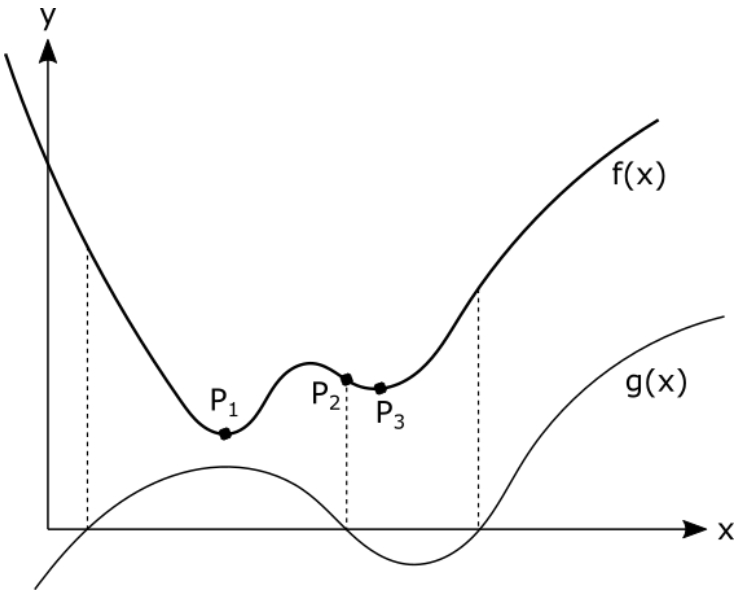
16. OPTIMIZATION WITH CONSTRAINTS

In the case of optimization with constraints, we search for the minimum of the function in such a way that the points must also satisfy some constraint or condition. There can be one or more conditions in the task, they can be given by equations or inequalities (see the following figure), they can be linear or non-linear. Different methods can be used in different cases (e.g. Lagrange method, penalty function method, Karush-Kuhn-Tucker conditions, linear programming).



1 THE MINIMUM OF THE FUNCTION: P_1 – WITHOUT CONSTRAINTS, P_2- WITH G(X)=0 CONSTRAINT, P_3 – WITH G(X)<0 CONSTRAINT

CONSTRAINT GIVEN BY EQUALITY

Among the constrained optimization problems, select those that have only equality conditions and those that have inequality conditions. The solution of the former is much simpler, for example it can be done using the Lagrange method or the penalty function method. Let's look at an example of this type of task!

We want to maximize the profit of a factory manufacturing steel objects. The largest part of our costs is labor wages, and the cost of steel raw materials. A labor hour costs $20 and a ton of steel costs $170. We can give the profit with the following function:

$$f(h,s) = 200 \cdot h^{2/3} \cdot s^{1/3}$$

where *h* is the number of working hours and *s* is the amount of steel in tons.

In addition, there is a budget available: $25,000. We want to get the maximum profit from this budget. We can write the following function for the costs:

$$20 \cdot h + 170 \cdot s$$

In order to maximize profit, we use the entire available budget. In this case, the following condition must be met: $20 \cdot h + 170 \cdot s = 25000$

For the solution, we enter this constraint in zero-ordered form

$$g(h,s) = 20 \cdot h + 170 \cdot s - 25000 = 0$$

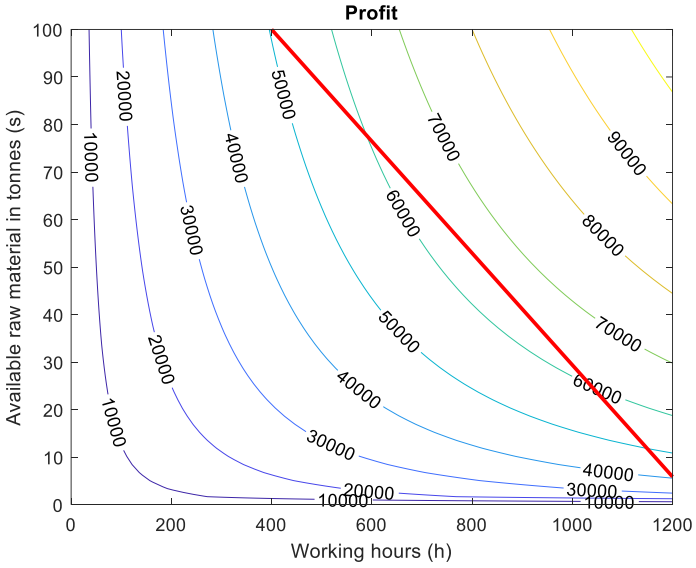
Note: The constraint is now the equation of a line, that is, we have a linear equality condition. The task could be solved in the same way in the case of a condition given by a non-linear equation (adjusted to zero).

Now we want to maximize the value of the bivariate surface *f* (objective function), so that the condition specified in function *g* is fulfilled. For the solution, we first represent the objective function *f* with contour lines, and the condition *g* with a curve.

The objective function can be visualized as a two-variable 3D surface with the **fsurf** command, or in contour map with the **fcontour** or **ezcontour** command. When using **ezcontour**, we can label the contour lines using **set** command, by turning on the **ShowText** property, and the contour line interval can be changed by entering the **LevelStep** or **LevelList** parameters.

Our constraint is given in an implicit form, we can draw it with the **fimplicit** command, if it given in a zero-ordered form.

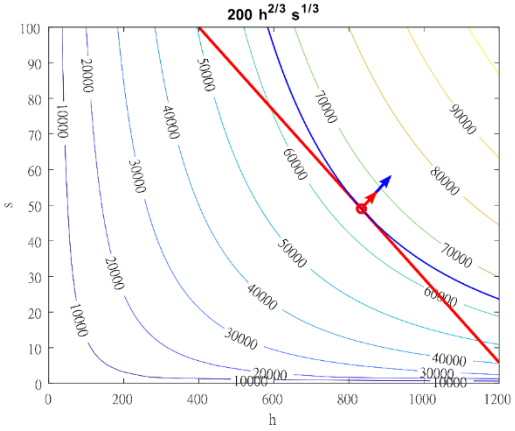
```
> clc; clear all; close all; format shortG;
> % define objective function
> f = @(h,s) 200*h.^(2/3).*s.^(1/3)
> % define budget condition: 20$/hour, 170$/ton: 20*h+170*s = 25000$
> g = @(h,s) 20*h+170*s-25000
> % plot labelled contour map
> figure(1); h1 = ezcontour(f,[0 1200 0 100])
> set(h1,'ShowText', 'on','LevelStep', 10000)
> % plot condition as an implicit function
> hold on; fimplicit(g,[0 1200 0 100],'r','LineWidth',2)
> xlabel('Working hours (h)')
> ylabel('Available raw material in tonnes (s)')
> title('Profit')
```



LAGRANGE-METHOD

How can we solve such a problem? Based on the figure, the maximum profit will be somewhere around $60-70,000 according to the given condition, where the condition is tangent to the contour lines. At this point, the normal of the contour line of the objective function, that is, the gradient of the surface, is parallel to the normal of the condition. Their size may differ by a proportionality factor λ (see the figure below). In the present case, this relationship can be written as follows[1]:

$$\nabla f(h,s) = \lambda \cdot \nabla g(h,s)$$



We rearrange this equation to zero:

$$\nabla f(h,s) - \lambda \cdot \nabla g(h,s) = 0$$

In addition to matching the gradients, of course the constraints must also be fulfilled, i.e. in this case the solution must lie on the specified line

$$g(h,s) = 0$$

In the case of two variables, if we have one constraint, then two equations can be written for matching the gradients and one for the constraint. In this case, we will get a system of equations consisting of 3 equations with 3 unknowns, where 2 unknowns are the 2 unknown variables, and the third is the proportionality factor, λ.

The Lagrange method is based on this principle, and the proportionality factor λ is called the Lagrange multiplier. In addition to the matching of the gradients, the condition must also be fulfilled. The beauty of the Lagrange method lies in the fact that the parallelism of the gradients and the fulfillment of the conditions can be specified with a single formula.

The Lagrange method can be used to find both constrained minimum and maximum locations, but only in the case of constraint(s) given by equality, not in the case of inequality. Using Lagrange-method, constrained optimization can be traced back to the solution of a system of equations (linear or non-linear).

Let's look at the Lagrange method in general. Instead of the original task, write the unconstrained minimization of the following function:

$$L(x,\lambda) = f(x) - \lambda^T \cdot g(x) = f(x) - \sum_{i=1}^{m} \lambda_i \cdot g_i(x)$$

where λ_i are the Lagrange multipliers, and vector x contains the unknowns. A necessary condition for the minimum is the vanishing of the partial derivatives, i.e.

[1] Parallelism is not a necessary requirement, only if $\nabla f \neq 0$ at the given location. When the condition intersects the local extreme value of the surface (where $\nabla f = 0$), then the condition will not necessarily be perpendicular to the contour lines. However, the equation $\nabla f = \lambda \cdot \nabla g$ will also be true in this case. A good example of this can be found here:
https://math.stackexchange.com/questions/2578903/lagrange-multipliers-tangency

$$\frac{d}{dx}L(x,\lambda) = \frac{d}{dx}f(x) - \lambda^T \cdot \frac{d}{dx}g(x) = \nabla f(x) - \sum_{i=1}^{m} \lambda_i \cdot \nabla g_i(x) = 0$$

$$\frac{d}{d\lambda_i}L(x,\lambda) = g_i(x) = 0$$

$i = 1,2,\cdots,m$, where m is the number of constraints given by equalities. The derivatives with respect to the variable x, will represent the parallelism of the gradients, and the derivative with respect to λ will return the original constraint.

A necessary condition for the extreme value is that the above equations are fulfilled. A sufficient condition for the minimum is that the Hessian matrix of the function $L(x,\lambda)$ is positive definite, i.e. the eigenvalues of the matrix are positive at the location of the extreme value. If the Hessian matrix is negative definite at the location of the solution, then the function has a local maximum under the given condition.

Let's solve the constrained problem given in the example using the Lagrange method! In the case of the constraint given by the nonlinear equation, we can write the following Lagrange function:

$$L(h,s,\lambda) = 200 \cdot h^{\frac{2}{3}} \cdot s^{\frac{1}{3}} - \lambda \cdot (20 \cdot h + 170 \cdot s - 25000)$$

A necessary condition for the minimum is the disappearance of the partial derivatives, i.e. the solution of the following system of equations:

$$\frac{dL(h,s,\lambda)}{dh} = 0 \qquad \frac{dL(h,s,\lambda)}{ds} = 0 \qquad \frac{dL(h,s,\lambda)}{d\lambda} = 0$$

After calculating the derivatives, we get the following system of equations:

$$\frac{400}{3} \cdot \frac{s^{1/3}}{h^{1/3}} - 20 \cdot \lambda = 0$$

$$\frac{200}{3} \cdot \frac{h^{2/3}}{s^{2/3}} - 170 \cdot \lambda = 0$$

$$20 \cdot h + 170 \cdot s - 25000 = 0$$

We see that using this formula, we obtained the two conditions for parallelism of the gradients (first 2 equations) and the original constraint itself (equation 3). Now we only have to solve this system of equations! Of course, Matlab can also be used to determine the derivatives using symbolic calculations.

```
> % define Lagrange function
> L = @(h,s,lambda) f(h,s)-lambda*g(h,s);
> % determine the derivatives using symbolic calculations
> syms h s lambda
> LS = L(h,s,lambda)
> % 200*h^(2/3)*s^(1/3) - lambda*(20*h + 170*s - 25000)
> dh=diff(LS,h) % (400*s^(1/3))/(3*h^(1/3)) - 20*lambda = 0
> ds=diff(LS,s) % (200*h^(2/3))/(3*s^(2/3)) - 170*lambda = 0
> dl=diff(LS,lambda) % 25000 - 170*s - 20*h = 0
```

It is a system of nonlinear equations for h,s,λ variables. Let's define the system of equations in Matlab and solve it using **fsolve**! First, all the derivatives must be put into

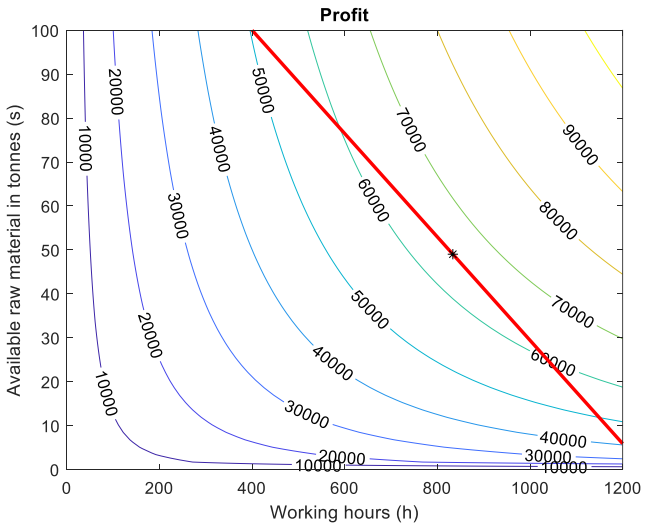
a system of equations (column vector), and then the symbolic expressions must be transformed back into functions. Please note that **fsolve** can handle multivariate functions only in vector variable form, so the multivariate function must be vectorized here as well!

```
> % Create system of equation
> FLsym = [dh;ds;dl]
> %  (400*s^(1/3))/(3*h^(1/3)) - 20*lambda
> % (200*h^(2/3))/(3*s^(2/3)) - 170*lambda
> %                    25000 - 170*s - 20*h
> % Change symbolic expression to Matlab function
> FL = matlabFunction(FLsym) % FL = @(h,lambda,s) [lambda.*2.0e+1+...
> % vectorization of system of nonlinear equation
> FL = @(v) FL(v(1),v(2),v(3)) % @(h,lambda,s) -> v=[h,lambda,s]
```

Initial values are also required for the solution. You must pay attention to the order of variables which Matlab created in FL function using **matlabFunction** command. The initial values must also be entered in the same order, to get the final result. Now the variables are listed in the order h, λ, s in the FL function. With the help of the contour map, we can choose a starting value for the h, s variables (where the curve approaches the contour line with the highest value). We cannot assign an initial value to the lambda variable, so let's choose its value as 1.

```
> x0 = [800; 1; 50] % intial value for h, lambda, s
> sol = fsolve(FL,x0,optimset('Display','iter'))
> h1 = sol(1) % 833.33
> l1 = sol(2) % 2.5927
> s1 = sol(3) % 49.02
> plot(h1,s1,'k*');
> zopt1 = f(h1,s1) % 64819
```

According to the obtained results, our profit will be maximum ($64,819) when we pay for 833 working hours and 49 tons of steel from the existing $25,000 budget. What is also interesting is that λ is not merely a proportionality factor (2.59), but can be deduced to indicate the magnitude of the change in profit as a function of the invested amount. If we invested $1 more than the current budget, our profit would increase by $2.59!



PENALTY FUNCTION METHOD

In the case of the solution using the Lagrange method, it was necessary to calculate the gradients, which is not always easy to calculate. There are other methods, such as the penalty function method, where there is no need to calculate gradients, and we can solve the equality-constrained optimization task without it. Let's look at another example for this!

Given the following surface, in the range $-0.5 \leq x \leq 0.5;\ -0.5 \leq y \leq 1$.

$$f(x,y) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)}$$

Determine the minimum along the given line

$$y = 0.6 \cdot x + 0.3$$

The constraint is an explicitly given linear equality (equation of a line). If we look at the objective function *f*, we see that it is now a much more complicated function than in the previous example. Of course, the Lagrange method could also be used here, but we would get much more complicated equations after the derivation. Now, however, let's look at a solution method that does not require the calculation of gradients, the penalty function method! This solution can be used in the case of constraints given by both linear and nonlinear equality, but not in the case of inequality.

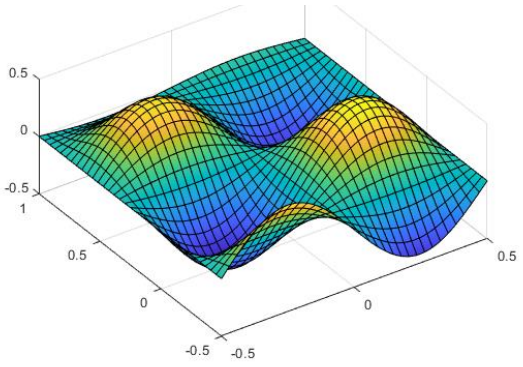
Let's define the objective function and the constraint in Matlab for the representation!

```
> clc; clear all; close all; format shortG;
> % Definition of objective function as a two-variable surface
> f = @(x,y) sin(2*pi*x).*cos(5*y)./((2+x.^3).*(1+2*y.^5))
> % Constraint definition, as an explicit, univariate function
> e = @(x) 0.6*x+0.3
```

3D REPRESENTATION OF OBJECTIVE FUNCTION AND CONSTRAINT[2]

The first step of the solution is, of course, to plot the objective function itself with the given constraints. We can do this on a spatial figure as a surface or with contour lines. In most cases, the contour map is sufficient, but for the sake of a better illustration, let's look at the 3D representation! First, we plot the surface of the objective function using the **fsurf** command (in some cases, the previous version, **ezsurf**, may be necessary).

```
> figure(1);
> fsurf(f,[-0.5 0.5 -0.5 1])
```



Now let's create the 3D figure of the constraint also! This can be done in two ways. One is to plot the constraint as a vertical surface (with **fsurf**). Another solution is to plot the 3D intersection line itself to the surface. For this, we define points with a suitable density along the condition, and then their corresponding heights on the surface is calculated. Finally,

[2] Supplementary material for home study

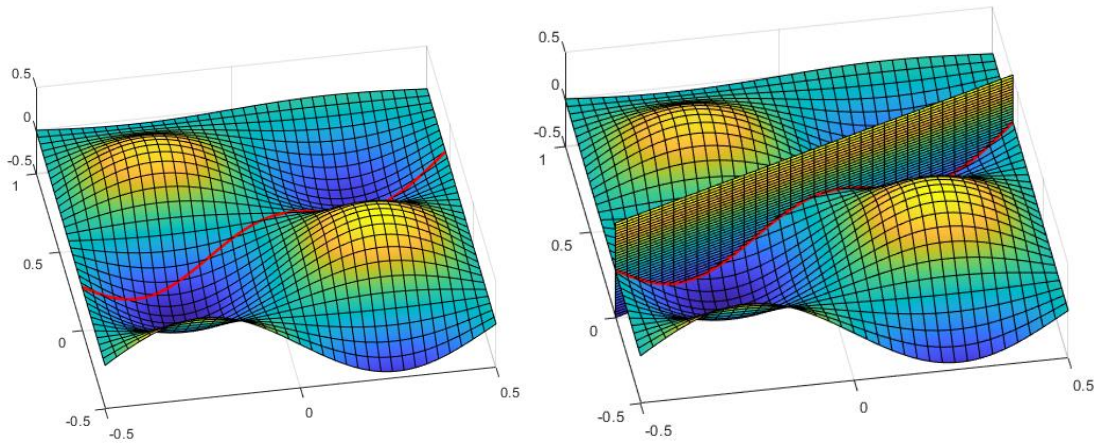
the spatial curve is plot into the surface (**plot3**). Let's look at both solutions. Let's start with the latter!

```
> % plot constraint as spatial curve
> xi = linspace(-0.5,0.5,50)'; % select 50 points in the range of x
> yi = e(xi); % calculate y coorinates along the line
> zi = f(xi,yi); % calculate corresponding surface heigths
> hold on; plot3(xi,yi,zi,'r','LineWidth',2)
> view([-10 70]) % point of view setting (azimuth, elevation angle)
```

See the result in the figure below on the left!

The constraint can also be specified as a vertical surface parametrically and plotted with the **fsurf** command (**`fsurf(funx,funy,funz,uvinterval)`**. Two parameters are required, one of them should be the z coordinate itself, we give this to the funz function, since every z value must be taken as a vertical surface. Our other parameter is the x coordinate, and y can be specified with the equation of the line as a function of x: $y = 0.6 \cdot x + 0.3$. The range of the two parameters (x,z), must also be specified. We know that $-0.5 \leq x \leq 0.5$, but based on the contour map, the values of z also remain within the range [-0.5,0.5].

```
> % constraint given with a vertical surface parametrically
> xp = @(x,z) x
> yp = @(x,z) 0.6*x+0.3
> zp = @(x,z) z
> fsurf(xp,yp,zp,[-0.5,0.5,-0.5,0.5])
```



Based on the figures, it is clear that we have two local minima along the curve. If we calculate the value of both, we can decide which of the two is smaller, which is the global minimum on the range along the given condition.

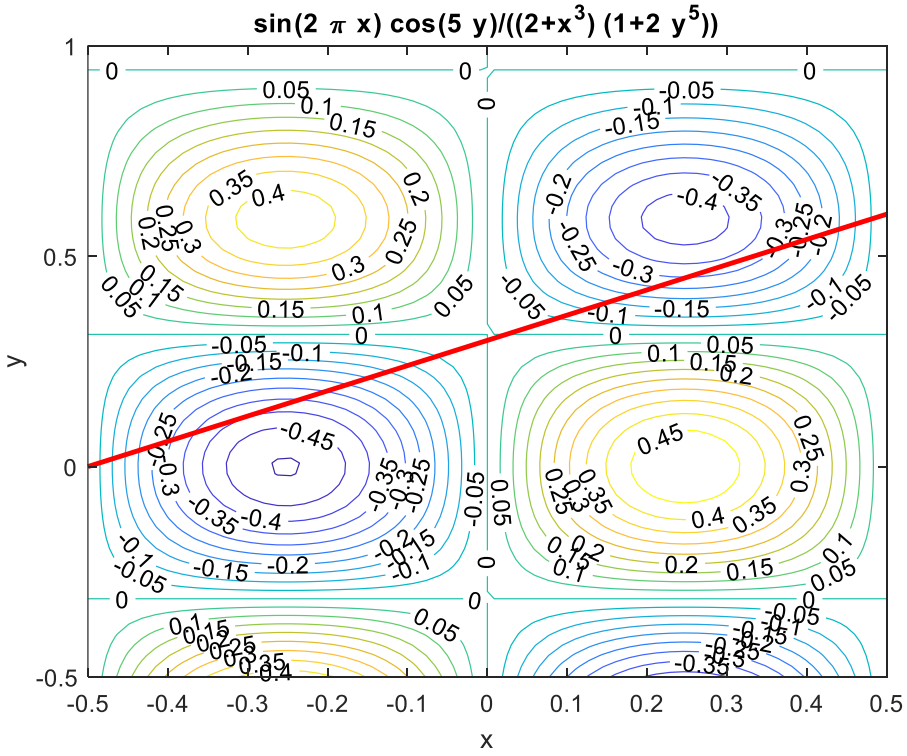
CONTOUR MAP REPRESENTATION

In order to find the solution, a contour map representation is usually sufficient, a spatial figure is not necessary. It is easier to choose a starting value from the contour map. If there are several local minima, it is also easier to decide which one will be the smallest, the global minimum, if the values of the contour lines are labeled.

First, plot the target function as a contour map! Let's use a contour interval of 0.05 and label the contour lines! After that, plot the constraint in the contour map with a different line type. The line given in explicit form can be drawn with the usual **fplot** command. If the constraint was given in an implicit form, we could use the **fimplicit** command.

```
> figure(2); h1 = ezcontour(f,[-0.5 0.5 -0.5 1])
> set(h1,'ShowText','on','LevelStep', 0.05)
> hold on
> e = @(x) 0.6*x+0.3
> fplot(e,[-0.5,0.5],'r','LineWidth',2)
```

Of course, it can also be seen from the contour map that the local minimum location with the smaller x coordinate will now also be the global minimum, since here the constraint passes by the contour line of -0.4, and in the other case by the contour line of -0.35.



SOLUTION WITH PENALTY FUNCTION METHOD

This method, similarly to the Lagrange method, is only suitable for optimization with conditions given by equality. The task is given in the following form:

Target function: $z = f(x, y)$

Constraints with equalities: $g_i(x, y) = 0$

The essence of the method is that it incorporates the conditions into the objective function, so we get an unconditional optimization task. Instead of constrained minimization, consider the following unconstrained minimization problem:

$$F(x, K) = f(x) + K \cdot g(x)^T \cdot g(x)$$

where K>0, a new parameter. The higher its value, the more we 'penalize' the deviation of the g(x) condition from zero, considering the g(x) condition squared. By increasing the value of the parameter K, the solution of the new, unconstrained problem converges to the solution of the original problem with the equality constraint.

The larger K is, the larger the "penalty" for the deviation of g(x) from zero. This quadratic penalty function is called the Courant penalty function. The current task can be written in this form:

$$F(x,y,K) = f(x,y) + K \cdot g(x,y)^2$$

Note: if we had not only one, but two constraints, we could write the task in the following form: $F(x,y,K) = f(x,y) + K \cdot (g_1(x,y)^2 + g_2(x,y)^2)$

Let's determine the location of the conditional minimum using the penalty function method! Let's try several K parameters and see how the solution changes! The scalar parameter of the penalty function should be 10, 100, 1000 and 10000, respectively. After reducing the task to an unconditional optimization task, this task can be solved either with **fminunc**, which uses the quasi-Newton method, or **fminsearch**, which uses the simplex method.

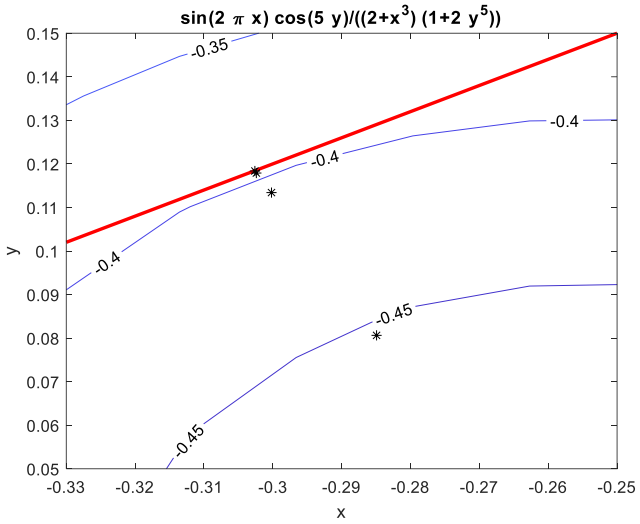
Since in this case the condition is not in a zero-ordered form, we must first rearrange it to zero.

$$g(x,y) = 0.6 \cdot x + 0.3 - y = 0$$

The other important aspect that we must pay attention to in Matlab is that we must transform both the objective function *f* and the constraint *g* into vector variable function!

```
> % Penalty function method
> g = @(x,y) 0.6*x+0.3-y % rearrange to zero
> F = @(v) f(v(1),v(2)); % vectorization
> G = @(v) g(v(1),v(2)); % vectorization
> % penalty function with different K parameters
> P10 = @(v)  F(v) + 10* G(v).^2;
> P100 = @(v)  F(v) + 100* G(v).^2;
> P1000 = @(v)  F(v) + 1000* G(v).^2;
> P10000 = @(v)  F(v) + 10000* G(v).^2;
> % the first local minimum for different K values
> x01 = [-0.25;0.1] % initial values for x,y
> [sol1 min1] = fminsearch(P10, x01)
> [sol2 min2] = fminsearch(P100, x01)
> [sol3 min3] = fminsearch(P1000, x01)
> [sol4 min4] = fminsearch(P10000, x01)
> sol = [sol1,sol2,sol3,sol4]
>       -0.28493     -0.30019     -0.30239     -0.30239
>       0.080633      0.11341       0.1179       0.1179
> fsol = [min1,min2,min3,min4]
>       -0.43069     -0.40224     -0.39835     -0.39835
> % check the constraint
> felt = g(sol(1,:),sol(2,:))
>   0.048408    0.0064735   0.00066395   0.00066395
> % Plot solutions
> figure(2); plot(sol(1,:),sol(2,:),'k*')
> axis([-0.33 -0.25 0.05 0.15])
```

It can be seen that by increasing the parameter K, the solution gets closer and closer to the condition g(x,y)=0, to the point where the condition touches the level lines of the objective function, which we also searched with the Lagrange method.



BUILT-IN MATLAB FUNCTION (FMINCON) – CONSTRAINT GIVEN BY LINEAR EQUATION

Of course, Matlab also has its own built-in function for the constrained optimization, this is the **fmincon** function (find minimum of constrained function). We can call it in the following form to use with linear equality constraint:

`[x,fval] = fmincon(`*fun,x0,A,b,Aeq,beq,lb,ub*`)`

Here *'fun'* is the objective function (with vector variable) and *'x0'* is the vector of the initial values. Constraints given by linear inequality could be specified in the variables *A,b*, and with linear equalities in variables *Aeq,beq*, in the following form:

$$Aeq \cdot x = beq$$

The range of the variables can be set by specifying the lower (*lb*) and upper bounds (*ub*). Let's solve the previous task with the built-in function as well!

Objective function: $$f(x,y) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)}$$

Constraint given by the equation of the line: $y = 0.6 \cdot x + 0.3$

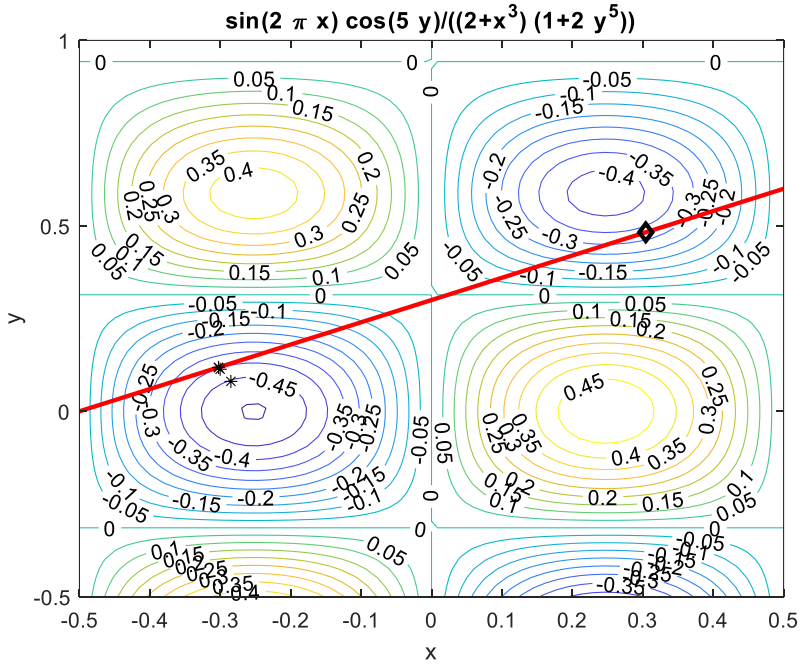
The specified lower/upper bounds: $x \in [-0.5,0.5]; y \in [-0.5,1]$

The constraint given by linear equality must be given in the form $Aeq \cdot v = beq$, where v vector contains the unknowns (*x,y* in this problem), the coefficients of the unknowns are in the matrix *Aeq*, and the constants are in the vector *beq*. Let's transform the constraint into this form!

$$0.6 \cdot x - y = -0.3 \qquad Aeq = [0.6 \quad -1];\ beq = -0.3$$

Since we do not have linear constraints given by inequalities, *A,b* will be empty matrices. Lower/upper limits are specified (*lb,ub*). It is also necessary to enter initial values. Now let's find the second local minimum along the condition with the built-in function!

```
> % Solution with fmincon
> axis([-0.5 0.5 -0.5 1]) % zoom to the range
> x02 = [0.3;0.4] % initial values for x,y
> A=[]; b=[]; % There is no constraint given by a linear inequality
> Aeq=[0.6 -1]; beq=[-0.3]; % Constraint with a linear equation
> % Lower and upper bounds for x,y
> lb=[-0.5,-0.5]; ub=[0.5,1];
> % Solution - F with vector variable!
> [sol2 fsol2] = fmincon(F,x02,A,b,Aeq,beq,lb,ub)
> % sol2 =   [0.30431; 0.48259];   fsol2 =  -0.3294
> plot(sol2(1),sol2(2),'kd','LineWidth',2)
```



CONSTRAINED OPTIMIZATION INCLUDING INEQUALITIES (FMINCON IN GENERAL FORM)

The **fmincon** function is a general-purpose constrained optimization algorithm. Constraints can be specified not only with equations, but also with inequalities, they can be either linear or non-linear, with upper/lower bounds and other options. It can be called in the following form (with more/less inputs and outputs):

```
[x,fval] =  fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Parameters:

- *fun*: objective function
- *x0*: initial values
- *A,b*: $A \cdot x < b$ constraints given by linear inequalities
- *Aeq,beq*: $Aeq \cdot x = beq$ constraints given by linear equations
- *lb,ub*: $lb < x < ub$, lower/upper bounds
- *nonlcon*: nonlinear constraints: $c(x) \leq 0$ and $ceq(x) = 0$
- *options*

The conditions must be entered in the specified order, if one type of constraint is missing, an empty matrix must be put in its place.

Let's solve the previous task with different conditions! Given the following objective function in the range $-0.5 \leq x \leq 0.5;\ -0.5 \leq y \leq 1$.

$$f(x,y) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)}$$

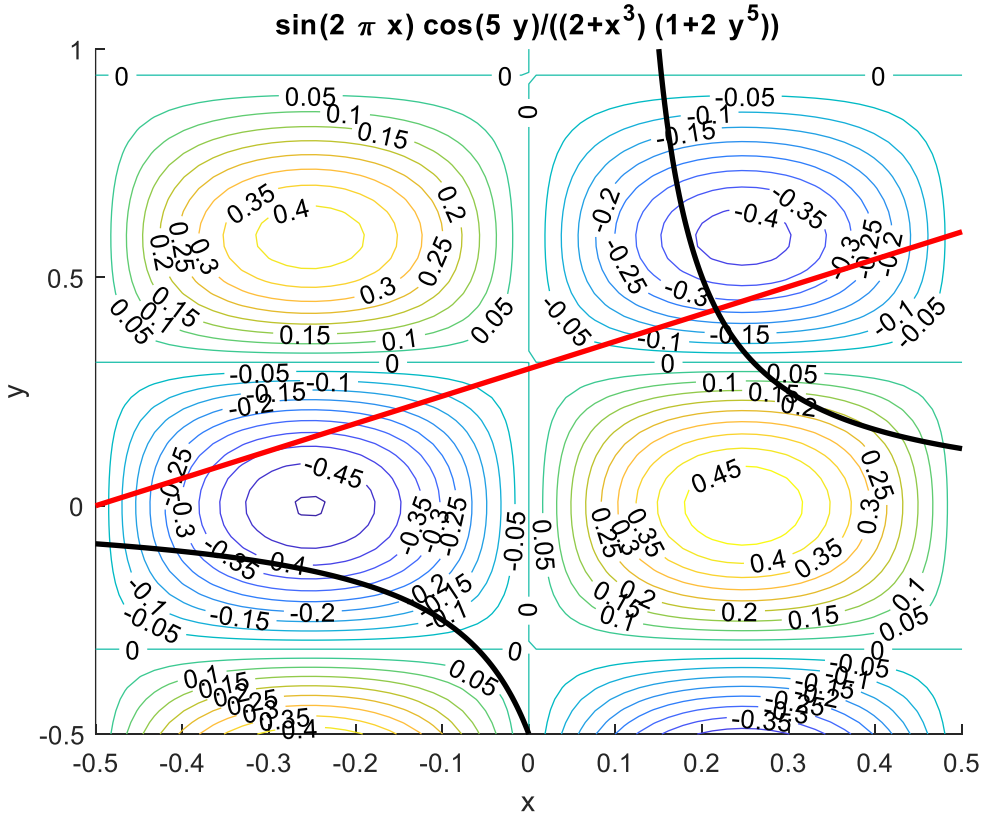
Determine the minimum under the given conditions!

$$20 \cdot x \cdot y - 2 \cdot y = 1$$

$$y > 0.6 \cdot x + 0.3$$

The first constraint is a non-linear equality given in an implicit form, and the second constraint is a linear inequality, which tells us to look for the minimum in the area located above the previously specified line and in the given nonlinear curve. Let's plot in a new figure the contour map and the two constraints! The original contour map and the straight line can be taken from the previous example. The nonlinear condition is not specified explicitly, but implicitly, so we now use the **fimplicit** command instead of the **fplot** command to plot it, after reordering the constraint to 0!

```
> clc; clear all; close all; format shortG;
> f = @(x,y) sin(2*pi*x).*cos(5*y)./((2+x.^3).*(1+2*y.^5))
> figure(3); hold on;
> h1 = ezcontour(f,[-0.5 0.5 -0.5 1])
> set(h1,'ShowText','on','LevelStep', 0.05)
> e = @(x) 0.6*x+0.3;
> fplot(e,[-0.5,0.5],'r','LineWidth',2)
> g1 = @(x,y) 20*x.*y-2*y-1
> fimplicit(g1,'k','LineWidth',2)
```



Let's first look at the constraint given by the linear inequality $y > 0.6 \cdot x + 0.3$! This means that we are looking for the minimum location in the area above the line ($y = 0.6 \cdot x + 0.3$). The condition should be put in the following form:

$$A \cdot x < b$$

Where the unknowns are on the smaller side of the relational sign, and the constant is on the larger side (or, in the case of more inequality constraints, the constants are in a vector). Let's transform the $y > 0.6 \cdot x + 0.3$ inequaliy into this form to determine the coefficients *A,b*!

$$0.6 \cdot x - y < -0.3 \qquad A = [0.6 \quad -1];\ b = -0.3$$

Note: if we were to look for the minimum below the line, then the $y < 0.6 \cdot x + 0.3$ inequality should be put in this form:

$$-0.6 \cdot x + y < 0.3 \qquad A = [-0.6 \quad 1];\ b = 0.3$$

Now we do not have constraint given by linear equations, so *Aeq* and *beq* must be replaced by an empty matrix. Lower/upper bounds remained the same as before.

```
> A=[0.6 -1]; b=-0.3; % linear inequality
> Aeq=[]; beq=[]; % there is no linear equation constraint
> lb=[-0.5,-0.5]; ub=[0.5,1]; % Lower/upper bound for x,y
```

Now let's look at the nonlinear constraints! The non-linear constraints can also be specified with an equation or inequality, these can be specified in the *nonlcon* parameter.

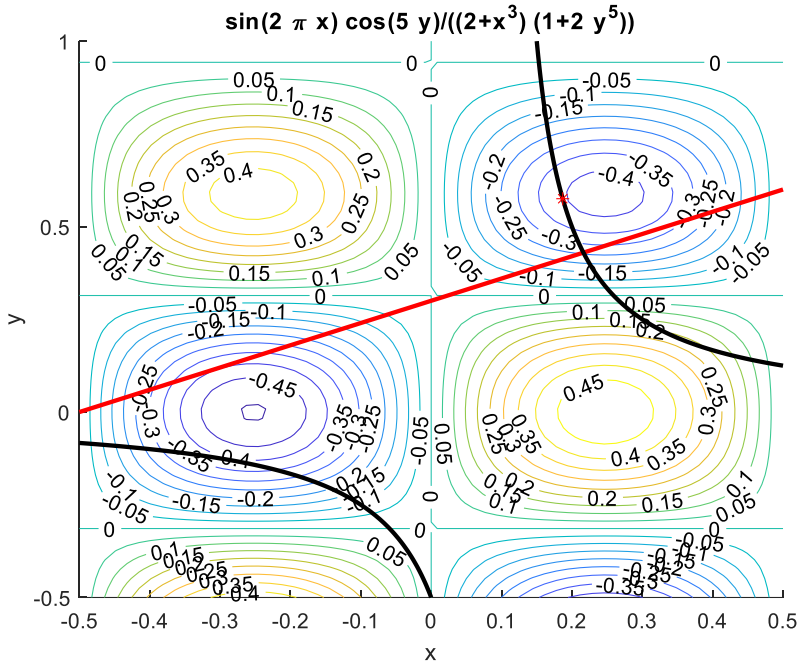
Inequality: $c(x) \leq 0$ Equation: $ceq(x) = 0$

Here, two types of conditions must be specified in one function (*nonlcon*). A function with two outputs must be generated, this can be done with the **deal** command. The conditions must be specified in the form $c(x) \leq 0$ and $ceq(x) = 0$. If we have only one of the two types of conditions, the other will be an empty matrix. A vector variable function must be defined here as well.

```
> % there are nonlinear constraints given by equality
> c = []; % no inequality
> ceq = @(x,y) 20*x.*y-2*y-1 % rearranged to zero
> nonlcon = @(v) deal(c, ceq(v(1),v(2))) % constraint with vector
  variable
```

For the solution, the objective function must also have vector variable. We need to specify an initial value which can be chosen from the contour map. We have to chose a point where the nonlinear constraint is above the line, and it approaches the smallest contour line. Optional parameters can also be specified, e.g. displaying the iteration steps, determining for example the tolerance for the function value as 10^{-9}.

```
> F = @(v) f(v(1),v(2)) % objective function with vector variable
> x0 = [0.2;0.5] % initial value from the figure
> opc = optimset('Display','iter','TolFun',1e-9);
> sol  = fmincon(F,x0,A,b,Aeq,beq,lb,ub,nonlcon,opc)
> % 0.18693,  0.57515
> plot(sol(1),sol(2),'r*')
```



What would happen if we looked for the minimum along a horizontal or vertical line? For example, along the lines x=-0.2, or y = 0.5?

$x = -0.2$	$A \cdot x = b$ alakban:	$1 \cdot x + 0 \cdot y = -0.2$	$Aeq = [1 \quad 0]; beq = 0.2$
$y = 0.5$	$A \cdot x = b$ alakban:	$0 \cdot x + 1 \cdot y = 0.5$	$Aeq = [0 \quad 1]; beq = 0.5$

In case, e.g. we have the condition given by the inequality x<-0.2, then we can enter the values of *A,b* as described earlier, or we can simply set the upper limit of *x* to -0.2.

LINEAR PROGRAMMING PROBLEM[3]

Linear programming is a very useful optimization technique for engineering applications. The word 'linear' refers to the fact that both the objective function and the constraints are only linear functions of non-negative variables. To be able to use this method, three conditions must be met:

1) The objective function is a linear function of the variables
2) The constraints are also linear functions of the variables
3) Variables can only take non-negative values

There are several techniques to solve this kind of problems, e.g. simplex method, interior-point method, etc. Now we will not go into details, we will use Matlab's built-in **linprog** command:

```
[x fval] = linprog(f,A,b,Aeq,beq,lb,ub,options)
```

We can see that the parameters are very similar to those of the **fmincon** function. Here are the parameters:

[3] Supplementary material for home study

- *fun*: object function to be minimized given as a vector
- *A,b*: $A \cdot x < b$ constraints given by linear inequalities
- *Aeq,beq*: $Aeq \cdot x = beq$ constraints given by linear equations
- *lb,ub*: $lb < x < ub$, lower/upper bound
- *options*

It does not include an initial value, as it is not needed in this case, which is a big advantage in many cases. It is also not possible to specify a non-linear condition. Specifying the linear conditions works the same as for **fmincon**, except for the objective function to be minimized. Let's look at a simple example!

Linear objective function to be minimized: $z = -2 \cdot x + 8 \cdot y$

Linear constraints:

$$3 \cdot x + 4 \cdot y \leq 80$$

$$-3 \cdot x + 4 \cdot y \geq 8$$

$$x + 4 \cdot y \geq 40$$

$$x, y \geq 0$$

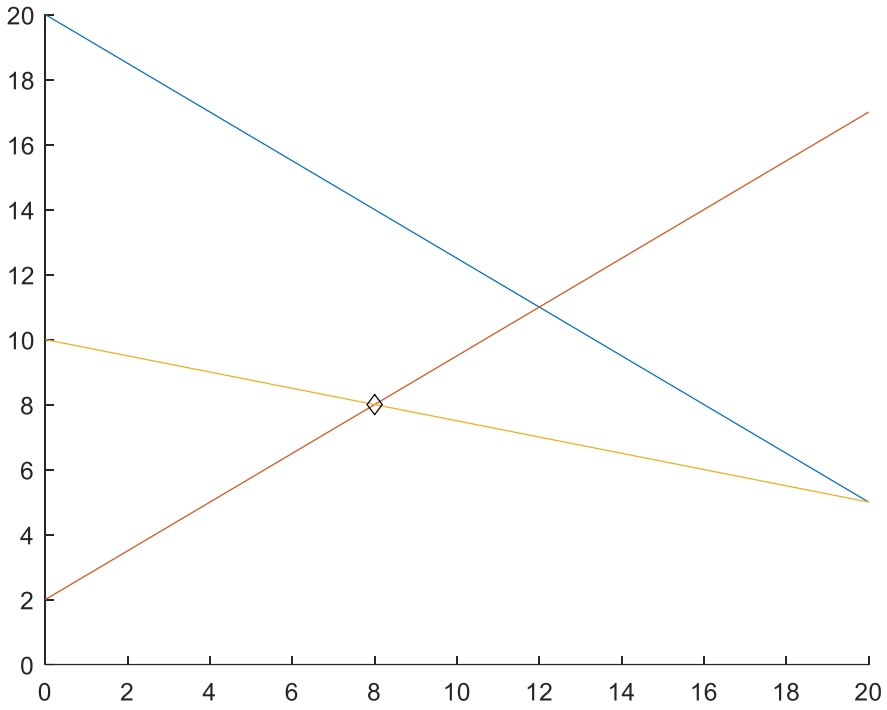
Specifying the objective function as a vector means specifying the coefficients of the unknowns: [-2 8]

Inequalities must be put in the form $A \cdot x < b$ (as before), where the variables are on the smaller side of the relational sign, and the constants on the larger side.

$$\begin{array}{ll} 3 \cdot x + 4 \cdot y \leq 80 & 3 \cdot x + 4 \cdot y \leq 80 \\ -3 \cdot x + 4 \cdot y \geq 8 \quad \rightarrow & 3 \cdot x - 4 \cdot y \leq -8 \\ x + 4 \cdot y \geq 40 & -x - 4 \cdot y \leq 40 \end{array}$$

Let's solve the problem in Matlab!

```
> clear all; clc; close all;
> % coefficient vector of the objective function
> c=[-2 8];
> % Plot constraints
> g1 = @(x,y) 3*x+4*y-80
> g2 = @(x,y) -3*x+4*y-8
> g3 = @(x,y) x+4*y-40
> figure(1); hold on; fimplicit(g1,[0,20]); fimplicit(g2,[0,20]);
  fimplicit(g3,[0,20]);
> % Linear constraint with inequality in the form of A*x <= b
> A = [3 4; 3 -4; -1 -4];
> b = [80; -8; -40];
> Aeq = []; beq = []; % Linear constraint with equality: none
> lb = [0 0]; % There is a lower limit for variables
> ub = [];% Upper limit for variables: none
> sol = linprog(c,A,b,Aeq,beq,lb,ub,optimset('Display','iter'))
> % sol = 8 8
> plot(sol(1),sol(2),'kd')
```



PRACTICE EXERCISES

PRACTICE EXERCISE 1. – LAGRANGE-METHOD

Let's solve the task in the penalty function method subsection with Lagrange method now! Given the following surface; in the range $-0.5 \leq x \leq 0.5;\ -0.5 \leq y \leq 1$.

$$f(x,y) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)}$$

Determine the minimum of the surface using the Lagrange method under the following constraint:

$$y = 0.6 \cdot x + 0.3$$

In the case of linear equation constraint, we can write the following Lagrange function (first rearranging the constraint to zero):

$$L(x,\mathrm{y},\lambda) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)} - \lambda \cdot (0.6 \cdot x + 0.3 - y)$$

A necessary condition for the minimum is the disappearance of the partial derivatives, i.e. the solution of the following system of equations:

$$\frac{dL(x,\mathrm{y},\lambda)}{dx} = 0 \qquad \frac{dL(x,\mathrm{y},\lambda)}{dy} = 0 \qquad \frac{dL(x,\mathrm{y},\lambda)}{d\lambda} = 0$$

After calculating the derivatives, we get the following system of equations:

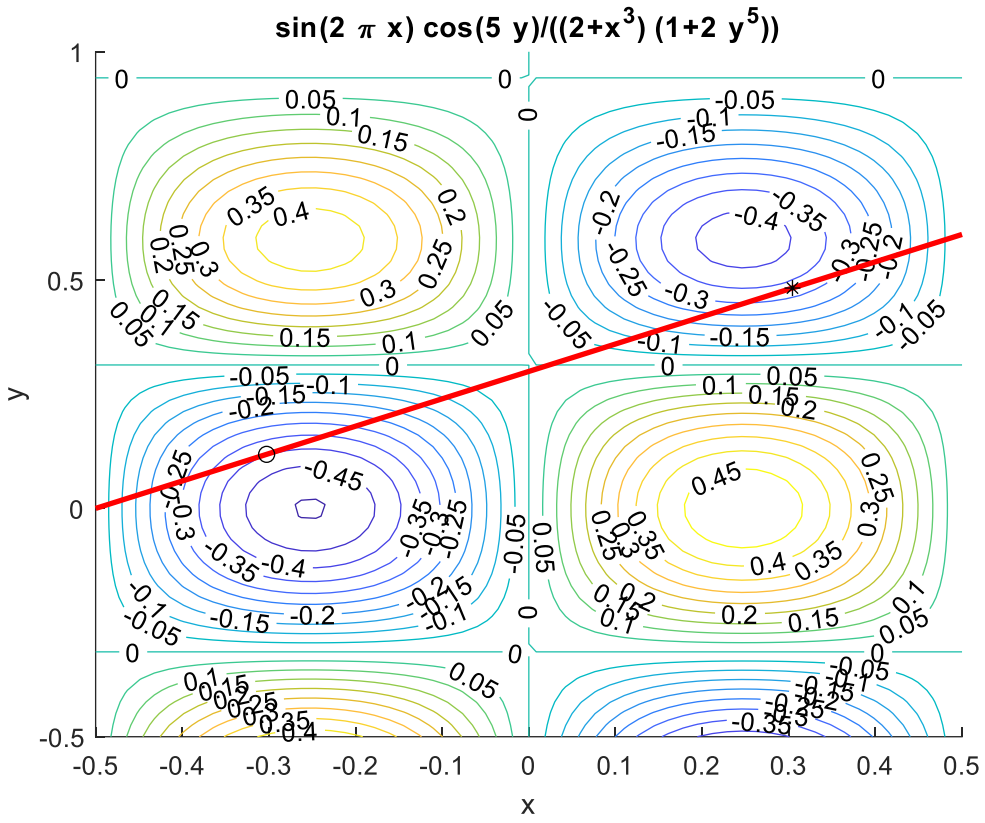
$$\frac{\cos(5\cdot y)}{1+2\cdot y^5}\cdot\left(\frac{\cos(2\cdot\pi\cdot x)\cdot 2\cdot\pi}{(2+x^3)}-\frac{\sin(2\cdot\pi\cdot x)\cdot 3\cdot x^2}{(2+x^3)^2}\right)-0.6\cdot\lambda\ =0$$

$$\frac{\sin(2\cdot\pi\cdot x)}{(2+x^3)}\cdot\left(\frac{-\sin(5\cdot y)\cdot 5}{(1+2\cdot y^5)}-\frac{\cos(5\cdot y)\cdot 10\cdot y^4}{(1+2\cdot y^5)^2}\right)+\lambda=0$$

$$0.6\cdot x+0.3-y=0$$

In this case, it is not so easy to determine the partial derivatives, but of course, Matlab can also be used for this, using symbolic calculations.

```
%% Solution by Lagrange method - linear constraint
clc; clear all; close all; format shortG;
f = @(x,y) sin(2*pi*x).*cos(5*y)./((2+x.^3).*(1+2*y.^5))
figure(3); hold on;
h1 = ezcontour(f,[-0.5 0.5 -0.5 1])
set(h1,'ShowText','on','LevelStep', 0.05)
e = @(x) 0.6*x+0.3;
fplot(e,[-0.5,0.5],'r','LineWidth',2)

g = @(x,y) 0.6*x+0.3-y % rearrange to 0!
L = @(x,y,lambda) f(x,y)-lambda*g(x,y);

syms x y lambda
dx=diff(L(x,y,lambda),x)
% (2*pi*cos(5*y)*cos(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)) - (3*lambda)/5
- (3*x^2*cos(5*y)*sin(2*pi*x))/((x^3 + 2)^2*(2*y^5 + 1))
dy=diff(L(x,y,lambda),y)
% lambda - (5*sin(5*y)*sin(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)) -
(10*y^4*cos(5*y)*sin(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)^2)
dl=diff(L(x,y,lambda),lambda)
% y - (3*x)/5 - 3/10

% Solving systems of nonlinear equations with fsolve
FLsym = [dx;dy;dl]
FL = matlabFunction(FLsym) % @(lambda,x,y)...
% Vectorization of the system of nonlinear equations
FL = @(v) FL(v(1),v(2),v(3)) % v = (lambda,x,y)

% a megoldás
x01 = [1; -0.3;0.1];  % 1st initial value, order: lambda,x,y
x02 = [1; 0.3;0.5];  % 2nd initial value, order: lambda,x,y

xyl1 = fsolve(FL,x01,optimset('Display','iter'))
% xyl1 = [-1.3387,-0.30265,0.11841]
xyl2 = fsolve(FL,x02,optimset('Display','iter'))
% xyl2 = [1.3002, 0.30431,0.48259]
plot(xyl1(2),xyl1(3),'ko'); plot(xyl2(2),xyl2(3),'k*')
zopt1 = f(xyl1(2),xyl1(3)) %  -0.3979
zopt2 = f(xyl2(2),xyl2(3)) %  -0.3294
```



PRACTICE EXAMPLE 2. – CONE WITH MINIMUM SURFACE

We would like to determine the data (radius, height) of a cone with a minimum surface and unit volume.

The surface of the cone: $A = r^2 \cdot \pi + \pi \cdot r \cdot a$

where a is the slant height of the cone, the length of which is: $a = \sqrt{r^2 + h^2}$

The volume of the cone: $V = \frac{r^2 \cdot \pi \cdot h}{3} = 1$

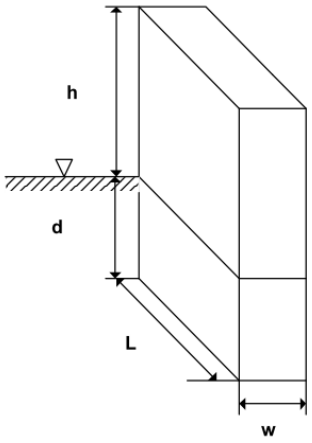
a) Write the Matlab function of the surface of the cone depending on the radius and height. Also write the Matlab function of the constraint corresponding to the unit volume!
b) Solve the constrained optimization problem for radius and height using different methods. In each case, check the fulfillment of the constraint and determine what the ratio of the height and the radius will be, and how big the resulting surface will be?
   i. using Matlab built-in command
   ii. with Lagrange method
   iii. with penalty function method (K=1000)

Solution:

```
> %% 1A - cone surface
> clc; clear all; close all;
>
> % A=R^2*pi+pi*R*a, a = sqrt(r^2+h^2)
> % V=pi*r^2*h/3=1;
> % a)
> A = @(r,h) r.^2*pi+pi*r.*sqrt(r.^2+h.^2)
> V = @(r,h) pi*r.^2.*h/3-1
> A = @(v) A(v(1),v(2)); V = @(u) V(u(1),u(2));
>
> % b-i) Matlab built-in function
> nonlcon = @(u) deal([],V(u))
> v0 = [0.5, 0.5]
> x = fmincon(A,v0,[],[],[],[],[0,0],[],nonlcon)
> r = x(1) % 0.69632
> h = x(2) % 1.9695
> ratio = h/r % 2.8284
> S = A([r,h]) % 6.0929
> V([r,h]) %  -4.0023e-10
>
> % Lagrange method
> syms r h lam
> Lfv = A([r h])+lam*V([r h])
> % lam*((h*pi*r^2)/3 - 1) + pi*r^2 + pi*r*(h^2 + r^2)^(1/2)
> F = gradient(Lfv,[r,h,lam])
> %   pi*(h^2 + r^2)^(1/2) + 2*pi*r + (pi*r^2)/(h^2 + r^2)^(1/2) +
  (2*pi*h*lam*r)/3
> %  (pi*lam*r^2)/3 + (pi*h*r)/(h^2 + r^2)^(1/2)
> %  (h*pi*r^2)/3 - 1
> sol=solve(F)
> %       h: [3×1 sym]
> %     lam: [3×1 sym]
> %       r: [3×1 sym]
> double([sol.r sol.h sol.lam])
> % 0.69632 + 0i 1.9695 + 0i -4.062 + 0i
> % 0.34816 - 0.60303i   -0.98475 + 1.7056i   2.031 + 3.5178i
> % 0.34816 + 0.60303i   -0.98475 - 1.7056i   2.031 - 3.5178i
> r = double(sol.r(1)) % 0.69632
> h = double(sol.h(1)) % 1.9695
> ratio = h/r % 2.8284
> S = A([r,h]) % 6.0929
> V([r,h]) % 0
>
> %b-iii) penalty function method
> Bfv=@(u) A(u)+1000*V(u).^2
> x =fminsearch(Bfv,v0)
> % x =    0.69585       1.9681
> ratio = x(2)/x(1) % 2.8284
> S = A(x) % 6.0847
> V(x) % -0.0020296
```

PRACTICE EXAMPLE 3. – COMPLEX CIVIL ENGINEERING PROBLEM[4]

Let's use Matlab's built-in function to solve a more complicated two-variable constrained optimization problem, where there is a specified lower/upper limit, and constraints given by linear and nonlinear inequalities.



In order to save energy costs, a building partially in the ground should be designed. The total floor area of the 25-story building must be at least 20,000 m^2. The prescribed ratio of the width w and length L of the building is w/L = 1/1.618, and L can be no more than 50 m. The height of each floor is 3.5 m. The energy cost of the building is 100$/year/m^2 based on the surface of the part above ground. Total energy costs per year must not exceed $225,000. Determine the dimensions of the building so that the cost of earthworks (which is proportional to the volume of the underground part of the building) should be minimal!

The value of the objective function to be minimized:

$$f(d,w) = cons \cdot 1.618 \cdot d \cdot w^2. \qquad \text{(cons. = constant = 1/10000)}$$

The inequality constraints:

$$g_1(d,w) = 20000 - 25 \cdot 1.618 w^2 \le 0 \qquad \text{(total floor area for 25 levels)}$$

$$g_2(d,w) = 1.618 w - 50 \le 0 \qquad \text{(building length)}$$

$$g_3(d,w) = 45815 w - 523.6 wd + 161.8 w^2 - 225000 \le 0 \qquad \text{(annual total energy costs)}$$

Lower bounds on the variables:

$$d > 0$$

$$w > 0$$

Solution steps:

1. In order to use MatLab's multivariate constraint optimization mehtod, we must specify the necessary linear inequality constraints and the lower bounds on the variables
2. Write a function for the nonlinear inequality constraints.
3. Starting from the initial values of d = 50, w = 10, we determine the solution of the problem using Matlab's built-in procedure
4. What will be the active inequality constraints?

```
> %% optimization with inequality constraints
> clc; clear all; close all
>
> % Required parameterization of the built-in fmincon function:
> %  X = fmincon(FUN,X0,A,B,Aeq,Beq,LB,UB,NONLCON,OPTIONS)
```

[4] From the exercise book by Béla Paláncz (2012)

```
>  % min F(X)  subject to: A*X  <= B, Aeq*X  = Beq (linear constraints)
>  %  X                    C(X) <= 0, Ceq(X) = 0 (nonlinear constraints)
>  %                      LB <= X <= UB        (bounds)
>
>  % the objective function (with vector variable):1.618 d w^2 / 10000
>  f = @(x) 1.618*x(1)*x(2)^2/1e4;
>
>  % Linear constraint with inequality: -50 + 1.618 w <= 0
>  A = [0 1.618]; b = [50];
>
>  % There is no linear constraint with equality
>  Aeq = [ ];  beq = [ ];
>
>  % There is a lower limit for the variables: d > 0 and w > 0
>  lb = [0; 0];
>  % There is no upper limit for variables
>  ub = [ ];
>
>  % There is no nonlinear equality constraint
>  ceq = [ ];
>  % Nonlinear inequalities: C(X) <= 0
>  g1 = @(d,w) 20000 - 25*1.618*w.^2
>  g2 = @(d,w) 45815*w - 523.6*w.*d + 161.8*w.^2 - 225000
>  % In vector, the two constraints, with vector variables
>  c = @(v) [g1(v(1),v(2)); g2(v(1),v(2))]
>  % Note:
>  % fmincon needs two output values ([c ceq]) generating functions,
>  % so we generate them with the built-in deal function
>  nonlincon = @(v) deal(c(v), ceq)
>  % nonlincon = @(v) deal([20000-25*1.618*v(2)^2;
>  %              45815*v(2)-523.6*v(2)*v(1)+161.8*v(2)^2-225000], []);
>
>  % Alternative solution: separate function in m-file:
>  % function [c ceq] = nonlincon(x)
>  % c = [20000 - 25*1.618*x(2)^2;
>  %      45815*x(2) - 523.6*x(2)*x(1) + 161.8*x(2)^2 - 225000];
>  % ceq = [];
>  % end
>
>  % The initial value
>  x0 = [50; 10];
>
>  % The solution is with Matlab's built-in function:
>  x = fmincon(f, x0, A, b, Aeq, beq, lb, ub, nonlincon, optimset
    ('Display', 'iter''TolFun',1e-9))
>  % x = 75.0459; 22.2360
>
>  % Note: if nonlincon is called as an external function, you need
    @nonlincon as a parameter:
>  % fmincon(f, x0, A, b, Aeq, beq, lb, ub, @nonlincon)
>
>  % check the values of the nonlinear constraints:
>  [f1,f2] = nonlincon(x)
>  % f1 =  1.0e-05 * [-0.0113;  -0.5822]
>  % f2 = []
```

PRACTICE EXERCISES 4. – LINEAR PROGRAMMING IN WATER PURIFICATION[5]

A linear programming problem is the case when both the objective function and the constraints are linear. Let's look at such an example! There are four water treatment plants operating on the river (1-4) and its tributary (2-3) shown in the figure, which process P (mg/day) wastewater from the nearby big cities at a ratio of x, i.e. the pollution entering the river:

$$W = (1 - x) \cdot P$$

If x_i is the degree of purification of the sewage treatment plant in city *i*, then the removed pollution is $x_i \cdot P_i$.

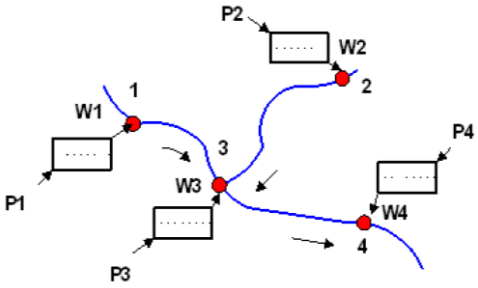
When the wastewater W_i enters the river, it is assumed that it is completely mixed with the pollutant with flow rate $Q_{i,j}$ (i→j) and concentration c_i arriving with the river. That is, the value of the wastewater concentrations (mg/L) after mixing in each section:

$$c_1 = \frac{1 - x_1}{Q_{13}} \cdot P_1$$

$$c_2 = \frac{1 - x_2}{Q_{23}} \cdot P_2$$

$$c_3 = \frac{R_{13} \cdot Q_{13} \cdot c_1 + R_{23} \cdot Q_{23} \cdot c_2 + (1 - x_3) \cdot P_3}{Q_{34}}$$

$$c_4 = \frac{R_{34} \cdot Q_{34} \cdot c_3 + (1 - x_4) \cdot P_4}{Q_{45}}$$



Natural decomposition also occurs in the corresponding (i→j) river section. This is expressed by the R_{ij} factors. The available data is contained in the table below:

City *i*	Wastewater load P_i (mg/day)	Wastewater treatment cost, d_i (Ft/mg)	River section i→j	Flow rate of section, $Q_{i,j}$ (L/day)	Natural decomposition ratio R_{ij}	Allowable impurity concentration, c_{si} (mg/L)
1	$1.0 \cdot 10^9$	$2 \cdot 10^{-6}$	1→3	$1.0 \cdot 10^7$	0.5	20
2	$2.0 \cdot 10^9$	$2 \cdot 10^{-6}$	2→3	$5.0 \cdot 10^7$	0.35	20
3	$4.0 \cdot 10^9$	$4 \cdot 10^{-6}$	3→4	$1.1 \cdot 10^8$	0.6	20
4	$2.5 \cdot 10^9$	$4 \cdot 10^{-6}$	4→5	$2.5 \cdot 10^8$	-	20

Operating cost of daily wastewater treatment:

$$Z(x) = \sum_{i=1}^{4} d_i \cdot P_i \cdot x_i$$

The optimization task is to determine the degree of cleaning (x_i) of each plant in such a way that the operating cost (Z) is minimal, but at the same time the concentration of pollution in each node does not exceed the permissible level, e.g $c_i < c_{si}$ .The objective function is linear and so are the constraints.

[5] Paláncz Béla példatárából (2012)

Substituting the appropriate values into the objective function:

$$Z = 2000 \cdot x_1 + 4000 \cdot x_2 + 16000 \cdot x_3 + 10000 \cdot x_4$$

And the restrictions on concentrations:

$$100 \cdot (1 - x_1) \leq 20$$

$$40 \cdot (1 - x_2) \leq 20$$

$$47.2727 - 4.54545 \cdot x_1 - 6.36364 \cdot x_2 - 36.3636 \cdot x_3 \leq 20$$

$$22.48 - 1.2 \cdot x_1 - 1.68 \cdot x_2 - 9.6 \cdot x_3 - 10 \cdot x_4 \leq 20$$

Additional restrictions on the degree of purification:

$$0 \leq x_i \leq 1$$

The task is therefore a standard linear programming problem!

```
> %% linear programming
> clear all; clc
> % the coefficient vector of the objective function
> c=[2000;4000;16000;10000];
> % Linear constraint with inequality
> % A*x <= b
> A = [-100, 0, 0, 0; 0, -40, 0, 0; - 4.54545,-6.36364,-36.3636,0; -
  1.2, -1.68,-9.6, -10];
> b = [-80;-20; -27.2727;-2.48];
> % There is no linear constraint with equality
> Aeq = []; beq = [];
> % Lower limit for variables
> lb = [0 0 0 0];
> % Upper limit for variables
> ub = [1 1 1 1];
> sol = linprog(c,A,b,Aeq,beq,lb,ub,optimset('Display','iter'))
> % sol = 0.8; 0.5; 0.5625; 0
```

NEW FUNCTIONS USED IN THE CHAPTER

fmincon	-	Find minimum of constrained nonlinear multivariable function
deal	-	Distribute inputs to outputs
linprog	-	Solve linear programming problems