueThres) {
454      if (vNext > vCurr) {
455        if ((vNext - vCurr) > valueThres) {
456          i = stateTarget
457          break
458        }
459      }
460    }
461
462    if (vCurr < valueThres) {
463      if (vNext > vCurr) {
464        if ((vNext - vCurr) > valueThres) {
465          i = stateTarget
466          break
467        }
468      }
469    }
470
471    if (vCurr < valueThres) {
472      if (vNext > vCurr) {
473        if ((vNext - vCurr) > valueThres) {
474          i = stateTarget
475          break
476        }
477      }
478    }
479
480    if (vCurr < valueThres) {
481      if (vNext > vCurr) {
482        if ((vNext - vCurr) > valueThres) {
483          i = stateTarget
484          break
485        }
486      }
487    }
488
489    if (vCurr < valueThres) {
490      if (vNext > vCurr) {
491        if ((vNext - vCurr) > valueThres) {
492          i = stateTarget
493          break
494        }
495      }
496    }
497
498    if (vCurr < valueThres) {
499      if (vNext > vCurr) {
500        if ((vNext - vCurr) > valueThres) {
501          i = stateTarget
502          break
503        }
504      }
505    }
506
507    if (vCurr < valueThres) {
508      if (vNext > vCurr) {
509        if ((vNext - vCurr) > valueThres) {
510          i = stateTarget
511          break
512        }
513      }
514    }
515
516    if (vCurr < valueThres) {
517      if (vNext > vCurr) {
518        if ((vNext - vCurr) > valueThres) {
519          i = stateTarget
520          break
521        }
522      }
523    }
524
525    if (vCurr < valueThres) {
526      if (vNext > vCurr) {
527        if ((vNext - vCurr) > valueThres) {
528          i = stateTarget
529          break
530        }
531      }
532    }
533
534    if (vCurr < valueThres) {
535      if (vNext > vCurr) {
536        if ((vNext - vCurr) > valueThres) {
537          i = stateTarget
538          break
539        }
540      }
541    }
542
543    if (vCurr < valueThres) {
544      if (vNext > vCurr) {
545        if ((vNext - vCurr) > valueThres) {
546          i = stateTarget
547          break
548        }
549      }
550    }
551
552    if (vCurr < valueThres) {
553      if (vNext > vCurr) {
554        if ((vNext - vCurr) > valueThres) {
555          i = stateTarget
556          break
557        }
558      }
559    }
560
561    if (vCurr < valueThres) {
562      if (vNext > vCurr) {
563        if ((vNext - vCurr) > valueThres) {
564          i = stateTarget
565          break
566        }
567      }
568    }
569
570    if (vCurr < valueThres) {
571      if (vNext > vCurr) {
572        if ((vNext - vCurr) > valueThres) {
573          i = stateTarget
574          break
575        }
576      }
577    }
578
579    if (vCurr < valueThres) {
580      if (vNext > vCurr) {
581        if ((vNext - vCurr) > valueThres) {
582          i = stateTarget
583          break
584        }
585      }
586    }
587
588    if (vCurr < valueThres) {
589      if (vNext > vCurr) {
590        if ((vNext - vCurr) > valueThres) {
591          i = stateTarget
592          break
593        }
594      }
595    }
596
597    if (vCurr < valueThres) {
598      if (vNext > vCurr) {
599        if ((vNext - vCurr) > valueThres) {
600          i = stateTarget
601          break
602        }
603      }
604    }
605
606    if (vCurr < valueThres) {
607      if (vNext > vCurr) {
608        if ((vNext - vCurr) > valueThres) {
609          i = stateTarget
610          break
611        }
612      }
613    }
614
615    if (vCurr < valueThres) {
616      if (vNext > vCurr) {
617        if ((vNext - vCurr) > valueThres) {
618          i = stateTarget
619          break
620        }
621      }
622    }
623
624    if (vCurr < valueThres) {
625      if (vNext > vCurr) {
626        if ((vNext - vCurr) > valueThres) {
627          i = stateTarget
628          break
629        }
630      }
631    }
632
633    if (vCurr < valueThres) {
634      if (vNext > vCurr) {
635        if ((vNext - vCurr) > valueThres) {
636          i = stateTarget
637          break
638        }
639      }
640    }
641
642    if (vCurr < valueThres) {
643      if (vNext > vCurr) {
644        if ((vNext - vCurr) > valueThres) {
645          i = stateTarget
646          break
647        }
648      }
649    }
650
651    if (vCurr < valueThres) {
652      if (vNext > vCurr) {
653        if ((vNext - vCurr) > valueThres) {
654          i = stateTarget
655          break
656        }
657      }
658    }
659
660    if (vCurr < valueThres) {
661      if (vNext > vCurr) {
662        if ((vNext - vCurr) > valueThres) {
663          i = stateTarget
664          break
665        }
666      }
667    }
668
669    if (vCurr < valueThres) {
670      if (vNext > vCurr) {
671        if ((vNext - vCurr) > valueThres) {
672          i = stateTarget
673          break
674        }
675      }
676    }
677
678    if (vCurr < valueThres) {
679      if (vNext > vCurr) {
680        if ((vNext - vCurr) > valueThres) {
681          i = stateTarget
682          break
683        }
684      }
685    }
686
687    if (vCurr < valueThres) {
688      if (vNext > vCurr) {
689        if ((vNext - vCurr) > valueThres) {
690          i = stateTarget
691          break
692        }
693      }
694    }
695
696    if (vCurr < valueThres) {
697      if (vNext > vCurr) {
698        if ((vNext - vCurr) > valueThres) {
699          i = stateTarget
700          break
701        }
702      }
703    }
704
705    if (vCurr < valueThres) {
706      if (vNext > vCurr) {
707        if ((vNext - vCurr) > valueThres) {
708          i = stateTarget
709          break
710        }
711      }
712    }
713
714    if (vCurr < valueThres) {
715      if (vNext > vCurr) {
716        if ((vNext - vCurr) > valueThres) {
717          i = stateTarget
718          break
719        }
720      }
721    }
722
723    if (vCurr < valueThres) {
724      if (vNext > vCurr) {
725        if ((vNext - vCurr) > valueThres) {
726          i = stateTarget
727          break
728        }
729      }
730    }
731
732    if (vCurr < valueThres) {
733      if (vNext > vCurr) {
734        if ((vNext - vCurr) > valueThres) {
735          i = stateTarget
736          break
737        }
738      }
739    }
740
741    if (vCurr < valueThres) {
742      if (vNext > vCurr) {
743        if ((vNext - vCurr) > valueThres) {
744          i = stateTarget
745          break
746        }
747      }
748    }
749
750    if (vCurr < valueThres) {
751      if (vNext > vCurr) {
752        if ((vNext - vCurr) > valueThres) {
753          i = stateTarget
754          break
755        }
756      }
757    }
758
759    if (vCurr < valueThres) {
760      if (vNext > vCurr) {
761        if ((vNext - vCurr) > valueThres) {
762          i = stateTarget
763          break
764        }
765      }
766    }
767
768    if (vCurr < valueThres) {
769      if (vNext > vCurr) {
770        if ((vNext - vCurr) > valueThres) {
771          i = stateTarget
772          break
773        }
774      }
775    }
776
777    if (vCurr < valueThres) {
778      if (vNext > vCurr) {
779        if ((vNext - vCurr) > valueThres) {
780          i = stateTarget
781          break
782        }
783      }
784    }
785
786    if (vCurr < valueThres) {
787      if (vNext > vCurr) {
788        if ((vNext - vCurr) > valueThres) {
789          i = stateTarget
790          break
791        }
792      }
793    }
794
795    if (vCurr < valueThres) {
796      if (vNext > vCurr) {
797        if ((vNext - vCurr) > valueThres) {
798          i = stateTarget
799          break
800        }
801      }
802    }
803
804    if (vCurr < valueThres) {
805      if (vNext > vCurr) {
806        if ((vNext - vCurr) > valueThres) {
807          i = stateTarget
808          break
809        }
810      }
811    }
812
813    if (vCurr < valueThres) {
814      if (vNext > vCurr) {
815        if ((vNext - vCurr) > valueThres) {
816          i = stateTarget
817          break
818        }
819      }
820    }
821
822    if (vCurr < valueThres) {
823      if (vNext > vCurr) {
824        if ((vNext - vCurr) > valueThres) {
825          i = stateTarget
826          break
827        }
828      }
829    }
830
831    if (vCurr < valueThres) {
832      if (vNext > vCurr) {
833        if ((vNext - vCurr) > valueThres) {
834          i = stateTarget
835          break
836        }
837      }
838    }
839
840    if (vCurr < valueThres) {
841      if (vNext > vCurr) {
842        if ((vNext - vCurr) > valueThres) {
843          i = stateTarget
844          break
845        }
846      }
847    }
848
849    if (vCurr < valueThres) {
850      if (vNext > vCurr) {
851        if ((vNext - vCurr) > valueThres) {
852          i = stateTarget
853          break
854        }
855      }
856    }
857
858    if (vCurr < valueThres) {
859      if (vNext > vCurr) {
860        if ((vNext - vCurr) > valueThres) {
861          i = stateTarget
862          break
863        }
864      }
865    }
866
867    if (vCurr < valueThres) {
868      if (vNext > vCurr) {
869        if ((vNext - vCurr) > valueThres) {
870          i = stateTarget
871          break
872        }
873      }
874    }
875
876    if (vCurr < valueThres) {
877      if (vNext > vCurr) {
878        if ((vNext - vCurr) > valueThres) {
879          i = stateTarget
880          break
881        }
882      }
883    }
884
885    if (vCurr < valueThres) {
886      if (vNext > vCurr) {
887        if ((vNext - vCurr) > valueThres) {
888          i = stateTarget
889          break
890        }
891      }
892    }
893
894    if (vCurr < valueThres) {
895      if (vNext > vCurr) {
896        if ((vNext - vCurr) > valueThres) {
897          i = stateTarget
898          break
899       
```

R-functions considers by walkDice and walkXYZ ... (1)

```

30     max2={ ;# next-state under max2 method      30     min1={ ;# next-state pivot min1 method
31         iNext = which.is.max(row)            31         iNext = which.is.min(row)
32         row[iNext] = 0                      32         row[iNext] = valueMax ;# remove 1 edge
33         col[iNext] = 0 ;# remove 2 edges       33         },
34     },                                         34         min2={ ;# next-state under min2 method
35         max3={ ;# next-state under max3 method  35         iNext = which.is.min(row)
36             iNext = which.is.max(row)           36         row[iNext] = valueMax ;
37             row = allZeros ;# set to 0 entire row 37         col[iNext] = valueMax ;# remove 2 edges
38             col = allZeros ;# set to 0 entire col 38         },
39         }                                         39         min3={ ;# next-state under min3 method
40     )                                         40         iNext = which.is.min(row)
41         return(list(iNext, row, col))          41         vNext = P[i,iNext]
42     })(nStates, P, i, method)                 42         row = allMax ; col = allMax
43
44     iNext=pivot.sel[[1]]                     43
45     P[i,]=pivot.sel[[2]] ; P[,i]=pivot.sel[[3]] 44
46     i    =iNext   ;  walkSeqVec=c(walkSeqVec, i) 45     return(list(iNext, vNext, row, col))
47
48 # first-passage-time break                46     })(nStates, P, i, method)
49 if (i==stateTarget) {isCensored=FALSE ; break} 47     iNext=pivot.sel[[1]] ; vNext=pivot.sel[[2]]
50 # first occurrence of a trapped pivot break 48     P[i,]=pivot.sel[[3]] ; P[,i]=pivot.sel[[4]]
51 if (method == "max3") {                   49     i = iNext ; vCurr=vNext;
52     if (length(setdiff(P[i,],allZeros))==0      50     walkSeqVec = c(walkSeqVec, i)
53         && length(setdiff(P[,i],allZeros))==0) { 51     # first-passage-time break
54         isTrapped=TRUE ; isCensored=FALSE ; break 52     if (i==stateTarget) {isCensored=FALSE ; break}
55     }                                         53     # first occurrence of a trapped pivot break
56 }                                         54     if (method == "min3") {
57 }                                         55     if (length(setdiff(P[i,],allMax))==0
58     return(list(walkSeqVec,isCensored,isTrapped,P)) 56         && length(setdiff(P[,i],allMax))==0) {
59 })(stateInit, stateTarget, P, walkLmt, method) 57         isTrapped=TRUE ; isCensored=FALSE ; break
60 #
61 #
62 #
63 # summarize results of this walk
64 walkSeqVec=walkSeq[[1]] ; P = walkSeq[[4]]      58
65 isCensored=walkSeq[[2]] ; isTrapped =walkSeq[[3]] 59     }
66 walkLength=length(walkSeqVec) - 1               60     }
67 ....
68 } ;# end of markov.fpt.stpm                  61     return(list(walkSeqVec,isCensored,isTrapped,P))
69 })(stateInit, stateTarget, P, walkLmt, method)      62 })(stateInit, stateTarget, P, walkLmt, method)
70
71 #
72 #
73 #
74 # summarize results of this walk
75 walkSeqVec=walkSeq[[1]] ; P=walkSeq[[4]]          63     # summarize results of this walk
76 isCensored=walkSeq[[2]] ; isTrapped=walkSeq[[3]]  64     walkSeqVec=walkSeq[[1]] ; P=walkSeq[[4]]
77 walkLength=length(walkSeqVec) - 1                 65     isCensored=walkSeq[[2]] ; isTrapped=walkSeq[[3]]
78 ....
79 } ;# end of markov.fpt.adjm

```

CSC801-002

Stochastic Optimization I: State of the Art and Beyond

Spring 2018

Course Syllabus (tentative)

3 credits

Instructor: Franc Brglez, Ph.D.

Office: Monteith Research Center, Room 456

Phone: 919-515-9675

Instructor E-mail: brglez@ncsu.edu

Lectures:

- Fridays 1:45 PM – 3:00 PM ; 3:15 PM – 4:30 PM
- room in EB2 TBD
- enrollment is limited, see below

Office Hours: TBD

Summary: This is a research-oriented, project-based, and limited-enrollment course on stochastic optimization. The textbook provides a ready-to-start framework in R. The course will not only cover the relevant algorithms/solvers and optimization problems introduced in the textbook but also explore new solver prototypes in R that will challenge and potentially outperform algorithms published in the book and elsewhere.

The prototypes rely on the markov chain framework by introducing, in discrete and continuous variable domains, a number of walk strategies. The experimental designs measure, in the context of standardized families of problems that are increasing in size, the asymptotic complexity of these algorithms in terms of the *uncensored mean first-passage-time*.

In the continuous domain, such families may include

- a variety of hard test functions,
- instances of Lennard-Jones clusters,
- clusters of folded proteins,
- hard instance brought to class by students.

In the discrete domain, such families may include

- instances of the knapsack problem,
- the maximum graph clique problem,
- the maximum satisfiability problem,
- the linear ordering problem,
- the job-shop scheduling problem,
- hard instance brought to class by students.

Textbooks:

Modern Optimization with R, Paulo Cortez, Springer-Verlag, 2014. The pdf file is freely downloadable from the NCSU library. For a review, see <https://www.jstatsoft.org/article/view/v070b03>

Markov Chains and Random Walks: two chapters from *Introduction to Probability* by Charles M. Grinstead and J. Laurie Snell, 2-nd revised edition, AMS, 2000. The pdf file of this book is freely available at http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/amsbook.mac.pdf

Supplementary Resources:

The R-language environment and packages described in the textbook on optimization are freely available from <http://www.r-project.org>

The material that complements and extends the chapters on markov chains and random walks will be sourced from publications authored and co-authored by instructor during 2007 – 2017; under <https://people.engr.ncsu.edu/brglez/publications.html>

The material relevant to this course will include articles in public domain such as https://en.wikipedia.org/wiki/Lennard-Jones_potential. Student in the class will also critically review articles such as *Effect of hybridizing Biogeography-Based Optimization (BBO) technique with Artificial Immune Algorithm (AIA) and Ant Colony Optimization (ACO)*, under <http://www.sciencedirect.com/science/article/pii/S1568494614001173> and articles such as *Metaheuristics – the metaphor exposed* under <http://onlinelibrary.wiley.com/doi/10.1111/itor.12001/abstract>; all downloadable from NCSU library.

Prerequisites: Undergraduate level knowledge of Discrete Mathematics [CSC 226], Data structures [CSC 316], Linear Algebra, and programming.

Learning outcomes:

- Understand the fundamental principles of stochastic optimization, both in continuous and discrete domains.
- Learn how to design computational experiments to rigorously evaluate performance difference between two or more solvers on well-defined sets of problem instances.
- Critically review a number of articles and learn about search strategies that have a good chance to improve the current generation of optimization solvers.
- Raise the level of curiosity and motivation for research in experimental algorithmics while also getting experience in rapid prototyping of new solvers in R while using LaTex to efficiently produce beautiful documents.

Limited enrollment: Enrollment is limited to a maximum of 10 graduate and senior-level undergraduate students. This course is a prerequisite for a follow-up 3-credit hour independent study course on Stochastic Optimization II where students will continue to pursue challenges and latest advances in stochastic optimization, including prototyping state-of-the-art stochastic solvers in **Julia**, **Swift**, and **C/C++**. Interested students should email a request for an interview to brglez@ncsu.edu, attaching an informal description of relevant courses, projects, and any references.

Course structure and schedule: The class will meet once a week for a total of 2.75 hours (including a 15-minute break) in a projector-equipped small seminar room, with participants bringing their own laptop computers (macOSX and linux preferred). The instructor will initiate a series of informal seminar-like lectures that will cover material from textbooks and relevant publications. Each week, students will be paired as a team to work on homework assignment selections. During the class on the following week, students will briefly initiate the discussion with few slides, outlining solutions in R. By Week 4, research projects will be identified and assigned to each student pair. Students will be able to initiate and test the computational projects on their own personal computers. For more extensive computational experiments, they will make use of the resources under <https://vcl.ncsu.edu/>.

On GoogleDrive, instructor will aggregate and maintain a class directory of lecture notes, homework/project slides, relevant publications, and a LaTex directory of a collaborative draft manuscript that will merge and compile contributed sections from the instructor and student teams. Students will be expected to maintain similar structures of their own course-related material on their GoogleDrive, including the final R-code and latex draft of their projects. When the course is completed, instructor will archive the relevant code under <https://github.com>, publish the jointly-authored manuscript under <https://arxiv.org/> and submit the manuscript for review to <https://www.jstatsoft.org/>.

The schedule in the table below is tentative. An updated schedule, along with all course material, will be maintained on the course home page under <https://drive.google.com/open?id=0B6mJTvscUcVfdnVwcGdRdEQwdm8>.

Dates:	Topics:	ToolKits:	Homeworks:
Jan 12	Introduction DE algorithm	R-packages DEoptim	Exercises with R Exercises with DEoptim
Jan 19	Genetic algorithm Evolutionary algorithm	GA genalg	Exercises with GA Exercises with genalg
Jan 26	Particle swarm algorithm Classification of problems	pso xyzLib, diceLib	Exercises with pso Exercises with *Lib
Feb 2	Comparative experimentation Q&A and project assignments		Projects: pass 1
Feb 9	Markov chains & dice graphs Minimum first-passage walkLength	stpm.spectral	Projects: pass 2
Feb 16	Significance of sparse dice graphs Merits of self-avoiding walks	landscapeGraph stpm.fpt.walk	Projects: pass 3
Feb 23	New xyzWalk algorithm New diceWalk algorithm	xyzWalk diceWalk	Projects: pass 4
Mar 2	Initiating experiments under VCL Mid-term project reports in latex		Projects: pass 5, VCL
Mar 9	Spring break		

Mar 9	Spring break	
Mar 16	Can we solve 'harder' problems? Problems to be added to projects	More 'harder' problems? Projects: pass 6, VCL
Mar 23	*Walk versus project solvers Project updates under VCL	team tests of *Walk Projects: pass 7, VCL
Mar 30	Spring Holiday	
Apr 6	Are there more 'harder' problems? Project updates under VCL	Projects: pass 8, VCL
Apr 13	Project drafts in latex due Final project updates under VCL	Final project report
Apr 20	manuscript editing begins	Final project edits
Apr 27	manuscript editing ends	

Grading: There is only one grade in this course: Pass. Students who are failing to participate with homeworks as described in the table above, will be asked to withdraw before the drop-course deadline. The final project report and participation in editing the joint manuscript is in lieu of the final exam.

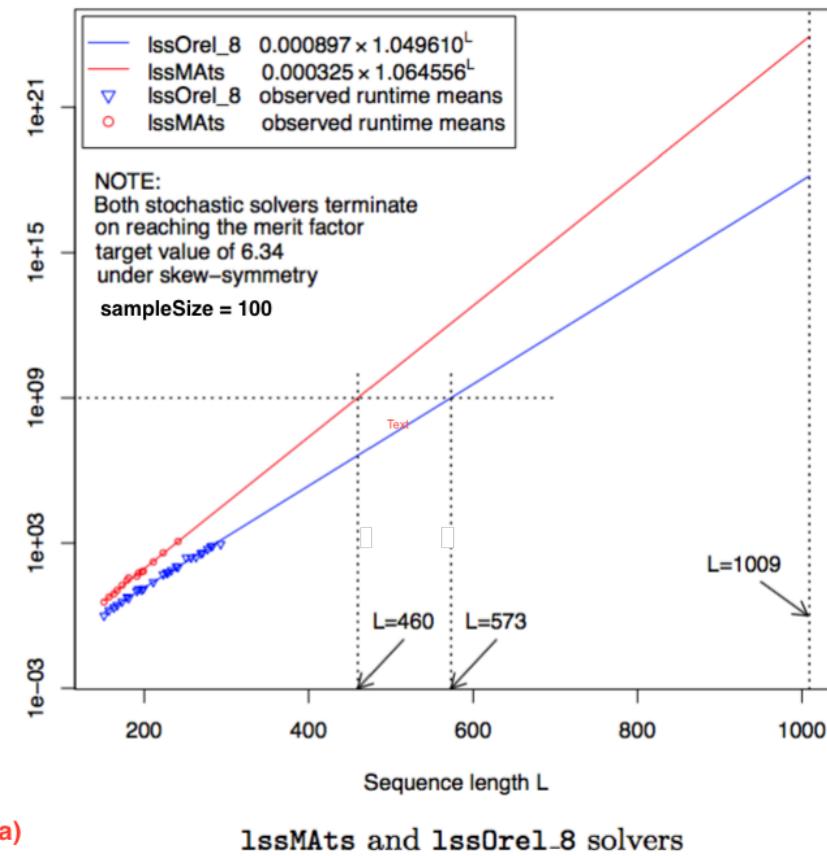
Examples of projects (details and decisions will be discussed in class): The metaphor of *the uncensored mean first-passage-time* is central to the performance evaluation of two optimization solvers in this project. We will have learned about this metaphor when reviewing markov chains and random walks. In our experiments with R, we do not measure the actual runtime directly since this would not provide us with a platform-independent metrics. Just like when comparing two sorting algorithms, *we count the number of primitive operations*, such as the number of function evaluations. For each instance, we invoke the optimization solver from a different randomly selected starting point $O(100)$ times and stop the search as soon as the best-known target value is reached for the first time.

- Evaluate *the uncensored mean first-passage-time* of the solver while searching for optima in continuous domains using a progression of test functions from the same family but of increasing size: in addition to 'standard' functions, consider also new and hard test functions such as represented by Lennard-Jones clusters, clusters of folded proteins, etc.
- Evaluate *the uncensored mean first-passage-time* of the solver while searching for optima in discrete domains using a progression of test instances from the same but of increasing size: the knapsack problem, the maximum graph clique problem, the maximum satisfiability problem, the low autocorrelation binary sequences problem, the optimum Golomb ruler problem, the linear ordering problem, the job-shop scheduling problem, the TSP problem, etc.
- In class, we shall rigorously compare, on identical instances, asymptotic performances of various stochastic solvers identified in the textbook with the new solvers walkXYZ and walkDice introduced by the instructor.

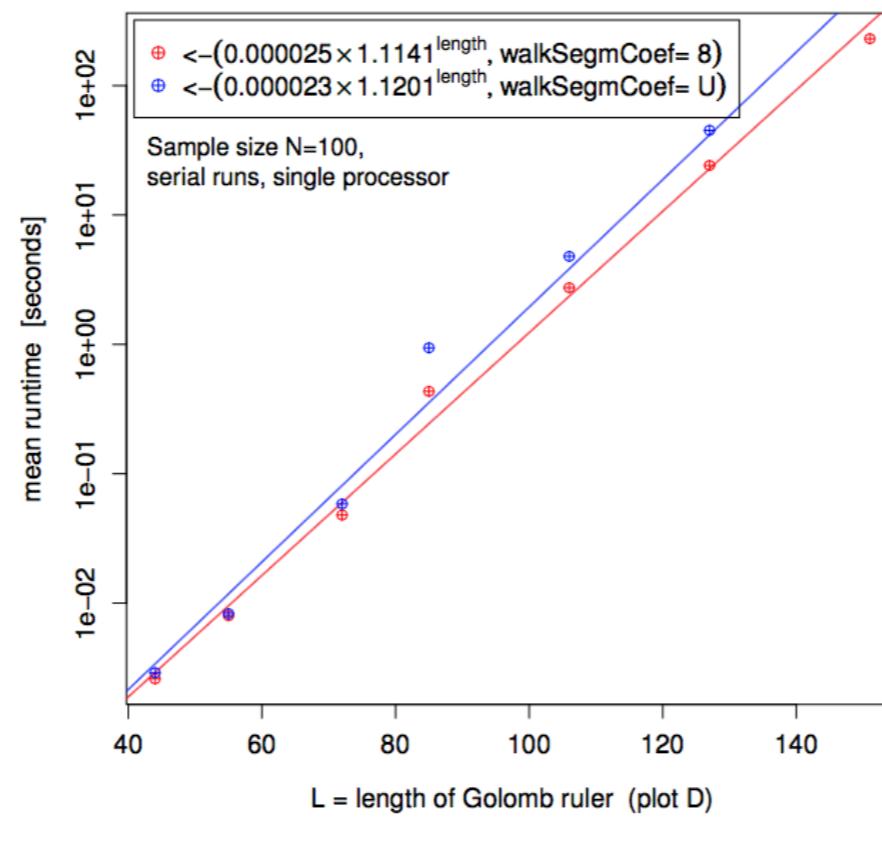
Examples of templates for the joint manuscript

Templates such as ones below will be used to summarize experimental results in the manuscript to be completed at the end of the course
 (data in these plots has been copied from existing articles)

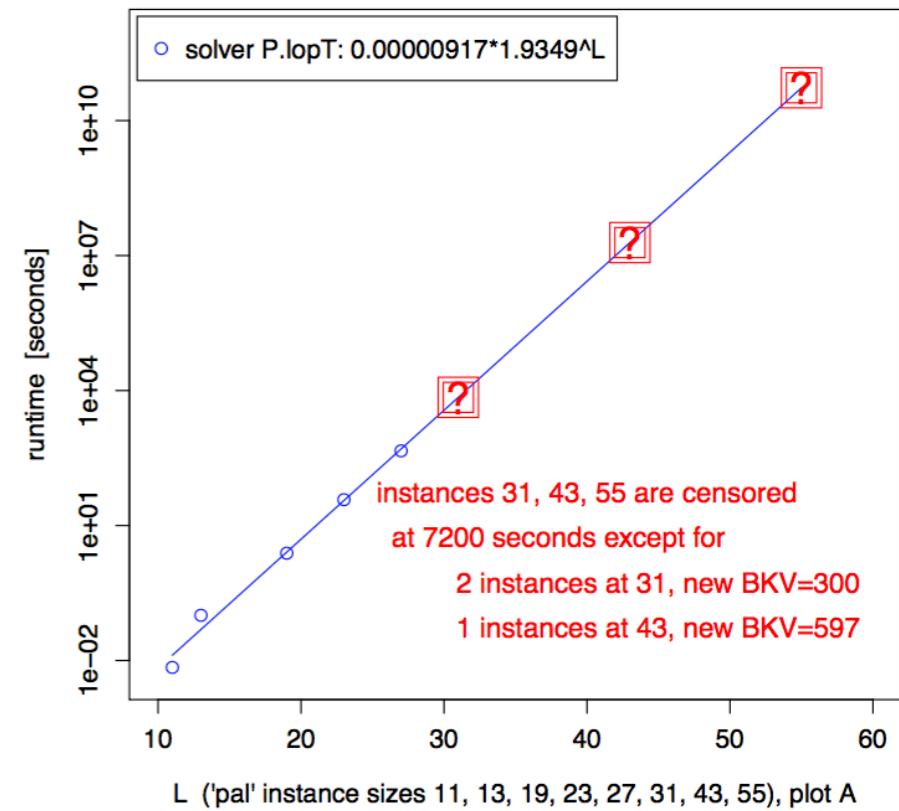
Problem class:
 Solver1
 Solver2



Problem class:
 Solver1
 Solver2



Problem class:
 Solver1
 Solver2



Preparing for the running start of this course

By Thanksgiving break, I will post a number of items that will give everybody a running start for the course on January 12, 2018. These will include:

- a library of R-functions such as 'wild(x)' invoked by DEoptim on page 9
- a script that encapsulates the solver DEoptim, so you can reproduce the statistical experiment on page 10
- suggestions about the initial R-packages to download for the course
- a copy of an excellent supplementary book about getting started with R

In the meantime, I suggest you follow the links in the updated syllabus

<https://people.engr.ncsu.edu/brglez/courses.html>

and download the two textbooks and and install and test your own R-environment!