Minimierung der Rosebrock - Funktion mittels des Trust-Region-Verfahrens

(Debug) In[1]:=

```
(* Die Parameter *)
γ1 = 0.5;
γ2 = 2;
η1 = 0.1;
η2 = 0.9;
β = 2;
α = 0.5;
ϵ = 10^(-3);

(* Cauchy-Schritt(sc = tau * sl) und Newton-Schritt sn *)
Tau[G_, H_, Δ_] := If[G.H.G ≤ 0, 1, Min[(Norm[G]^3) / (Δ * G.H.G) , 1]];
                   ⌊wenn             ⌊klein⋯ ⌊Norm
CauchySchritt[G_, H_, Δ_] := -Tau[G, H, Δ] * ((Δ * G) / (Norm[G]));
                                                          ⌊Norm
NewtonSchritt[G_, H_] := -Inverse[H].G;
                          ⌊inverse Matrix
(* Die Cauchy-Abstiegsbedingung entscheidet, ob Newton- oder Cauchy-
 Schritt gewählt werden muss. Zusätzlich werden diese Schritte auch
  jeweils mit C für Cauchy- und mit N für Newton-Schritt benennt. *)
              ⌊Konstante                ⌊numerischer Wert
CABedingung[G_, H_, N_, C_, Δ_] := Module[{Tmp, Tmp2},
                                   ⌊Modul
   Tmp = If[(-G.N - 0.5 * N.H.N ≥ α * (-G.C - 0.5 * C.H.C)),
         ⌊wenn   ⌊numerisc⋯ ⌊n⋯ ⌊numerischer⋯ ⌊Konstante ⌊Ko⋯⌊Konstante
     Tmp2 = "N"; N,
            ⌊n⋯ ⌊numerischer Wert
     Tmp2 = "C"; C];
            ⌊K⋯ ⌊Konstante
   Tmp = If[Norm[Tmp] ≤ β Δ, Tmp, C];
         ⌊⋯ ⌊Norm                ⌊Konstante
   {Tmp, Tmp2}
  ];
(* Rho dient für die Bewertung der Qualität des berechneten Schrittes *)
Rho[x_, s_, G_, H_, f_] := (f[x] - f[x + s]) / (-G.s - 0.5 * s.H.s);
(* Je nach Rho ρ wird entweder der Radius Δ verkleinert,
beibehalten oder vergrößert *)
UpdateTrustRegion[Δ_, rho_] := Module[{Δout},
                               ⌊Modul
   If[rho ≤ η1, Δout = γ1 Δ];
   ⌊wenn
   If[η1 < rho ≤ η2, Δout = 0.75 Δ];
   ⌊wenn
```

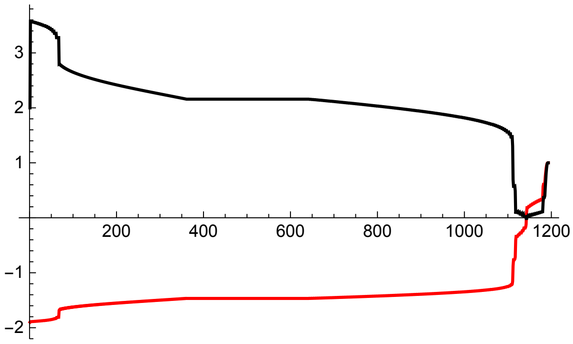
```
    wenn
    If[η2 < rho, Δout = γ2 Δ];
    wenn
    Δout];
(* Ist Rho ρ mindestens größer als η1, so wird der Schritt akzeptiert *)
AkzeptiereSchritt[x_, s_, rho_] := If[rho > η1, x + s, x];
                                   wenn

(* An das Verfahren wird eine Testfunktion f,
ein Startwert x0 und höchstanzahl an Iterationen n
 übergeben. Zuerst werden die erste und zweite Ableitungen,
bzw. G und H, der Tesfunktion f an der Stelle x berechnet. Diese
  werden dann später an oben aufgeführen Definitionen übergeben und
  an der Stelle x ausgewärtet. Zunächst werden Cauchy- und Newton-
 Schritt berechnet und mit Hilfe von Cauchy-Abstiegsbedingung geprüft,
welche der beiden gewählt werden muss. Danach wird
 Rho berechnet und mit AkzeptiereSchritt geprüft,
ob die neue x^(k+1) die Anforderung genügt. Zuletzt wird mit UpdateTrustRegion
  die am Anfang als Δ=2 festgesetzte Radius aktualisiert. *)
TrustRegionVerfahren[f_, x0_, n_] := Module[{G, H, x = x0, X = {x0},
                                     Modul
     s, Δ = 2, rho, k, x01, x02, Gnow, Hnow, Taunow, Cnow, Nnow, Kind},
    G[{x1_, x2_}] := Evaluate[{D[f[{x01, x02}], x01], D[f[{x01, x02}], x02]} /.
                     werte aus  leite ab                leite ab
      {x01 → x1, x02 → x2}];
    H[{x1_, x2_}] := Evaluate[{D[G[{x01, x02}], x01], D[G[{x01, x02}], x02]} /.
                     werte aus  leite ab                leite ab
      {x01 → x1, x02 → x2}];
    Catch[
    fange ab
     Do[
     iteriere
      If[Norm[G[x]] < ϵ, Throw[Print[k]]];
      … Norm              werfe  gebe aus

      Gnow = G[x];
      Hnow = H[x];
      Taunow = Tau[x, Gnow, Hnow, Δ];
      Cnow = CauchySchritt[Gnow, Hnow, Δ];
      Nnow = NewtonSchritt[Gnow, Hnow];
      {s, Kind} = CABedingung[Gnow, Hnow, Nnow, Cnow, Δ];
      rho = Rho[x, s, Gnow, Hnow, f];
      x = AkzeptiereSchritt[x, s, rho];
      Δ = UpdateTrustRegion[Δ, rho];
      X = Append[X, x],
          hänge an
      {k, 0, n}];
    ];
```

```
f[{x1_, x2_}] := 100 (x2 - x1^2)^2 + (1 - x1)^2;
X = TrustRegionVerfahren[f, x0 = {-1.9, 2.}, 10 000] // N
```

nu

```
(* Die Iterationsverlauf der x1 und x2 *)
ListLinePlot[Transpose[X], PlotStyle → {Red, Black}, PlotRange → All]
```

Liniengrafik einer ·· transponiere Grafikstil rot schwarz Definitions- un··· alle




```
ContourPlot[f[{x1, x2}], {x1, -2.5, 2}, {x2, -1., 4},
```

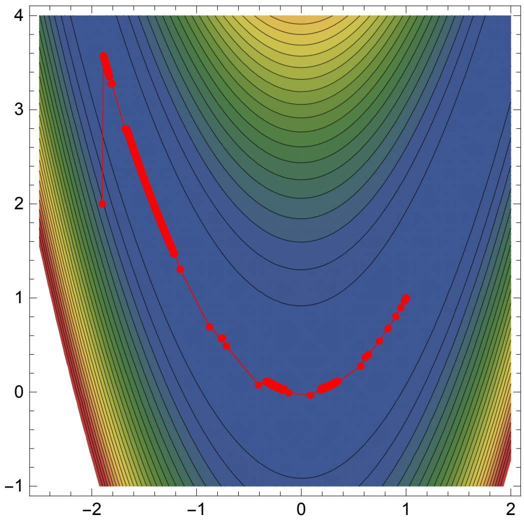
Konturgraphik

```
 Epilog → {Red, PointSize[0.015], Map[Point, X], Line[X]},
```

Epilog rot Punktgröße w··· Punkt Linie

```
 ColorFunction → "DarkRainbow",  Contours → 25]
```

Farbfunktion Konturen



```
T1 = Plot3D[f[{x1, x2}], {x1, -2.5, 2}, {x2, -1, 4}, ColorFunction → "DarkRainbow",
      grafische Darstellung 3D                                 Farbfunktion
    Boxed → False, AxesLabel → Automatic, Mesh → None];
    einger··· falsch  Achsenbeschri··· automatisch  Netz  keine
T2 = Graphics3D[{PointSize[0.02], Red, Point[Table[
      3D-Graphik   Punktgröße          rot  Punkt  Tabelle
       {X[[i, 1]], X[[i, 2]], f[{X[[i, 1]], X[[i, 2]]}]}, {i, 1, Length[X]}]]}];
                                                          Länge
T3 = Graphics3D[{Red, Line[Table[{X[[i, 1]], X[[i, 2]],
      3D-Graphik   rot  Linie  Tabelle
        f[{X[[i, 1]], X[[i, 2]]}]}, {i, 1, Length[X]}]]}];
                                       Länge
Show[
zeige an
 T1,
 T2,
 T3]
```