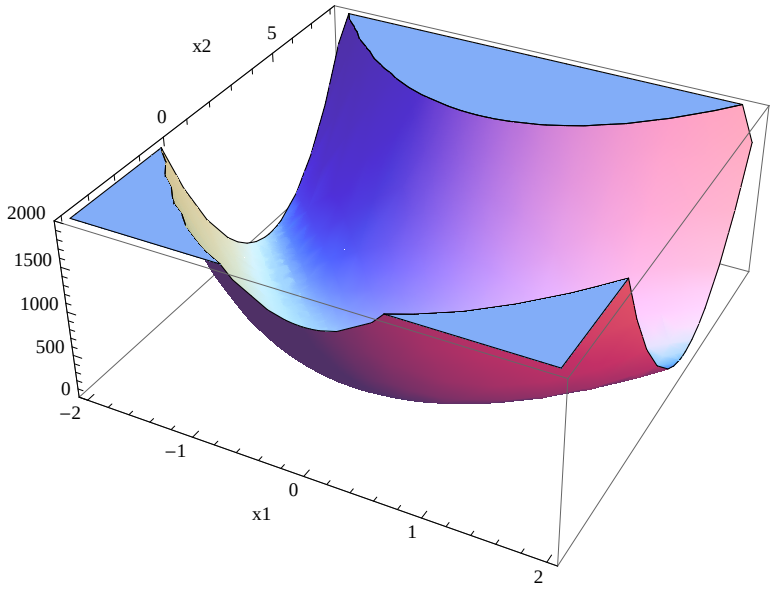
Search

Scott Greenwald
May 2, 2011
MAS864

Rosenbrock

- **(a) Plot**

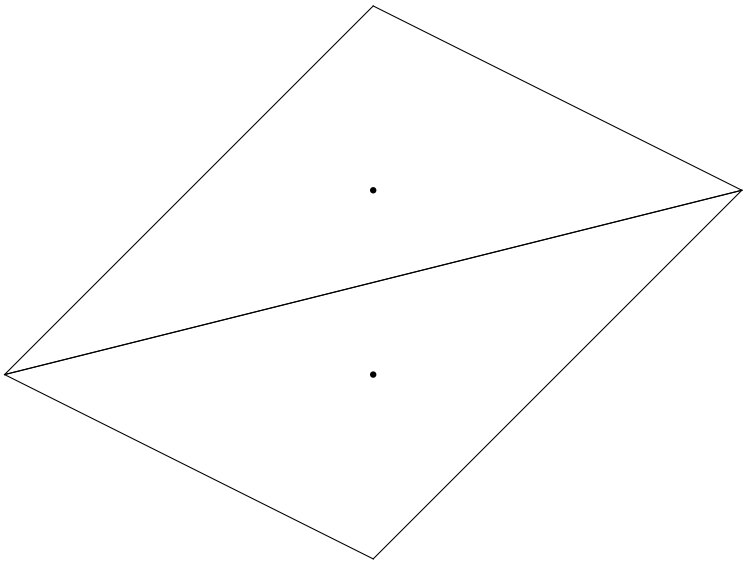
```
Clear[rosenbrock]
rosenbrock[{x1_, x2_}] := (1 - x1)^2 + 100 (x2 - x1^2)^2;
Plot3D[rosenbrock[{x1, x2}], {x1, -2, 2}, {x2, -4, 8},
 Mesh → False, PlotRange → {0, 2000}, AxesLabel → {"x1", "x2"}]
```



■ (b) downhill simplex

```
init := (setAttr[p1, coord, {1, 8} // N];
   setAttr[p1, value, rosenbrock[coord[p1]] // N];
   setAttr[p2, coord, {-1, 6} // N];
   setAttr[p2, value, rosenbrock[coord[p2]] // N];
   setAttr[p3, coord, {3, 7} // N];
   setAttr[p3, value, rosenbrock[coord[p3]] // N];
   prims = {};
   addPrims[prims, {p1, p2, p3}];
   cnt = 0; Print["initialized"]);
init;
simplexStep[rosenbrock, {p1, p2, p3}];
Graphics[prims]
```

```
initialized
```



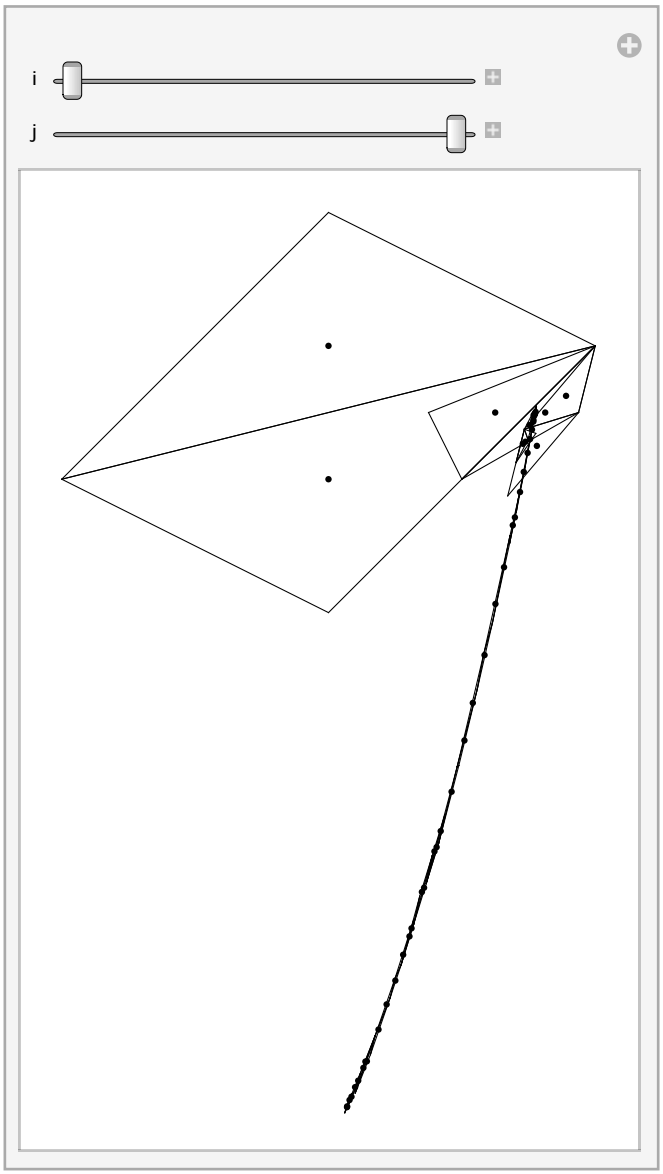
```
AbsoluteTiming[Table[simplexStep[rosenbrock, {p1, p2, p3}(* ,True *)], {16}]];
targetThreshold = 10^-7;
cnt = 0;
itr = 0;
itrMax = 500;
threshold = 1;
init;
While[threshold > targetThreshold && itr < itrMax,
  itr += 1; threshold = Abs[Subtract @@ (simplexStep[rosenbrock, {p1, p2, p3}])]];
{threshold, itr, cnt}
primP = prims;
```

```
initialized
```

$$\{5.56878 \times 10^{-8}, 81, 165\}$$

How does exponent of threshold relate to number of iterations?

```
DynamicModule[{prims = primP},
 Manipulate[Graphics[Take[prims, {i, Min[i + (4 j - 1), Length[prims]]}]],
  {{i, Length[prims] - 11}, 1, Length[prims] - 11, 4}, {{j, 3}, 1, 50, 1}]]
```



Source

```
Clear[setAttr]
setAttr[mod_, attr_, val_] :=
    (Unprotect[mod];
     attr[mod] ^= val;
     Protect[mod]);
Attributes[simplexStep] = HoldAll;
simplexStep[function_, ps_,verbose_:False] :=
    Module[ {countFn, maxVal, delt, deltB, pM,dp,vPrint},
        vPrint[msg_] :=
            If[ verbose,
                Print[msg]
            ];
        delt[p_] :=
            coord[p] - coord[pM];
        deltB[p_] :=
            coord[p] - coord[pB];
        shrinkToBest[p_] :=
            setAttr[p, coord, coord[p] - deltB[p]/2];
        countFn[arg_] :=
            (cnt += 1;
             function[arg]);
        (* wrap function with a counter *)
        maxVal = Max[value /@ ps];
        pX = First[Select[ps, value[#] == maxVal &]];
        setAttr[pM, coord, Mean[coord /@ Complement[ps,{pX}]]];
        vPrint[coord[pM]];
        vPrint["Reflecting"];
        dp = {coord[pM],coord[pX],delt[pX]};
        setAttr[pX, coord, coord[pM] - delt[pX]];
        setAttr[pX, value, countFn[coord[pX]]];
        (* addPrims[prims,ps]; *)
        Which[
         (* CASE: pX is now the best *)
         value[pX] == Min[value /@ ps],
         (* grow? *)
         vPrint["checking for grow"];
         If[ countFn[coord[pX] + delt[pX]] < value[pX],
             vPrint["grow"];
             setAttr[pX, coord, coord[pX] + delt[pX]],
             vPrint["no grow"]
         ],
         (* CASE: pX is still the worst *)
         value[pX] == Max[value /@ ps],
         (* shrink? *)
         vPrint["still the worst, going to try shrinking"];
         dp = {coord[pM],coord[pX],delt[pX]};
         setAttr[pX, coord, coord[pX] - delt[pX]/2];
         setAttr[pX, value, countFn[coord[pX]]];
         If[
          (* CASE: pX is still the worst *)
               value[pX] == Max[value /@ ps],
             vPrint["still still worst, going to try reflecting back across"];
             setAttr[pX, coord, coord[pX] - 2 delt[pX]];
             setAttr[pX, value, countFn[coord[pX]]];
             If[
              (* CASE: pX is still the worst *)
                   value[pX] == Max[value /@ ps],
                vPrint["still still still worst, going to shrink towards best"];
                bestVal = Min[value /@ ps];
                pB = First[Select[ps, value[#] == bestVal &]];
                shrinkToBest /@ ps
             ]
         ]];
        addPrims[prims,ps];
        {Min[value /@ ps], Max[value /@ ps]}
    ];
```

Junk