Optimisation

Derivative-free optimization:
From Nelder-Mead to global methods

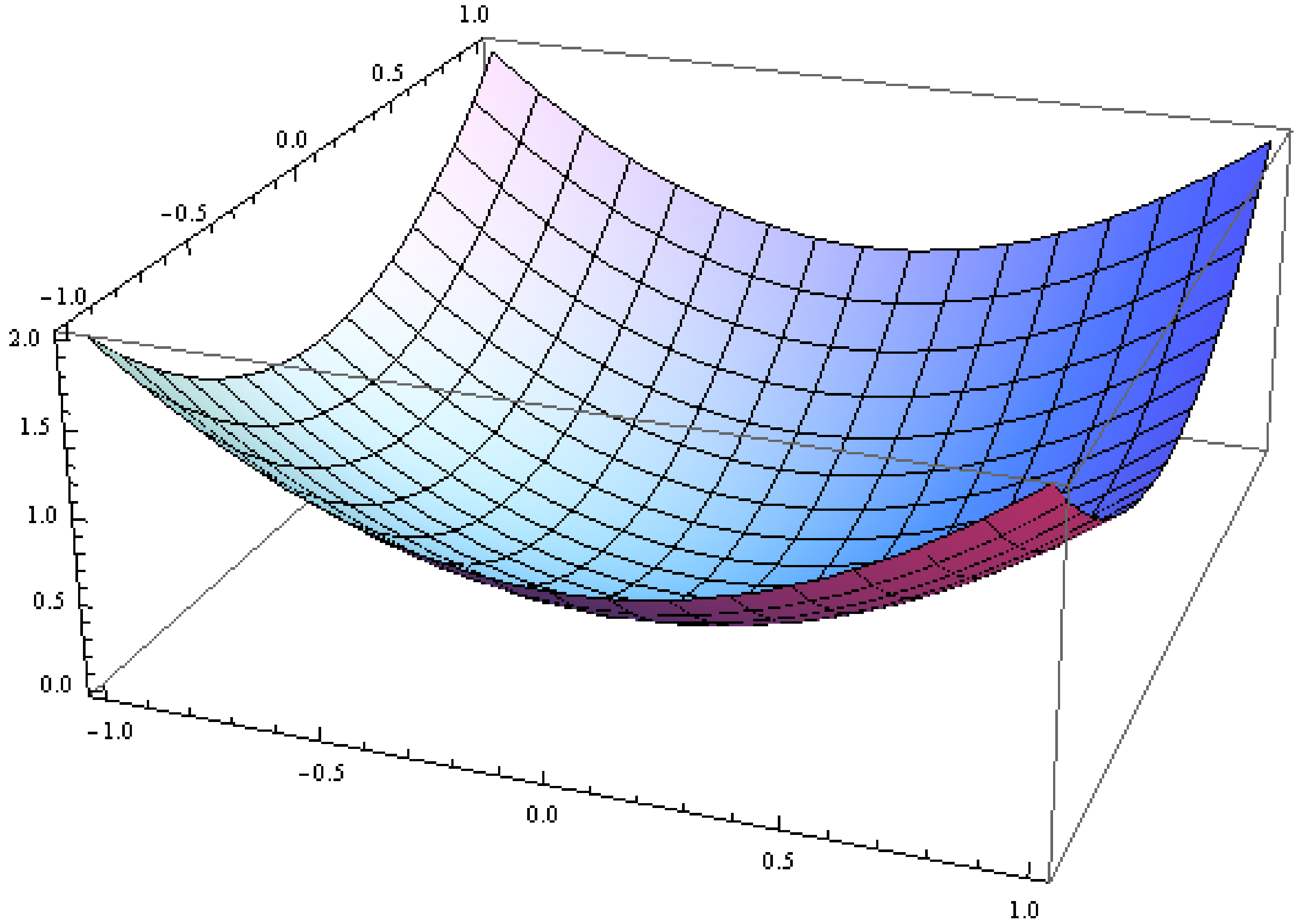
Definition

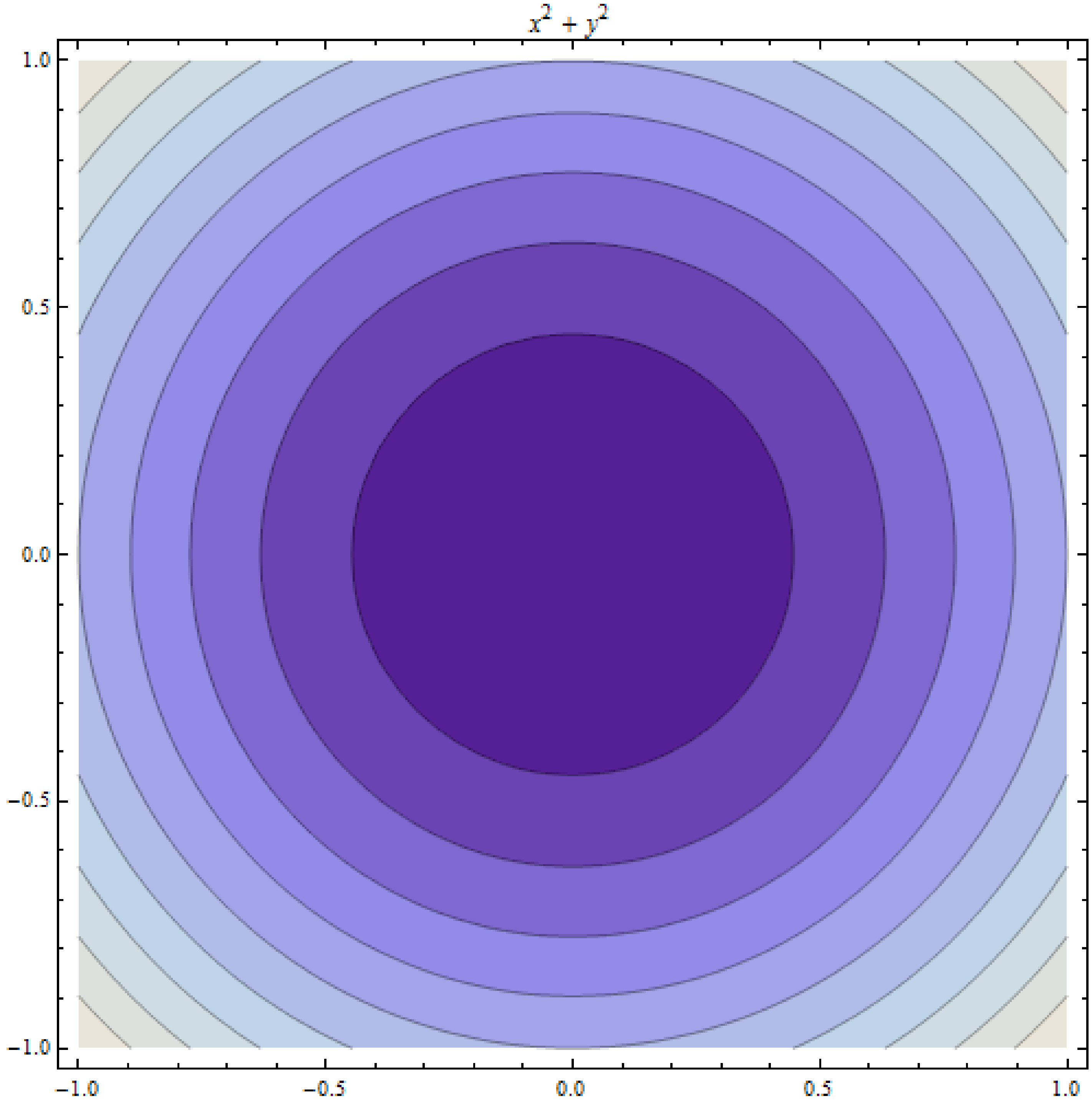
- Optimizing a function is looking for the set of values of the variables that will maximize (or minimize) the function.
- Optimization is usually a very complex problem. There are many different techniques, each being adapted to a specific kind of problems.
- There is no universal method, but a set of tools which requires a lot of experience to be used properly.

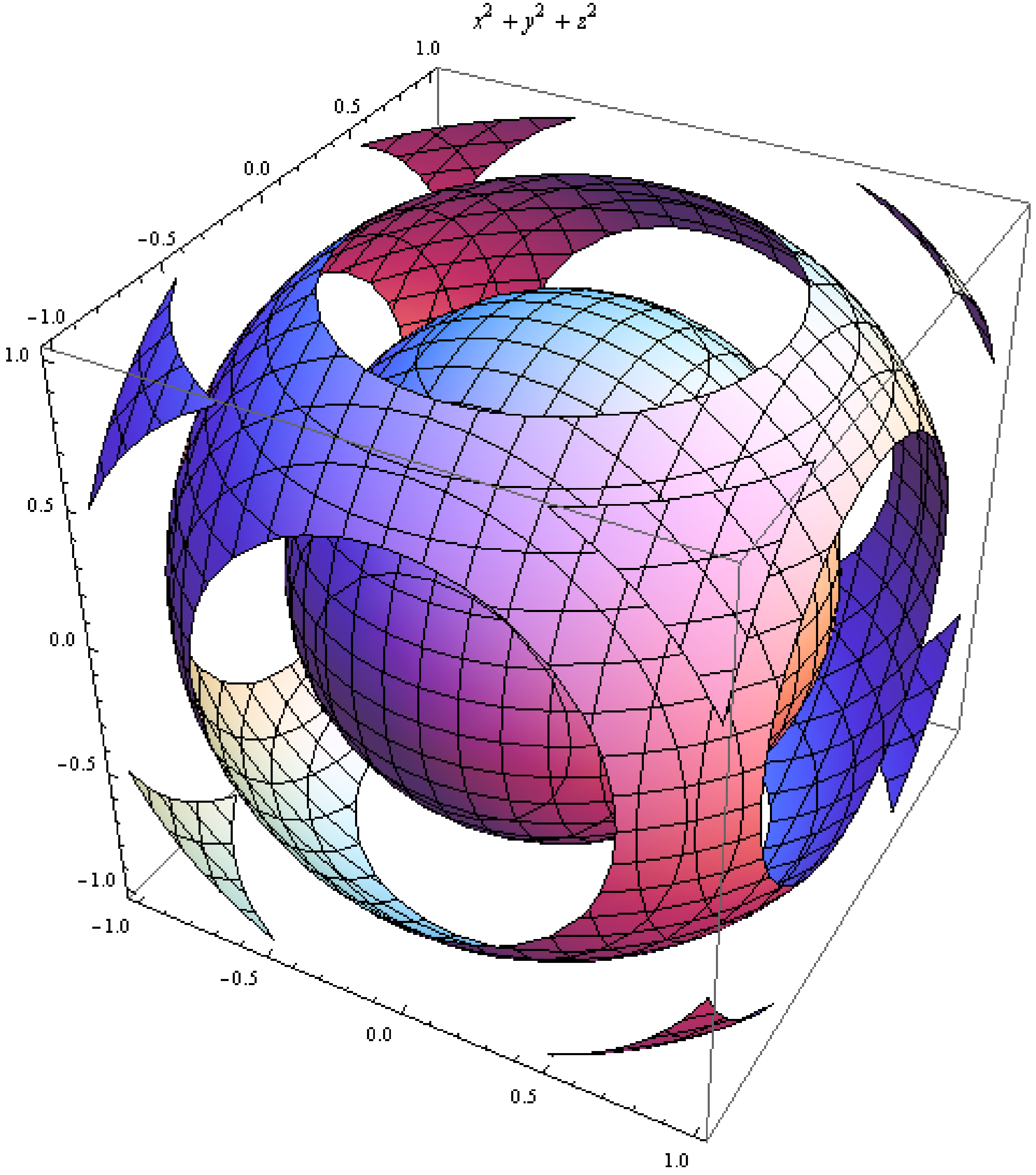
Optimization caracteristics

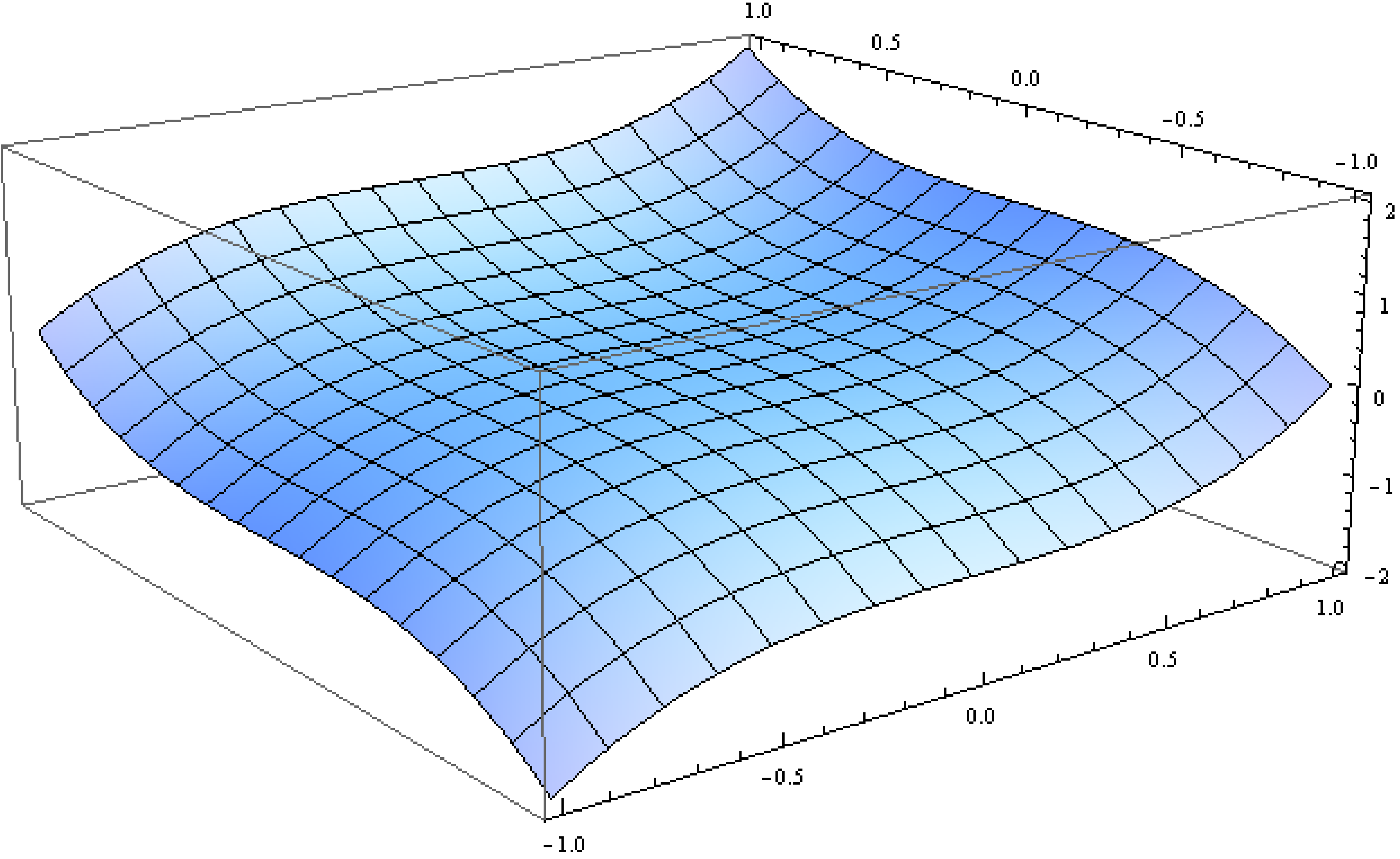
- Global/local optimization
  - Global optimization is searching for the absolute extremum of the function over its entire definition domain
  - Local optimization is looking for the extremum of the function in the vicinity of a given point
- Stochastic / deterministic methods
  - A stochastic method searches the definition domain of the function in a random way . Two succesive runs can give different résults.
  - A deterministic method always walks the search space in the same way, and always gives the same results.

x^2+y^2

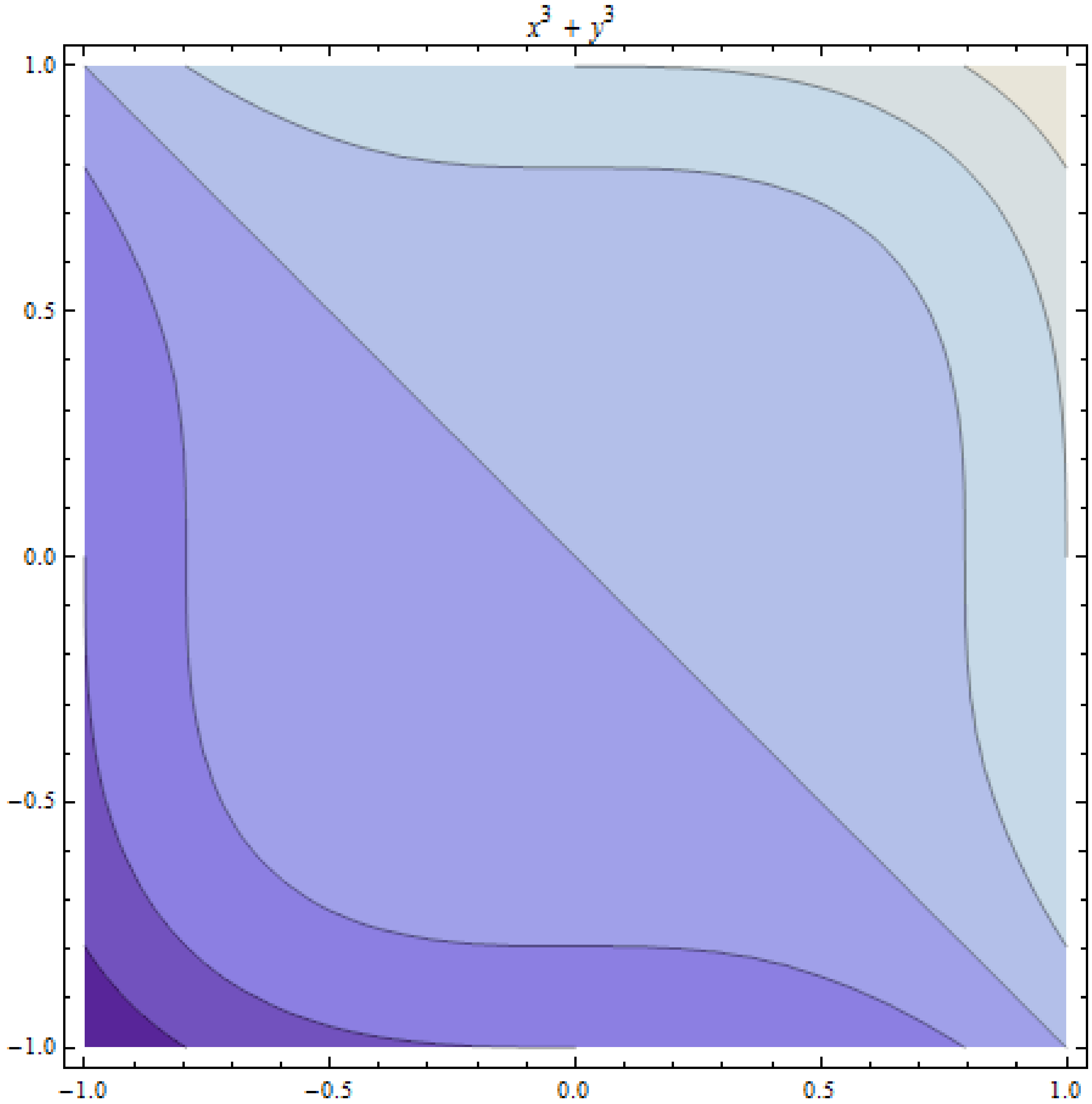


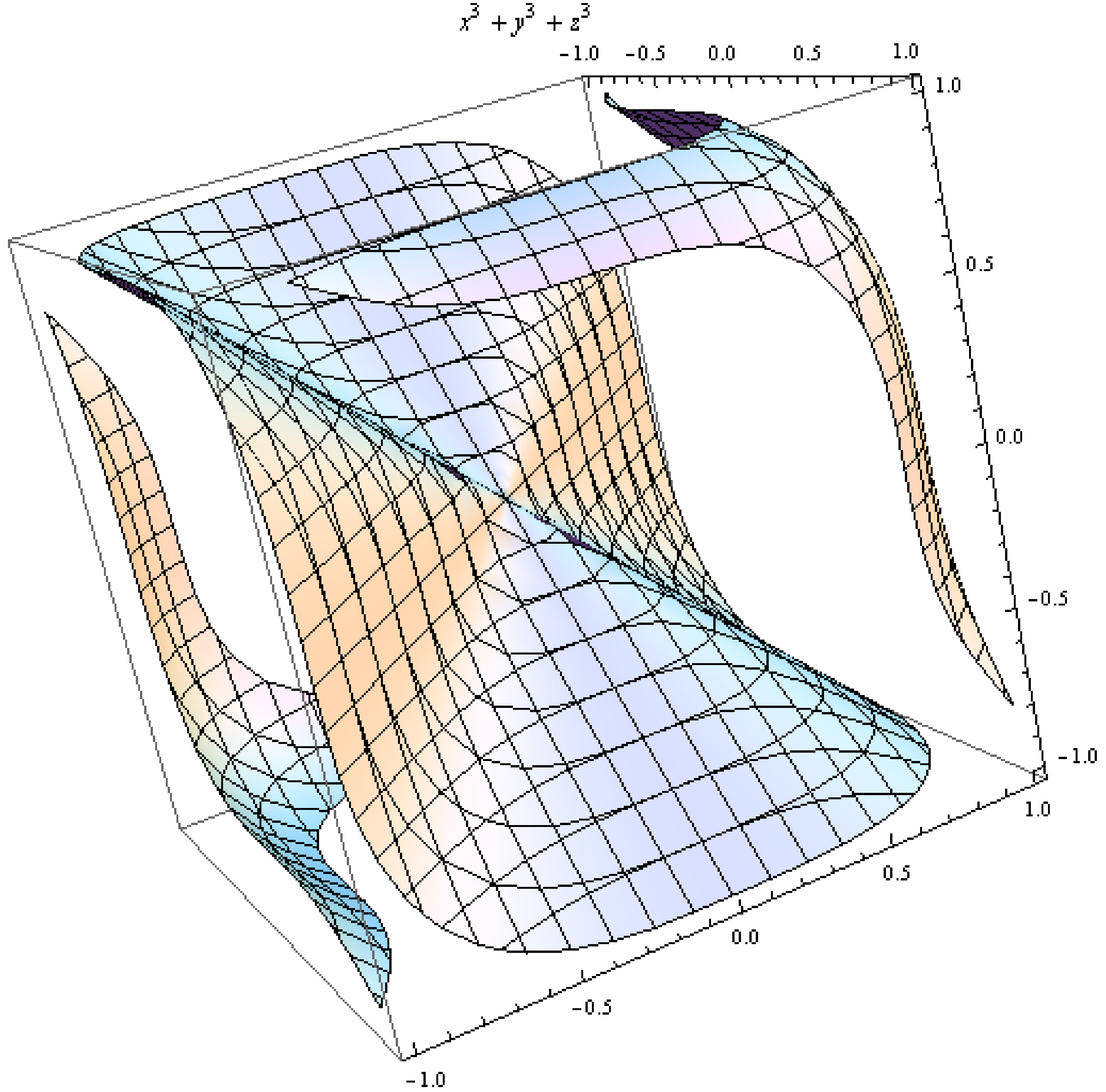

$x^2 + y^2$
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


$x^2 + y^2 + z^2$
1.0
0.5
0.0
-0.5
-1.0
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0

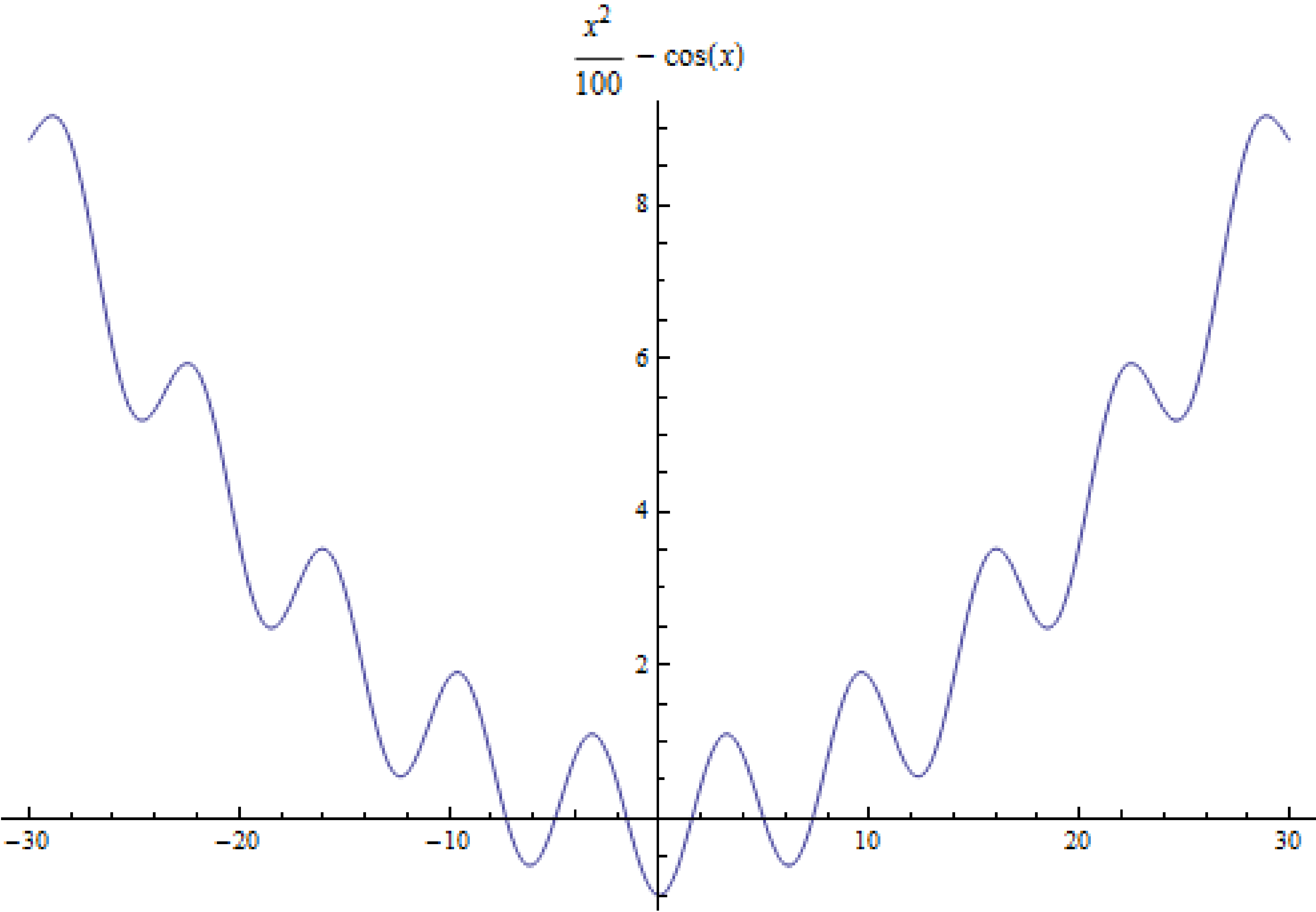


1.0
0.5
0.0
-0.5
-1.0
2
1
0
-1
-2
1.0
0.5
0.0
-0.5
-1.0


$x^3 + y^3$
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

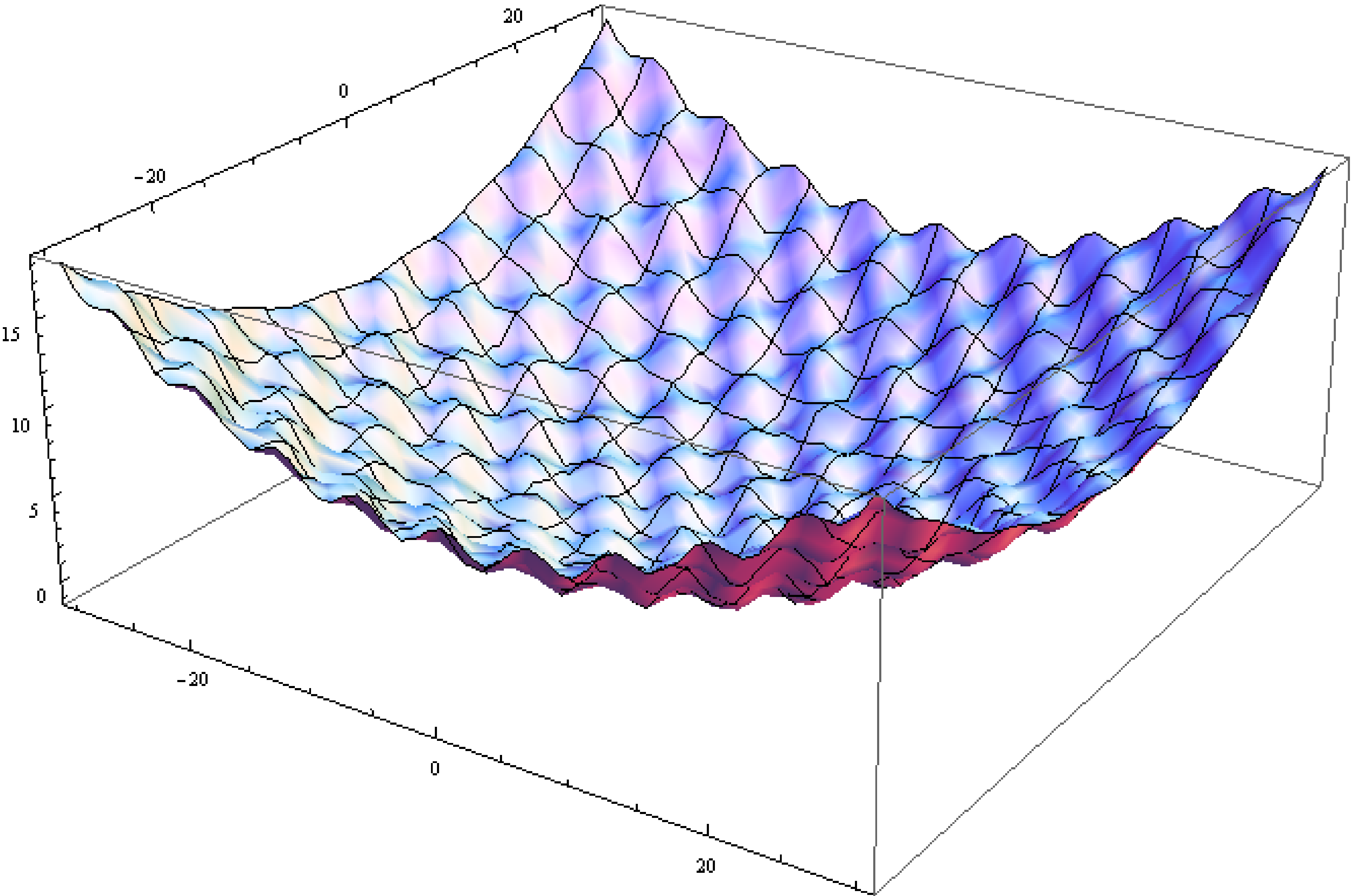


$x^3 + y^3 + z^3$
-1.0
-0.5
0.0
0.5
1.0
1.0
0.5
0.0
-0.5
-1.0
1.0
0.5
0.0
-0.5
-1.0

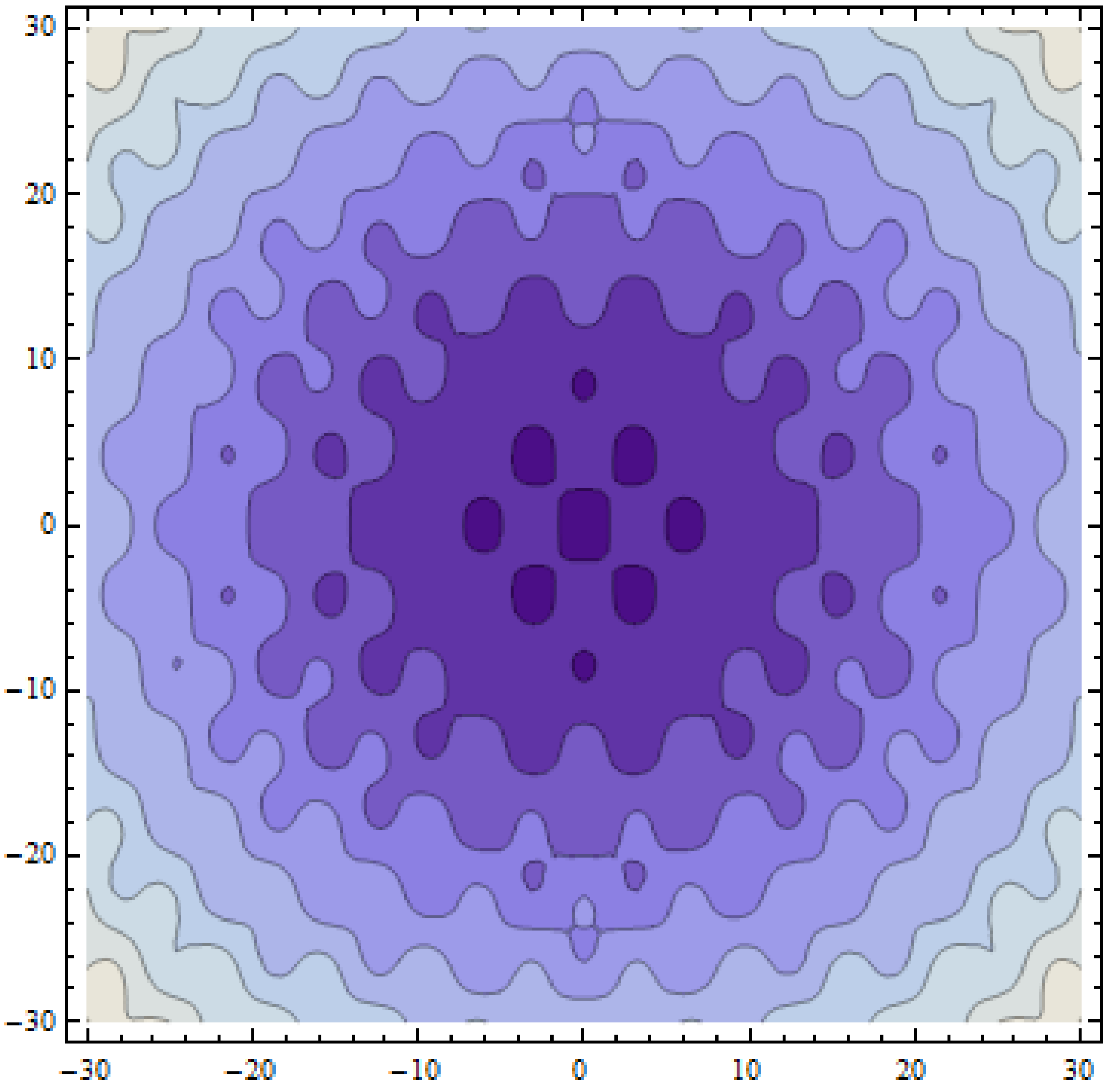


$\frac{x^2}{100} - \cos(x)$
8
6
4
2
−30
−20
−10
10
20
30

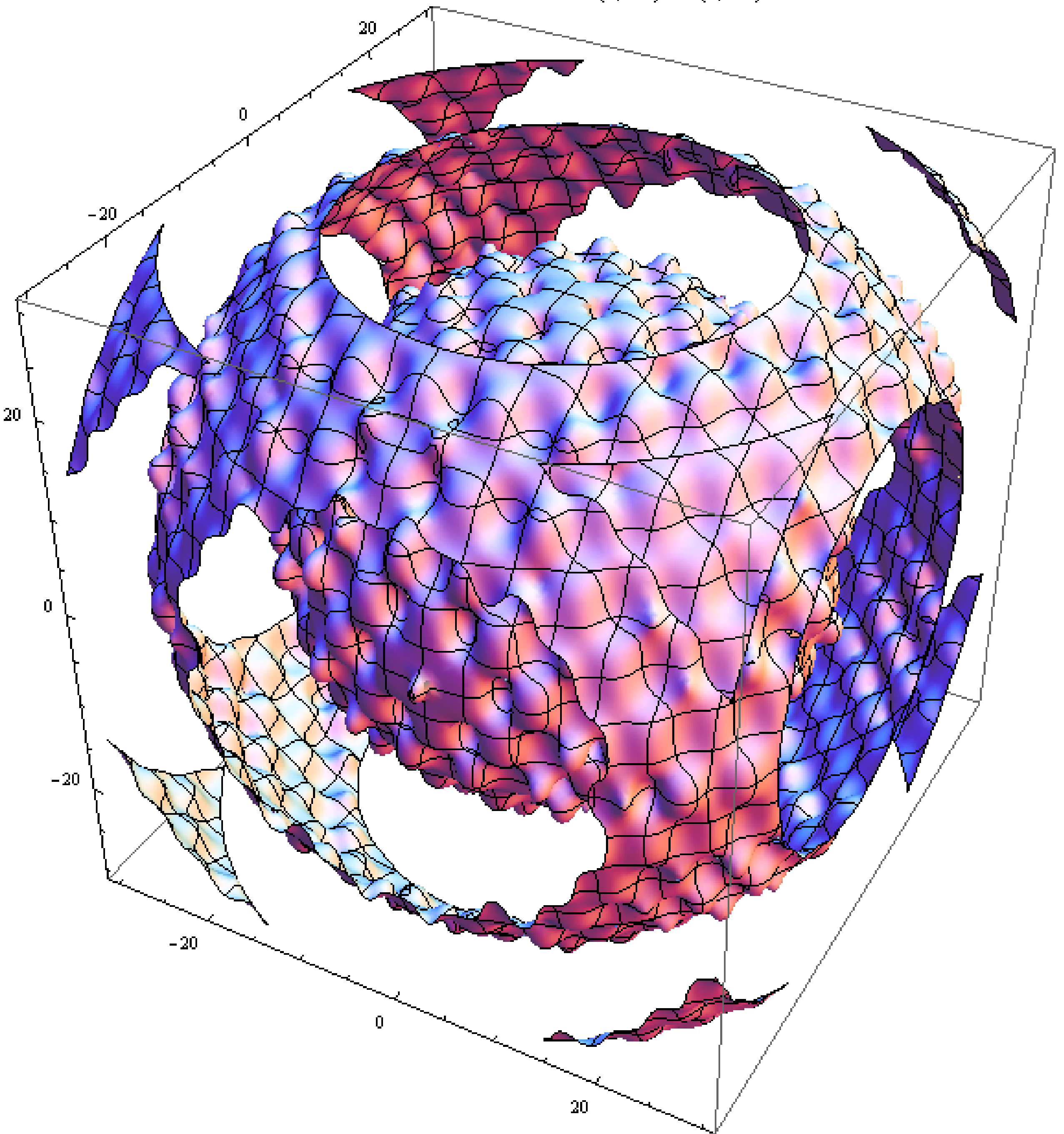
$$\frac{1}{100}(x^2+y^2)-\cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)$$



$$\frac{1}{100}\left(x^2+y^2\right)-\cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)$$



$$\frac{1}{100}\left(x^2+y^2+z^2\right)-\cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)\cos\left(\frac{z}{\sqrt{3}}\right)$$



Part I
Local deterministic methods

Derivation (deterministic)

- When it is possible to compute and solve f'(x)=0, then we know that the extrema of the function are in the set of solutions.
- This method can only be used for very simple analytic functions

Gradient method (deterministic and local)

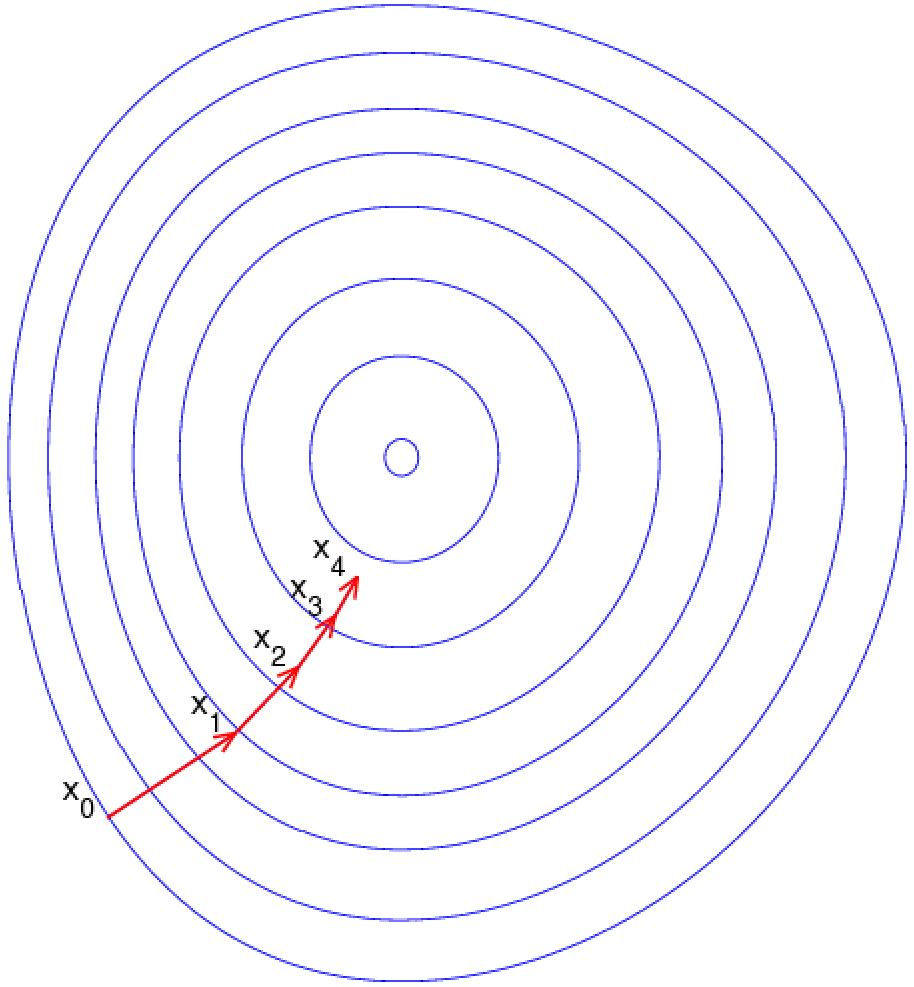
If f(X) is a real valued function of a real valued vector X, and we can calculate f'(X), we compute:

$X_{n+1} = X_n - a\, f'(X_n),\ a>0$

The best choice of a>0 is done by minimizing:

$G(a)=f(X_n - a\, f'(X_n))$

It's usually impossible to solve the above equation and approximate methods are used.



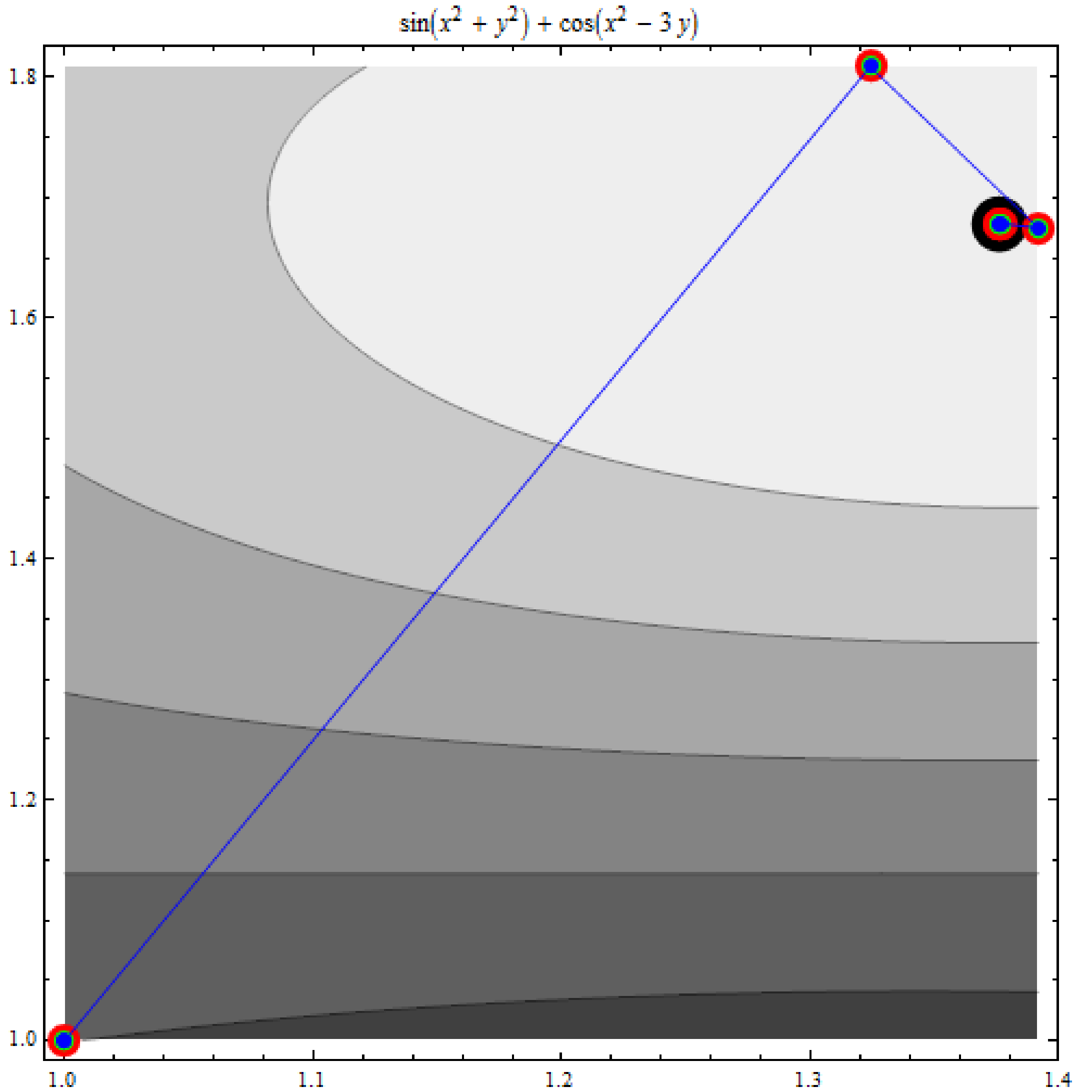
Local, deterministic, order 2, methods.

- To accelerate computation we use the computation of the first and second order derivatives of the function
- We need to be able to compute both, in a reasonnable amount of time.

Local, deterministic, order 2, methods.

- $f(y) = f(x) + f'(x)\,(y-x) + \frac{1}{2} f''(x)\,(y-x)^2 + d$
- We minimize the y quadratic form:
  - $f'(x)+f''(x)(y-x)=0 \Rightarrow y = x - f'(x)/f''(x)$
- Algorithm:
  - $x_{n+1} = x_n - f'(x_n) \,/\, f''(x_n)$
- Known as Newton method
- Convergence is (much) faster than the simple gradient method.

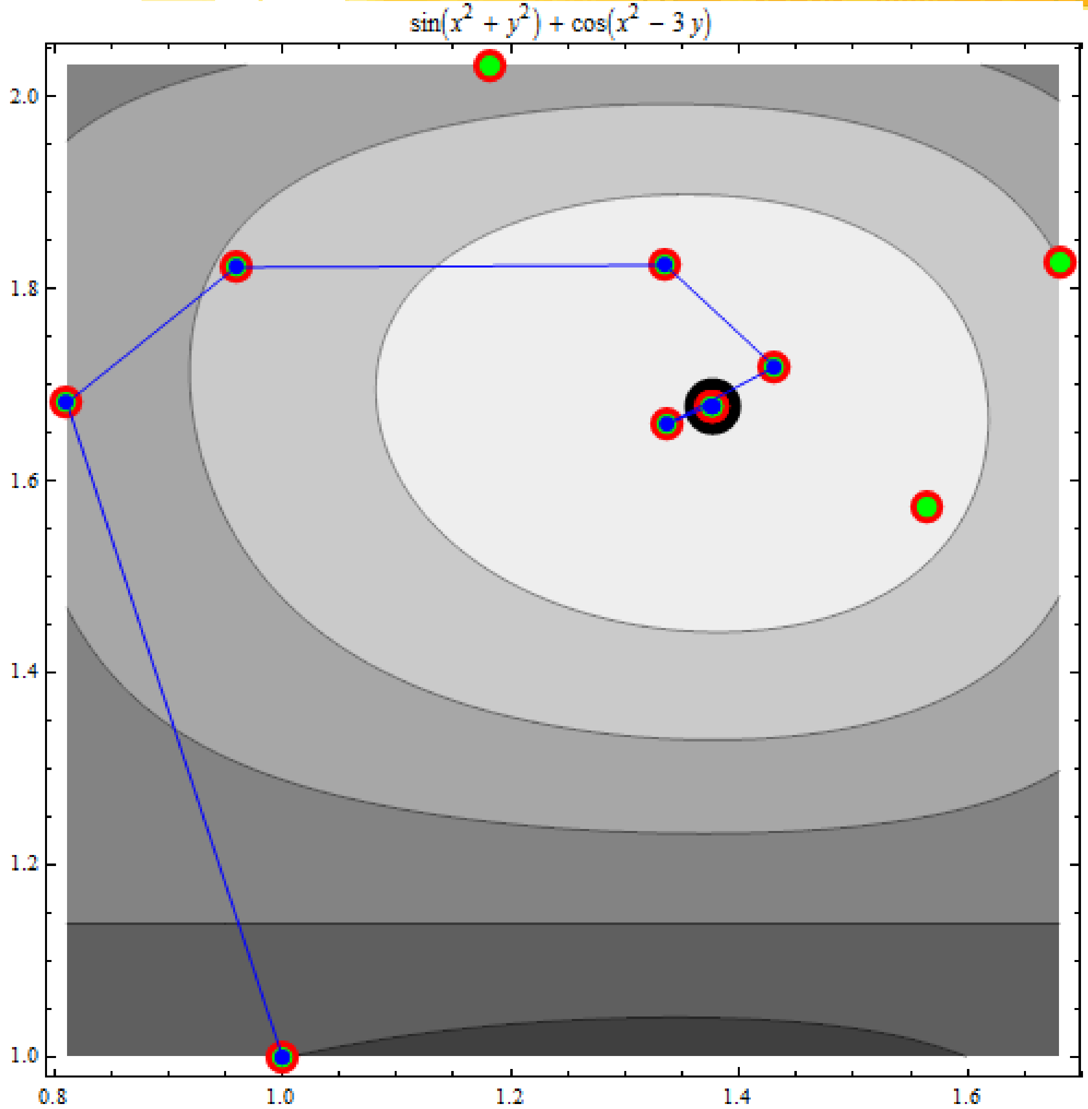
Newton



Deterministic method: BFGS

- BFGS approximates the hessian matrix without explicitly computing the hessian
- It only requires knowledge of the first order derivative.
- It’s faster than gradient, slower (but much more practical) than Newton
- One of the most used method.

BFGS



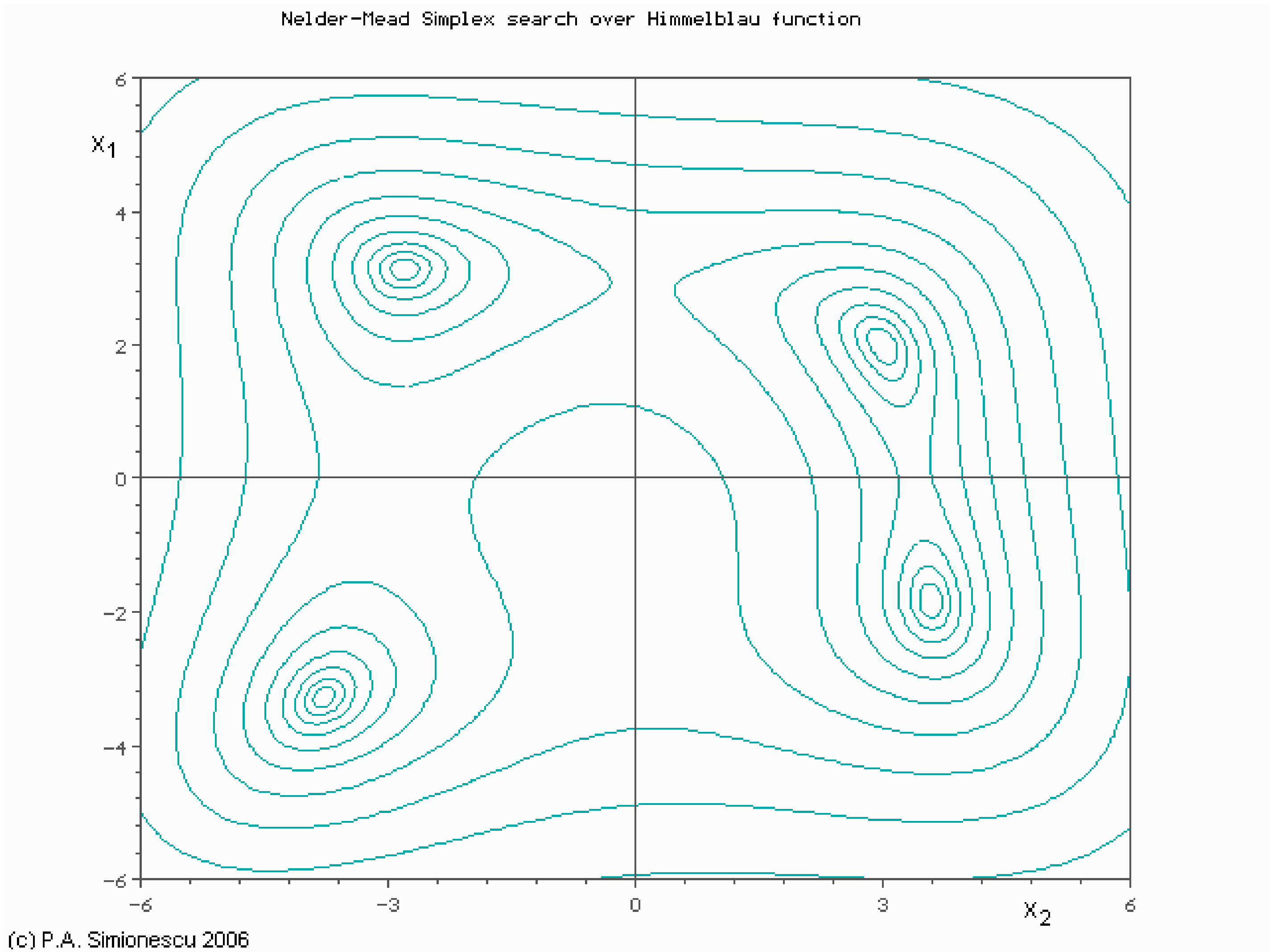
Local deterministic: Nelder-Mead simplex

- Works by building an n+1 points polytope for an n variables function, and by shrinking, expanding and moving the polytope.
- There’s no need to compute the first or second order derivative, or even to know the analytic form of f(x), which makes NMS very easy to use.
- The algorithm is very simple.

Nelder-Mead simplex

- Choose n+1 points (x_1,..x_{n+1})
- Sort: $f(x_1)<f(x_2)...<f(x_{n+1})$
- Compute barycenter: $x_0 = (x_1+...+x_n)/n$
- Reflection of x_{n+1}/x_0: $x_r=x_0+(x_0-x_{n+1})$
- If $f(x_r)<f(x_1)$, $x_e=x_0+2(x_0-x_{n+1})$. If $f(x_e)<f(x_r)$, $x_{n+1}<-x_e$, else $x_{n+1}<-x_r$, back to sort.
- If $f(x_n)<f(x_r)$, $x_c=x_{n+1}+(x_0-x_{n+1})/2$.If $f(x_c)<f(x_r)$ $x_{n+1}<-x_c$, back to sort
- Otherwise: $x_i <- x_0+(x_i-x_1)/2$. Back to sort.

Nelder Mead



Nelder-Mead Simplex search over Himmelblau function
x1
x2
6
4
2
0
-2
-4
-6
-6
-3
0
3
6
(c) P.A. Simionescu 2006

Part II
Global stochastic methods

Stochastic optimization

- Do not require any regularity (functions do no even need to be continuous)
- Usually expensive regarding computation time, and do not guarantee optimality
- There are some theoretical convergence results, but they usually don’t apply in day to day problems.

Simulated annealing

- Generate one random starting point $x_0$inside the search space.
  - Build $x_{n+1}=x_n+B(0,s)$
  - Compute: $t_{n+1}=H(t_n)$
  - If $f(x_{n+1})<f(x_n)$ then keep x_{n+1}
  - If $f(x_{n+1})>f(x_n)$ then :
    - If $|f(x_{n+1})-f(x_n)|<e^{-k\,t}$ then keep x_{n+1}
    - Si $|f(x_{n+1})-f(x_n)|>e^{-k\,t}$ then keep x_n

Important parameters

- H (the annealing schedule):
  - Too fast=>the algorithm converges very quickly to a local minimum
  - Too slow=>the algorithm converges painfully slowly.
- Deplacement: B(0,s) must search the whole space, and mustn't jump too far or too close either

Efficiency

- SA can be useful on problems too difficult for « classical methods »
- Genetic algorithms are usually more efficient when it is possible to build a « meaningful » crossover

Genetic algorithms (GA)

- Search heuristic that « mimics » the process of natural evolution:
  - Reproduction/selection
  - Crossover
  - Mutation
- John Holland (1960/1970)
- David Goldberg (1980/1990).

Coding / population generation

- If x is a variable of f(x), to optimize on the interval $[x_{min},x_{max}]$.
- We rewrite x :$2^n (x-x_{min})/(x_{max}-x_{min})$
- This gives an n bits string:
  - For n=8: 01001110
  - For n=16: 0100010111010010
- A complete population of N (n bits string) is generated.

Crossover

- Two parents :
  - 01100111
  - 10010111
- One crossover point (3):
  - 011|00111
  - 100|10111
- Two children:
  - 011|10111
  - 100|00111

Mutation

- One randomly chosen element:
  - 01101110
- One mutation site (5):
  - 01101110
- Flip bit value:
  - 01100110

Reproduction/selection

- For each x_i compute $f(x_i)$
- Compute $S=\Sigma(f(x_i))$
- Then for each x_i :
  - $p(x_i)=f(x_i)/S$
- The n elements of the new population are picked from the pool of the n elements of the old population with a bias equal to $p(x_i)$.
- Better adapted elements are more reproduced

Exemple de reproduction

- $f(x)=4x(1-x)$
- x in [0,1[

Séquence	Valeur	$U(x)$	% de chance de reproduction	% cumulés	Après reproduction
10111010	0.7265625	0.794678	0.794678 / 2.595947 = 0.31	0.31	11011110
11011110	0.8671875	0.460693	0.460693 / 2.595947 = 0.18	0.31+0.18=0.49	10111010
00011010	0.1015625	0.364990	0.364990 / 2.595947 = 0.14	0.49+0.14=0.63	01101100
01101100	0.4218750	0.975586	0.975586 / 2.595947 = 0.37	0.62+0.37=1.00	01101100
=		2.595947			

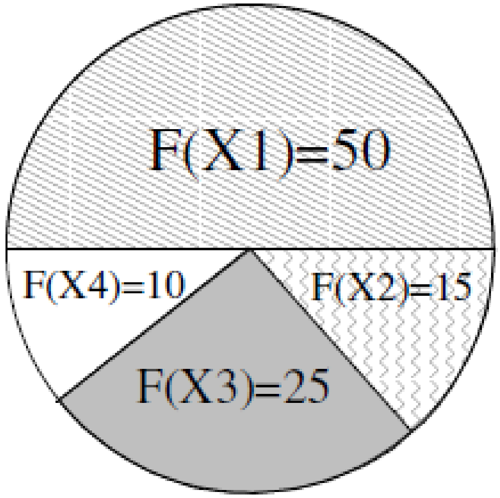
AG main steps

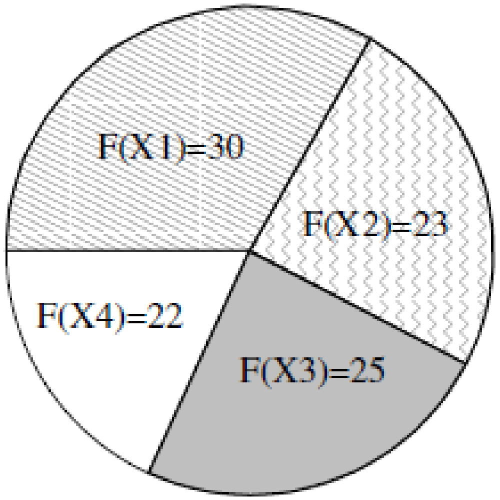
- Step 1: reproduction/selection
- Step 2: crossing
- Step 3: mutation
- Step 4: End test.

Scaling

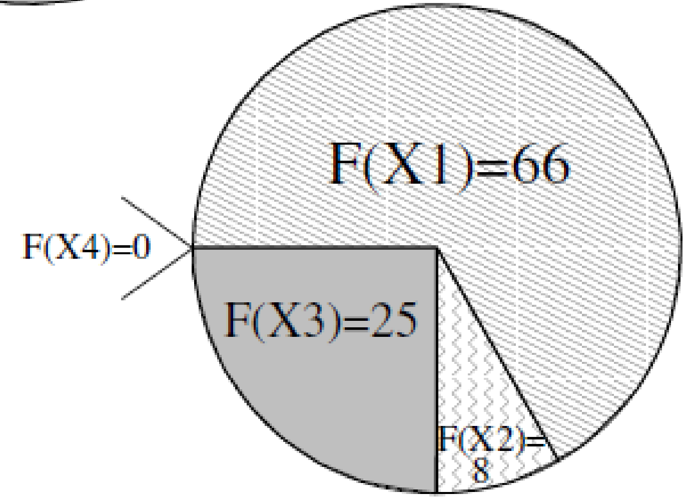
- Fact: in the « simple » AG, the fitness of an element x is equal to f(x)
- Instead of using f(x) as fitness, f is « scaled » by using an increasing function.
- Exemples:
  - 5 (f(x)-10)/3: increase selection pressure
  - 0.2 f + 20 : diminishes selection pressure
- There are also non-linear scaling functions

Scaling examples





0.2 F + 20



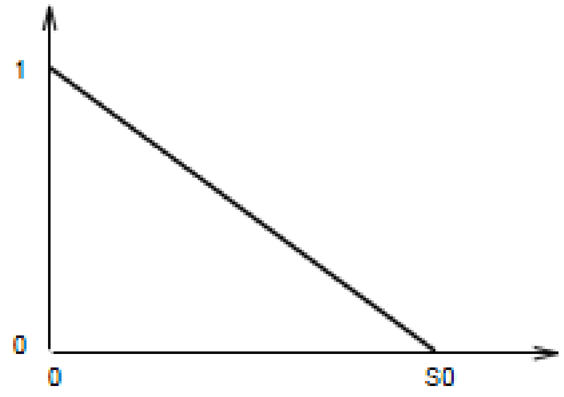
5/3 (F – 10)

Sharing

- Selection pressure can induce too fast convergence to local extrema.
- Sharing modifies fitness depending on the number of neighbours of an element:
  - $f_s(x_i)=f(x_i)/\Sigma_j\ s(d(x_i,x_j))$
  - s is a decreasing function.
  - $d(x_i,x_j)$ is a distance measurement between i et j

Sharing

- To use sharing, you need a distance function over variables space
- General shape of s:



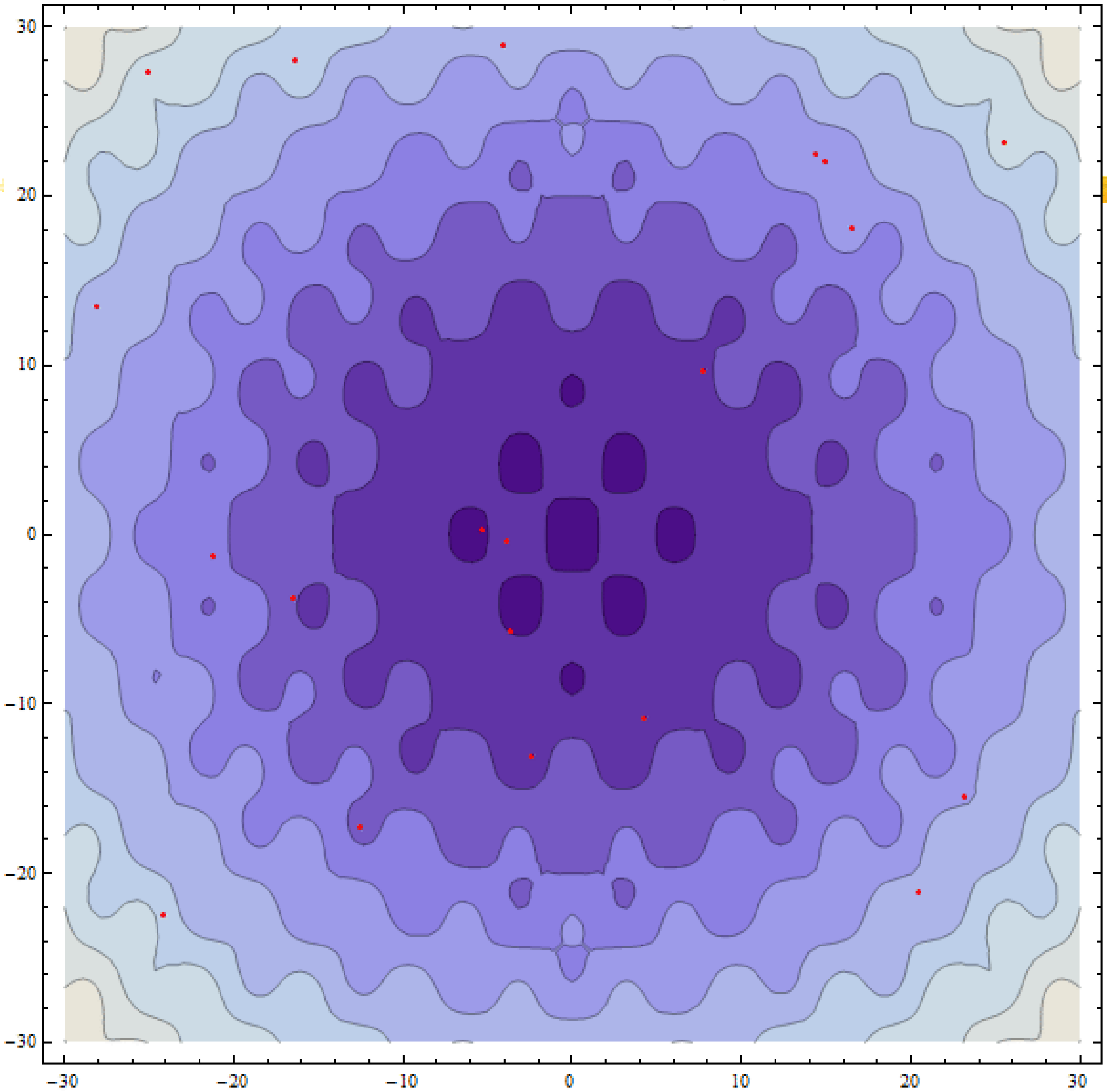
Bit string coding problem

- Two very different bit strings can represent elements which are very close to each other:
  - If encoding real values in [0,1] with 8 bits:
    - 10000000 et 01111111 represent almost the same value (1/2) but their hamming distance is maximal (8).
  - Necessity to use Grey encoding.

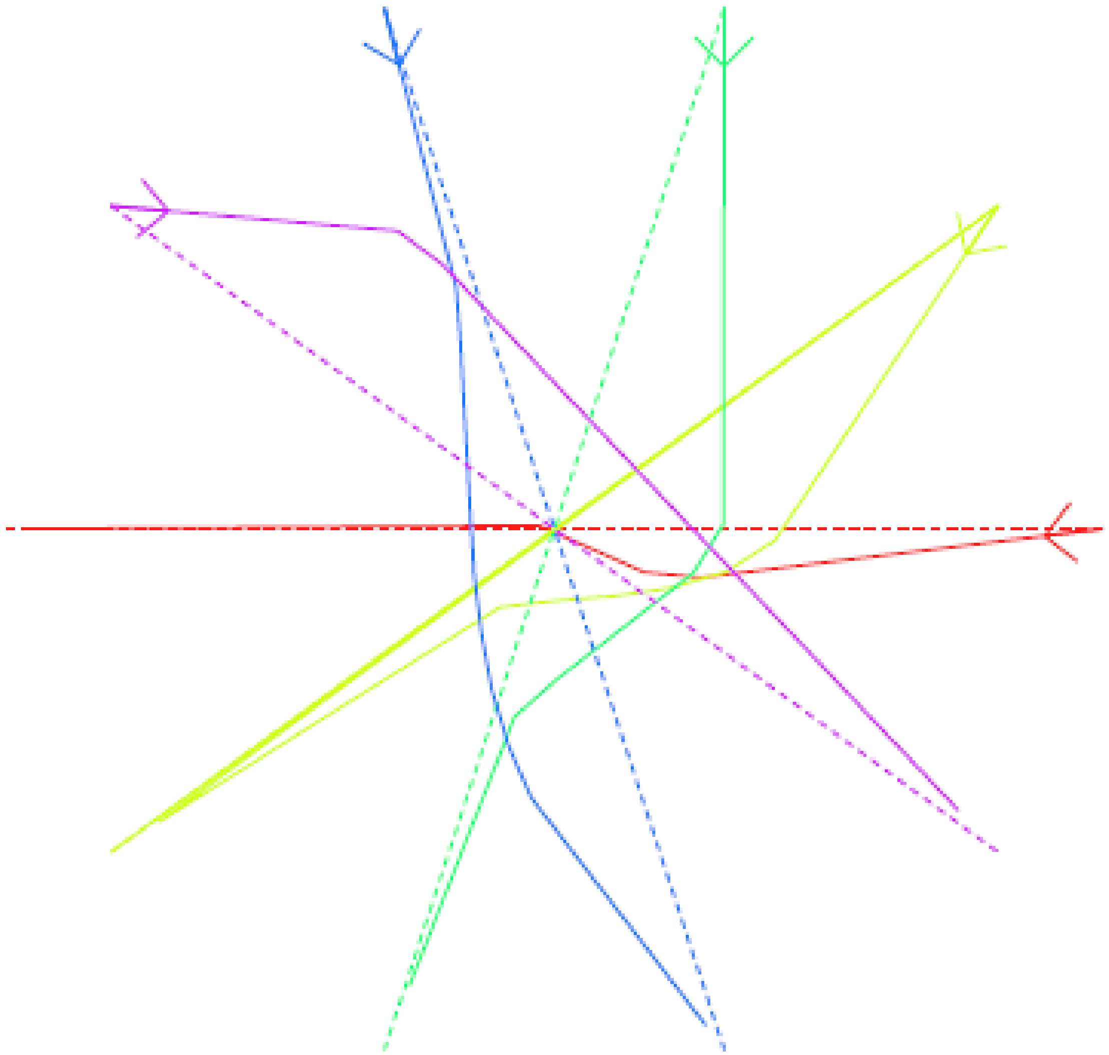
Using a proper coding

- For real variable functions, real variable encoding is used
- Crossover:
  - $y_1 = \alpha\, x_1 + (1-\alpha)\, x_2$
  - $y_2 = (1-\alpha)\, x_1 + \alpha\, x_2$
  - α randomly picked in [0.5,1.5]
- Mutation:
  - $y_1 = x_1 + B(0,\sigma)$

$$\frac{1}{100}\left(x^2+y^2\right)-\cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)$$



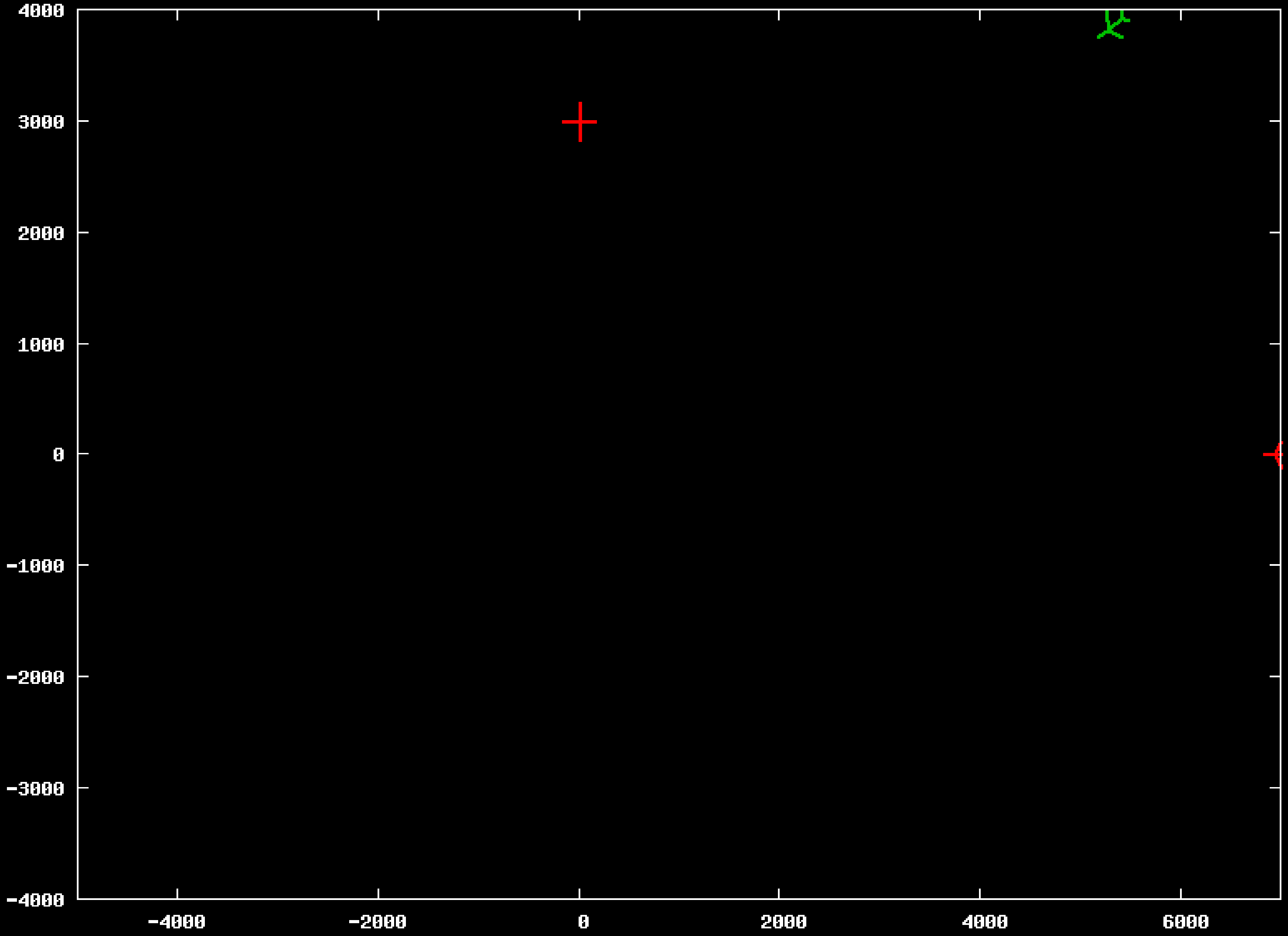
Aircraft conflit resolution



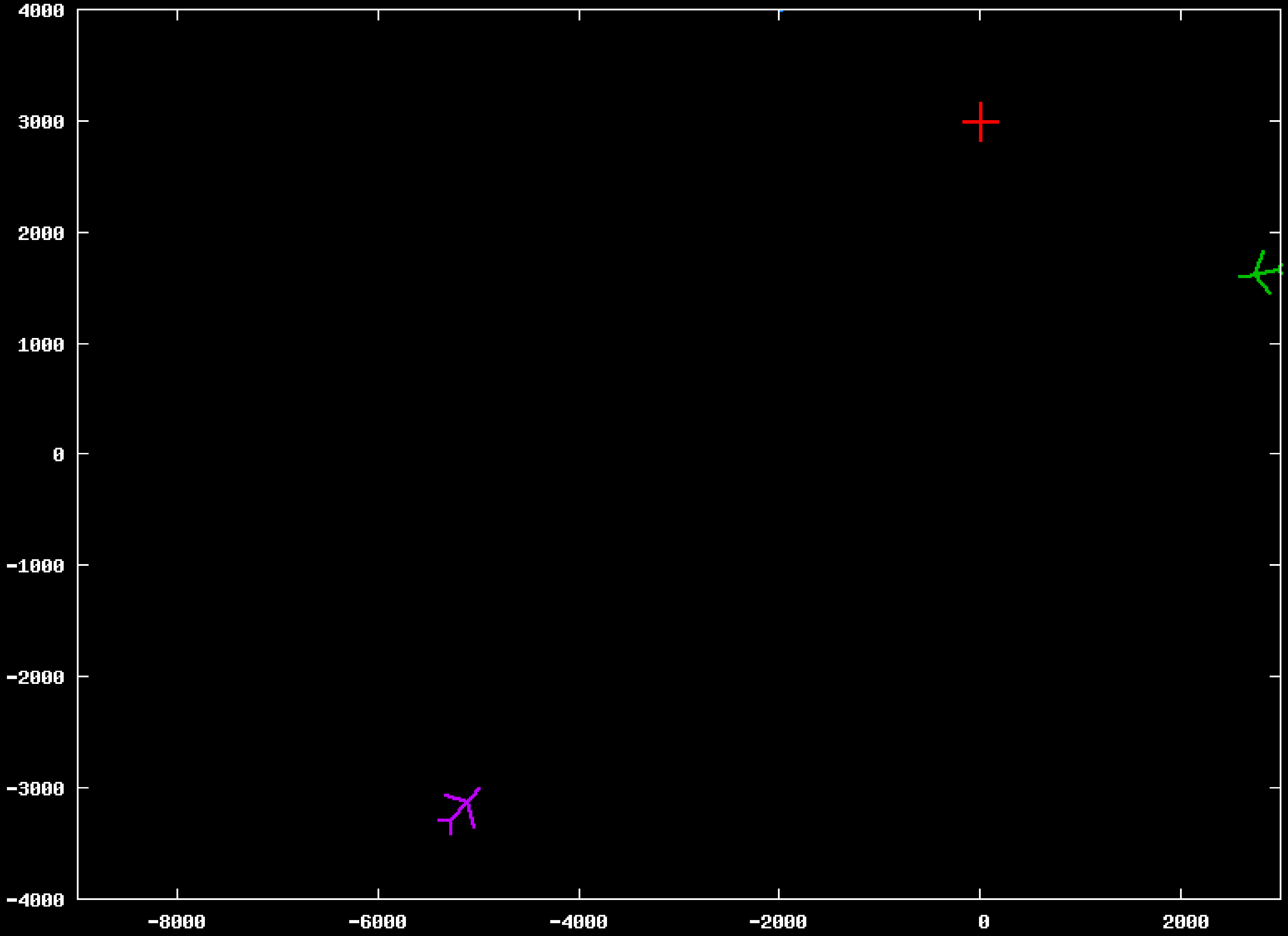
Modeling

- Only one manoeuver maximum by aircraft
  - 10, 20 or 30 degrees deviation right or left
    - Then return to destination
  - Offset
- Variables: 3 n
  - T0: start of manoeuver
  - T1: end of manoeuver
  - A: angle of deviation
- Uncertainties on speeds

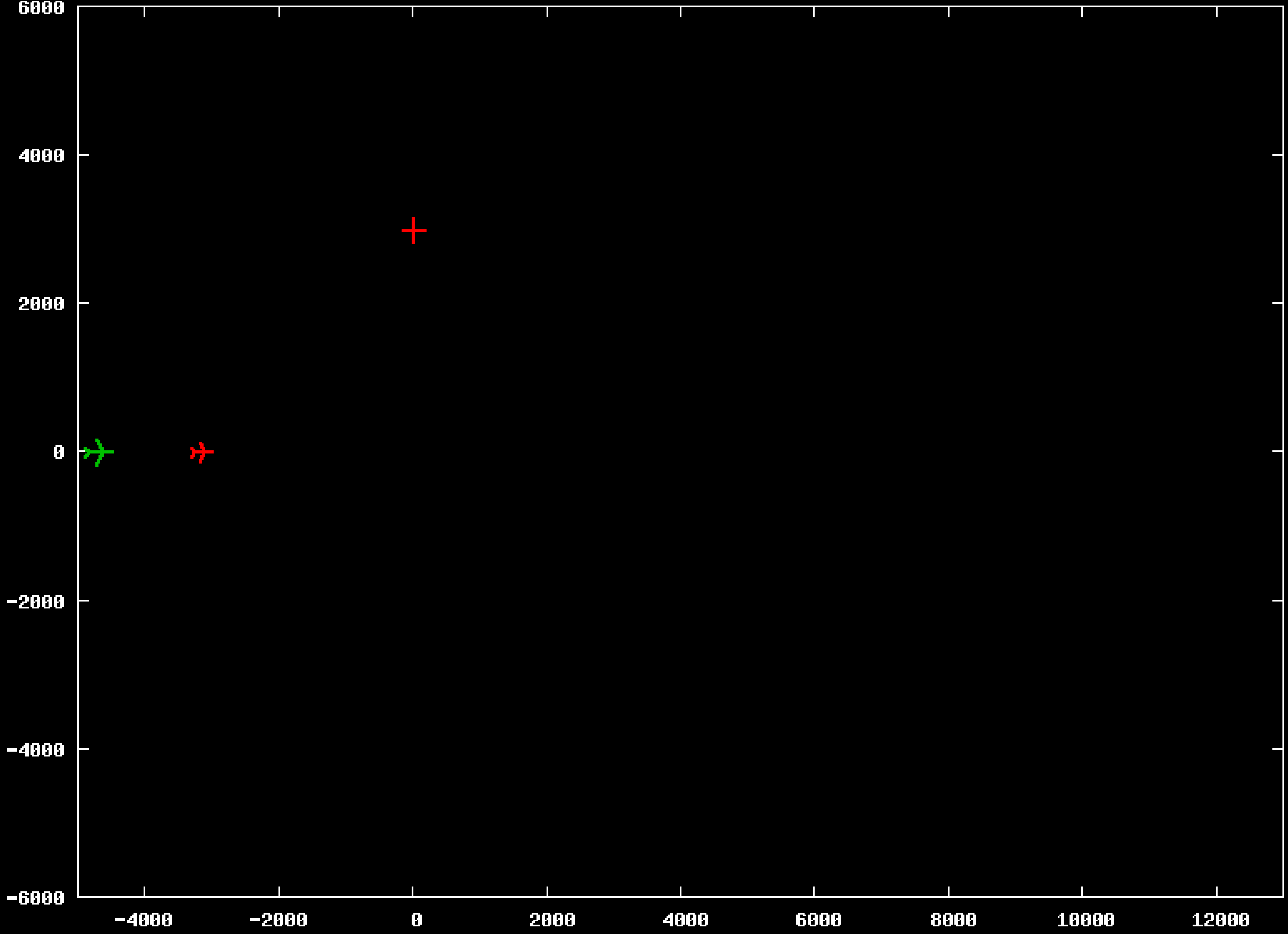
0: 0: 6 - Prevision 20 mn - Opti toutes les 4 mn EN COURS - Incertitude 4%   0.829489

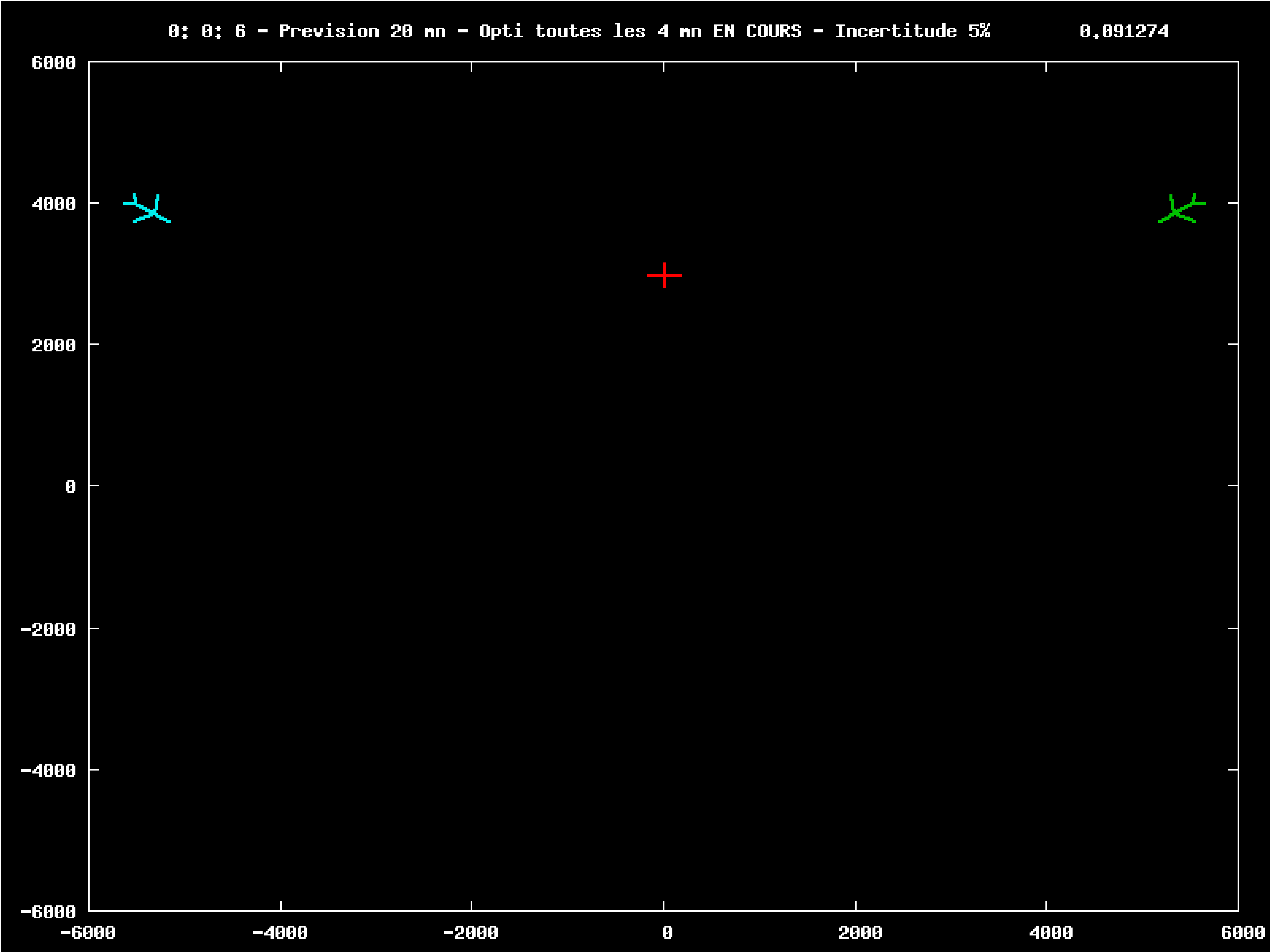


0: 0: 6 - Prevision 20 mn - Opti toutes les 4 mn EN COURS - Incertitude 5%   0.405680

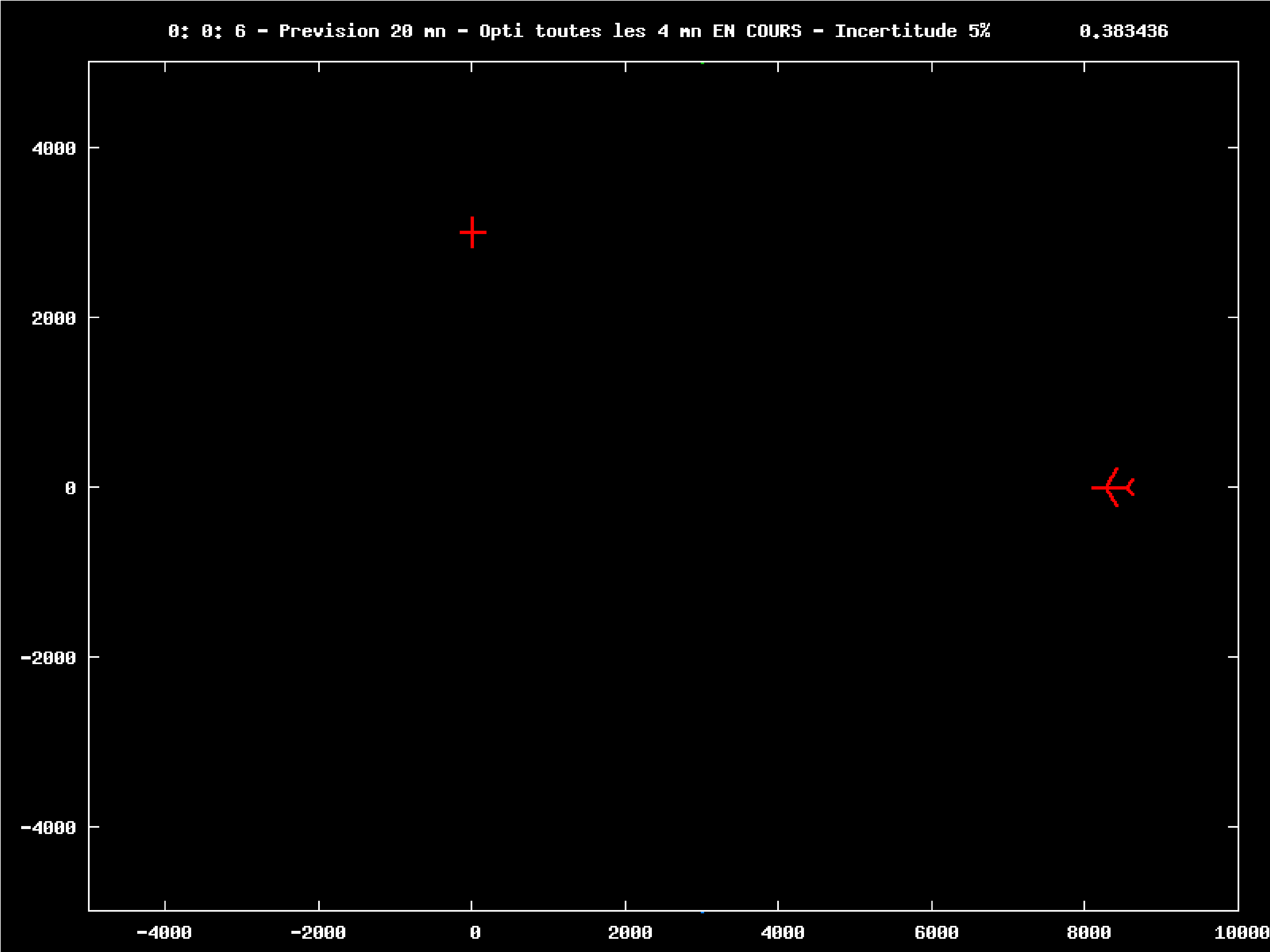


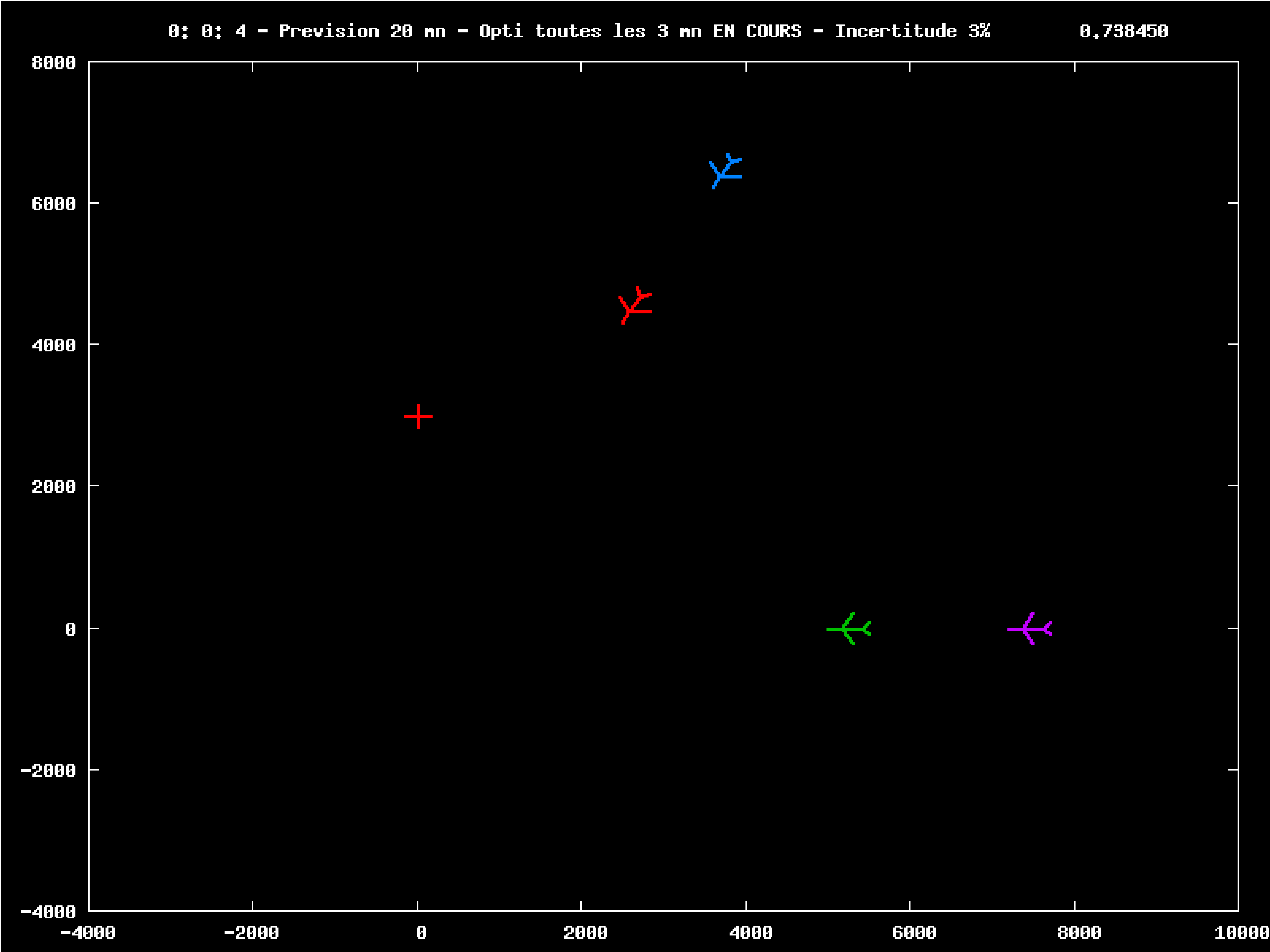
0: 0: 6 - Prevision 20 mn - Opti toutes les 4 mn EN COURS - Incertitude 5%   0.116225





```
0: 0: 6 - Prevision 20 mn - Opti toutes les 4 mn EN COURS - Incertitude 5%        0.091274
```

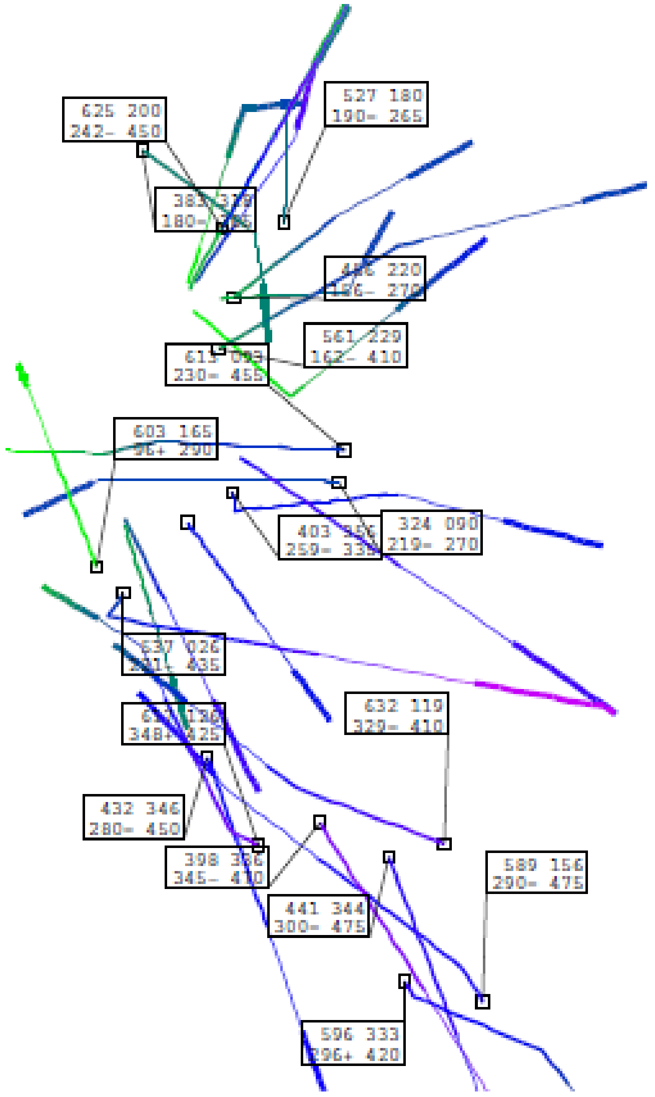

0: 0: 6 - Prevision 20 mn - Opti toutes les 4 mn EN COURS - Incertitude 5%
0.383436
4000
2000
0
-2000
-4000
-4000
-2000
0
2000
4000
6000
8000
10000


0: 0: 4 - Prevision 20 mn - Opti toutes les 3 mn EN COURS - Incertitude 3%
0.738450
8000
6000
4000
2000
0
-2000
-4000
-4000
-2000
0
2000
4000
6000
8000
10000

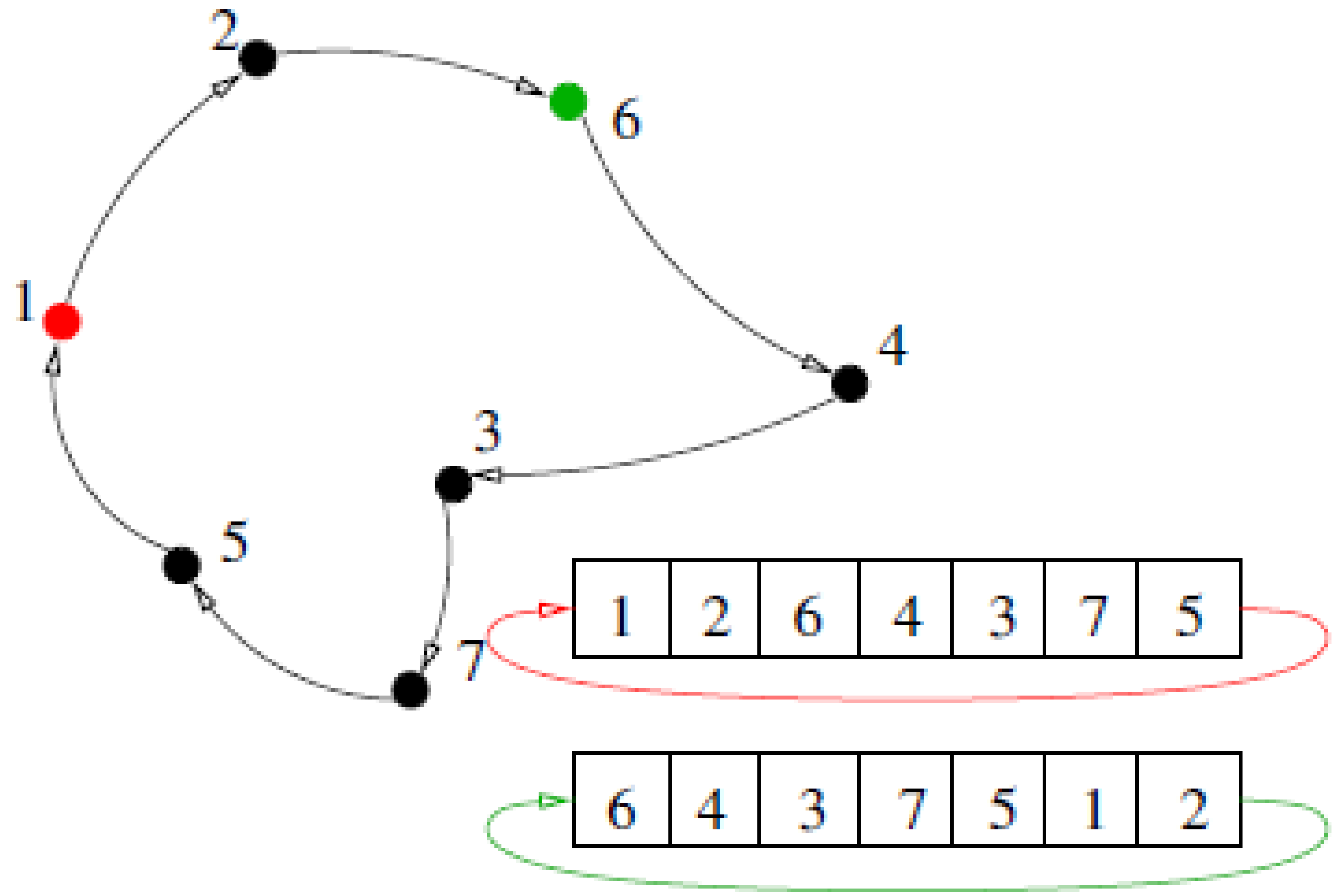
Results

Résultats

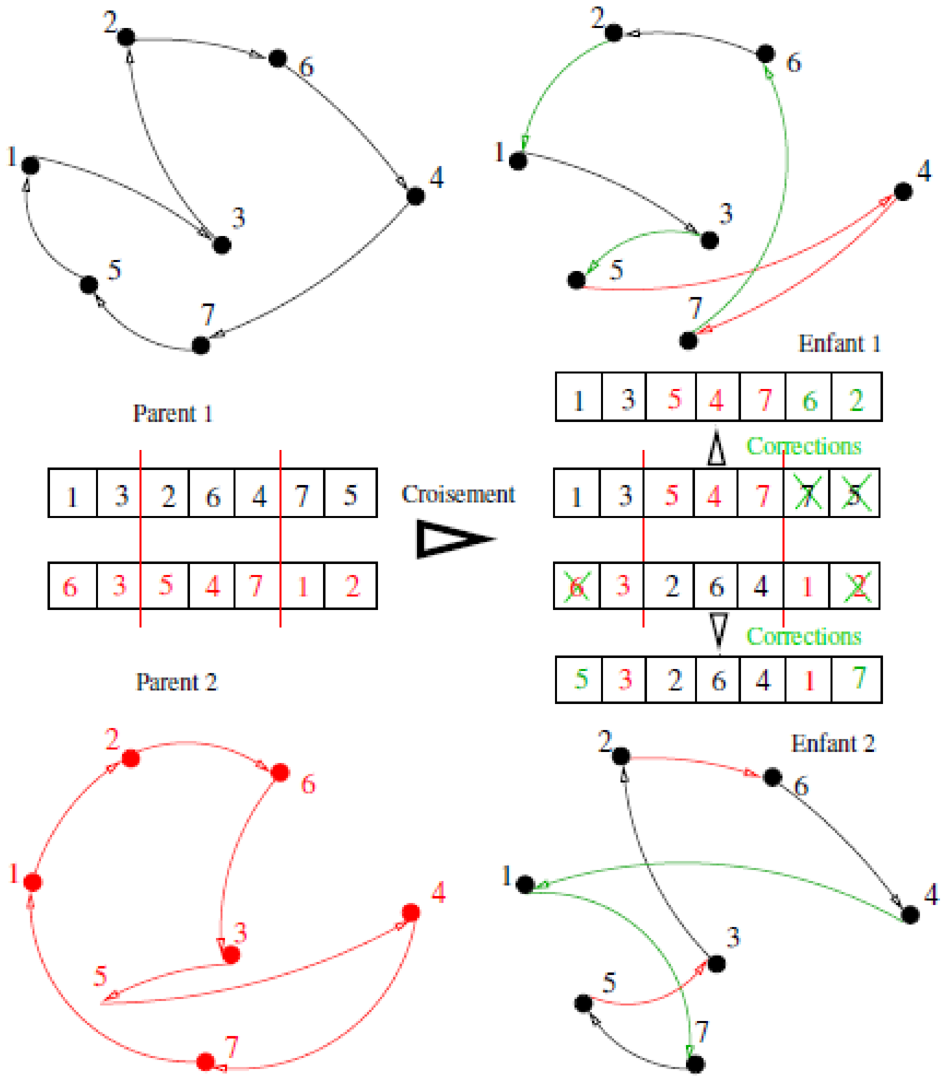
- Résout des gros conflits (30 avions)
- Intégration dans un outil de simulation (CATS/OPAS)
- Testé sur des journées de trafic réel
- Peu de restrictions sur la modélisation
- Pas de garantie d'optimalité



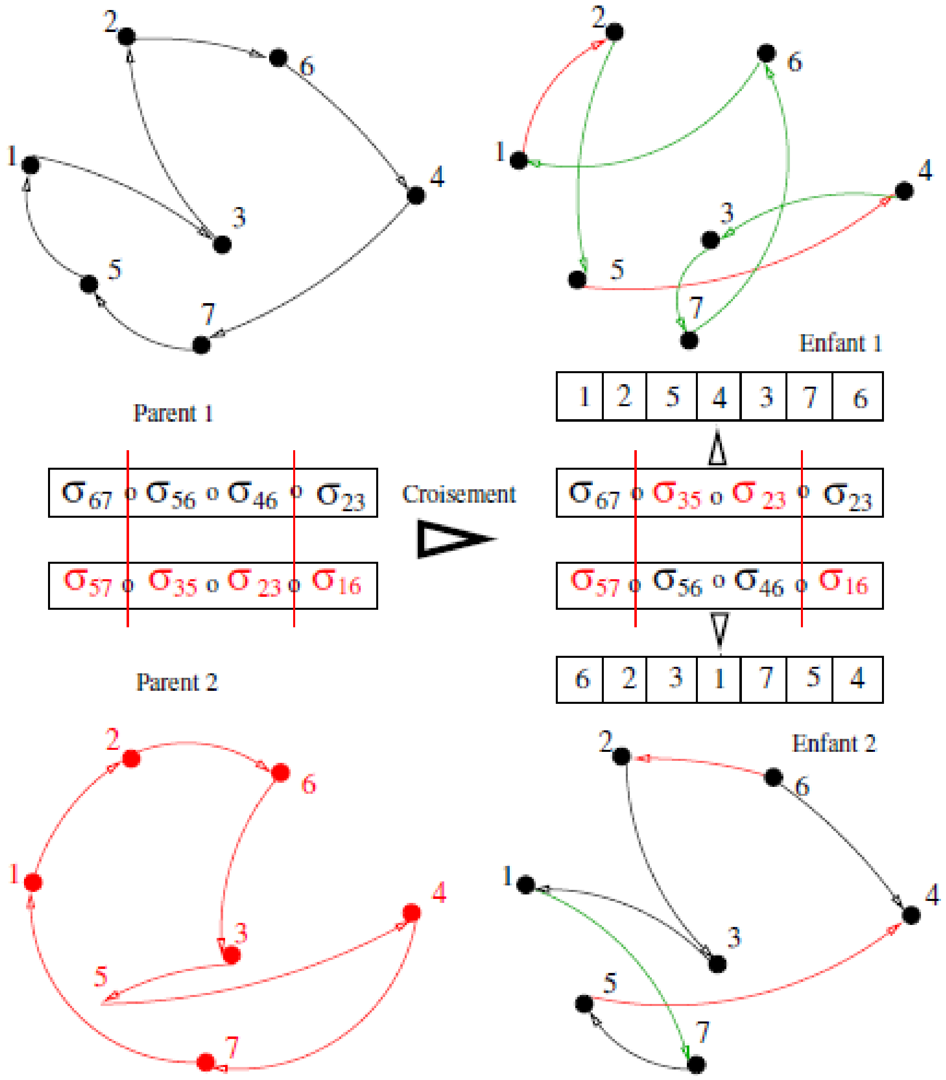
Traveling Salesman Problem (TSP)



TSP: crossover

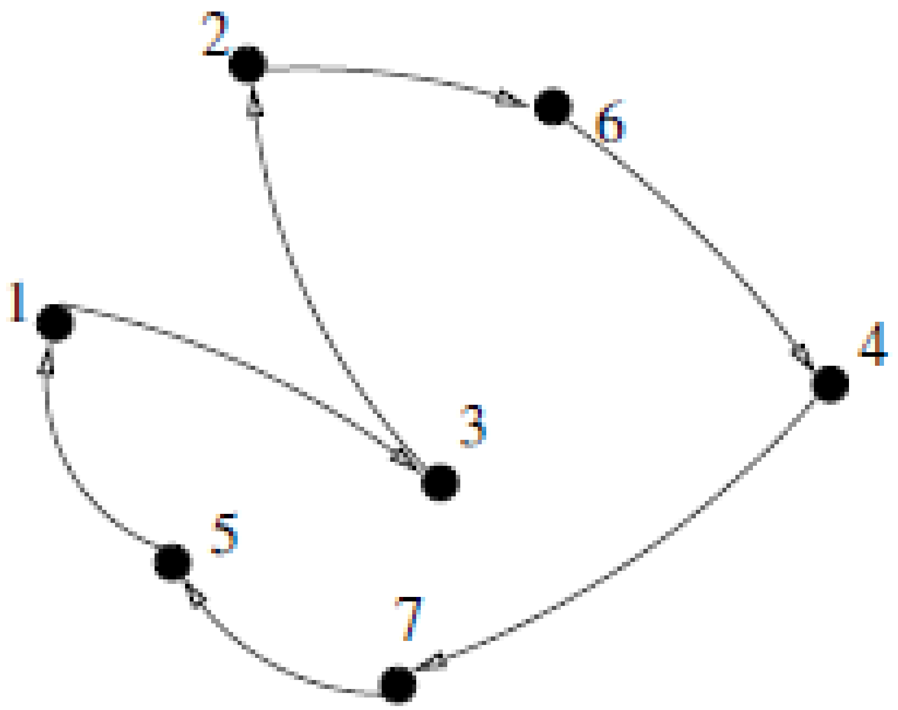


TSP: new crossover

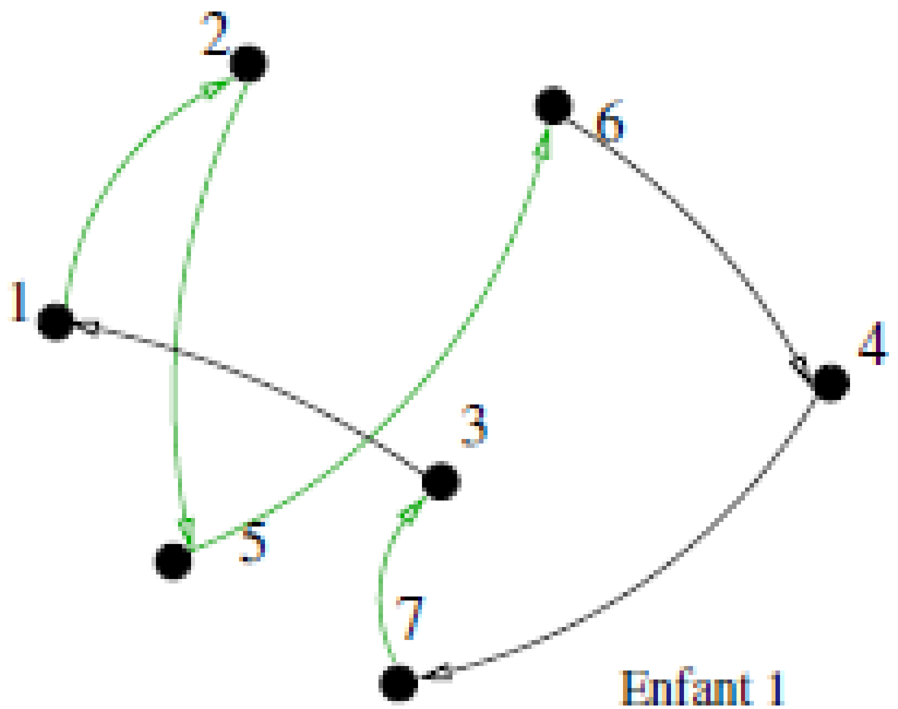


Parent 1
$\sigma_{67} \circ \sigma_{56} \circ \sigma_{46} \circ \sigma_{23}$
$\sigma_{57} \circ \sigma_{35} \circ \sigma_{23} \circ \sigma_{16}$
Parent 2
Croisement
Enfant 1
1 2 5 4 3 7 6
$\sigma_{67} \circ \sigma_{35} \circ \sigma_{23} \circ \sigma_{23}$
$\sigma_{57} \circ \sigma_{56} \circ \sigma_{46} \circ \sigma_{16}$
6 2 3 1 7 5 4
Enfant 2

TSP: mutation



Parent 1



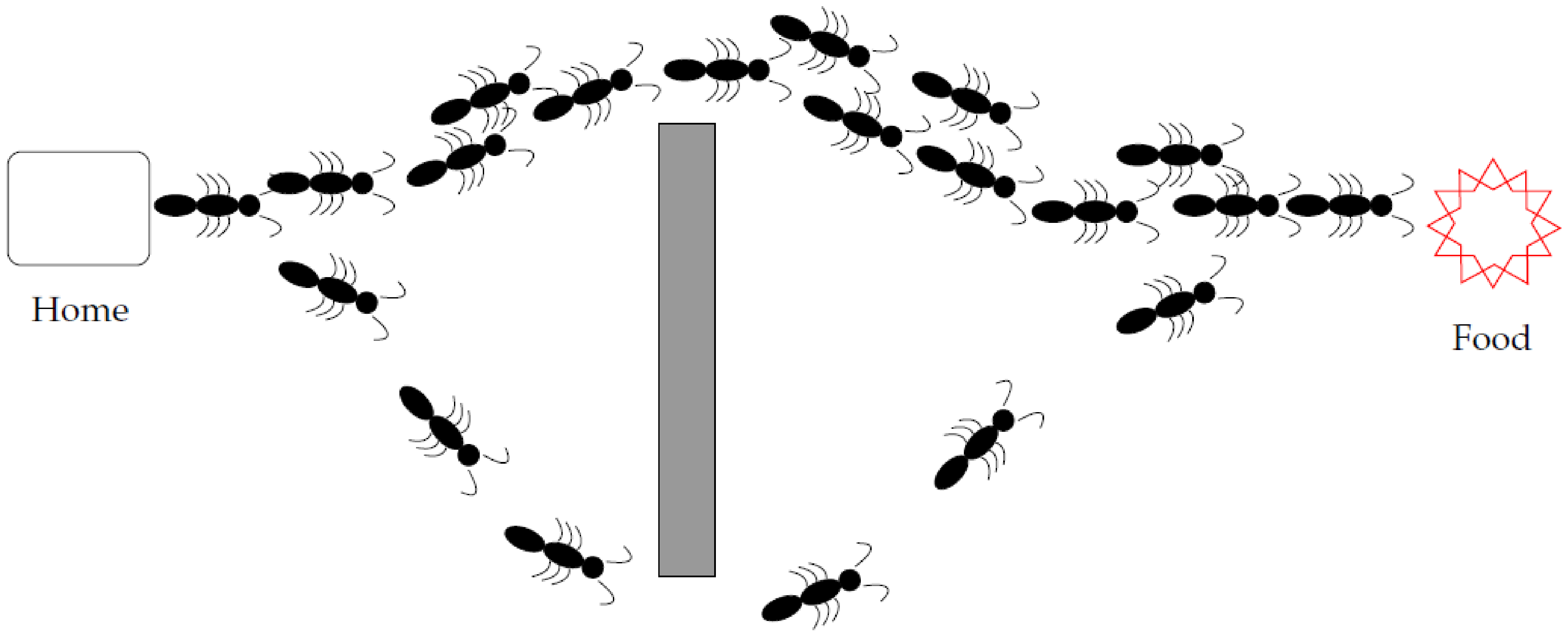
Enfant 1

1	2	5	4	3	7	6

$$\boxed{\sigma_{67} \circ \sigma_{56} \circ \sigma_{46} \circ \sigma_{23}} \xrightarrow{\text{Mutation}} \boxed{\sigma_{67} \circ \sigma_{56} \circ \sigma_{46} \circ \sigma_{35}}$$

Ant Colony Optimization (ACO)

- Mimic the ants trying to find the shortest path to food

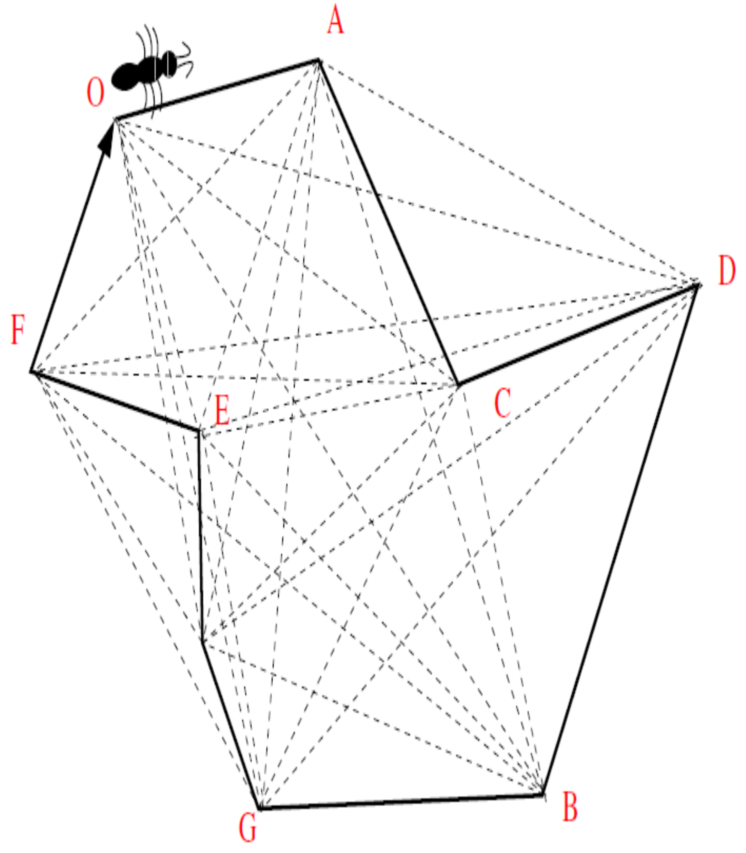


ACO

- Ants deposit pheromones according to the quality of path
- Ants more likely to follow paths with the most pheromones
- Evaporation process to prevent early convergence
- Stop when no more improvement

ACO for the TSP

- Each ant builds a path
- Choice of next city influenced by pheromones already present
- Ants deposit pheromons on the path chosen
- At each iteration, pheromons evaporate



Differential Evolution

- Pick:
  - NP vector elements population with n variables
  - F in [0,2] (differential weight)
  - CR in [0,1] (Crossover probability)
- For each vector element x
  - Pick randomly 3 distinct vectors a,b,c in population
  - Pick a random index R in [1,n]
  - For each i in [1,n] pick randomly r_i in [1,n]
    - If r_i<CR or i=R then $y_i=a_i+F(b_i-c_i)$ else $y_i=x_i$
  - If f(y) better than f(x) replace x by y in population

Other evolutionary techniques

- Particle Swarm optimization
- Evolutionary strategies
- Genetic Programming
- .....

Part III
Global deterministic methods

B&B and interval programming (global deterministic methods)

- With:

  $f(x,y) = 333.75\, y^6 + x^2\, (11\, x^2 y^2 - y^6 - 121\, y^4 - 2) + 5.5\, y^8 + x\, /\, (2y)$
- If we compute f(77617,33096), we get 1.172603.
- The correct value is -0.827396.
- Interval program was initially designed to circumvent improper rounding.

Elementary operations

- If X=[a,b] and Y=[c,d]
- X+Y=[a+c,b+d] and X-Y=[a-d,b-c]
- X*Y=
  - [ac,bd] si a>0 et c>0
  - [bc,bd] si a>0 et c<0<d
  - [bc,ad] si a>0 et d<0
  - [ad,bc] si a<0<b et c>0
  - [bd,ad] si a<0<b et d<0
  - [ad,bc] si b<0 et c>0
  - [ad,ac] si b<0 et c<0<d
  - [bd,ac] si b<0 et d<0
  - [min(bc,ad),max(ac,bd)] si a<0<b et c<0<d

Divide

- R is extended using +∝/–∝
- X/Y=
  - [b/c,+ ∝] if b<0 and d=0
  - [- ∝,b/d] and [b/c,+ ∝] if b<0 and c<0<d
  - [- ∝,+ ∝] if a<0<b
  - [- ∝,a/c] if a>0 et d=0
  - [- ∝,a/c] and [a/d,+ ∝] if a>0 et c<0<d
  - [a/d,+ ∝] if a>0 and c=0

Other operations

- All operations can be extended to interval arithmetic.
- For monotonous functions:
  - F([a,b])=[f(a),f(b)] if f is increasing
  - F([a,b])=[f(b),f(a)] if f is decreasing
  - Example: Exp([a,b])=[e^a,e^b]
- Composing functions is done by composing interval extensions of these functions

Problems

- If X=[a,b], X-X = [a-b,b-a]<>[0,0]!
- In the same way (X-1)(X+1) <> X^2-1
- ([0,2]-1)([0,2]+1)=[-1,1]*[1,3]=[-3,3]
- $[0,2]^2$-1=[0,4]-1=[-1,3]
- Associativity is preserved:
  - A+(B+C)=(A+B)+C
  - A(BC)=(AB)C
- Distributivity is lost: A(B+C)<>AB+AC

Branch and bound

- Generic name for all methods that divide and cut part of the search space.
- Here, search space is divided by cutting intervals in two, and bounds are generated by estimating the function value over each sub-interval.

Minimization

- Set: L<-{[a,b]} et e<-estimator of f on [a,b]
  - Extract I=[c,d] top of L. If e<c, redo. If I is too small, redo. If L is empty: end.
  - Build $I_1=[c,(c+d)/2]$ and $I_2=[(c+d)/2,d]$.
  - Compute $F(I_1)=[x_1,y_1]$, $F(I_2)=[x_2,y_2]$, e_1 et e_2.
  - Set e=min(e,e1,e2)
  - If $x_1<e$ then insert I_1 in L
  - If $x_2<e$ then insert I_2 in L
  - Back to start.

Computation of the estimator

- Let X=[a,b]. Different ways:
  - Easiest: e=f((a+b)/2)
  - Sampling: take n points equally spaced in X
  - Stochastic: draw randomly n points in X
  - Computer f'(x) and F'(X) et check if the sign of f'(x) is the same on X => f is monotonous and the extremum is on one side of the interval

How to sort the list of intervals

- Many ways:
  - First In First Out
  - Largest first
  - Best estimator first
  - Smaller lower bound first
  - etc...

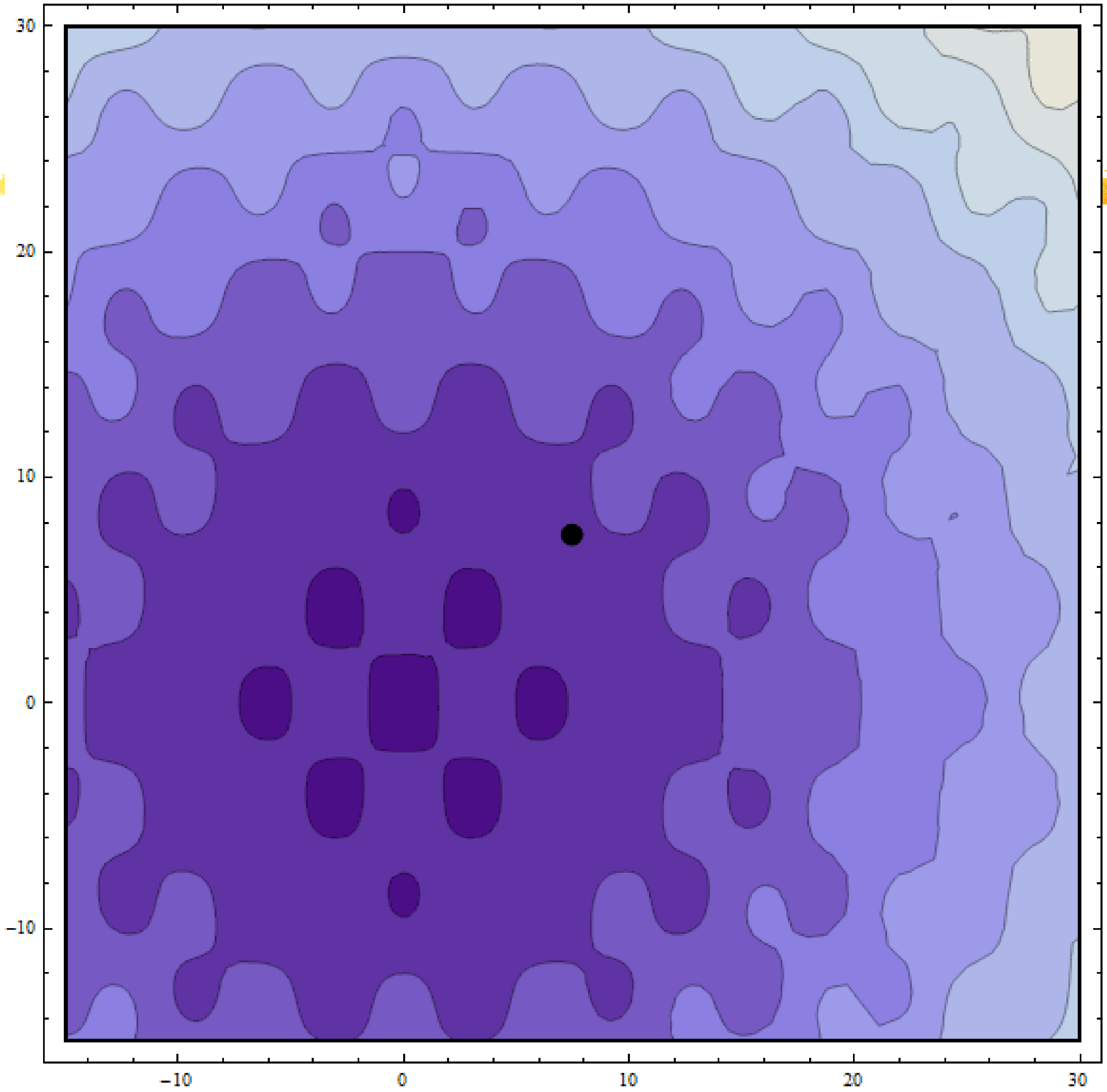
End test

- Many ways:
  - The size of the interval is smaller than a defined value
  - The size of the image of the function is smaller than a defined value
  - Etc...

More than one dimension

- For a multiple dimension functions, cutting is done on each variable in turn.
- It’s usually the largest interval which is cut first.
- The end test is modified accordingly.

$$\frac{1}{100}(x^2+y^2)-\cos(x)\cos\left(\frac{y}{\sqrt{2}}\right)$$



When to use it

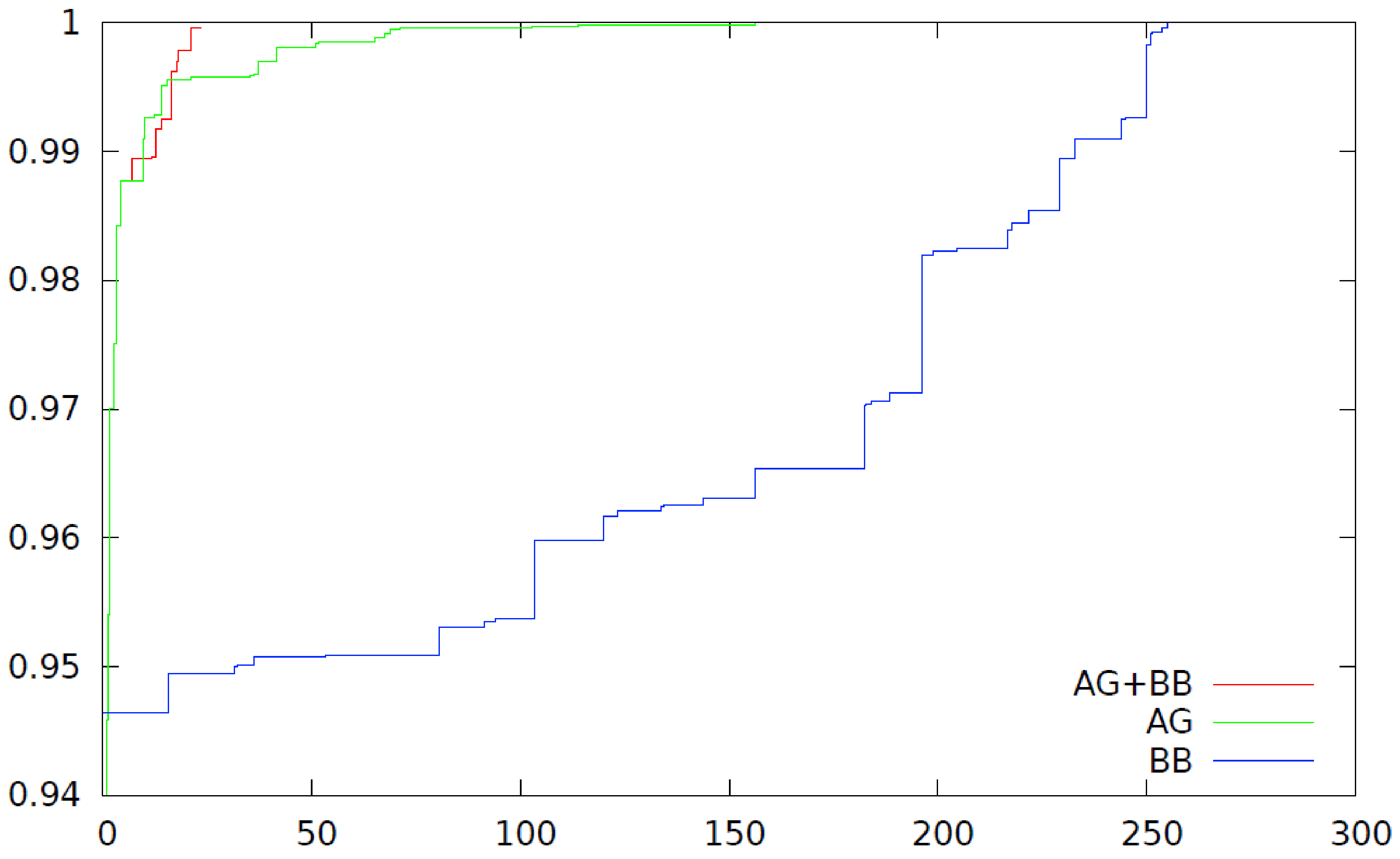
- The program computing the function can be « easily » extended to interval arithmetic.
- Method efficient when there are not too many variables.
- In theory, computation time grows as 2^N with N being the number of variables.

Part IV
Cooperation

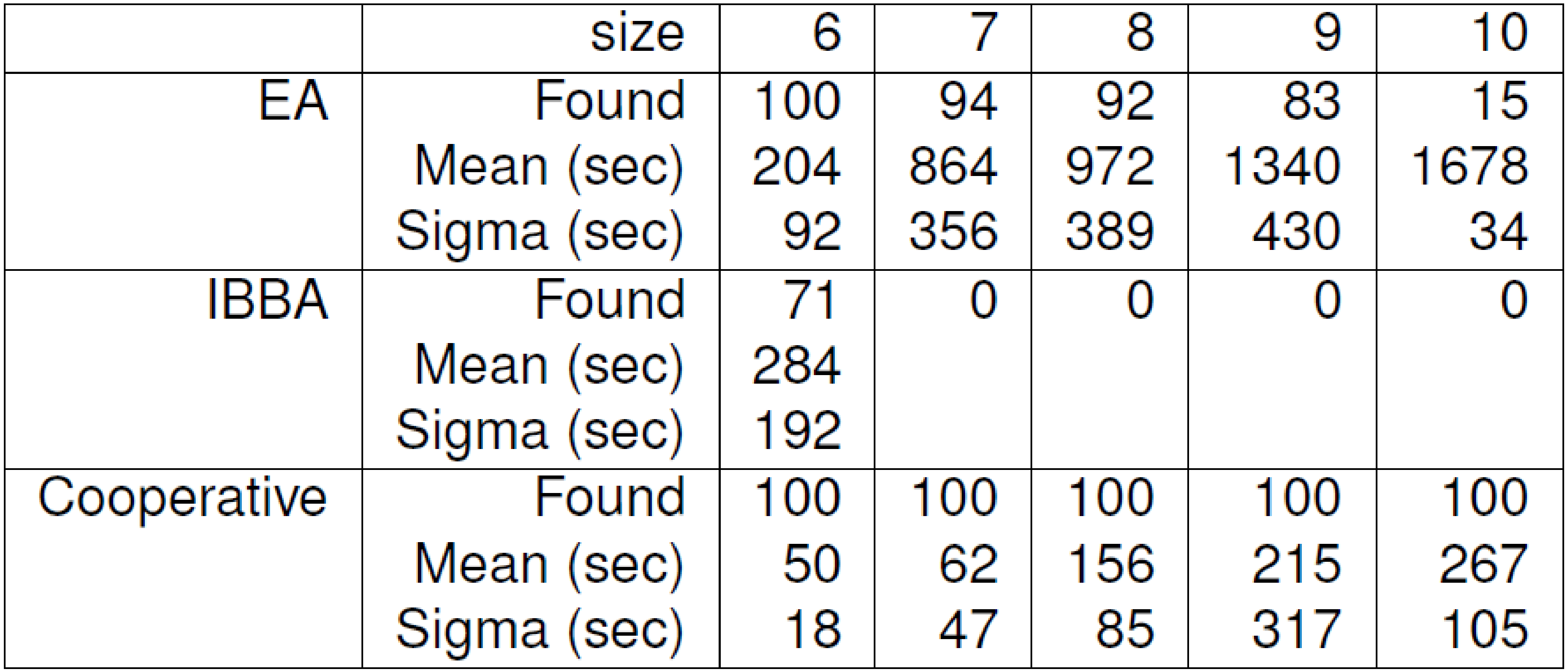
Cooperative algorithm

- IBBA thread
  - Gets from shared memory best EA element
    - =>speeds up the cutting process
  - Sends to shared memory its best element
- EA thread
  - Sends to shared memory its best element
  - Replace worst element with best IBBA element
- Update thread
  - Updates admissible domains/cleans up IBBA queue
  - Projects EA elements into the closest box

Cooperative algorithm Griewank D=6



Cooperative algorithm Statistical results



	size	6	7	8	9	10
EA	Found	100	94	92	83	15
	Mean (sec)	204	864	972	1340	1678
	Sigma (sec)	92	356	389	430	34
IBBA	Found	71	0	0	0	0
	Mean (sec)	284				
	Sigma (sec)	192				
Cooperative	Found	100	100	100	100	100
	Mean (sec)	50	62	156	215	267
	Sigma (sec)	18	47	85	317	105

Cooperative algorithm

- Useful when the extremum has to be proved
- Advantages of both algorithms and more
  - Faster than both IBBA and GA
- Same constraints as the IBBA
  - Needs code that can be extended to interval arithmetics