

Pathfinding

Gesucht ist ein optimaler Weg durch eine Gebirgslandschaft,

Länge, Steigungen und Hindernisse sind zu berücksichtigen.

Illustration

```
In [1]: using Plots; pyplot();
        using Optim;
```

definiere Höhenprofil, möglichst wellig

```
In [2]: h(x,y)=cos(sqrt(x^2+y^2));
```

bereiten Grafik vor - ein 3d-Bilde des "Gebirges"

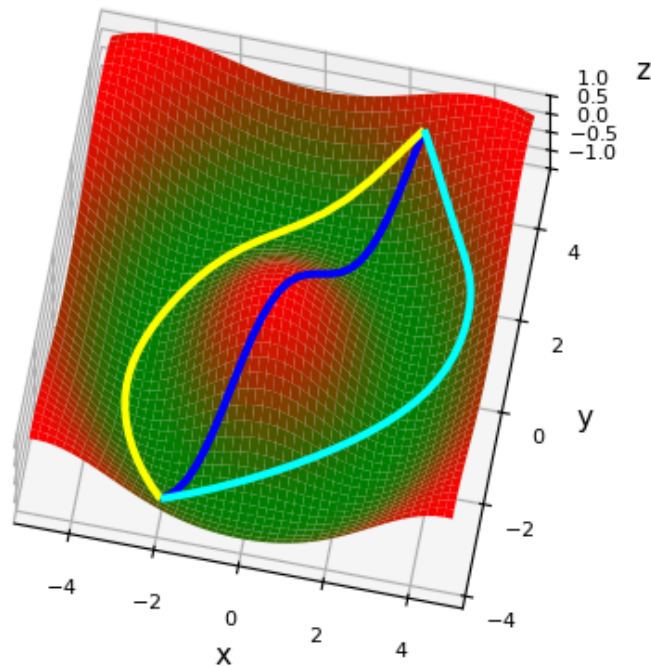
```
In [3]: xgrd=-5:0.2:5;
        ygrd=-4:0.2:5;
        cg=cgrad([:green,:red])
        surface(xgrd,ygrd,h,color=cg,colorbar=false);
```

wählen (willkürlich) drei Pfade und zeichnen sie ein

```
In [4]: A=[-2.0; -3.5]; B=[2.5;+4.5];
        xtr=range(A[1],stop=B[1],length=30);
        ytr=range(A[2],stop=B[2],length=30);
        ztr=map(h,xtr,ytr);
        xtr1=range(A[1],stop=B[1],length=30).+
              0.2*(ytr.-ytr[1]).*(ytr.-ytr[end]);
        ztr1=map(h,xtr1,ytr);
        xtr2=range(A[1],stop=B[1],length=30).-
              0.22*(ytr.-ytr[1]).*(ytr.-ytr[end]);
        ztr2=map(h,xtr1,ytr);
```

```
In [5]: surface(xgrd,ygrd,h,color=cg,colorbar=false);
        plot!(xtr,ytr,ztr,linewidth=3,color=:blue,label="")
        plot!(xtr1,ytr,ztr1,linewidth=3,color=:yellow,label="")
        plot!(xtr2,ytr,ztr2,linewidth=3,color=:cyan,label="")
        plot!(camera=[11,77],xlabel="x",ylabel="y",zlabel="z")
```

Out[5]:



offenbar gibt es einen direkten, aber bergigen, und zwei vergleichsweise flache Alternativrouten ... und noch viele weitere Variationen

Weglänge

```
In [6]: fl(x,y,z)=sum(sqrt.(diff(x).^2+diff(y).^2+diff(z).^2));
        fl8(x,y,z)=sum((diff(x).^2+diff(y).^2+diff(z).^2).^8)^0.125;
```

fl dient als Zielfunktion, es drückt die euklidische Länge des durch die Knotenkordinaten x,y,z gegebenen Polygonzuges aus

fl8 ist ein Surrogat für die maximale Länge eines Segments, also den Abstand zwischen zwei Knoten

```
In [7]: L=fl(xtr,ytr,ztr)
        L1=fl(xtr1,ytr,ztr1)
        L2=fl(xtr2,ytr,ztr1)
        [L,L1,L2]
```

```
Out[7]: 3-element Vector{Float64}:
         11.123092053437384
         11.766106248843908
         12.33410793277631
```

offenbar sind beide (willkürlich gewählten) Umwege tatsächlich länger

und natürlich sind sie länger als der direkte gerade Weg (Luftlinie)

```
In [8]: Ld=sqrt((xtr[end]-xtr[1])^2+(ytr[end]-ytr[1])^2+(ztr[end]-ztr[1])^2)
```

```
Out[8]: 9.238820141912802
```

die (fast) ∞ -Normen der Schrittweiten hätten wir gern in der Nähe der Weglänge geteilt durch die Zahl der Schritte - Hauptsache, die Knoten "verklumpen" nicht

```
In [9]: L8=f18(xtr,ytr,ztr)
        L18=f18(xtr1,ytr,ztr1)
        L28=f18(xtr2,ytr,ztr1)
        [L8,L18,L28]./length(xtr)
```

```
Out[9]: 3-element Vector{Float64}:
         0.00867862735160449
         0.02281454081184193
         0.017416273200428083
```

Minimierung der Gesamtlänge mit den Knotenkoordinaten als freien Variablen führt dazu, dass die Knoten sich häufen, wobei die Haufen auf der Vogelfluglinie von A nach B liegen

deshalb bestrafen wir große Segmentlängen und führen als Zielfunktion die Länge plus Strafterm ein. Der Penalty-Term hat die Form Faktor mal fl8

es erweist sich, dass für das gegebene Beispiel ein Faktor von mehr als 0.084 nötig ist, um die Knoten etwa gleichmäßig zu verteilen

```
In [10]: function ffl(tr,mu=0.1)
          n=Int(length(tr)/2);
          xtr=tr[1:n];
          ytr=tr[n+1:2*n];
          xtr=[A[1];xtr;B[1]];
          ytr=[A[2];ytr;B[2]];
          ztr=h.(xtr,ytr);
          return fl(xtr,ytr,ztr)+mu*f18(xtr,ytr,ztr); # 0.084 - Limit
        end;
# ffl ist die Längenbasierte Zielfunktion mit penalty
```

als Ausgangspfad nehmen wir die Trajektorie xtr,ytr, ohne die Endpunkte, in einen Vektor zusammengefasst

```
In [11]: tr0=[xtr[2:end-1];ytr[2:end-1]];
```

```
In [12]: ffl(tr0)
```

```
Out[12]: 11.149127935492197
```

die Zielfunktion ist etwas länger als die tatsächliche Länge, den Unterschied macht der Penaltyterm aus

das BFGS-Verfahren liefert schnell ein Minimum

```
In [13]: soll=optimize(fft, tr0, BFGS());
          trl=soll.minimizer;
          println("gelöst in $(soll.iterations) BFGS-Iterationsschritten")
```

gelöst in 321 BFGS-Iterationsschritten

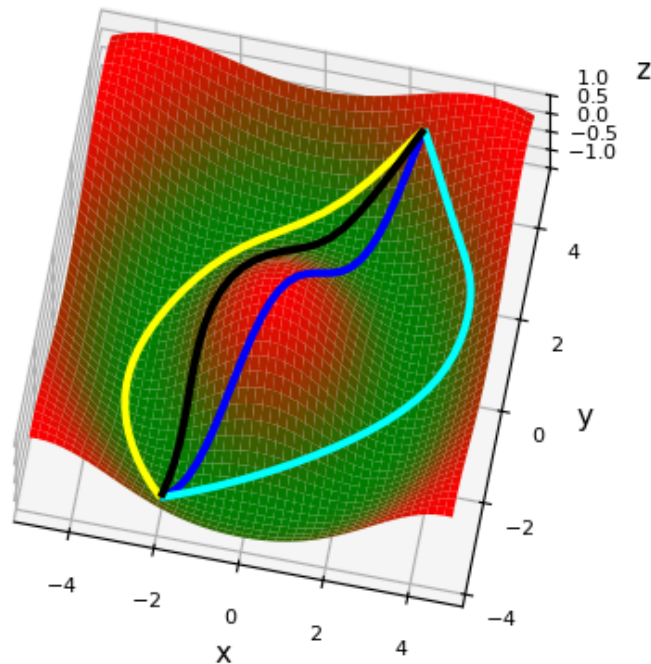
zwecks Auswertung ist anschließend die Lösung wieder in Komponenten aufzuteilen, um die Endpunkte zu ergänzen und die Höhen zu ermitteln

die optimale Länge ist kleiner als die der geratenen Ausgangskurve und länger als die Luftlinie

```
In [14]: trl2=reshape(trl,Int(length(trl)/2),2);
xtrl=[A[1];trl2[:,1];B[1]];
ytrl=[A[2];trl2[:,2];B[2]];
ztrl=map(h,xtrl,ytrl);
println("minimale Länge $(f1(xtrl,ytrl,ztrl))")
plot!(xtrl,ytrl,ztrl,linewidth=3,color=:black,label="")
```

minimale Länge 10.655731627968093

Out[14]:



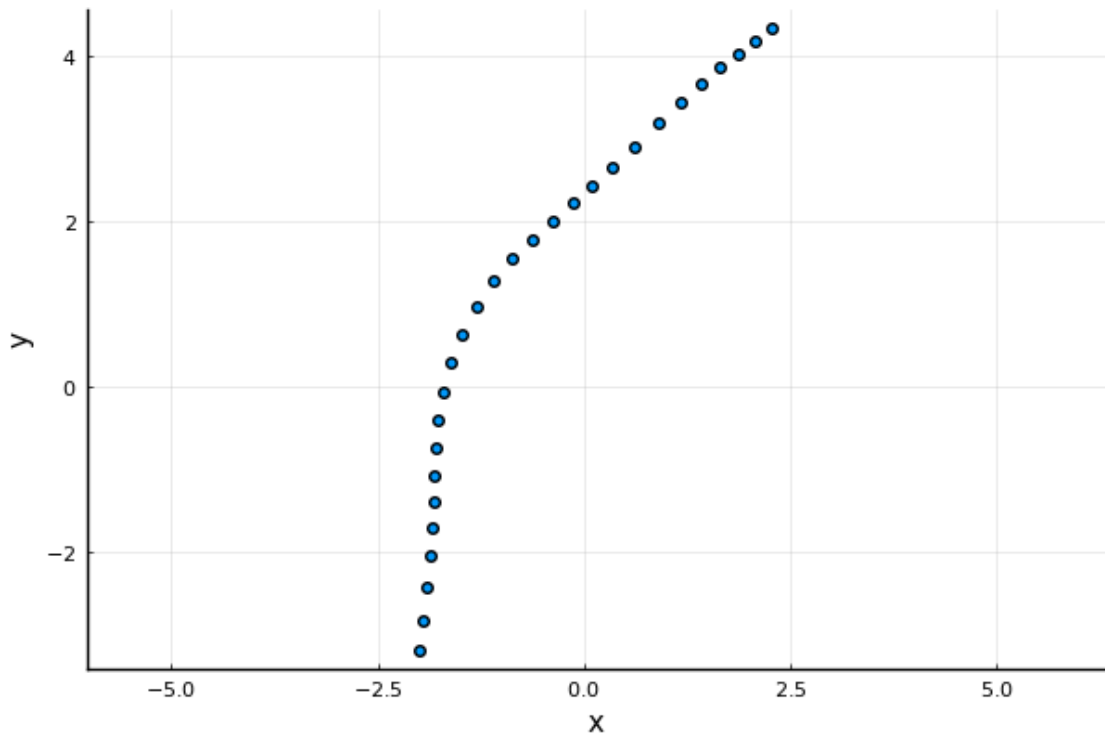
eine Vergleichsrechnung mit Faktor 0.084 demonstriert das Verklumpen

```
In [15]: solla=optimize(x->ffl(x,0.084), tr0, BFGS());
trla=solla.minimizer;
println("gelöst in $(solla.iterations) BFGS-Iterationsschritten")
trla2=reshape(trla,Int(length(trla)/2),2);
xtrla=[A[1];trla2[:,1];B[1]];
ytrla=[A[2];trla2[:,2];B[2]];
ztrla=map(h,xtrla,ytrla);
println("minimale Länge $(f1(xtrla,ytrla,ztrla))")
scatter(trla2[:,1],trla2[:,2],legend=false,aspect_ratio=1.0,
        xlabel="x",ylabel="y")
```

gelöst in 303 BFGS-Iterationsschritten

minimale Länge 10.655574526854418

Out[15]:



```

sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored

```

der lückenhafte Weg kürzt einfach über die Täler ab oder tunnelt die Hügel!

um dies zu verhindern kann man statt Penalty auch eine Nebenbedingung auf die Schrittweite auferlegen

als weiteren Test starten wir von einem alternativen Ausgangspfad, der rechten Trajektorie xtr2, ytr2

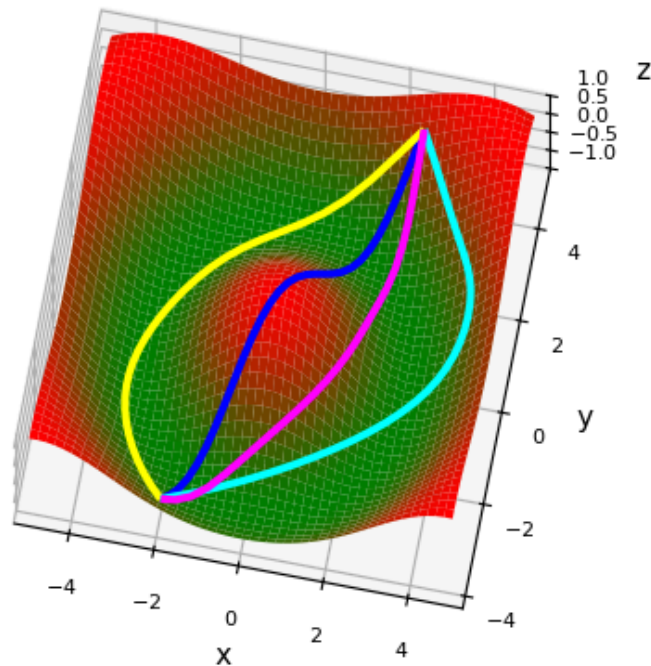
```

In [16]: tr2=[xtr2[2:end-1];ytr[2:end-1]];
sollb=optimize(ffl, tr2, BFGS());
trlb=sollb.minimizer;
println("gelöst in $(solla.iterations) BFGS-Iterationsschritten")
trlb2=reshape(trlb,Int(length(trlb)/2),2);
xtrlb=[A[1];trlb2[:,1];B[1]];
ytrlb=[A[2];trlb2[:,2];B[2]];
ztrlb=map(h,xtrlb,ytrlb);
println("minimale Länge $(fl(xtrlb,ytrlb,ztrlb))")
surface(xgrd,ygrd,h,color=cg,colorbar=false);
plot!(xtr,ytr,ztr,linewidth=3,color=:blue,label="")
plot!(xtr1,ytr,ztr1,linewidth=3,color=:yellow,label="")
plot!(xtr2,ytr,ztr2,linewidth=3,color=:cyan,label="")
plot!(camera=[11,77],xlabel="x",ylabel="y",zlabel="z")
plot!(xtrlb,ytrlb,ztrlb,linewidth=3,color=:magenta,label="")

```

gelöst in 303 BFGS-Iterationsschritten
minimale Länge 10.691471733925296

Out[16]:



offensichtlich gibt es mehrere Minima, leider auch solche, die den Berg (beliebig oft) umkreisen

ausprobieren!

Gefälle

besonders beschwerliche Wegstücke mit steilem Anstieg oder Gefälle sollen vermieden werden - so dass ein insgesamt längerer, dafür flacherer Weg bevorzugt wird

gegebenfalls sind Anstieg und Gefälle unterschiedlich zu bewerten

hierzu berechnen wir weitere Terme für die Zielfunktion

```
In [17]: function ffs1p(tr,mu=10,nu=1)
    n=Int(length(tr)/2);
    xtr=tr[1:n];
    ytr=tr[n+1:2*n];
    xtr=[A[1];xtr;B[1]];
    ytr=[A[2];ytr;B[2]];
    ztr=h.(xtr,ytr);
    str=sqrt.(diff(xtr).^2.0.+diff(ytr).^2.0);
    str3=sqrt.(diff(xtr).^2.0.+diff(ytr).^2.0.+diff(ztr).^2.0);
    mtr=diff(ztr)./str;
    return fl(xtr,ytr,ztr)+mu*f18(xtr,ytr,ztr)+nu*sum(mtr.^2.0.*str3);
end;
# das slp steht für slope = Anstieg
```

hier ist str der Vektor aller Schrittweiten, mtr der aller Anstiege

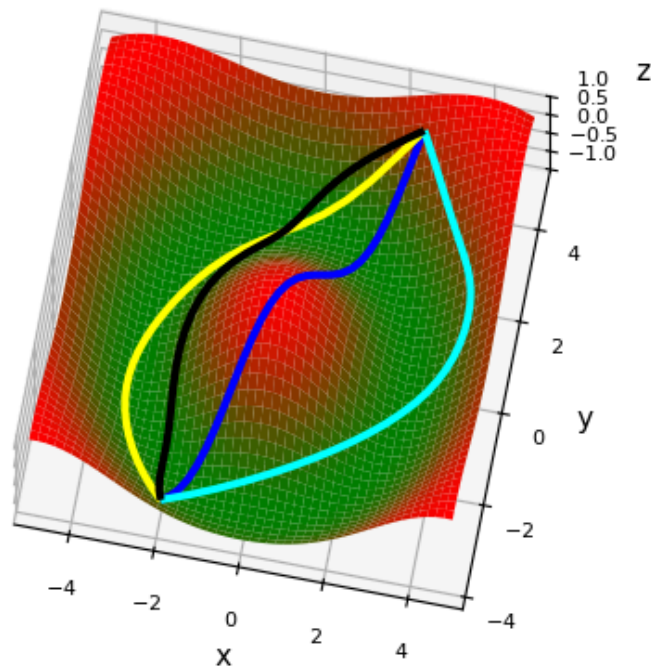
wir bestrafen Steigungen unabhängig vom Vorzeichen mit 4 mal dem Quadrat der Steigung

```
In [18]: solslp=optimize(ffslp, tr0, BFGS());
        trslp=solslp.minimizer;
        println("gelöst in $(solslp.iterations) BFGS-Iterationsschritten")
        trslp2=reshape(trslp,Int(length(trslp)/2),2);
        xtrslp=[A[1];trslp2[:,1];B[1]];
        ytrslp=[A[2];trslp2[:,2];B[2]];
        ztrslp=map(h,xtrslp,ytrslp);
        println("minimale Länge $(f1(xtrslp,ytrslp,ztrslp))")
        println("minimale Kosten $(solslp.minimum)")
```

```
gelöst in 93 BFGS-Iterationsschritten
minimale Länge 10.958070095360085
minimale Kosten 14.218888049365836
```

```
In [19]: surface(xgrd,ygrd,h,color=:cg,colorbar=false);
        plot!(xtr,ytr,ztr,linewidth=3,color=:blue,label="")
        plot!(xtr1,ytr,ztr1,linewidth=3,color=:yellow,label="")
        plot!(xtr2,ytr,ztr2,linewidth=3,color=:cyan,label="")
        plot!(camera=[11,77],xlabel="x",ylabel="y",zlabel="z")
        plot!(xtrslp,ytrslp,ztrslp,linewidth=3,color=:black,label="")
```

Out[19]:



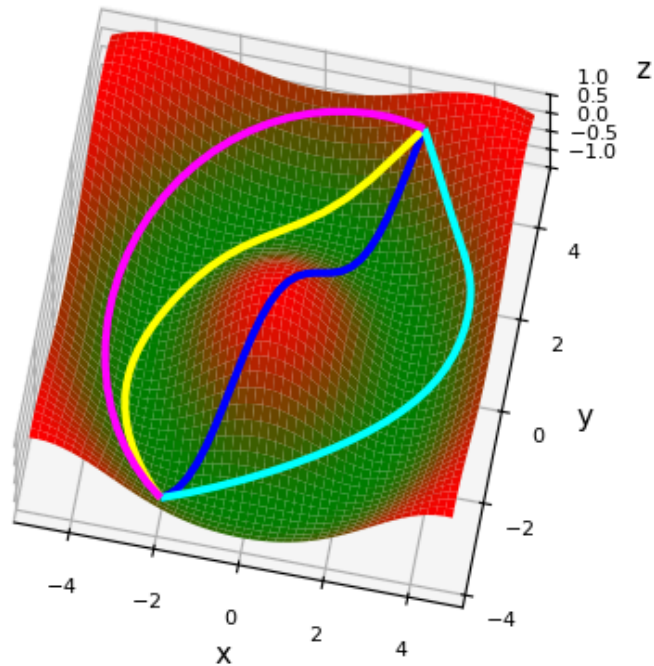
bei gleichem Startpfad und Straffaktor 10 für Gefälle geht der gefundene Weg in einem sehr glatten Bogen um den Berg

```
In [20]: solslpa=optimize(x->ffslp(x,10,10), tr0, BFGS());
        trslpa=solslpa.minimizer;
        println("gelöst in $(solslpa.iterations) BFGS-Iterationsschritten")
        trslpa2=reshape(trslpa,Int(length(trslpa)/2),2);
        xtrslpa=[A[1];trslpa2[:,1];B[1]];
        ytrslpa=[A[2];trslpa2[:,2];B[2]];
        ztrslpa=map(h,xtrslpa,ytrslpa);
        println("minimale Länge $(f1(xtrslpa,ytrslpa,ztrslp))")
        println("minimale Kosten $(solslpa.minimum)")
```

gelöst in 204 BFGS-Iterationsschritten
 minimale Länge 13.886787548746685
 minimale Kosten 18.338200070715615

```
In [21]: surface(xgrd,ygrd,h,color=cg,colorbar=false);
plot!(xtr,ytr,ztr,linewidth=3,color=:blue,label="")
plot!(xtr1,ytr,ztr1,linewidth=3,color=:yellow,label="")
plot!(xtr2,ytr,ztr2,linewidth=3,color=:cyan,label="")
plot!(camera=[11,77],xlabel="x",ylabel="y",zlabel="z")
plot!(xtrslpa,ytrslpa,ztrslpa,linewidth=3,color=:magenta,label="")
```

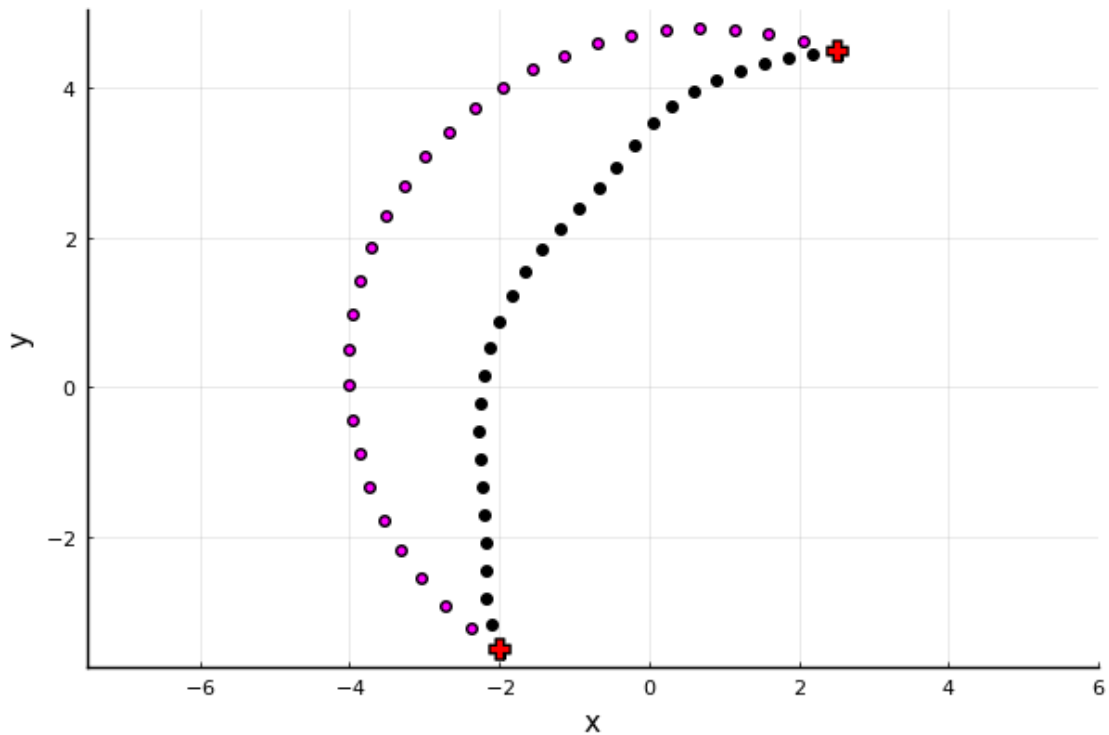
Out[21]:



in jedem Fall ist der Penalty-Term im Vergleich zu zur nackten Längenminimierung zu vergrößern, da die Tendenz zur Clusterbildung der Knoten hier größer ist (hier $\mu=10$)

```
In [22]: scatter(trslp2[:,1],trslp2[:,2],color=:black,
               legend=false,aspect_ratio=1.0,
               xlabel="x",ylabel="y")
scatter!(trslpa2[:,1],trslpa2[:,2],color=:magenta)
scatter!([A[1];B[1]], [A[2];B[2]],color=:red,shape=:cross,markersize=8)
```


Out[22]:



```

sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored

```

Bikeology

im vorigen Abschnitt wurden Anstieg und Abstieg gleich bewertet, steil ist schlecht, egal ob bergauf oder bergab

jetzt wiederholen wir die Untersuchung mit einseitiger Bewertung, mögen nur Anstiege unangenehm sein, bergab rollt es ...

```

In [23]: function ffuph(tr,mu=10,nu=4)
    n=Int(length(tr)/2);
    xtr=tr[1:n];
    ytr=tr[n+1:2*n];
    xtr=[A[1];xtr;B[1]];
    ytr=[A[2];ytr;B[2]];
    ztr=h.(xtr,ytr);
    str=sqrt.(diff(xtr).^2.0+diff(ytr).^2.0);
    str3=sqrt.(diff(xtr).^2.0+diff(ytr).^2.0+diff(ztr).^2.0);
    mtr=max.(0,diff(ztr)./str);
    return fl(xtr,ytr,ztr)+mu*fl8(xtr,ytr,ztr)+nu*sum(mtr.^2.0.*str3);
end;
# das uph steht für uphill

```

```

In [24]: ffuph(trslp),ffslp(trslp)

```

```

Out[24]: (16.605647781759096, 14.218888049365836)

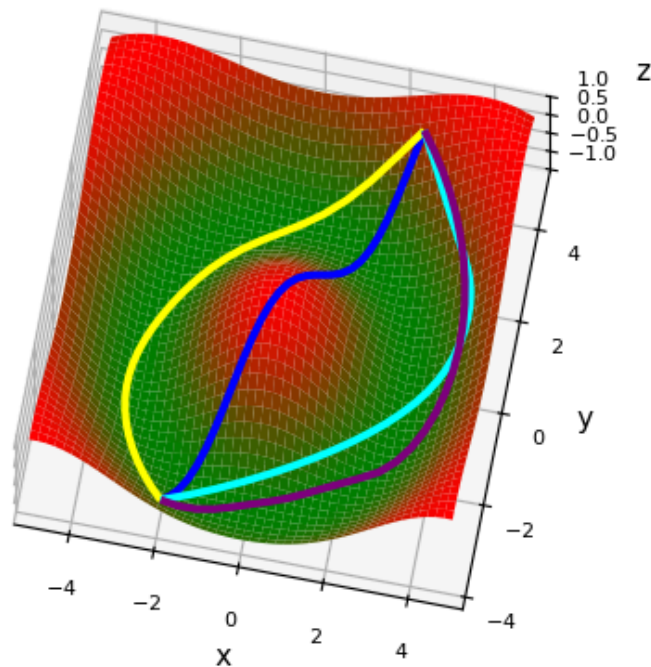
```

```
In [25]: soluph=optimize(x->ffuph(x,10.0,10.0), tr2, BFGS());
truph=soluph.minimizer;
println("gelöst in $(soluph.iterations) BFGS-Iterationsschritten")
truph2=reshape(truph,Int(length(truph)/2),2);
xtruph=[A[1];truph2[:,1];B[1]];
ytruph=[A[2];truph2[:,2];B[2]];
ztruph=map(h,xtruph,ytruph);
println("minimale Länge $(f1(xtruph,ytruph,ztruph))")
println("minimale Kosten $(soluph.minimum)")
```

```
gelöst in 92 BFGS-Iterationsschritten
minimale Länge 12.493822824863893
minimale Kosten 18.24484584702399
```

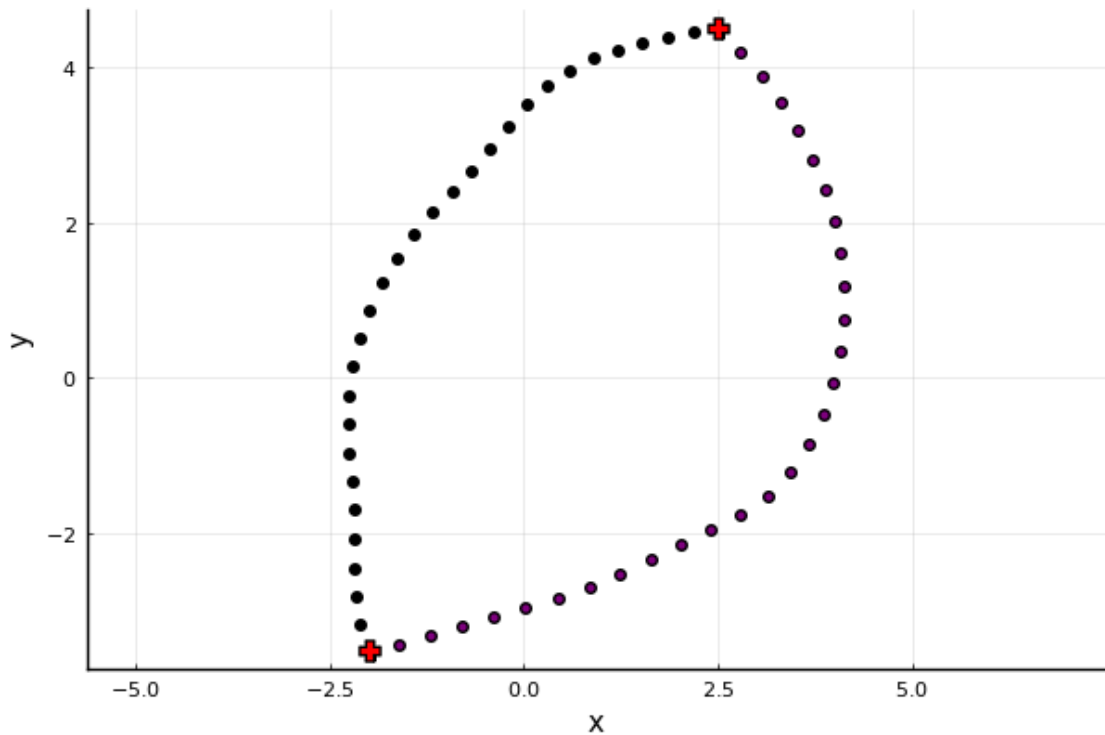
```
In [26]: surface(xgrd,ygrd,h,color=:cg,colorbar=false);
plot!(xtr,ytr,ztr,linewidth=3,color=:blue,label="")
plot!(xtr1,ytr,ztr1,linewidth=3,color=:yellow,label="")
plot!(xtr2,ytr,ztr2,linewidth=3,color=:cyan,label="")
plot!(camera=[11,77],xlabel="x",ylabel="y",zlabel="z")
plot!(xtruph,ytruph,ztruph,linewidth=3,color=:purple,label="")
```

Out[26]:



```
In [27]: scatter(trslp2[:,1],trslp2[:,2],color=:black,
               legend=false,aspect_ratio=1.0,
               xlabel="x",ylabel="y")
scatter!(truph2[:,1],truph2[:,2],color=:purple)
scatter!([A[1];B[1]], [A[2];B[2]],color=:red,shape=:cross,markersize=8)
```

Out[27]:



```

sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored
sys:1: UserWarning: No data for colormapping provided via 'c'. Parameters 'vmin',
'vmax' will be ignored

```

Zusammenfassung

die Suche nach dem günstigsten Weg wurde hier mittels Approximation durch Polygonzüge und die unrestringierte numerische Minimierung mit einem Quasi-Newton-Verfahren durchgeführt

um die Polygonzüge quasi-äquidistant zu halten wurde ein Penalty-Term eingeführt

desweiteren wurden Terme berücksichtigt, welche die Kosten steiler Anstiege bzw. Gefälle modellieren

diverse Fälle wurden getestet

alternativ können andere Optimierungsroutinen genutzt werden, die etwa Nebenbedingungen erlauben (nlopt) oder ableitungsfrei arbeiten (Nelder-Mead)

eine Lösung mittels Variationsrechnung unter Nutzung der Euler-Lagrange-Gleichungen kommt auch in Frage, doch dies sprengt den Rahmen dieser Demo

In []: