```
A=[6 8 8; 21 6 3; 18 8 6]
using Symbolics
@variables x,y
toy(i)=7-2*i
tox(j)=2*j-1
R=[((x-tox(j))^2+(y-toy(i))^2)^0.5 for i=1:3,j=1:3]
z=3.2+1.7*sum(A.*R.^0.91)/84
fs="fz(x,y)="*string(z)
eval(Meta.parse(fs))
using Plots
contour(0:0.05:6,0:0.05:6,fz)
```

Contour plot: x axis 0–6, y axis 0–6; colour bar 6.5–10.5 (6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0, 10.5).

Annotations on the plot:
- not good place here
- contours of the average time to get to the fire if the station is located at (x,y)
- starting point for Newton
- (2,3)
- min average time to get to the fire.

$\nabla z = 0$ and solve for (x,y).

$g(x,y) = \nabla z$

Symbolic gradient.

```
julia> gradf=Symbolics.gradient(fz(x,y),[x,y])
2-element Vector{Num}:
 0.02023809523809524((16.38(-1 + x)) / (sqrt((-1 +
99999999) + (7.28(-3 + x)) / (sqrt((-3 + x)^2 + (-
+ (2.73(-5 + x)) / (sqrt((-5 + x)^2 + (-3 + y)^2)^
 + x)) / (sqrt((-3 + x)^2 + (-3 + y)^2)^1.08999999
sqrt((-5 + x)^2 + (-5 + y)^2)^1.0899999999999999)
 x)^2 + (-5 + y)^2)^1.0899999999999999) + (19.11(-
-3 + y)^2)^1.0899999999999999) + (7.28(-3 + x)) /
^1.0899999999999999) + (5.46(-5 + x)) / (sqrt((-5
999999999))
 0.02023809523809524((16.38(-1 + y)) / (sqrt((-1 +
99999999) + (5.46(-1 + y)) / (sqrt((-5 + x)^2 + (-
+ (19.11(-3 + y)) / (sqrt((-1 + x)^2 + (-3 + y)^2)
3 + y)) / (sqrt((-5 + x)^2 + (-3 + y)^2)^1.0899999
```

To help computer algebra system... let's write $\sqrt{\cdot}$ as $(\cdot)^{0.5}$

```
R=[((x-tox(j))^2+(y-toy(i))^2)^0.5 for i=1:3,j=1:3]
```

Now R is like this and

```
julia> gradf=Symbolics.gradient(fz(x,y),[x,y])
2-element Vector{Num}:
 0.020238095238095238095240((5.46(-1 + x)) / (((-1 + x)^
999) + (5.46(-5 + x)) / (((-5 + x)^2 + (-1 + y)^2
3 + x)) / (((-3 + x)^2 + (-5 + y)^2)^0.54499999999
5 + x)^2 + (-3 + y)^2)^0.5449999999999999) + (16.
1 + y)^2)^0.5449999999999999) + (5.46(-3 + x)) /
4999999999999) + (7.28(-3 + x)) / (((-3 + x)^2 +
) + (7.28(-5 + x)) / (((-5 + x)^2 + (-5 + y)^2)^0
+ x)) / (((-1 + x)^2 + (-3 + y)^2)^0.544999999999
 0.020238095238095238095240((2.73(-3 + y)) / (((-5 + x)^
999) + (7.28(-1 + y)) / (((-3 + x)^2 + (-1 + y)^2
-3 + y)) / (((-1 + x)^2 + (-3 + y)^2)^0.54499999999
-5 + x)^2 + (-5 + y)^2)^0.5449999999999999) + (16
```

Still difficult to solve for $\nabla z = 0$ so try Newton's method

$$g\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \nabla z$$

$$X_0 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Need the Jacobian matrix for g (or equivalently the Hessian of f)

$$X_1 = X_0 - Dg(X_0) \backslash g(X_0)$$

$$X_2 = X_1 - Dg(X_1) \backslash g(X_1)$$

$$X_3 = X_2 - Dg(X_2) \backslash g(X_2)$$

looks like a fn def.

[convert to expression to avoid Num type later

```
julia> gs="g(x,y)="*string(Symbolics.toexpr(gradf))
"g(x,y)=begin\n    #= /x/floa/ejolson/.julia/package
de.jl:685 =#\n    (SymbolicUtils.Code.create_array)(
(), Val{(2,)}(), (*)(0.02023809523809524, (+)((+)((+
)(5.46, (+)(-1, x)), (^)((+)((^)(" ⋯ 1537 bytes ⋯ "(-
+)(-3, y), 2)), 0.5449999999999999))), (/)((*)(7.28,
)(-3, x), 2), (^)((+)(-5, y), 2)), 0.544999999999999
y)), (^)((+)((^)((+)(-5, x), 2), (^)((+)(-1, y), 2))
nd"

julia> eval(Meta.parse(gs))
g (generic function with 1 method)
```

← fed back in to Julia...

```
julia> g(x,y)
2-element Vector{Num}:
 0.02023809523809524((5.46(-1 + x)) / (((-1 + x)^2 + (-5 + y)^2)^0
999) + (5.46(-5 + x)) / (((-5 + x)^2 + (-1 + y)^2)^0.544999999999
3 + x)) / (((-3 + x)^2 + (-5 + y)^2)^0.5449999999999999) + (2.73(-
5 + x)^2 + (-3 + y)^2)^0.5449999999999999) + (16.38(-1 + x)) / (((
1 + y)^2)^0.5449999999999999) + (5.46(-3 + x)) / (((-3 + x)^2 + (-
4999999999999) + (7.28(-3 + x)) / (((-3 + x)^2 + (-1 + y)^2)^0.54
) + (7.28(-5 + x)) / (((-5 + x)^2 + (-5 + y)^2)^0.544999999999999
+ x)) / (((-1 + x)^2 + (-3 + y)^2)^0.5449999999999999))
 0.02023809523809524((5.46(-5 + y)) / (((-1 + x)^2 + (-5 + y)^2)^0
999) + (2.73(-3 + y)) / (((-5 + x)^2 + (-3 + y)^2)^0.544999999999
1 + y)) / (((-3 + x)^2 + (-1 + y)^2)^0.5449999999999999) + (19.11(
-1 + x)^2 + (-3 + y)^2)^0.5449999999999999) + (7.28(-5 + y)) / (((
5 + y)^2)^0.5449999999999999) + (16.38(-1 + y)) / (((-1 + x)^2 + (
4499999999999999) + (5.46(-3 + y)) / (((-3 + x)^2 + (-3 + y)^2)^0.5
```

← ∇z is the function.

solve this ∇z = 0

gradient of z

```
gradf=Symbolics.gradient(fz(x,y),[x,y])
gs="g(x,y)="*string(Symbolics.toexpr(gradf))
eval(Meta.parse(gs))
jacg=Symbolics.jacobian(g(x,y),[x,y])
Dgs="Dg(x,y)="*string(Symbolics.toexpr(jacg))
eval(Meta.parse(Dgs))
```

← jacobian of the gradient... (same as hessian)

```
julia> xn=[2.0,3.0]
2-element Vector{Float64}:
 2.0
 3.0

julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.471133510162801
 2.6821612659034932
```

... means feed the elements of the vector xn as the arguments xn[1], xn[2] into the functions Dg and g.

← improved approx...

again ... is called "splat" operator and

iterate

```
julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.5639700775842522
 2.781032476478142

julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.6171501928978793
 2.7650714968565913

julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.6164879700559405
 2.76590753501586

julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.616487857748521
 2.765906920091334

julia> xn=xn-Dg(xn...)\g(xn...)
2-element Vector{Float64}:
 1.6164878577491917
 2.765906920090973
```

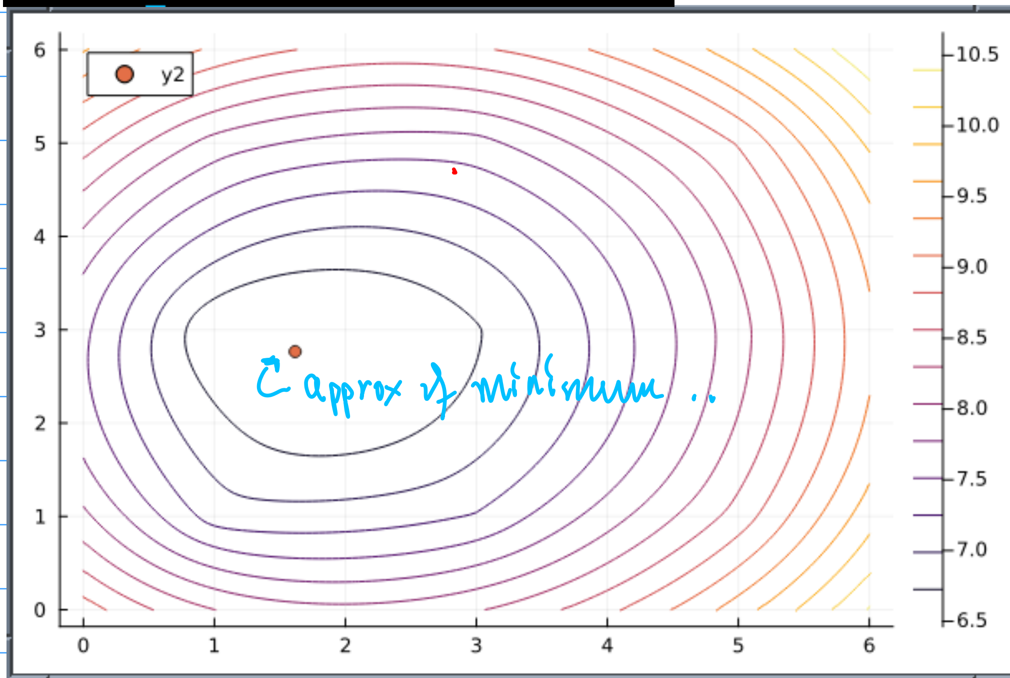
Dg(xn...) means Dg(xn[1], xn[2])

g(xn...) means g(xn[1], xn[2])

approx. of minimum ..

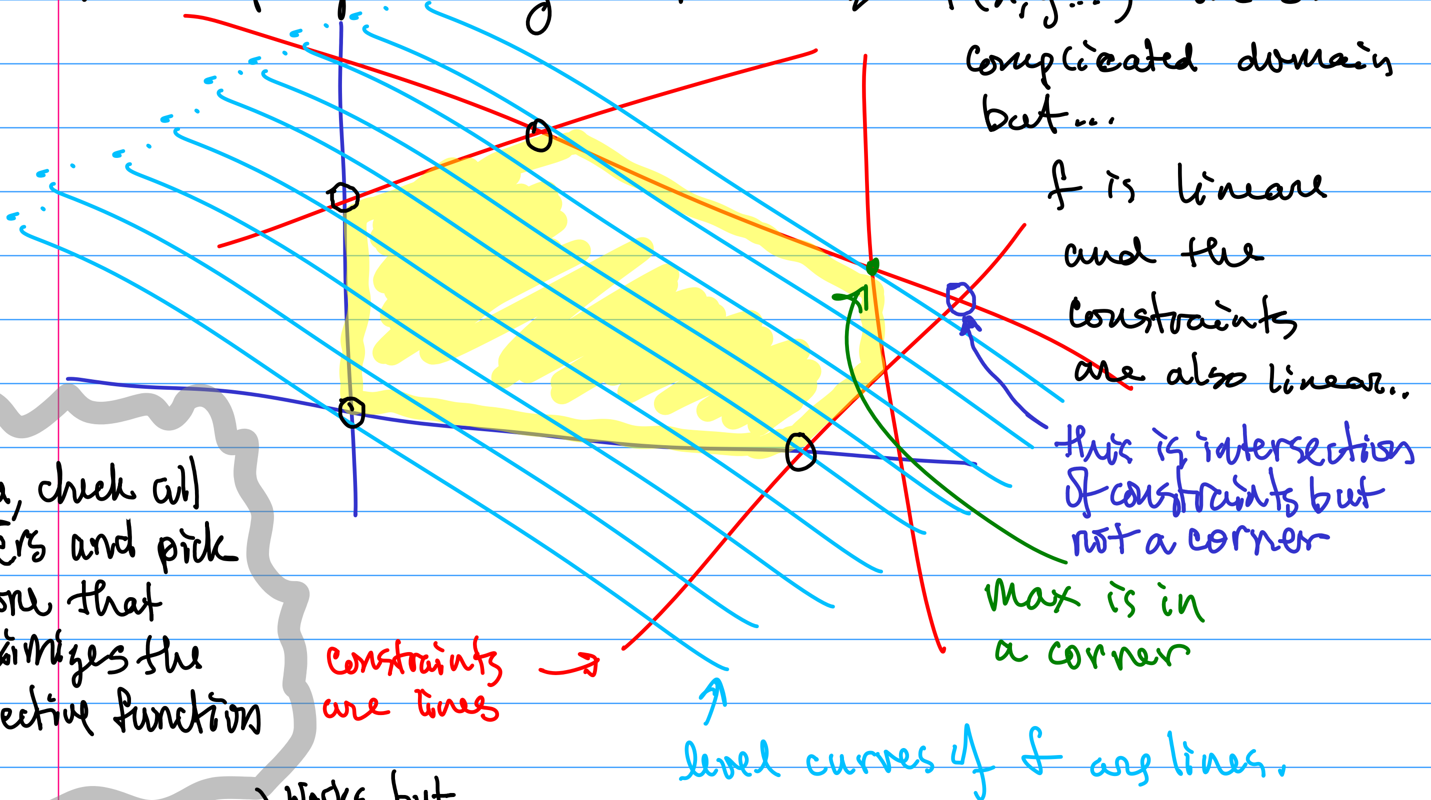
Plot the minimum we found on the contours ...

```
julia> scatter!((xn[1],xn[2]))
```



Chapter 3.3 Linear Programming..

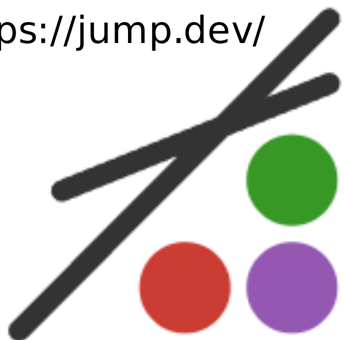
Linear programming: Maximize $f(x,y\ldots)$ on a complicated domain but... f is linear and the constraints are also linear..



Works, but

When there are lots of variables the constraints are hyperplanes and there is a factorial number of corners to check... The idea is to move from corner to corner in a way that maximizes f and avoids checking all the corners...

https://jump.dev/



JUMP

What is JuMP?