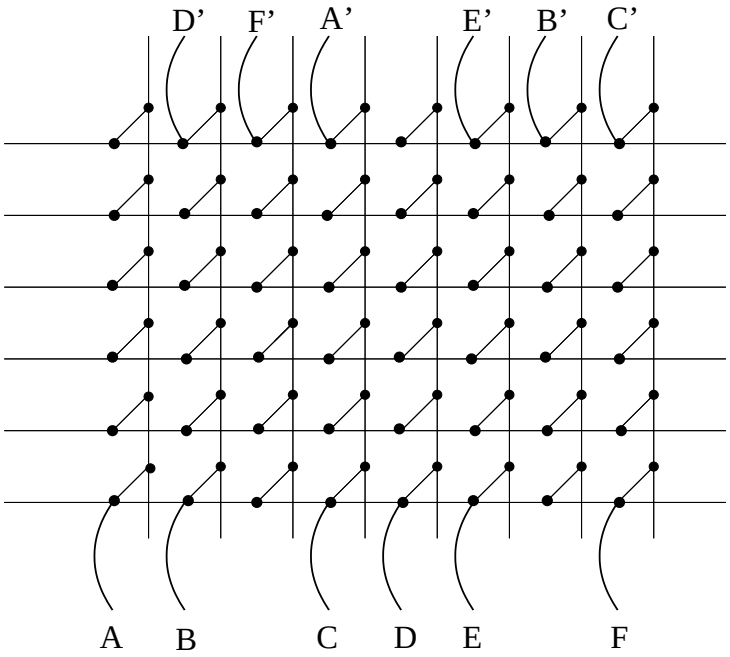
# Projekt 1: VLSI routing och Lagrange-dualitet

## 1 Introduktion

Syftet med detta projekt är att illustrera hur ett relativt svårlöst optimeringsproblem kan angripas med en Lagrange-dual metod. Under projektet kommer ett *Matlab*-program att skrivas för att lösa ett Lagrange-dualt problem. Till er hjälp finns två hjälpfunktioner som kan anropas från Matlab.

Vi betraktar ett ruttningsproblem inom "very large-scale integrated circuit design" (VLSI), nämligen en tillämpning av det som brukar kallas "the Manhattan channel routing problem." Den matematiska klassiciferingen av problemet brukar benämnas "vertex disjoint paths," och problemet och tillämpningen baseras på artikeln "Lagrangian relaxation for testing infeasibility in VLSI routing" av Thomas A. Feo och Dorit S. Hochbaum, som är publicerad i *Operations Research*, vol. 54, sid. 819–831 (1986).

Det program som skall framställas under projektet ska användas för att avgöra om en given placering av ett antal kontakter är möjlig att implementera med avseende på de kopplingar som behövs mellan komponenterna. Kortet där kopplingarna skall göras är sådant att på kortets framsida så kan bara ledningar dras horisontellt och på dess baksida bara vertikalt. På kortet finns ett antal fördefinierade vior (eller genomföringar) där det är möjligt att göra en koppling från ena till andra sidan på kortet.




Figur 1: Rutnät med kontaktpar A–A' till F–F'.

Vi illustrerar ett möjligt problem i Figur 1. I figuren finns sex horisonella ledningar på den ena sidan och åtta vertikala på den andra. Dessutom är sex kontakt-par, A–A', B–B', till och med F–F', inritade i figuren.

Först introducerar vi notation för att matematiskt beskriva nätverket och hur kontaktpunkterna kopplas samman. Sedan tecknar vi ett optimeringsproblem vars optimala målfunktionsvärde säger hurvida ruttningsproblemet har en lösning eller inte. Detta problem kommer vi att teckna Lagrange-dualen till, vilken sedan skall lösas med hjälp av subgradientoptimering. *Det är programkoden för subgradientoptimeringen som ni har till uppgift att skriva.*

Med hjälp av de gränser som ges på det optimala målfunktionsvärdet från Lagrangedualiteten kan slutsatser om huruvida ett problem saknar lösning dras i vissa situationer.

## 2 Uppgift 1

Studera problemet noggrant och skriv ner ett utkast på programkod som följer flödesschemat i Figur 2.

Er uppgift är att skriva programkoden för subgradientoptimeringen. Programmet skall testköras på de testproblem som finns att ladda ner från kurshemsidan. Filerna med namnen "p6.m", "p10.m" och "p11.m" innehåller definitioner för `dimX`, `dimY`, `k` och `com` (vilka beskrivs på sid. 5). Problemet som definieras i "p6.m" är det problem som visas i Figur 1, vilket passar bra att testa programkoden på. Detta problem saknar lösning! På hemsidan finns också de två hjälpfunktioner som beskrivs på sid. 4 och har namnen "gsp.c" och "visagrid.m".

Det som skall redovisas är:

- En utskrift av programkoden med era namn på.
- De "optimala" målfunktionsvärdena för de duala problemen till varje testproblem.
- Grafer för vardera testproblem, där det duala målfunktionsvärdet finns på y-axeln och iterationsnumret på x-axeln.
- Slutsats om vilka testproblem som saknar lösning.

## 3 Uppgift 2

I en andra fas av projektet skall ni också implementera en tillåtenhetsheuristik. I artikeln som projektet bygger på saknas en sådan; det är måhända naturligt, eftersom där i stället beskrivs hur uppdragasgivaren har primala heuristiker redan implementerade, men det vore inte helt tokigt att utnyttja Lagrangesubproblemlösningarna, när de nu redan finns till hands.

Er uppgift är att, baserat på utseendet på Lagrangesubproblemet och de bivillkor som Lagrangerelaxerats, uppfinna en heuristisk metod som manipulerar Lagrangesubproblemets lösning och finner en tillåten lösning till det ursprungliga problemet. Eftersom det är ett grafproblem bör proceduren vara baserad på någon form av grafoperation och/eller optimering. Notera att en Lagrangeheuristik helst inte ska vara alltför komplicerad och att den *måste* terminera, det vill säga, den får inte vara konstruerad så att vi kan råka behöva göra att oändligt antal steg. (I värsta fall får alltså tillåtenhetsheuristiken misslyckas med att finna en tillåten lösning.)

Nyttan med denna heuristik är flerfaldig, men de två mest självklara är (a) att den ger oss också pessimistiska uppskattningar av det optimala målfunktionsvärdet, vilket kan användas i avbrottskriterier för det duala schemat och (b) det ger oss en tillåten lösning till problemet när schemat avbryts och alltså ett förslag på en lösning att implementera.

Det som skall redovisas är:

- En utskrift av denna del av programkoden, tillsammans med en beskrivning i ord av metoden, med era namn på.
- De bästa tillåtna lösningarna som åstadkoms på detta sätt för varje testproblem.

- Grafer för vardera testproblem, där det primala och duala målfunktionsvärdet finns på y-axeln och iterationsnumret på x-axeln.

# 4 Matematisk modell

Vi introducerar notation för att beskriva ruttningen mellan kontaktparen i ett nätverk $\mathcal{G}$.

Låt mängden $\mathcal{V}$ innehålla noderna i nätverket och låt $\mathcal{A}$ innehålla de *riktade* bågar som finns mellan noderna. Låt $n$ vara antalet noder, dvs $|\mathcal{V}| = n$. Noderna motsvarar de punkter på kortet där man kan göra en koppling från ena till andra sidan, de så kallade "viorna". Viorna beskrivs av bågar mellan två noder, en på uppsidan av kortet och en på undersidan. En båge beskriver en möjlig koppling mellan två noder. I Figur 1 representerar varje linje mellan två noder två bågar, en i varje riktning.

Kontaktparen "startar" och "avslutas" i en nod. Låt $k$ vara antalet kontakterpar, och låt $s_l$ och $t_l$ definiera "start"- och "slut"-nod för paren $l = 1, \ldots, k$.

Vi låter $x_{ijl}$ beskriva hurvida kontaktpar $l$ använder bågen från nod $i$ till nod $j$ eller inte. Variablen $x_{ijl}$ får värdet ett om bågen används av paret och noll annars. Definiera dessutom variabler $x_{t_l s_l l}$ som beskriver en båge direkt från $t_l$ till $s_l$ för $l = 1, \ldots, k$. Dessa variabler kan tolkas som "direktbågar" från A' till A, B' till B, osv.

Problemet för att hitta en lösning där så många kontaktpar som möjligt är kopplade kan med hjälp av denna notation nu skrivas som

$$\text{maximera} \sum_{l=1}^{k} x_{t_l s_l l}, \tag{1a}$$

$$\text{då} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i \in \mathcal{V}, \quad l = 1, \ldots, k, \tag{1b}$$

$$\sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil} \leq 1, \qquad i \in \mathcal{V}, \tag{1c}$$

$$x_{ijl} \in \{0, 1\}, \qquad (i, j) \in \mathcal{A}, \quad l = 1, \ldots, k. \tag{1d}$$

Modellen kan tolkas som följer: villkoren (1d) ser till att ett kontaktpar kan passera en båge endera en eller ingen gång; villkoren (1c) ser till att varje nod endast kan passeras totalt en gång, dvs bara användas av maximalt ett kontaktpar; villkoren (1b) ser till att en nod som får en ingående anslutning också får en utgående; målfunktionen beskriver en maximering av antalet "direktbågar" som antar värdet ett, dvs vi vill maximera antalet kontaktpar som kopplas in.

Om det optimala målfunktionsvärdet är lägre än antalet kontaktpar finns ingen möjlighet att med den givna kortstorleken koppla de givna kontaktparen.

# 5 Lagrangedualt problem

Bilda ett Lagrangedualt problem där villkoren (1c) relaxeras. Antalet villkor i (1c) är lika många som antalet noder i nätverket, dvs $n$ stycken. Låt dualvariablerna till villkoren i (1c) betecknas av $\pi_i$, $i = 1, \ldots, n$. Eftersom dualvariablerna $\pi$ svarar mot $\leq$-villkor i ett maximeringsproblem tillkommer kravet $\pi_i \geq 0$, $i = 1, \ldots, n$. Det Lagrangeduala problemet kan skrivas som det att

$$\underset{\pi \geq 0^n}{\text{minimera}} \; h(\pi), \tag{2a}$$

där

$$h(\pi) = \text{maximum} \sum_{l=1}^{k} x_{t_l s_l l} + \sum_{i=1}^{n} \pi_i \left( 1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil} \right), \tag{2b}$$

$$\text{då} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i = 1, \ldots, n, \quad l = 1, \ldots, k, \tag{2c}$$

$$x_{ijl} \in \{0, 1\}, \qquad (i, j) \in \mathcal{A}, \quad l = 1, \ldots, k. \tag{2d}$$

Problemet (2b)–(2d) kallas *Lagrangesubproblem*. Målfunktionen (2b) i Lagrangesubproblemet kan skrivas om till

$$h(\pi) = \text{maximum} \sum_{i=1}^{n} \pi_i + \sum_{l=1}^{k} \left( x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \right). \tag{3}$$

Vi observerar här att eftersom $\pi_i$ är konstanta i Lagrangesubproblemet kan vi separara problemet i $k$ stycken problem, ett för varje kontaktpar, som vi kan lösa vart och ett för sig.

För ett givet kontaktpar $l$ är Lagrangesubproblemet det att

$$\text{maximera} \left\{ x_{t_l s_l l} - \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_i x_{jil} \right\}, \tag{4a}$$

$$\text{då} \quad \sum_{j=1}^{n} (x_{jil} - x_{ijl}) = 0, \qquad i = 1, \ldots, n, \tag{4b}$$

$$x_{ijl} \in \{0, 1\}, \qquad (i, j) \in \mathcal{A}. \tag{4c}$$

Detta problem löser vi i två steg. Först finner vi en billigaste väg mellan noderna $s_l$ och $t_l$ där kostnaderna för att passera en nod $i$ är $\pi_i$. Vi låter $z_i$ beskriva denna väg med $z_i = 1$ om nod $i$ passeras i vägen, annars låter vi $z_i = 0$. I steg två beräknar vi vägkostnaden $\sum_{i=1}^{n} \pi_i z_i$. Om vägkostnaden är mindre än ett löses problemet (4) av att sätta de $x$-variabler som motsvarar de i vägen ingående bågarna till 1. Om vägkostnaden överstiger ett sätts motsvarande $x$-variabler till 0.

Värdet av $h(\pi)$ är enligt teorin för Lagrangdualitet för ett maximeringsproblem en *övre* gräns (UBD) för det optimala målfunktionsvärdet till det ursprungliga problemet, det vill säga en *optimistisk* uppskattning.

Givet lösningarna till de $k$ Lagrangesubproblemen, dvs. värdet av $x$-variablerna, kan en subgradient $d$ i iteration $t$ beräknas enligt

$$d_i^t = 1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}, \qquad i = 1, \ldots, n. \tag{5}$$

(Detta uttryck för sökriktningen kan tolkas i det nätverk som ruttningen sker i. Summan $\sum_{j=1}^{n} x_{jil}$ kan tolkas för ett givet kontaktpar $l$ som antalet använda länkar, startande i noderna $j$, $j = 1, \ldots, n$, som ansluter mot nod $i$. Denna tolkning kan vara användbar då sökriktningen skall beräknas.)

Dualvariablerna kan nu uppdateras genom att vi tar ett steg i riktningen $-d$:

$$\pi_i^{t+1} = \max\{0, \pi_i^t - s^t d_i^t\}, \qquad i = 1, \ldots, n \tag{6}$$

(ett steg motriktat subgradientriktningen eftersom det Lagrangeduala problemet är ett minimeringsproblem), där steglängden $s^t > 0$ kan beräknas enligt formeln

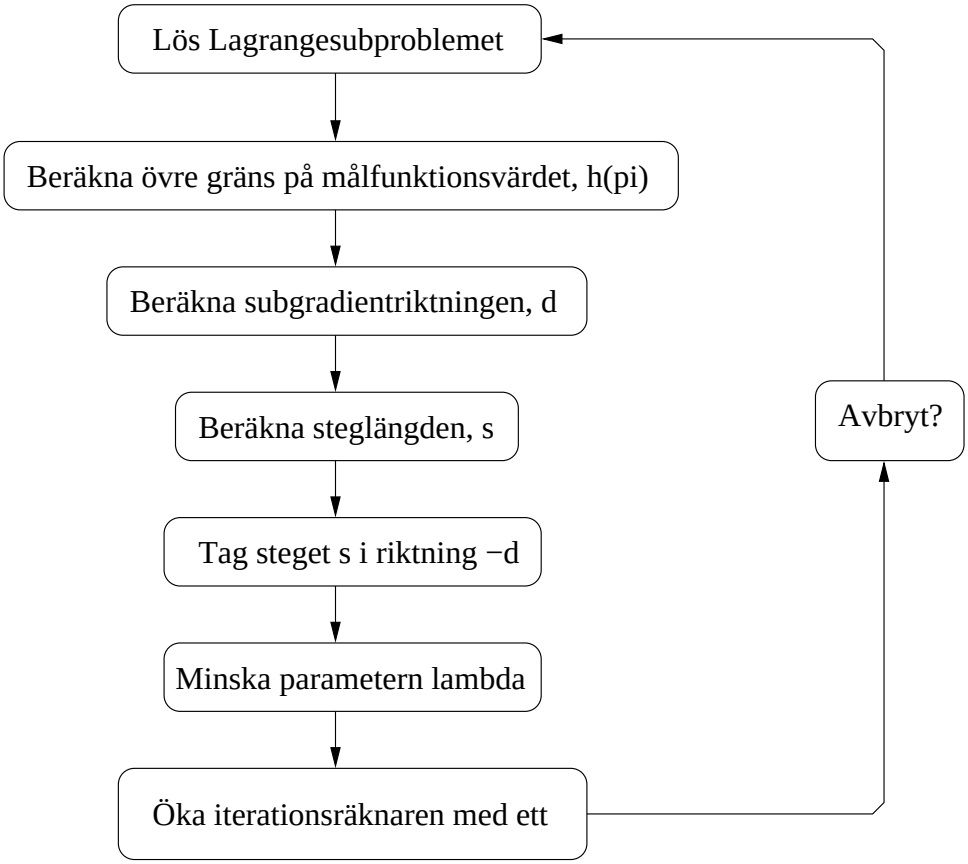
$$s^t = \lambda^t \frac{h(\pi) - LBD}{\sum_{i=1}^{n} \left(1 - \sum_{l=1}^{k} \sum_{j=1}^{n} x_{jil}\right)^2}. \tag{7}$$

Värdet $LBD$ skall vara värdet på en *undre* gräns på det optimala målfunktionsvärdet (en *pessimistisk* uppskattning). Vi väljer att sätta $LBD$ lika med noll. (Värdet $LBD$ skulle kunna vara värdet på en bästa kända tillåten lösning. Värdet på $LBD$ skulle till exempel kunna beräknas med hjälp av en tillåtenhetsheuristik.) Parametern $\lambda^t$ skall ha ett värde strikt mellan noll och två.

Vi noterar att max-operationen i (6) är nödvändig för att vara säker på att multiplikatorerna $\pi$ bibehåller sina teckenkrav även efter en uppdatering. (Vi startar givetvis proceduren i en icke-negativ punkt, t.ex. $\pi_i^0 = 0$ eller $\pi_i^0 = 1/n$ för alla $i$.)

Givet de nya värdena på dualvariablerna, $\pi_i^{t+1}$, $i = 1, \ldots, n$, enligt (6), löses Lagrangesubproblemet igen och så vidare tills dess att vi avbryter. Subgradientoptimeringen avbryter vi normalt efter ett fixt antal iterationer; ett lagom antal iterationer här kan vara runt 1000. Parametern $\lambda$ kan man till exempel initialt sätta till två för att sedan multipliceras med 0.95 var tionde iteration. (Se också artikeln för en alternativ regel.)

Beräkningsschemat för subgradientoptimeringen finns sammanfattat i Figur 2.




Figur 2: Beräkningsschema.

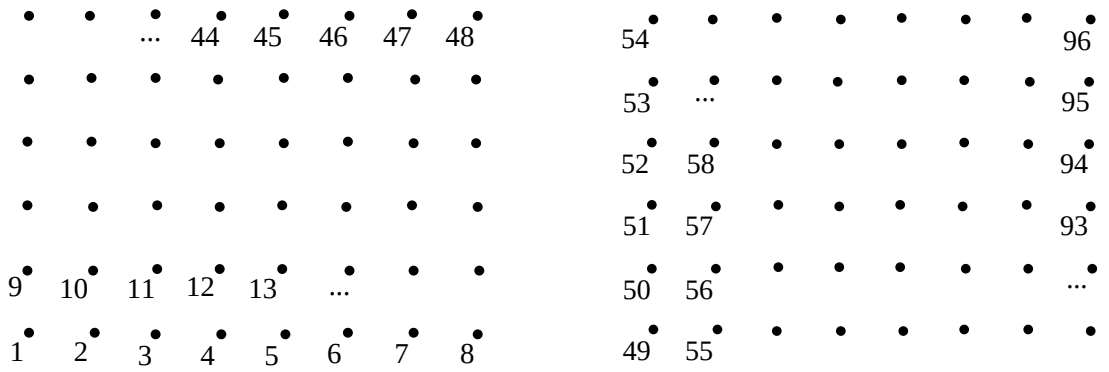
# 6 Hjälpfunktioner i Matlab

Till hands finns två hjälpfunktioner färdigskrivna i Matlab. Den ena är en rutin för att finna $k$ stycken billigaste vägar i ett nätverk. Den andra är en rutin för att grafiskt rita upp ett nätverk och en lösning.

`nl=gsp(dimX,dimY,pi,k,com)`

`dimX` och `dimY` är antalet "vior" i x respektive y-led. `pi` är en vektor med dualvariabelvärden, `pi` har `dimX*dimY*2` antal element. `k` är antalet kontaktpar och `com` är en matris som beskriver start och slutnod för varje kontaktpar. Tex `com = [1 48; 2 42; 3 43]` med `k=3` säger att första paret startar i 1 och slutar i 48, andra startar i 2 och slutar i 42 och tredje startar i 3 och slutar i 43. Resultatet `nl` är en vektor med nodnummer som beskriver de billigaste vägarna.

Det resultat som kommer ur funktionen `gsp` är en lista (`nl`) med nodnummer som beskriver en billigaste väg för varje kontaktpar. Noderna är för ett nätverk med `dimX` = 8 och `dimY` = 6 numrerade enligt Figur 3.




Figur 3: Nodnumrering för övre (vänstra) och undre (högra) lagret.

**Exempel:** Låt `k = 3` med kontaktpar `com = [1 48; 3 41; 5 42]`. Antag att alla element i $\pi$-vektorn har värdet 0.1. Genom att anropa `nl = gsp(8, 6, pi, k, com)` fås `nl = 48 47 46 45 44 43 42 41 54 53 52 51 50 49 1 42 60 59 58 57 56 55 2 43 66 65 64 63 62 61 3`. □

För att givet vektorerna `pi`, `nl`, `com` och `k` ta reda på vilka kontaktpar som har en vägkostnad (beräknad baserad på kostnaden $\pi_i$ för passera nod $i$) som är *lägre än ett* kan följande kod användas:

```
% Beräkna kostnaden per väg och tag bort de vägar som har
% en kostnad > 1 (Kräver vägar i nl och par i com-vektorer.)
last = 0;
for i = 1 : k;
    first = last+1;
    slask = find(nl(last+1:length(nl)) == com(i,1));
    last = slask(1)+first-1;
    if (sum(pi(nl(first:last))) < 1)
        okcom = [okcom i]; newnl = [newnl; nl(first:last)];
    end
end
```

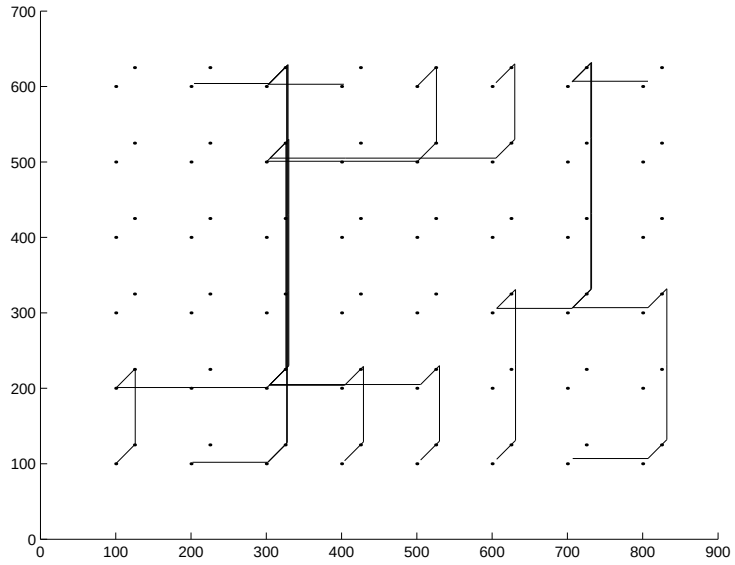
Efter att denna kod har körts finns en vektor `okcom` som innehåller indexen (radnummer i `com`-vektorn) på kontaktpar som har kostnad lägre än ett och en vektor `newnl` som

innehåller alla nodnummer för alla vägarna i dessa kontaktpar lagrade efter varandra i vektorn.

**Exempel (forts.):** Med `nl` beräknat enligt tidigare exemplel fås `okcom = 2 3` och `newnl = 42 60 59 58 57 56 55 2 43 66 65 64 63 62 61 3` som resultat från programkoden ovan. Vi observerar att kopplingen mellan 1 och 48 passerar 15 noder och alltså har en vägkostnad på $15 \cdot 0.1 > 1$. Denna koppling finns alltså inte med i `okcom` eller `newnl`. □

`visagrid(dimX,dimY,nl,com,pi,shift)`

`dimX` och `dimY` är antalet "vior" i x respektive y-led. `nl` är en lista med nodnummer som beskriver alla kopplingar mellan alla kontaktpar. `com` är en matris som beskriver start och slutnod för varje kontaktpar. `pi` är en vektor med dualvariabelvärden, `pi` har `dimX*dimY` antal element. `shift` är ett tal som beskriver hur stor viornas förskjutning skall vara i bilden. Lämpligt värde är 25.




Figur 4: Exempel på resultat från `visagrid`.

# A Appendix: En kort kommandolista för Matlab

Matlab-program skrivs lämpligen i texteditorn Emacs. Matlab-programmet sparas med ett namn som slutar på ".m". Programmet startas från Matlab genom att namnet på filen ni sparat (utan .m) skrivs vid prompten.

Generellt gäller att om en rad avslutas med ;-tecken så fås inte någon utskrift. Utan ;-tecken kommer Matlab att skriva ut resultatet av varje rad. Sätts ett %-tecken i början på en rad görs raden till en kommentar.

För att få hjälp om ett kommando kan man skriva "help < *kommando* >" vid Matlab-prompten. Kommandot "who" visar vilka variabler som finns definierade.

| Kommando | Exempel | Förklaring |
|---|---|---|
| | **Variabler och Matriser** | |
| | `A=[];` | Nollställer variabeln A. |
| | `A=[1 2 7;4 5 6];` | Tilldelar en matris med två rader och tre kolumner till variabeln A |
| | `A(2,1)=4;` | Sätter elementet på rad två kolumn ett till 4. |
| zeros | `A=zeros(3,1);` | Tilldelar A en matris med tre rader och en kolumn innehållande nollor |
| ones | `A=ones(3,1);` | Tilldelar A en matris med tre rader och en kolumn innehållande ettor. |
| | **Programkontroll** | |
| | `a=13;` | |
| while | `while (a<=100)`<br>`  a=a+1;`<br>`end` | Utför while-loop där a ökas med 1 tills dess att a blir 100. |
| for | `for i=1:100`<br>`  a=a+1;`<br>`end` | Utför for-loop där a ökas med ett 100 gånger. |
| if | `if a==100`<br>`  a=a+1;`<br>`else`<br>`  a=a-1;`<br>`end` | Kollar om a är 100, om så är fallet ökas a med 1, annars minskas a med 1 |
| | **Operatorer** | |
| max | `[a,b]=max(x)` | Tar fram största värdet och dess index i vektorn x. Här blir a värdet och b indexet. |
| mod | `a=mod(x,10)` | Här blir a noll om talet x är jämt delbart med tio, annars ett. |
| sum | `a=sum(x)` | a tilldelas summan av alla element i vektorn x. |

| | | |
|---|---|---|
| find | a=find(x>1.5) | a tar värdet på de index där vektorn x har värden som är större än 1.5. Finns fler värden större än 1.5 blir a en vektor av index. |

---

## Grafer

---

| | | |
|---|---|---|
| plot | plot(y) | Ger en graf över värden i vektorn y. |