A new derivative-free algorithm for unconstrained optimization

K. Scheinberg and Philippe Toint

Department of Mathematics, University of Namur, Belgium

(philippe.toint@fundp.ac.be)

University of Firenze, June 2009

An application of trust-regions: unconstrained DFO

Consider the unconstrained problem

$$\min_x f(x)$$

Gradient (and Hessian) of $f(x)$ unavailable

- physical measurement
- object code
- typically small-scale (but not always. . .)

$\Rightarrow$ "Derivative free optimization" (DFO)

$f(x)$ typically very costly

Exploit each evaluation of $f(x)$ to the utmost possible

considerable interest of practitioners

Interpolation methods for DFO

Idea: Winfield (1973), Powell (1994)

Until "convergence":

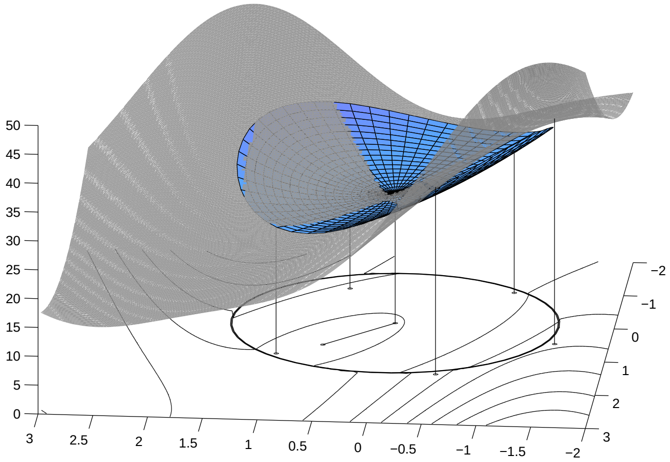
- Use the available function values to build a polynomial interpolation model m_k:

$$m_k(y_i) = f(y_i) \quad y_i \in Y;$$

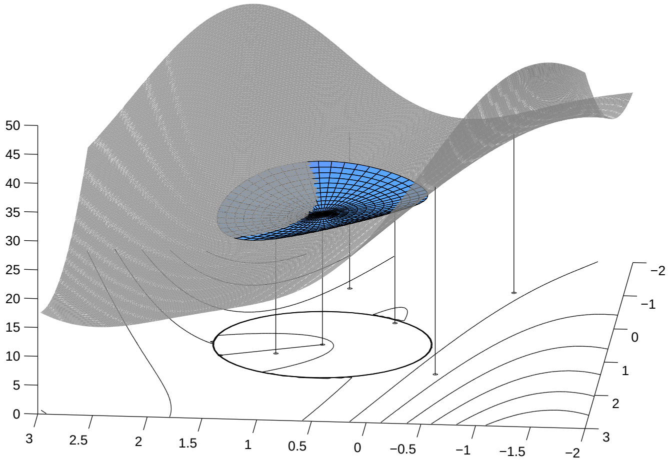
- Minimize the model in a "trust region", yielding a new potentially good point;
- Compute a new function value.

Y = interpolation set $\subseteq$ { points y_i at which $f(y_i)$ is known }

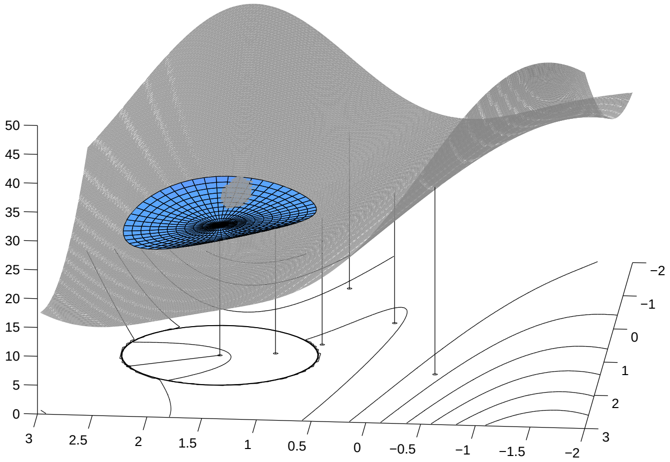
A naive trust-region method for DFO: illustration



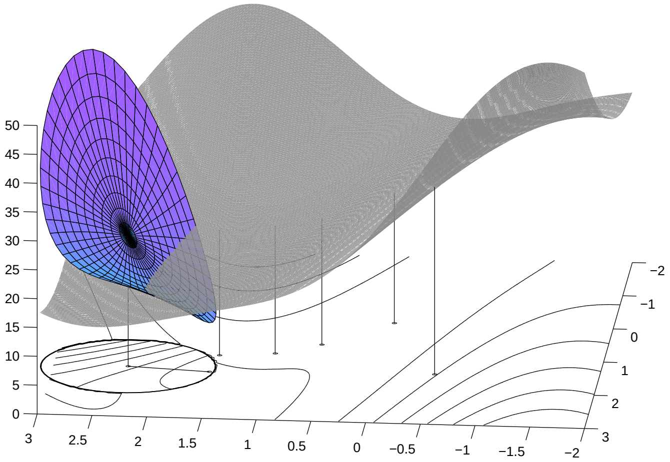
A naive trust-region method for DFO: illustration



A naive trust-region method for DFO: illustration



A naive trust-region method for DFO: illustration



Interpolation methods for DFO (2)

To be considered:

- poisedness of the interpolation set Y
- choice of models (linear, quadratic, in between, beyond)
- convergence theory
- numerical performance

Poisedness

Assume a quadratic model

$$m_k(x_k + s) = f_k + \langle g_k, s\rangle + \tfrac{1}{2}\langle s, H_k s\rangle$$

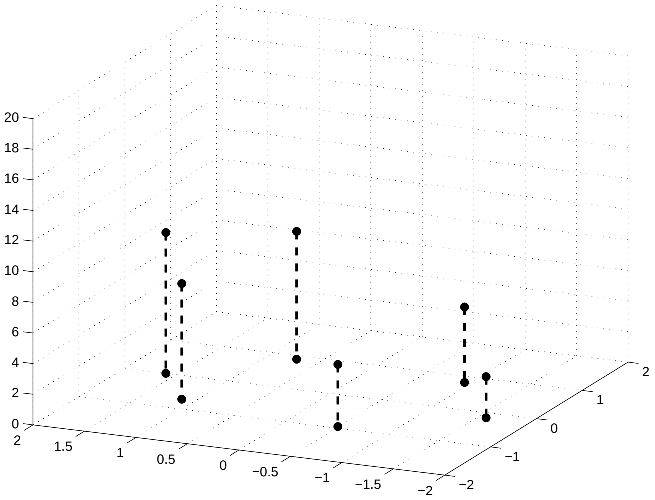
Thus

$$p = 1 + n + \tfrac{1}{2}n(n+1) = \tfrac{1}{2}(n+1)(n+2)$$

parameters to determine $\Rightarrow$ need p function values ($|Y| = p$)

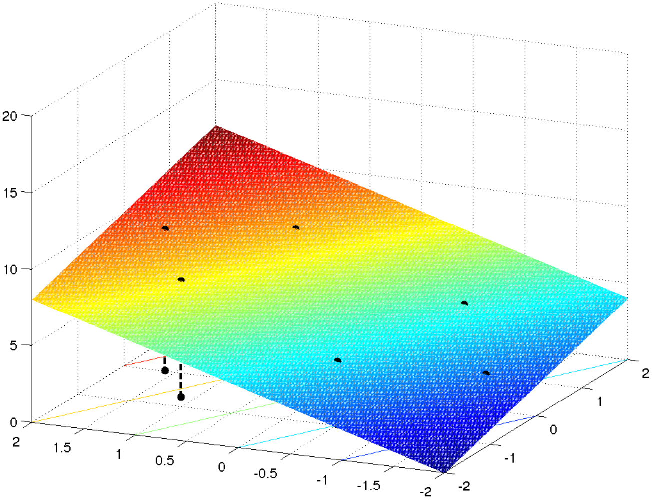
Not sufficient!

$\Rightarrow$ need geometric conditions for the points in Y . . .

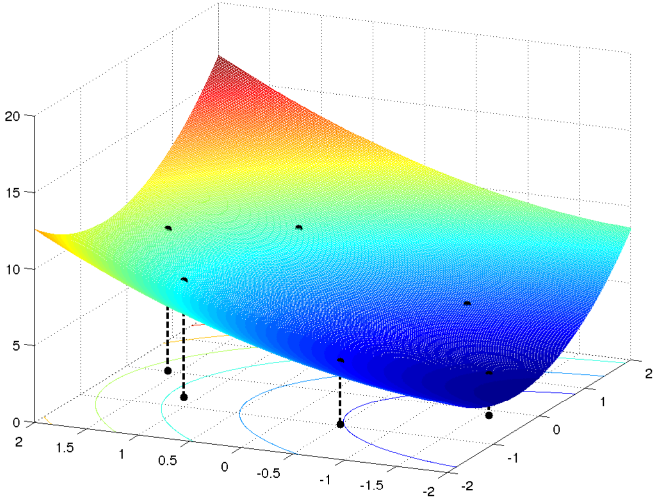


With these 6 data points in $\mathbb{R}^3$......

Poisedness: geometry with $n = 2$, $p = 6$

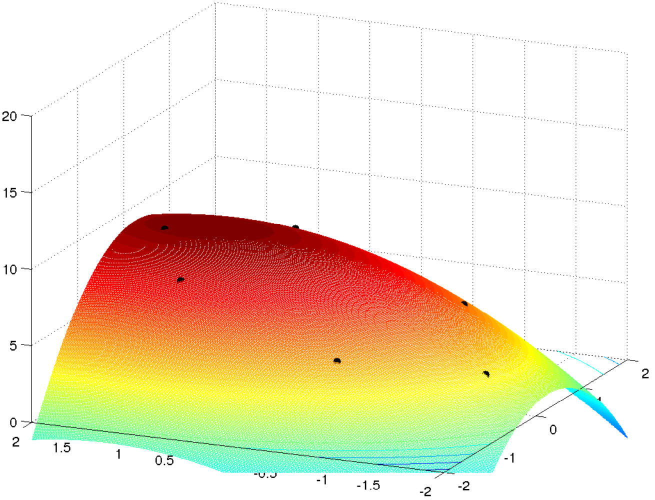


...is this the correct interpolation?



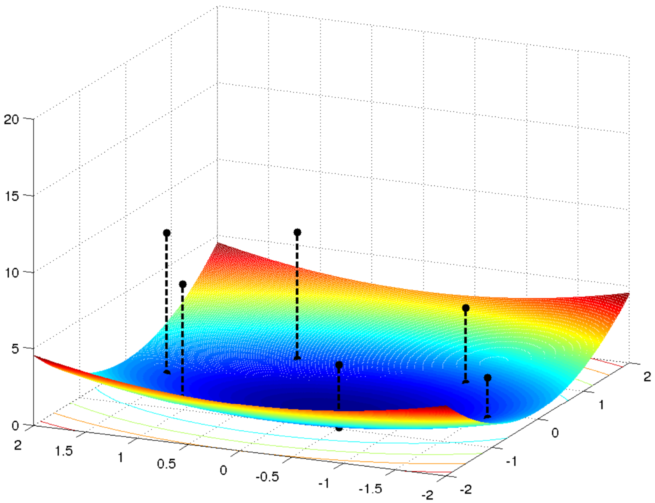
…or this?

Poisedness: geometry with $n = 2$, $p = 6$



...or this?

Poisedness: geometry with $n = 2$, $p = 6$



The difference ... is zero on a quadratic curve containing Y!

Poisedness: geometry (2)

If $\{\phi_i(\cdot)\}_{i=1}^p$ = basis for quadratic polynomials

$$\sum_{i=1}^{p} \alpha_i \phi_i(y_j) = f(y_j) \quad j = 1, \ldots, p$$

Possible poisedness measure:

$$\delta(Y) = \det \begin{pmatrix} \phi_1(y_1) & \cdots & \phi_p(y_1) \\ \vdots & & \vdots \\ \phi_1(y_p) & \cdots & \phi_p(y_p) \end{pmatrix}$$

Y (well) poised $\Leftrightarrow |\delta(Y)| \geq \epsilon$

- scale for the spread of the y_i's
- notion of geometry improvement

Lagrange polynomials

Remarkable: replace y_- by y_+ in Y:

$$\frac{\delta(Y_+)}{\delta(Y)} = L(y_+, y_-) \text{ is independent of the basis } \{\phi_i(\cdot)\}_{i=1}^p$$

where

$$\forall y \in Y \qquad L(y, y_-) = \begin{cases} 1 & \text{if } y = y_- \\ 0 & \text{if } y \neq y_- \end{cases}$$

is the Lagrange fundamental polynomial

Note: for quadratic interpolation, $L(\cdot, y)$ is a quadratic polynomial!

Powell (1994)

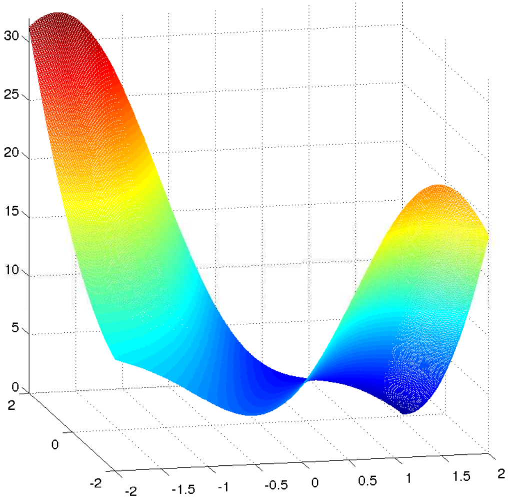
Idea: use the Lagrange polynomials to define the (quadratic) interpolant by

$$m_k(x_k + s) = \sum_{y \in Y_k} f(y) L_k(x_k + s, y)$$

And then...

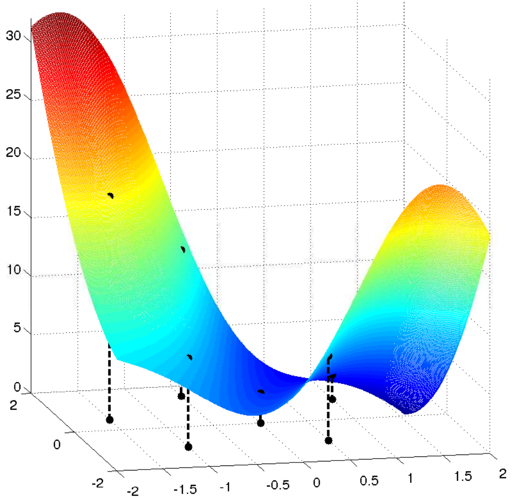
$$\|f(x_k + s) - m_k(x_k + s)\| \leq \kappa \sum_{y \in Y_k} \|x_k + s - y\|^2 |L_k(x_k + s, y)|$$

Interpolation using Lagrange polynomials: construction



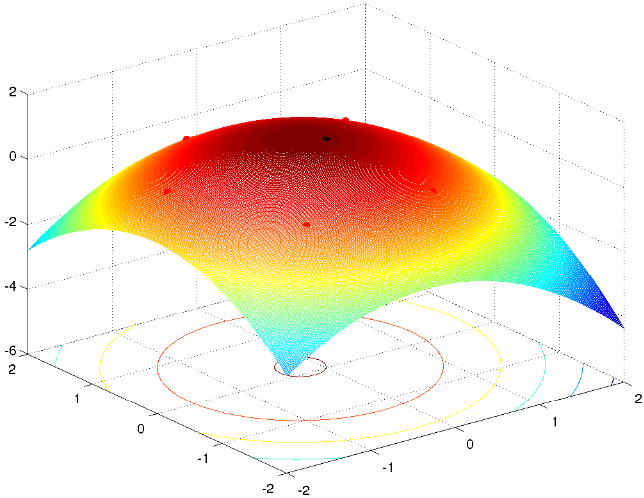
The original function. . .

Interpolation using Lagrange polynomials: construction



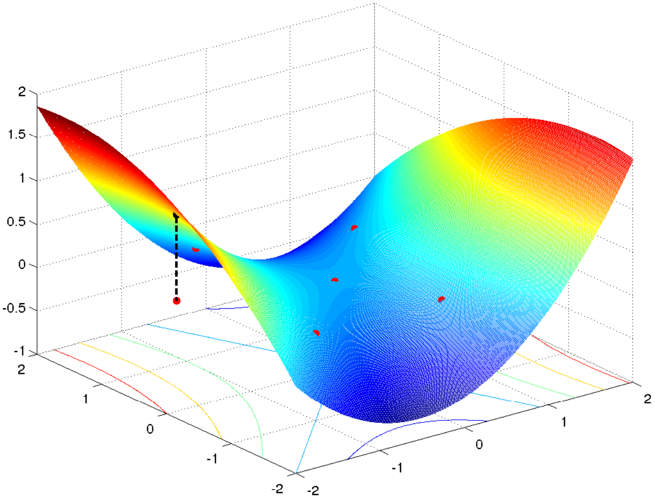
. . . and the interpolation set

Interpolation using Lagrange polynomials: construction



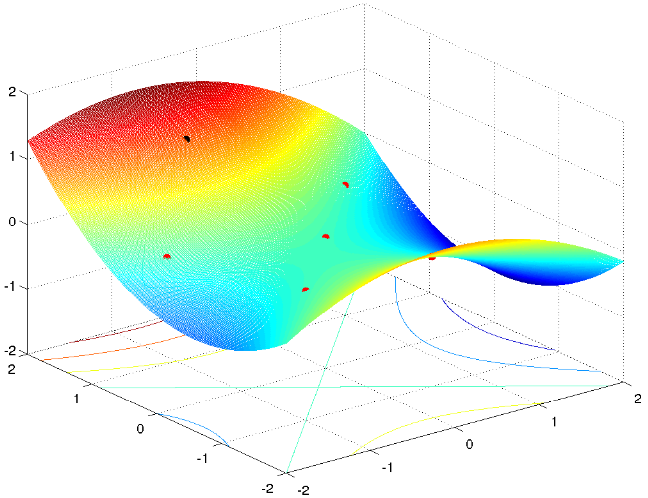
The first Lagrange polynomial

Interpolation using Lagrange polynomials: construction



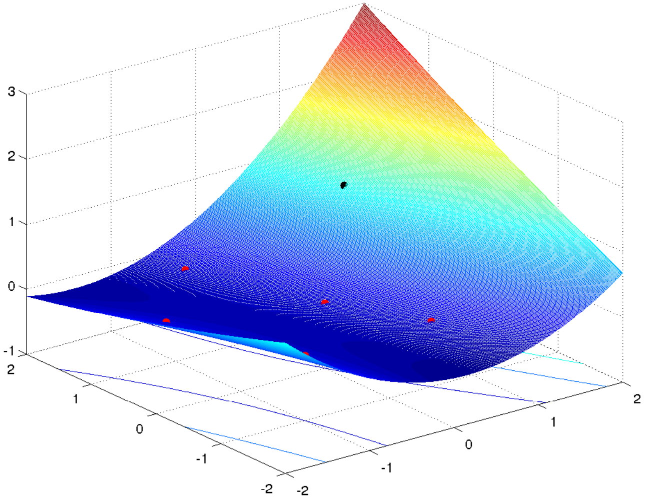
The second Lagrange polynomial

Interpolation using Lagrange polynomials: construction



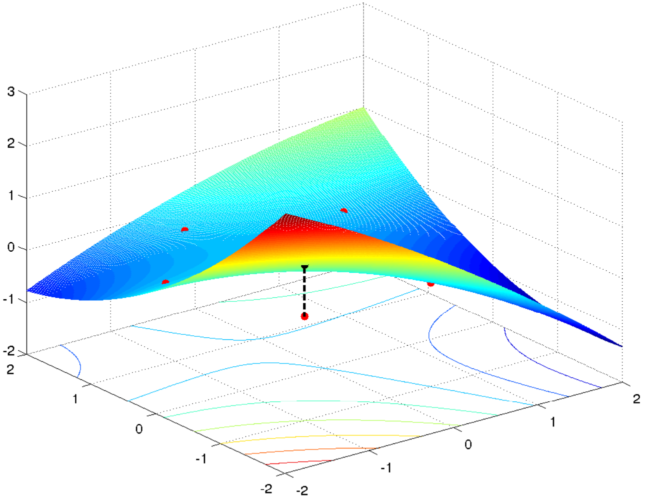
The third Lagrange polynomial

Interpolation using Lagrange polynomials: construction



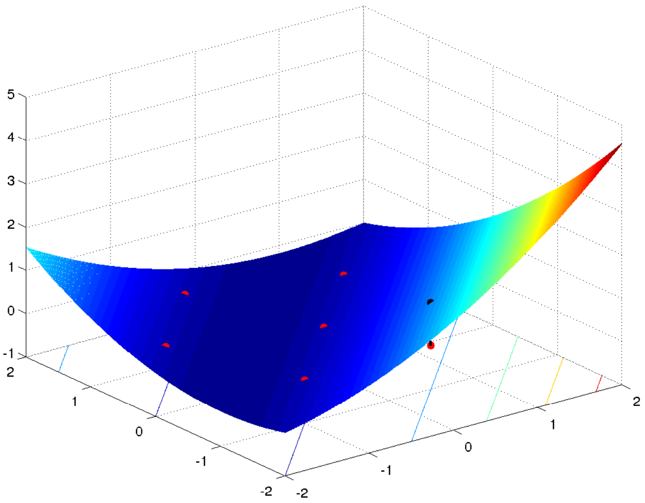
The fourth Lagrange polynomial

Interpolation using Lagrange polynomials: construction



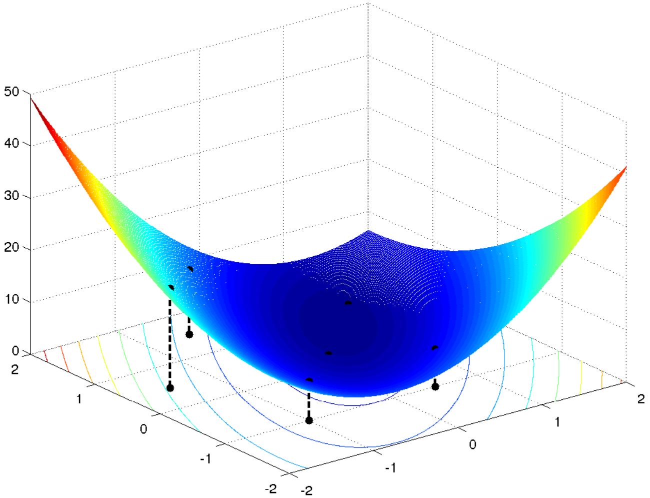
The fifth Lagrange polynomial

Interpolation using Lagrange polynomials: construction



The sixth Lagrange polynomial

Interpolation using Lagrange polynomials: construction



The final interpolating quadratic

Other algorithmic ingredients

- include a new point in the interpolation set
  - need to drop an existing interpolation point?
  - select which one to drop: make Y "as poised as possible"

Note: model/function minimizer may produce bad geometry!!

$\Rightarrow$ geometry improvement procedure ...

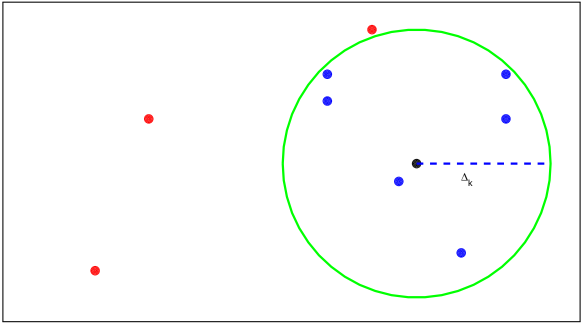
- trust-region radius management

$$\text{trust region} \; = \mathcal{B}_k = \{x_k + s \mid \|s\| \leq \Delta_k\}$$

  - standard: reduce Δ_k when "no progress"
  - DFO: more complicated! (Could reduce Δ to fast and prevent convergence...)

$\Rightarrow$ verify that Y is poised before reducing Δ_k

Improving the geometry in a ball



- attempt to reuse past points that are close to x_k
- attempt to replace a distant point of Y
- attempt to replace a close point of Y

good geometry for the current $\Delta_k \Leftrightarrow$ improvement impossible

Self-correction at unsuccessful iterations (1)

At iteration k, define the set of exchangeable far points:

$$\mathcal{F}_k = \{y \in Y_k \mid \|y - x_k\| > \Delta_k \text{ and } L_k(x_k + s_k, y) \neq 0\}$$

and the set of exchangeable close points (for some $\pi > 1$):

$$\mathcal{C}_k = \{y \in Y_k \backslash \{x_k\} \mid \|y - x_k\| \leq \Delta_k \text{ and } |L_k(x_k + s_k, y)| \geq \pi\}$$

Self-correction at unsuccessful iterations (2)

Remarkably,

Whenever

- iteration k is unsuccessful,
- $\mathcal{F}_k = \emptyset$
- Δ_k is small w.r.t. $\|g_k\|$,

then $\mathcal{C}_k \neq \emptyset$.

(an improvement of the geometry by a factor π is always possible at unsuccessful iterations when Δ_k is small and all exchangeable far points have been considered)

$\Rightarrow$ no need to reduce Δ_k forever!

Trust-region algorithm for DFO (1)

Algorithm 0.1: TR for DFO

Step 0: Initialization. Given: x_0, Δ_0, Y_0 $(\rightarrow L_0(\cdot, y))$. Set $k = 0$.

Step 1: Criticality test [complicated and not discussed here]

Step 2: Solve the subproblem. Compute s_k that sufficiently reduces $m_k(x_k + s)$ within the trust region,

Step 3: Evaluation. Compute $f(x_k + s_k)$ and

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}.$$

Step 4: Define the next iterate and interpolation set.

the big question

Step 5: Update the Lagrange polynomials.

Trust-region algorithm for DFO (2)

Algorithm 0.2: Step 4: Define x_{k+1} and Y_{k+1}

Step 4a: Successful iteration. If $\rho_k \geq \eta_1$, accept $x_k + s_k$, increase Δ_k and exchange $x_k + s_k$ with

$$y = \arg\max_{y \in Y_k} \|y - (x_k + s_k)\|^2 |L_k(x_k + s_k, y)|$$

Step 4b: Replace far point. If $\rho_k < \eta_1$ (+ other technical condition) and $\mathcal{F}_k \neq \emptyset$, reject $x_k + s_k$, keep Δ_k and exchange $x_k + s_k$ with

$$y = \arg\max_{y \in \mathcal{F}_k} \|y - (x_k + s_k)\|^2 |L_k(x_k + s_k, y)|$$

Step 4c: Replace close point. If $\rho_k < \eta_1$ (+ other technical condition) and $\mathcal{C}_k \neq \emptyset$, reject $x_k + s_k$, keep Δ_k and exchange $x_k + s_k$ with

$$y = \arg\max_{y \in \mathcal{C}_k} \|y - (x_k + s_k)\|^2 |L_k(x_k + s_k, y)|$$

Step 4d: Decrease the radius. Otherwise, reject $x_k + s_k$, keep Y_k, and reduce Δ_k.

Global convergence results

If the model is at least fully linear, then

$$\liminf_{k\to\infty} \|\nabla_x f(x_k)\| = \liminf_{k\to\infty} \|g_k\| = 0$$

Scheinberg and T. (2009)

With more costly algorithm:

If the model is at least fully linear, then

$$\lim_{k\to\infty} \|\nabla_x f(x_k)\| = \lim_{k\to\infty} \|g_k\| = 0$$

If the model at least fully quadratic, then iterates converge to 2nd-order critical points

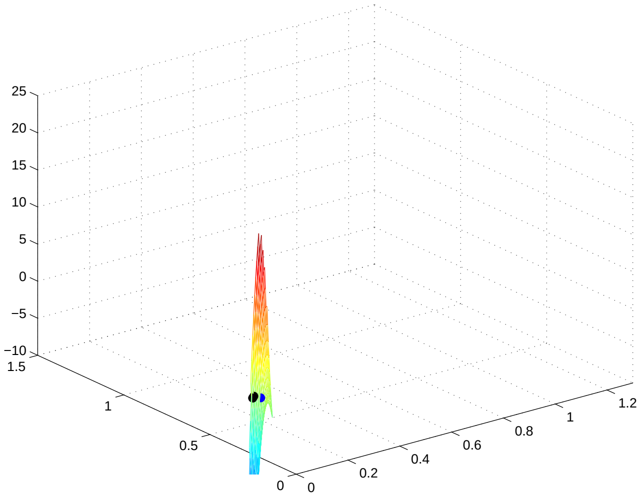
For an efficient numerical method...

Many more issues:

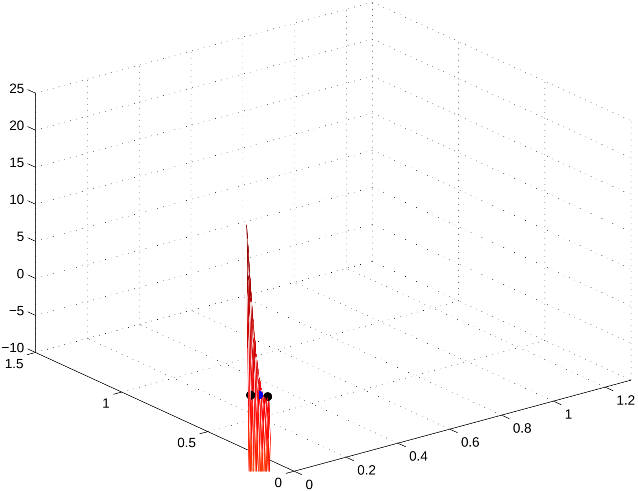
- which Hessian approximation?
  (full/vs diagonal or structured)
- details of criticality tests difficult
- details for numerically handling interpolation polynomials
  (Lagrange, Newton),
- reference shifts,
- ...

good codes around: NEWUOA, DFO $\Rightarrow$ efficient solvers

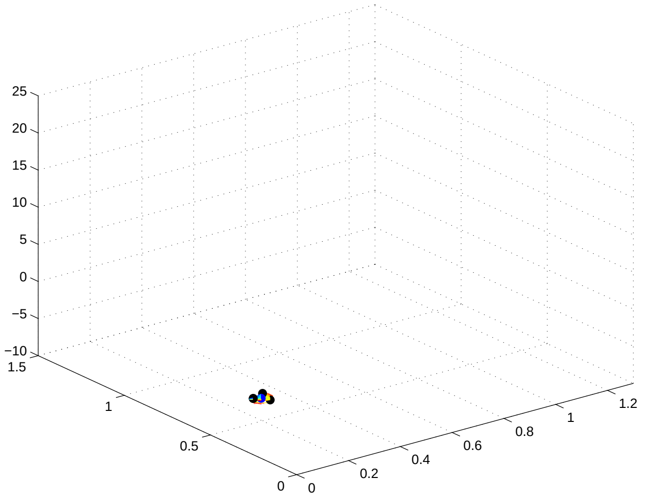
Powell (2008 and previously), Conn, Scheinberg and T. (1998)
Conn, Scheinberg and Vicente (2008)


25
20
15
10
5
0
−5
−10
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2

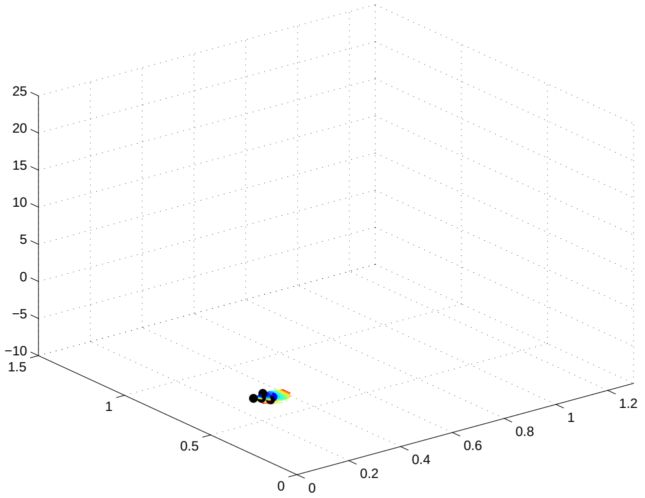
On the ever famous banana function...



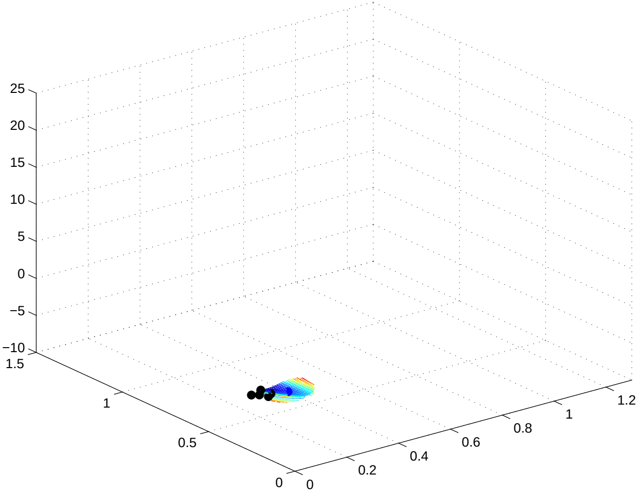
On the ever famous banana function. . .



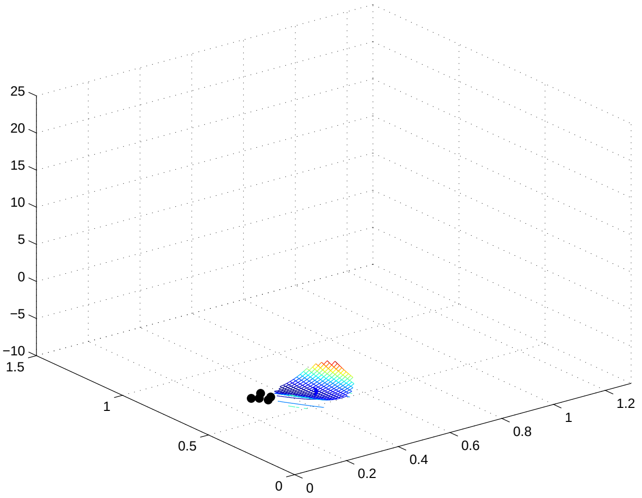
On the ever famous banana function. . .



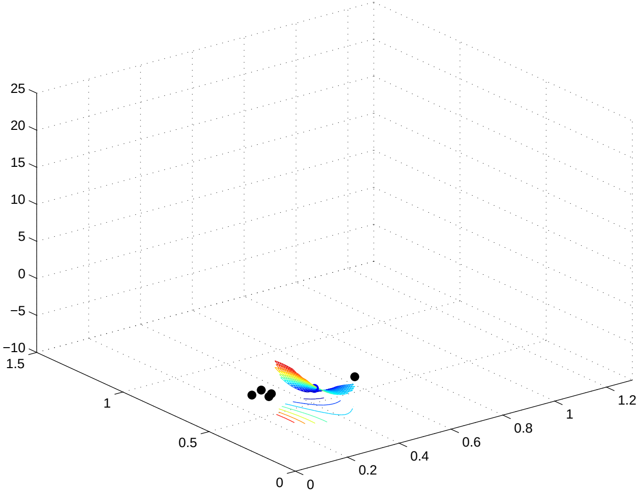
On the ever famous banana function. . .



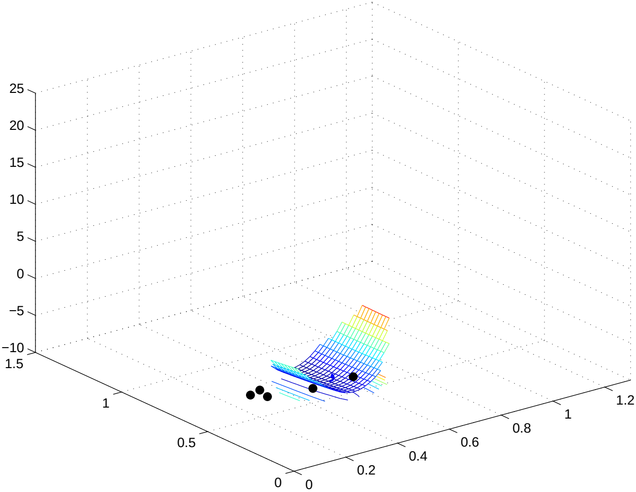
On the ever famous banana function. . .



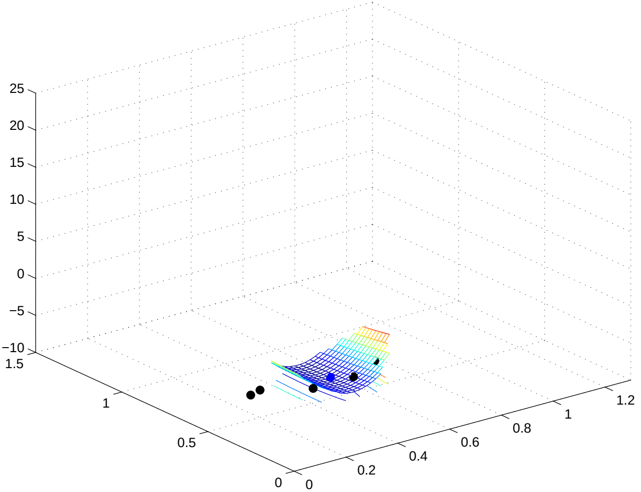
On the ever famous banana function. . .



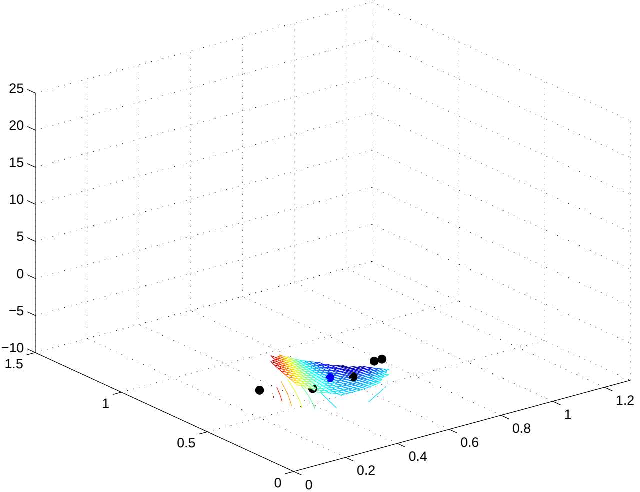
On the ever famous banana function. . .



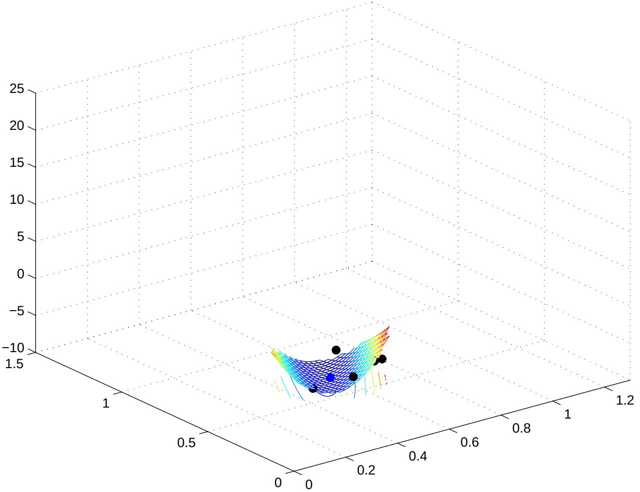
On the ever famous banana function...



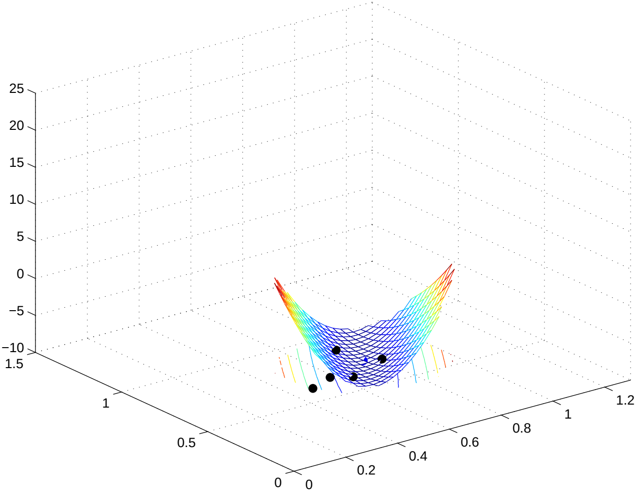
On the ever famous banana function...

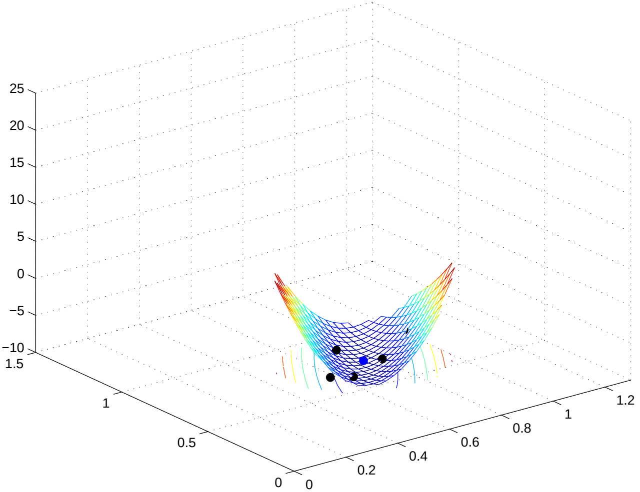


On the ever famous banana function. . .

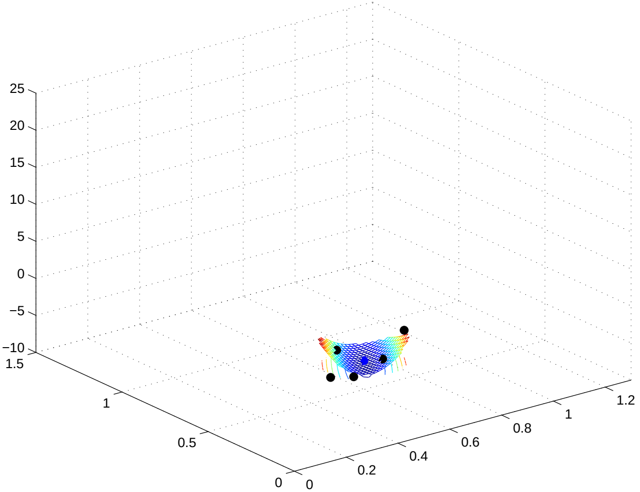


On the ever famous banana function...

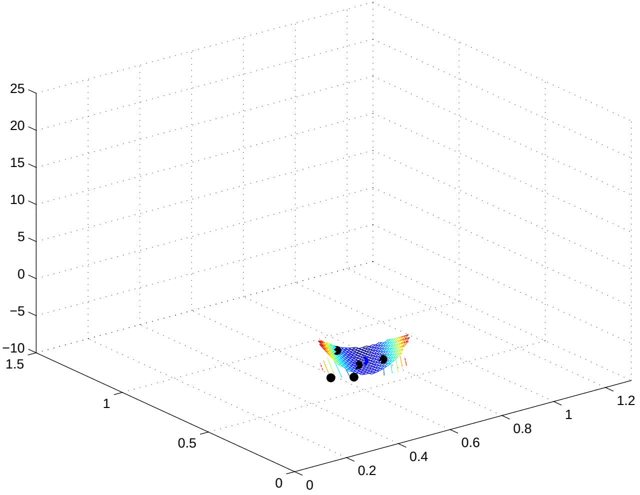



25
20
15
10
5
0
−5
−10
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2

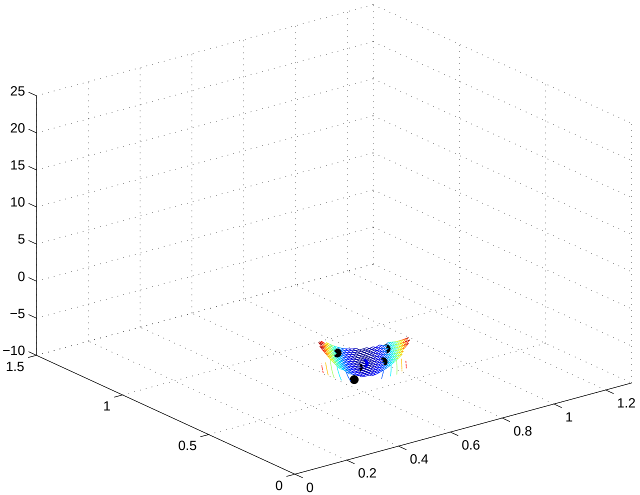
On the ever famous banana function. . .



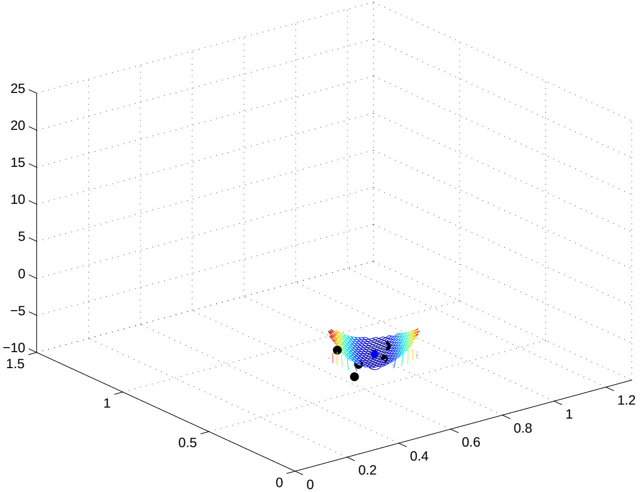
On the ever famous banana function. . .



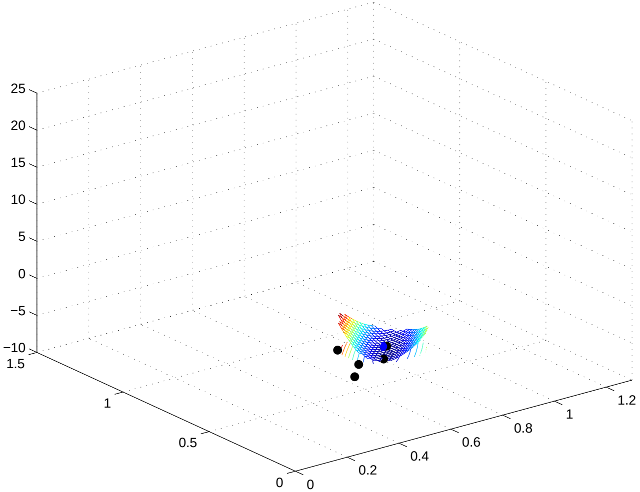
On the ever famous banana function...



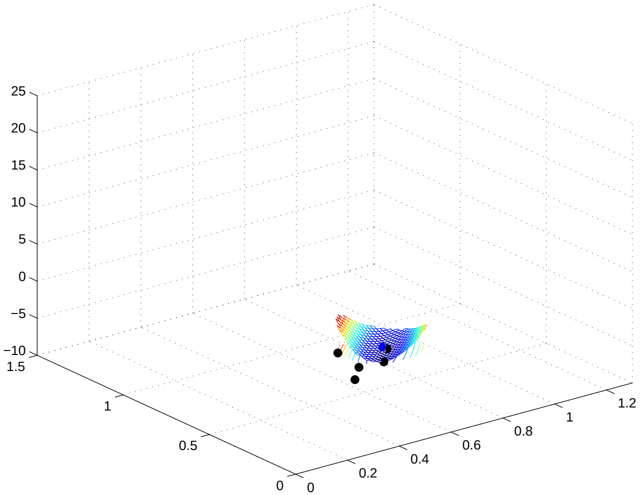
On the ever famous banana function. . .



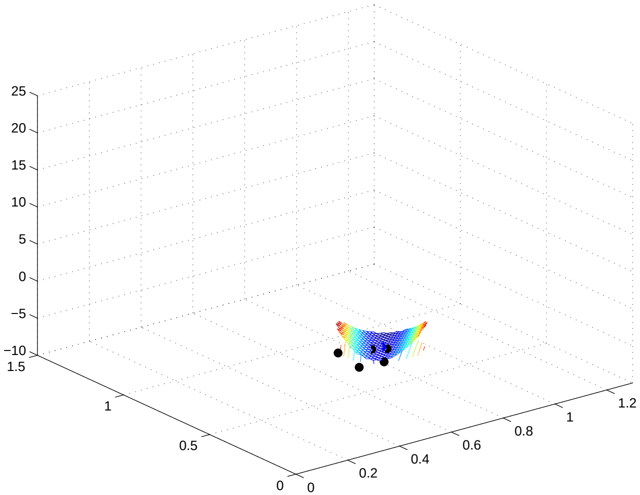
On the ever famous banana function. . .



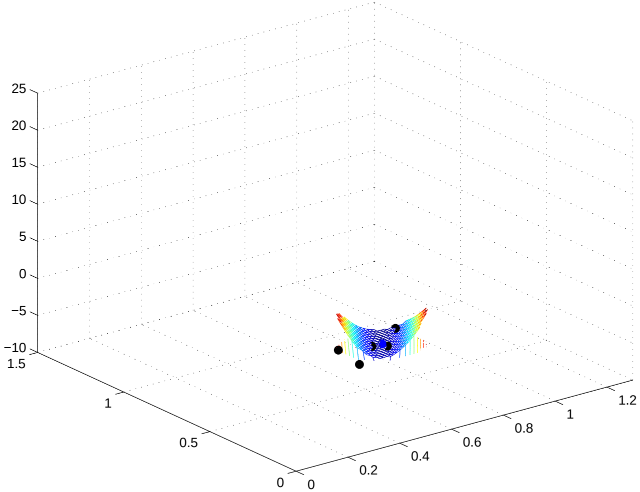
On the ever famous banana function. . .

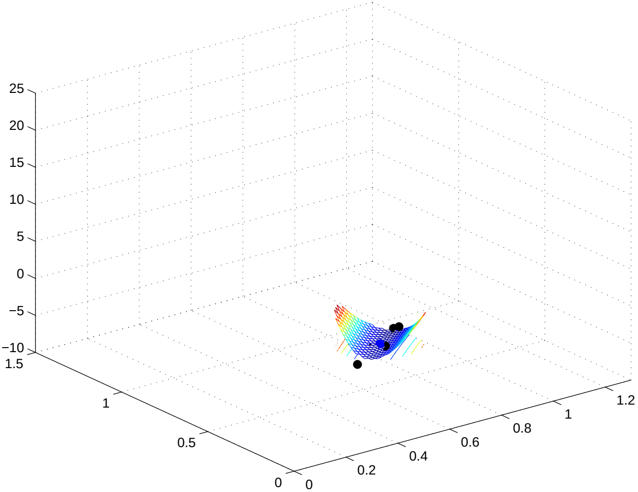


On the ever famous banana function. . .

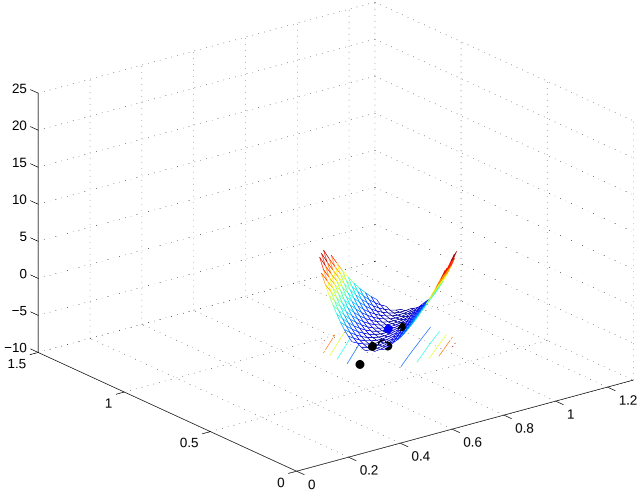


On the ever famous banana function. . .

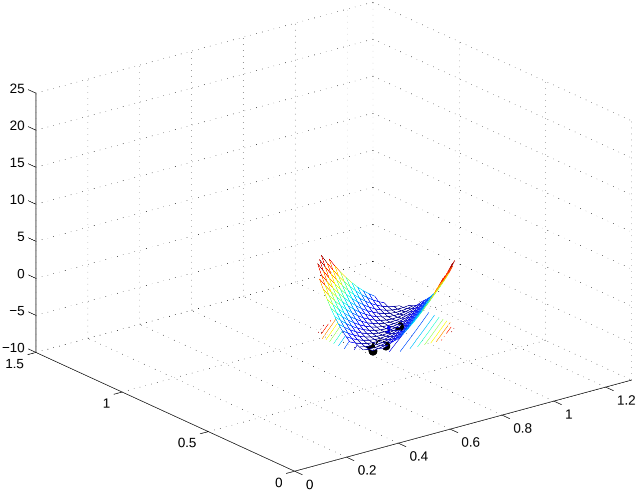



25
20
15
10
5
0
−5
−10
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2

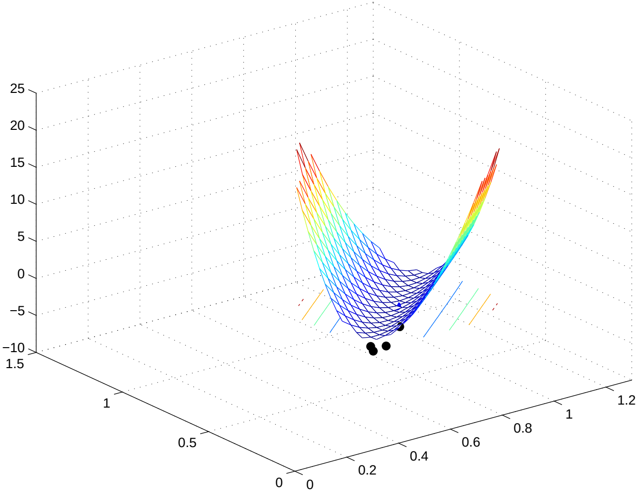
On the ever famous banana function. . .



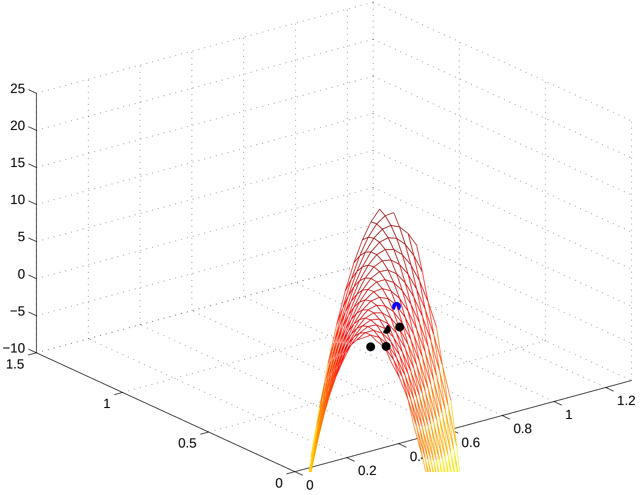
On the ever famous banana function...



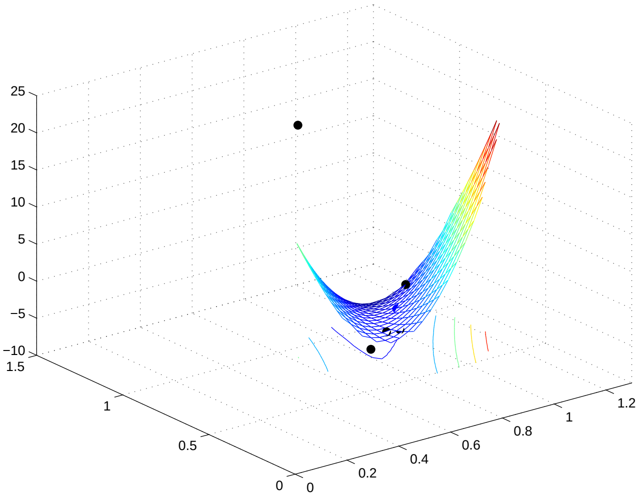
On the ever famous banana function...



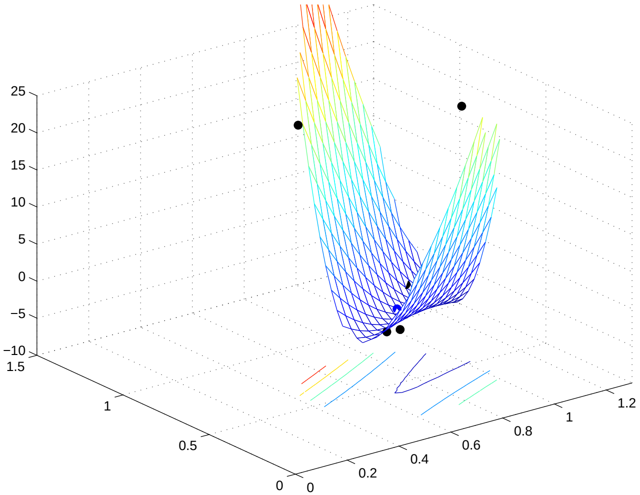
On the ever famous banana function. . .



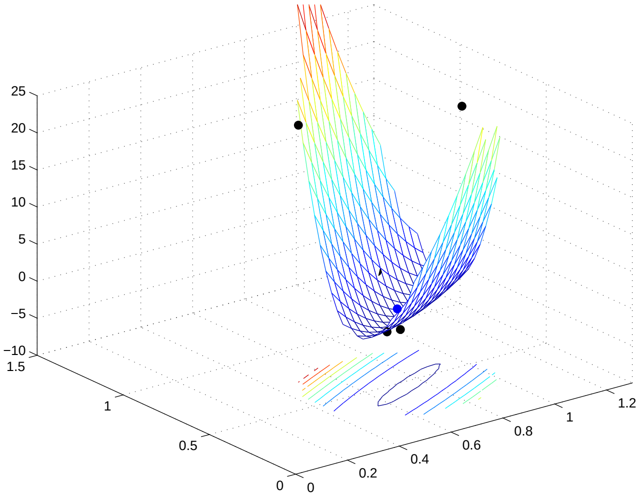
On the ever famous banana function. . .



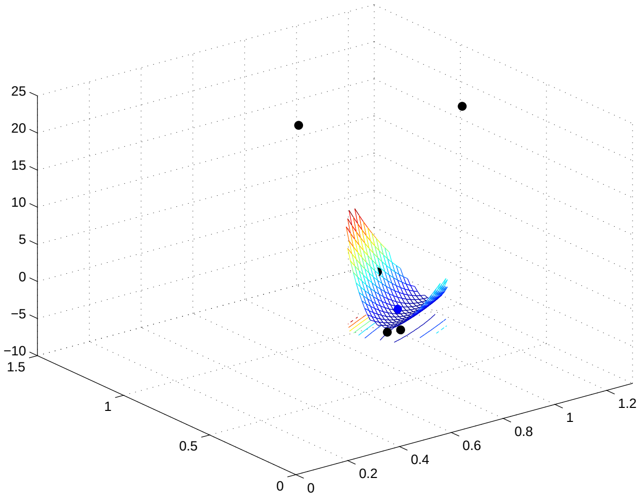
On the ever famous banana function. . .



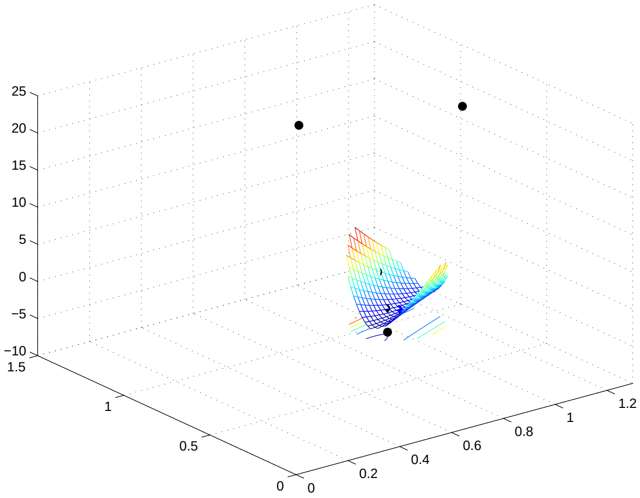
On the ever famous banana function...



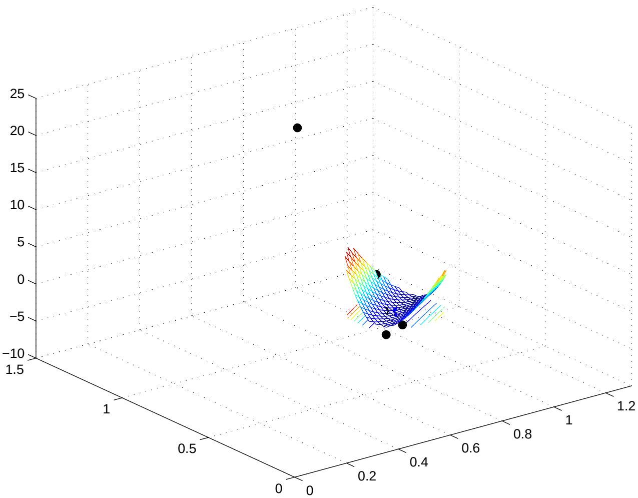
On the ever famous banana function...



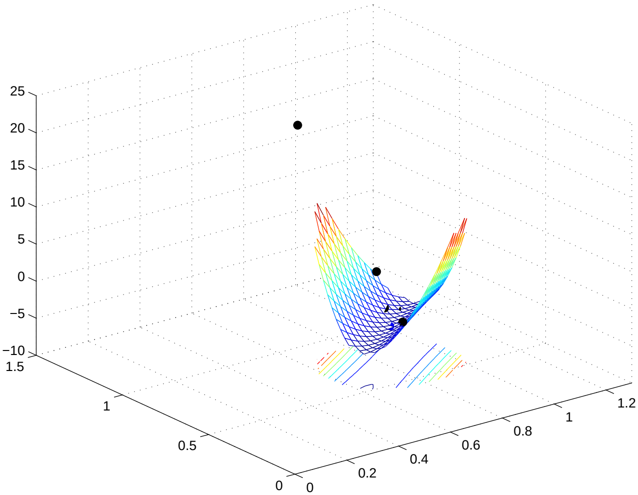
On the ever famous banana function. . .



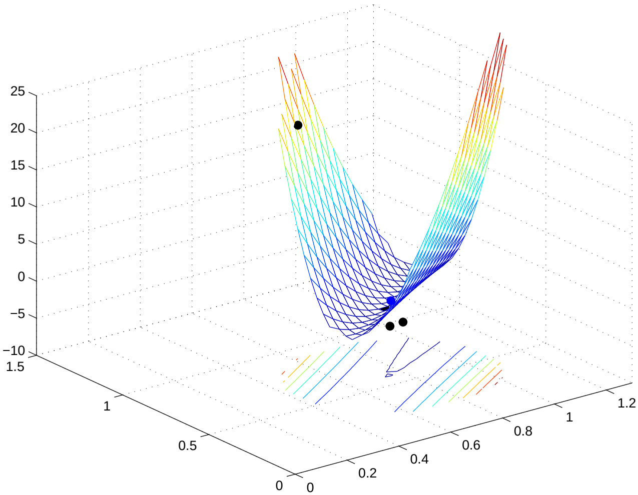
On the ever famous banana function. . .



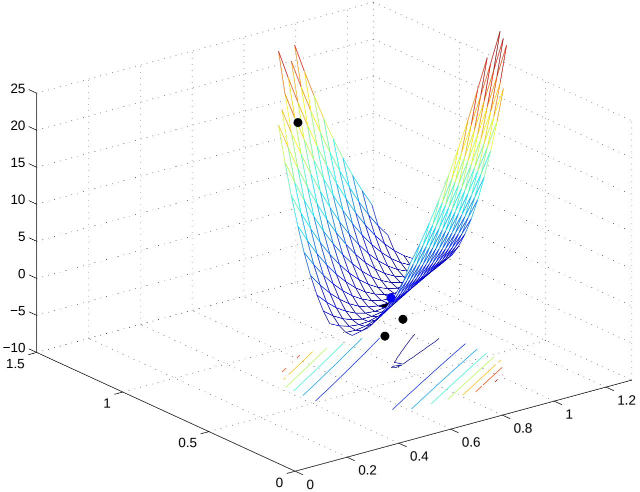
On the ever famous banana function. . .

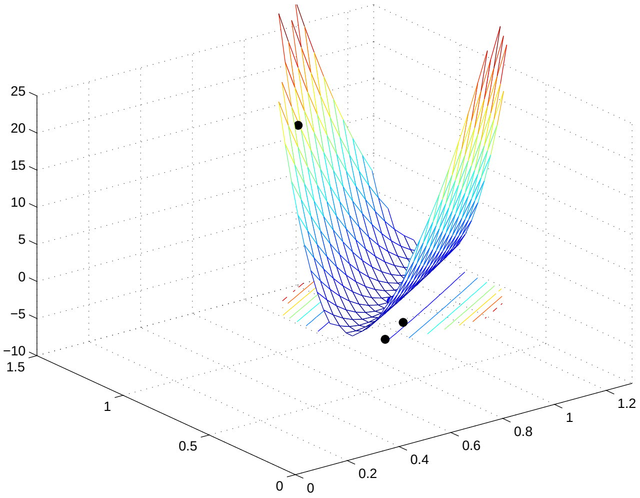


On the ever famous banana function...

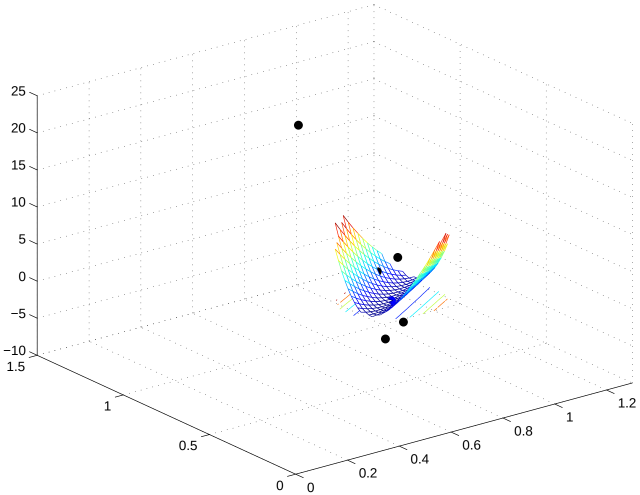


On the ever famous banana function. . .

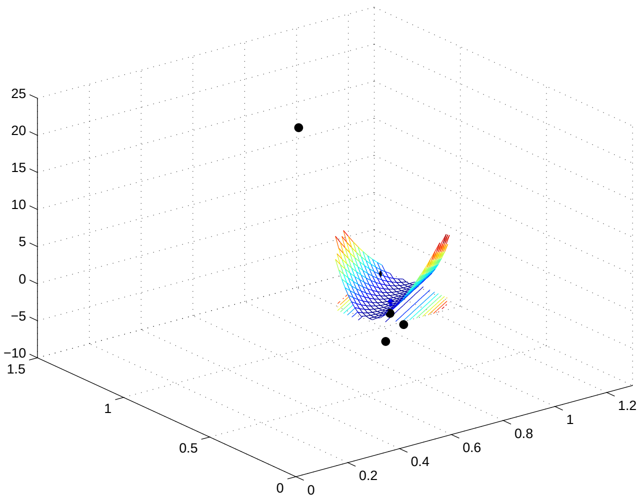



25
20
15
10
5
0
−5
−10
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2

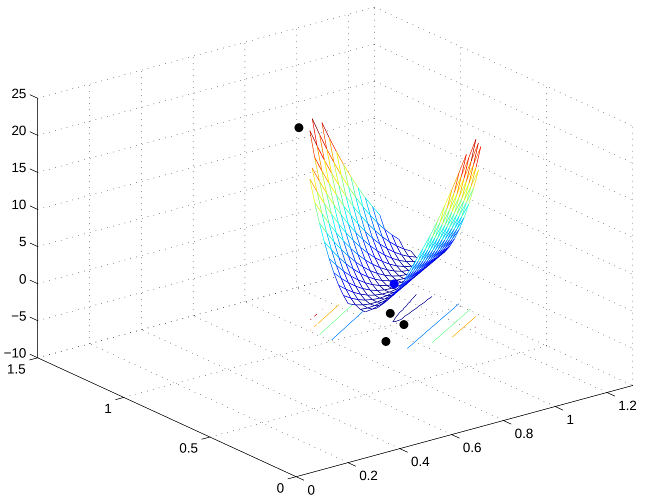
On the ever famous banana function...



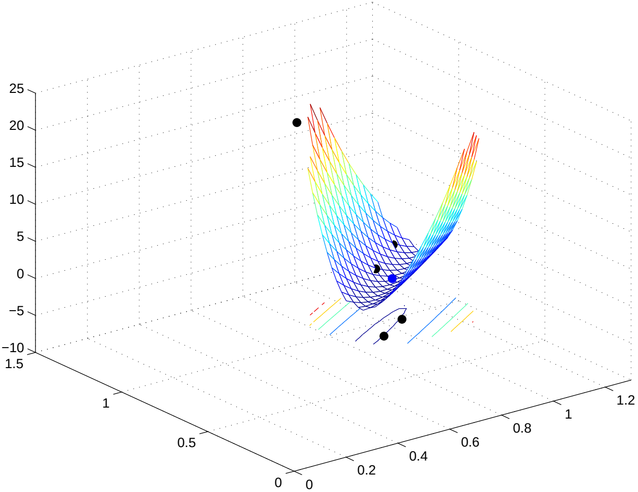
On the ever famous banana function. . .



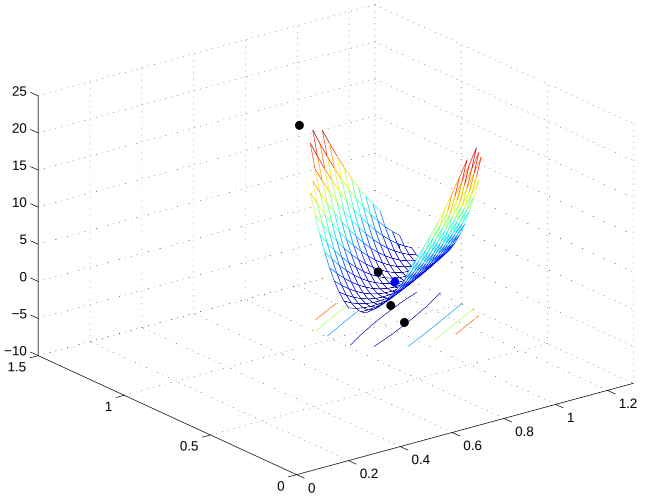
On the ever famous banana function...



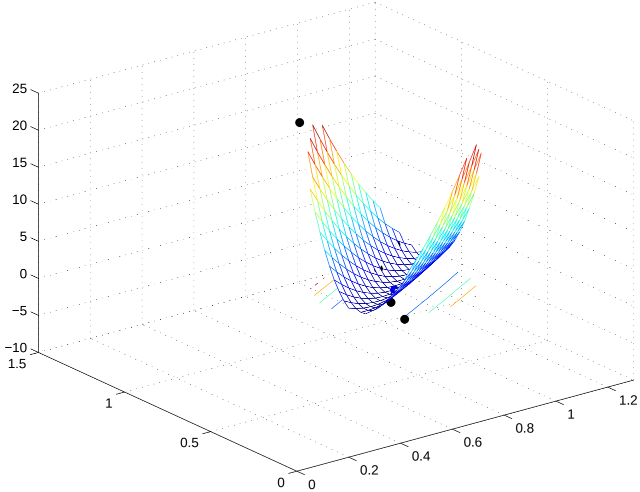
On the ever famous banana function. . .



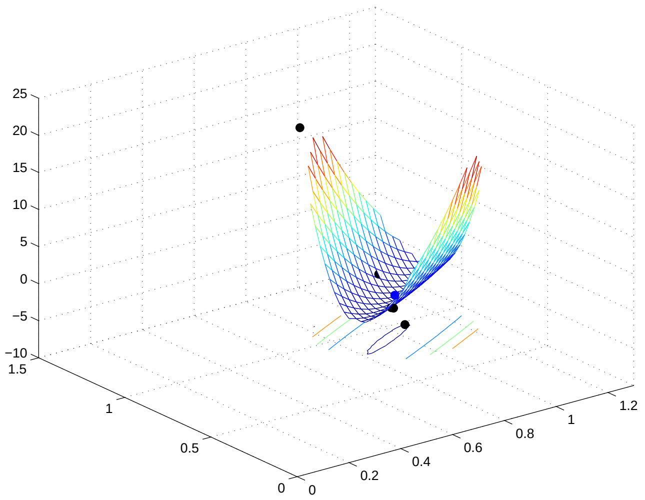
On the ever famous banana function. . .



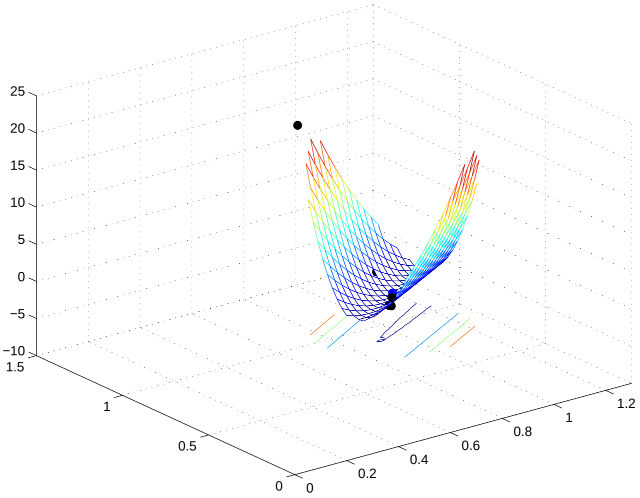
On the ever famous banana function. . .



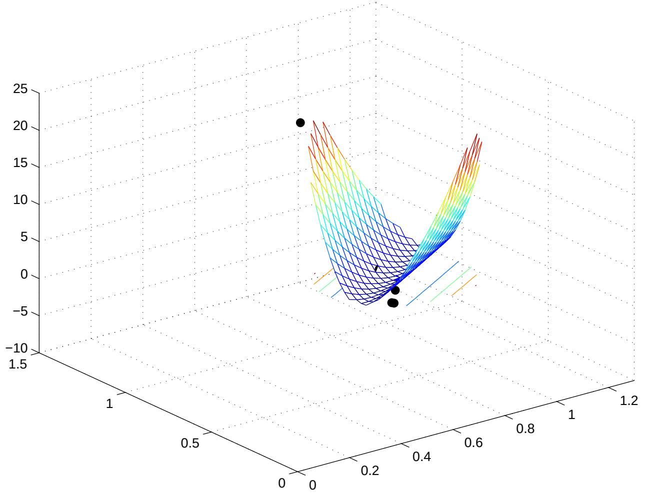
On the ever famous banana function. . .



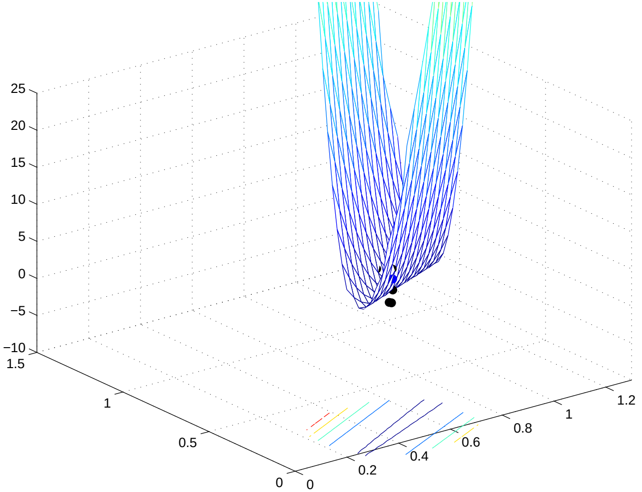
On the ever famous banana function. . .



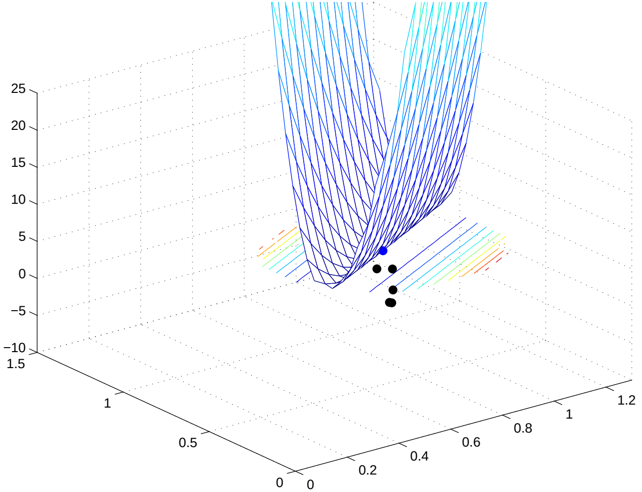
On the ever famous banana function...



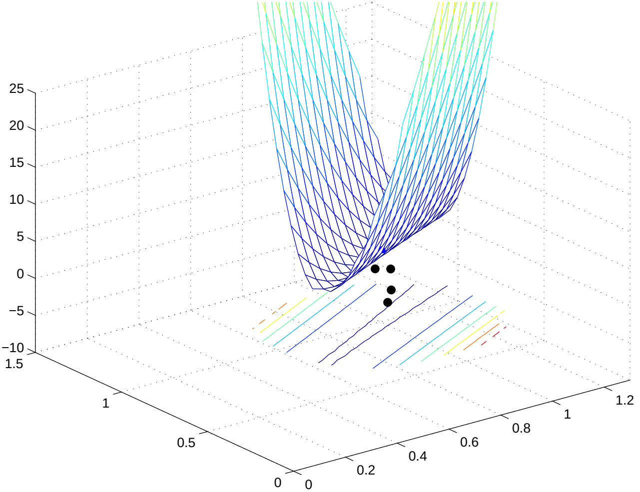
On the ever famous banana function. . .



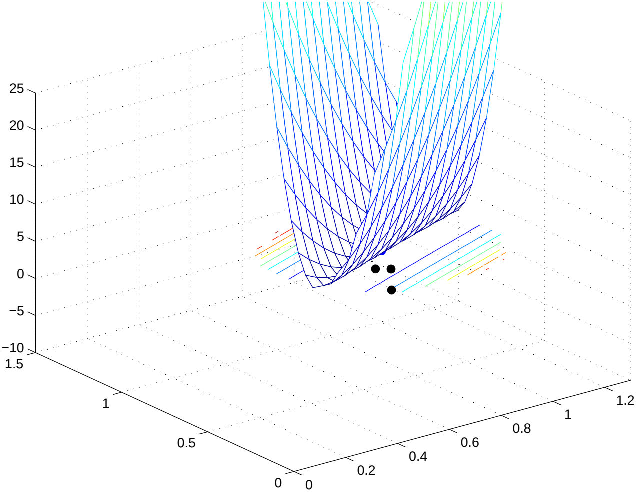
On the ever famous banana function. . .



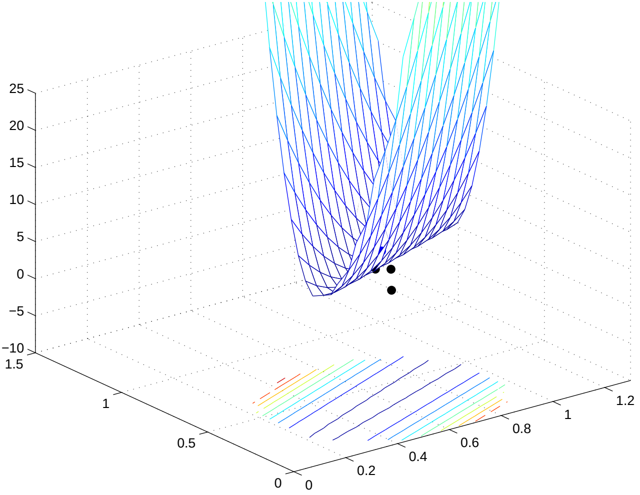
On the ever famous banana function. . .



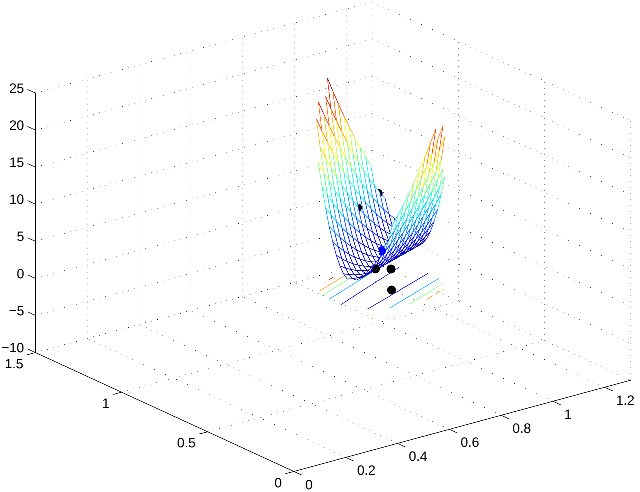
On the ever famous banana function. . .



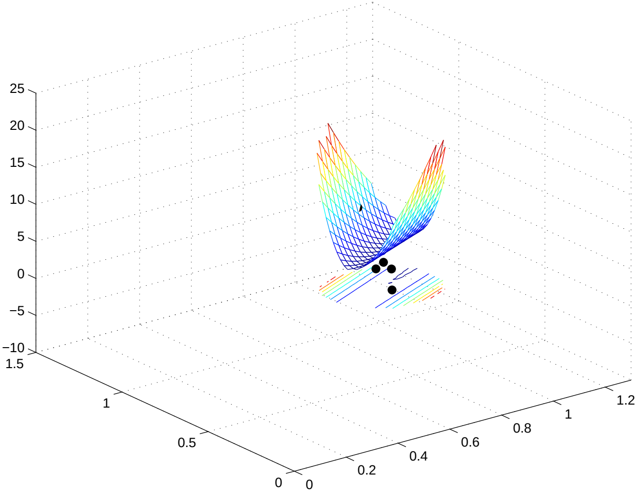
On the ever famous banana function. . .



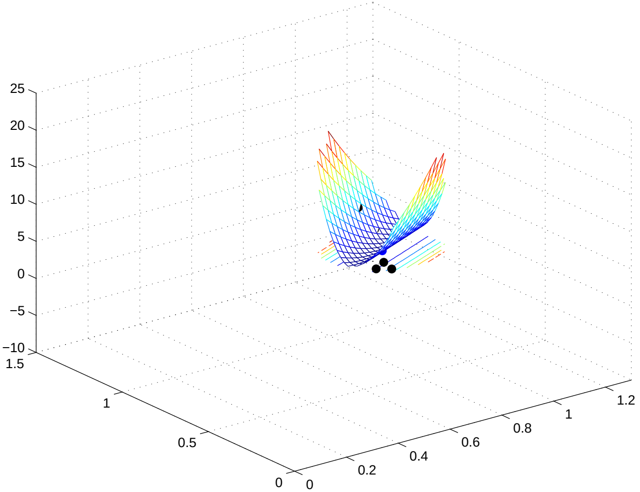
On the ever famous banana function. . .



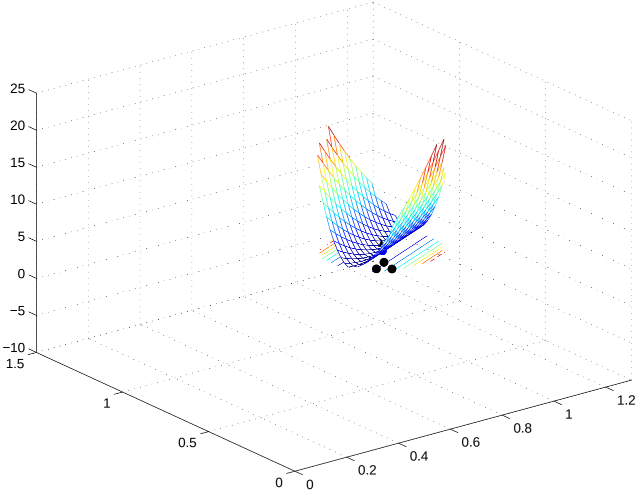
On the ever famous banana function...

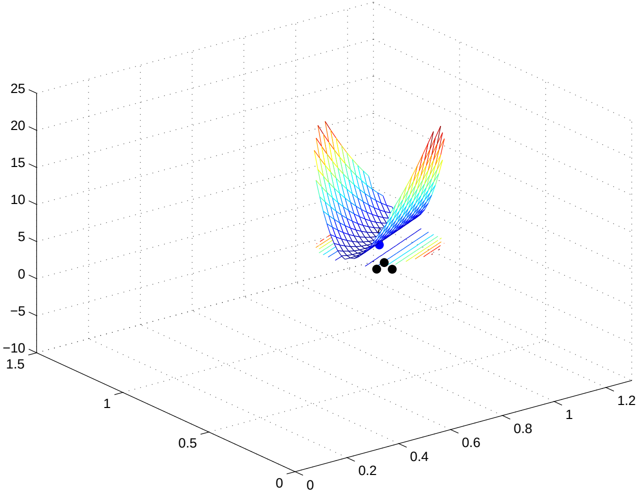


On the ever famous banana function. . .

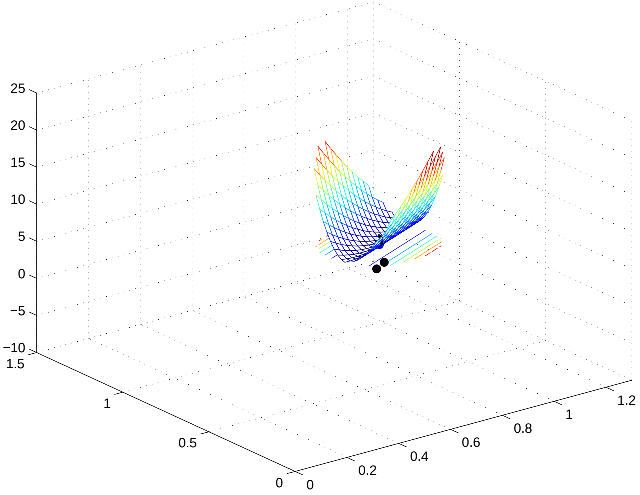


On the ever famous banana function. . .




25
20
15
10
5
0
−5
−10
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
1.2

On the ever famous banana function. . .



Conclusions

- necessity of —tbluegeometry management
- interesting auto-correction property
- a new efficient algorithm
- a Matlab code soon available
- . . .

Many thanks for your attention!