4 კონტროლის ოპერაციები

პრობლემა: ზეგამტარობა

მაგნიტურ ლევიტაციური ტრანსპორტი მოძრაობს, იმართება და ჩერდება ელექტრომაგნიტური ძალის მოქმედებით. ამ ტიპის ტრანსპორტს მიეკუთვნება სწრაფი მატარებელი (bullet train), რომელსაც შეუძლია განავითაროს სიჩქარე 300 მილი საათში. მატარებელი რელსებიდან დახლოებით 1 დუიმითაა დაცილებული განზიდულობის ძალის გამო, რომელსაც წარმოშობს მატარებელზე დამაგრებული ზეგამტარი მაგნიტი და კოჭა მიწაზე. ელექტრული სენსორები გრძნობენ მატარებლის მოახლოებას და არეგულირებენ ელექტრომაგნიტურ ველს. როცა სენსორებზე მატარებელი არ არის, ენერგია წყდება. ბორბლები შეიკეცება მატარებლის შიგნით, როცა ის მოძრაობს, მაგარმ ისინი დაეშვებიან, როცა მატარებელი უკლებს სიჩქარეს გასაჩერებლად.

საჭიროა ახალი მასალის გამოგონება, რომელიც იქნება უფრო მსუბუქი და უფრო მტკიცე ერთდროულად იმისათვის, რომ მოხდეს მომავლის სატრანსპორტო საშუალებათა სრულყოფა. ამ მიზნით შექმნილი ახალი მასალები გადიან მკაცრ და ხანგრძლივ ტესტირებას. ერთერთი მათგანია ტემპერატურის განაწილების ხასიათი. ამას დიდი მნიშვნელობა ენიჭება, როცა ეს მასალა გამოყენებული უნდა იქნას ძრავის ან მაღალ ტემპერატურული კომპონენტების მახლობლად.

შესავალი

MATLAB –ის პროგრამები, რომლებიც ჩვენ აქამდე დავწერეთ, შეიცავდა მხოლოდ მიმდევრობით საფეხურებს. ერთ ოპერაციას მეორე ცვლიდა ვიდრე არ დასრულდებოდა

საჭირო გამოთვლები. მაგრამ არსებობს მრავალი პრობლემა, სადაც საჭიროა შესარჩევი ბრძანება, რომელიც საშუალებას იძლევა გავუშვათ ბრძანებათა ერთი ციკლი, როცა გარკვეული პირობები სრულდება და ბრძანებათა სხვა ციკლი, როცა ეს პირობა არ სრულდება. ასევე დაგვჭირდება გავიმეოროთ ბრძანებათა გარკვეული ერთობლიობა რამდენიმეჯერ. ასეთი ტიპის ბრძანებებს კონტროლის ოპერაციებს უწოდებენ, რადგან ისინი საშუალებას გვაძლევს ვაკონტროლოთ თუ რომელი ბრძანებების გაშვებაა საჭირო.

if brZaneba

შერჩევის ბრძანება არის პირობითი მაკონტროლებელი ბრძანება, რომელიც საშუალებას იძლევა შემოწმდეს პირობა და ისე განისაზღვროს პროგრამის შემდგომი საფეხურები. შერჩევითი ბრძანების ძირითადი ფორმაა **if** ბრძანება. რამდენადაც იგი იყენებს შედარების და ლოგიკურ ოპერატორებს, განვიხილოთ როგორ ხორციელდება ისინი MATLAB–ში.

შედარების ოპერატორები

MATLAB აქვს შედარების 6 ოპერატორი:

შედარების ოპერატორი	ინტერპრეტაცია
<	ნაკლებია
<=	ნაკლებია ან ტოლია
>	მეტია
>=	მეტია ან ტოლია
==	ტოლია
~=	არ უდრის

მატრიცა და მატრიცული გამოსახულება შეიძლება შეგვხვდეს ამ ოპერატორთა ორივე მხარეს, მაგრამ შეიძლება შევადაროთ მხოლოდ ერთნაირი ზომის მატრიცები. შედეგად მიღებული იგივე ზომის მატრიცის შესაბამისი ელემენტი უდრის 1, თუ შერადების შედეგი ჭეშმარიტია და 0, თუ იგი მცდარია. გამოსახულებას, რომელიც შეიცავს შედარების ოპერატორს ლოგიკური გამოსახულება ეწოდება, რადგან შედეგად იძლევა მატრიცას, რომლის ელემენტების მნიშვნელობებია - 1 (ჭეშმარიტი) და 0 (მცდარი). შედეგად მიღებულ მატრიცას 0-1 მატრიცასაც უწოდებენ.

განვიხილოთ შემდეგი ლოგიკური გამოსახულება:

```
a < b
```

თუ a და b სკალარული სიდიდეებია, მაშინ შედეგად მივიღებთ 1, როცა იგი ჭეშმარიტია და 0, როცა იგი მცდარია. დავუშვათ a და b ვექტორებია:

```
a = [2 4 6]
b = [3 5 1]
```

ასეთ შემთხვევაში a < b ბრძანება მოგვცემს მატრიცას: [1 1 0], ხოლო a ~= b - [1 1 1].
ორი ლოგიკური გამოსახულება შეგვიძლია გავაერთიანოთ ლოგიკური ოპერატორით **და**, **ან**, **არ**. ეს ოპერატორები აღინიშნება სიმბოლოებით:

ლოგიკური ოპერატორი	სიმბოლო
და	&
ან	|
არ	~

ლოგიკური ოპერატორები საშუალებას იძლევა შევადაროთ ერთმანეთს შედარების ოპერატორებით მიღებული მატრიცები, მაგალითად:

```
a < b & b < c
```

ეს ოპერაცია შედეგს მოგვცემს, თუ a < b და b < c შედეგად მიღებული მატრიცები ერთნაირი ზომისაა. შესაბამისად მისი ელემენტები უდრის 1, როცა ორივე პირობა სრულდება, სხვა შემთხვევაში – 0.

როცა ორი ლოგიკური გამოსახულება შეერთებულია **ან (|)** ნიშნით, შედეგად მივიღებთ 0-1 მატრიცას, რომლის ელემენტები ტოლია 1, როცა ერთ-ერთი გამოსახულება მაინცაა ჭეშმარიტი, ხოლო 0-ის ტოლია, როცა ორივე გამოსახულება მცდარია. როცა ორი ლოგიკური გამოსახულება შეერთებულია ნიშნით - **და (&)**, შედეგად მიღებული მატრიცის შესაბამისი ელემენტი 1-ის ტოლია მხოლოდ და მხოლოდ მაშინ, როცა ორივე ლოგიკური გამოსახულება ჭეშმარიტია, სხვა შემთხვევაში 0-ის ტოლია. ცხრილი 4.1 შეიცავს A და B ლოგიკურ გამოსახულებებს შორის ლოგიკურ ოპერატორთა ყველა შესაძლო კომბინაციას:

A	B	~A	A | B	A & B
მცდარი	მცდარი	ჭეშმარიტი	მცდარი	მცდარი
მცდარი	ჭეშმარიტი	ჭეშმარიტი	ჭეშმარიტი	მცდარი
ჭეშმარიტი	მცდარი	მცდარი	ჭეშმარიტი	მცდარი
ჭეშმარიტი	ჭეშმარიტი	მცდარი	ჭეშმარიტი	ჭეშმარიტი

ლოგიკური ოპერატორი აერთიანებს დასრულებულ ლოგიკურ გამოსახულებებს. მაგალითად a > b & b > c ვარგისი გამოსახულებაა მაგრამ a > b & c არ არის მისი ექვივალენტური.

ლოგოკური გამოსახულება შეიძლება იწყებოდეს ~ ნიშნით. ეს ოპერატორი გამოსახულების მნიშვნელობას მისი საპირისპიროთი ცვლის. თუ a > b ჭეშმარიტია, მაშინ ~ (a > b) – მცდარია.

ლოგიკური გამოსახულება შესაძლოა შეიცავდეს რამდენიმე ლოგიკურ ოპერატორს:

```
~ (b == c | b == 5.5)
```

ლოგიკური ოპერატორების იერარქია (ზემოდან ქვემოთ) ასეთია: არ, და, ან. იერარქიის შესაცვლელად ფრჩხილები უნდა გამოვიყენოთ. ზემოთ მოცემულ გამოსახულებაში პირველ რიგში შესრულდება b == c და b == 5.5. დავუშვათ b = 3 c = 5. რადგან არცერთი გამოსახულება არ არის ჭეშმარიტი, გამოსახულება - b == c | b == 5.5 მცდარია, ფრჩხილების წინ მდგარი ~ ოპერატორი კი მას საპირისპირო მნიშვნელობას – ჭეშმარიტს მიანიჭებს. დავუშვათ ფრჩხილები არ გვაქვს ამ გამოსახულებაში:

~ b == c | b == 5.5

ამ შემთხვევაში გამოსახულება ~ b == c შეფასდება b == 5.5 გამოსახულებასთან ერთად . b და c მოცემულ მნიშვნელობათათვის ორივე გამოსახულება მცდარია, მთლიანად გამოსახულების მნიშვნელობაც მცდარია. შეიძლება გაგიჩნდათ კითხვა, როგორ შევაფასეთ ~b, როცა b თვითონ რიცხვს წარმოადგენს. MATLAB –ში ყველა არანულოვანი მნიშვნელობა შეფასდება როგორც ჭეშმარიტი, ხოლო ნოლოვანი მნიშვნელობა შეფასდება როგორც მცდარი. ამიტომ დიდი სიფრთხილეა საჭირო შედარების და ლოგიკური ოპერატორებით სარგებლობისას, რათა პროგრამის ყველა საფეხური ისე შესრულდეს, როგორც ამას ჩვენი ამოცანა მოითხოვს.

სავარჯიშო

განსაზღვრეთ ჭეშმარიტია თუ მცდარი შემდეგი გამოსახულებები. შეამოწმეთ პასუხები MATLAB საშუალებით. გაიხსენეთ, რომ პასუხის შესამოწმებლად საჭიროა ცვლადებისათვის მნიშვნელობათა მინიჭება და შემდეგ გამოსახულების შეყვანა კლავიატურიდან ბრძანებათა ფანჯარაში:

a = 5.5; b = 1.5; k = -3;

1. a < 10.0
2. a + b >= 6.5
3. k ~= 0
4. b – k > a
5. ~ (a == 3* b)
6. – k <= k + 6
7. a < 10 & a > 5
8. abs (k) > 3 | k < b – a

მარტივი if ბრძანება

პროგრამაში **if** ბრძანება შეიძლება გამოვიყენოთ მარტივი ფორმით, მხოლოდ if ბრძანება, ანდა მისი რთული ფორმა, რომელიც გარდა **if** ბრძანებისა, შეიცავს **else** და **elseif** ოპერატორებს. განვიხილოთ მარტივი ფორმა ამ ბრძანებისა, რომელსაც აქვს ასეთი სახე:

```
if ლოგიკური გამოსახულება
    ბრძანებათა ჯგუფი A
end
```

თუ ლოგიკური გამოსახულება ჭეშმარიტია, სრულდება ბრძანებები **if** და **end** შორის, ხოლო თუ ლოგიკური გამოსახულება მცდარია, ბრძანებათა ჯგუფი A გამოტოვებული იქნება და პროგრამა გადავა იმ ბრძანებათა შესრულებაზე, რომელიც **end** –ს მოჰყვება. პროგრამის დაწერისას იმისათვის, რომ გამოიკვეთოს **if** ბრძანებით გათვალისწინებული საფეხური, უკეთესია ბრძანებათა ეს ჯგუფი აბზაცით დაიწეროს. მაგალითად:

```
if a < 50
    count = count + 1;
    sum = sum +a;
end
```

დავუშვათ a სკალარია. თუ a < 50, ცვლადი count გაიზრდება ერთით და sum ცვლადის მნიშვნელობას დაემატება a მნიშვნელობა, თუ არა, პროგრამა გამოტოვებს ამ ორ ბრძანებას. თუ a სკალარი არ არის, ორივე ბრძანება შესრულდება მხოლოდ მაშინ, თუ a-ს ყველა ელემენტი < 50.

if ბრძანება შეიძლება იყოს ერთმანეთში ჩასმული:

```
if ლოგიკური გამოსახულება 1
    ბრძანებათა ჯგუფი A
    if ლოგიკური გამოსახულება 2
       ბრძანებათა ჯგუფი B
    end
    ბრძანებათა ჯგუფი C
end
ბრძანებათა ჯგუფი D
```

თუ 'ლოგიკური გამოსახულება 1' ჭეშმარიტია, შესრულდება ბრძანებათა ჯგუფი A და C, თუ 'ლოგიკური გამოსახულება 2' ასევე ჭეშმარიტია, შესრულდება ბრძანებათა ჯგუფი B-ც, ვიდრე შესრულდება ბრძანებათა ჯგუფი C. თუ 'ლოგიკური გამოსახულება 1' მცდარია, პროგრამა პირდაპირ გადადის ბრძანებათა D ჯგუფზე. აბზაცების გამოყენება **if** ბრძანების ასეთ სტრუქტურაში ძალზე მნიშვნელოვანია.
განვიხილოთ მაგალითი, რომელიც შეიცავს ერთმანეთში ჩასმულ **if** ბრძანებებს.

```
if a < 50
    count = count + 1;
    sum = sum +a;
    if b > a
       b = 0;
    end
end
```

დავუშვათ a და b სკალარებია, მაშინ როცა a < 50, ცვლადი count გაიზრდება 1 –ით და ცვლადს sum დაემატება a. გარდა ამისა, თუ b > a მაშინ b მიენიჭება მნიშვნელობა 0. თუ a არ არის 50-ზე ნაკლები, მაშინ პროგრამა პირდაპირ გადადის ბრძანებაზე, რომელიც მოჰყვება მეორე end –ს. თუ a სკალარი არ არის, მაშინ პირობა a < 50 მხოლოდ მაშინაა ჭეშმარიტი, როცა იგი სრულდება a-ს ყველა მნიშვნელობისათვის. თუ არც a და არც b არ არის სკალარი, მაშინ b მეტია a-ზე მხოლოდ მაშინ, როცა b ყველა ელემენტი მეტია a შესაბამის ელემენტზე. თუ a ან b სკალარია, მაშინ მატრიცის ელემენტები შედარდება სათითაოდ სკალარს და ისე შემოწმდება პირობა.

შედარების და ლოგიკური ფუნქციები

MATLAB აქვს შედარების და ლოგიკური ფუნქციები, რომლებიც გამოიყენება **if** ბრძანების სტრუქტურაში.

any (x)	x მატრიცის ყოველი სვეტისათვის გვაძლევს 1, თუ ამ სვეტის რომელიმე ელემენტი მაინც არის არანულოვანი, სხვა შემთხვევაში გვაძლევს 0
all (x)	x მატრიცის-ის ყოველი სვეტისათვის გვაძლევს 1, თუ ამ სვეტის ყველა ელემენტი არანულოვანია, სხვა შემთხვევაში გვაძლევს 0.
find (x)	გვაძლევს ვექტორს, რომელიც შეიცავს x ვექტორის არანულოვანი ელემენტების ინდექსებს. თუ x მატრიცაა, მაშინ იდექსები შეირჩევა x(:) ვექტორიდან. ეს არის ერთი გრძელი ვექტორი, რომელიც შედგება x მატრიცის ერთმანეთს მიყოლებული სვეტების ელემენტებისაგან.
exist ('A')	გვაძლევს 1, თუ მოცემულ სამუშაო სივრცეში არსებობს A ცვლადი , 2 თუ მოცემულ სამუშაო სივცეში არსებობს A ცვლადი ან A.m ფაილი, გვაძლევს 0 თუ ასეთი არც ცვლადი და არც .m ფაილი არ არსებობს. (ყურადღება მიაქციეთ: სახელი ბრჭყალებში უნდა იყოს ჩასმული)
isnan(x)	გვაძლევს მატრიცას, რომლის ელემენტები უდრის 1, როცა x მატრიცის შესაბამისი ელემენტები წარმოადგენს განუზღვრელობას, სხვა შემთხვევაში - 0.
finite (x)	გვაძლევს მატრიცას, რომლის ელემენტები ტოლია 1, როცა

	x მატრიცის შესაბამისი ელემენტები სასრული რიცხვებია, სხვა შემთხვევაში – 0.
isepty (x)	გვაძლევს 1, თუ x არის ცარიელი მატრიცა, თუ არა - 0.
isstr (x)	გვაძლევს 1, თუ x სიმბოლოებისგან შემდგარი სტრიქონია, 0 - თუ ეს ასე არაა
strcmp(y1,y2)	ერთმანეთს შეადარებს ორ სტრიქონს y1 და y2 და გვაძლევს 1, თუ ისინი იდენტურია, 0 – თუ ისინი განსხვავებულია.

დავუშვათ A სამსტრიქონიანი და სამსვეტიანი მატრიცაა. განვიხილოთ შემდეგი ბრძანება:

```
if all (A) == 0
     disp ('A contains all zeros')
end
```

სტრიქონი A contains all zeros დაიბეჭდება მხოლოდ იმ შემთხვევაში, როცა A ყველა ელემენტის მნიშვნელობა არის 0 –ის ტოლია.

სავარჯიშო

განსაზღვრეთ ყოველი გამოსახულების მნიშვნელობა. შეამოწმეთ პასუხები MATLAB საშუალებით. დავუშვათ მატრიცა B –ს აქვს მნიშვნელობა:

$$B = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 0 & 3 \\ 8 & 7 & 0 \end{bmatrix}$$

1. any (B)
2. find (B)
3. all (any (B))
4. any (all (B))
5. finite (B(:,3))
6. any (B(1:2,1:3))

else ოპერატორი

ეს ოპერატორი საშუალებას გვაძლევს გავუშვათ ბრძანებათა ერთი ჯგუფი, როცა ლოგიკური გამოსახულება ჭეშმარიტია, და ბრძანებათა სხვა ჯგუფი, როცა იგივე გამოსახულება მცდარია. **else** ოპერაორის შემცველი if ბრძანების ზოგადი ფორმაა:

```
if  ლოგიკური გამოსახულება
    ბრძანებათა ჯგუფი A
else
    ბრძანებათა ჯგუფი B
end
```

თუ 'ლოგიკური გამოსახულება' ჭეშმარიტია, სრულდება ბრძანებათა ჯგუფი A, თუ მცდარია, სრულდება ბრძანებათა ჯგუფი B. **else** შეიძლება გამოვიყენოთ ერთმანეთში ჩასმულ **if** ბრძანებებშიც.

საილუსტრაციოდ განვიხილოთ მაგალითი:

ბაგირგზა აკავშირებს ორ კოშკს. დავუშვათ სკალარი d წარმოადგენს მანძილის მნიშვნელობას ვაგონიდან უახლოეს კოშკამდე. თუ ვაგონი კოშკიდან 30-მდე ფუტითაა დაშორებული, სიჩქარე გამოითვლება ფორმულით:

$$velocity = 0.425 + 0.00175d^2$$

თუ ვაგონი კოშკიდან უფრო შორსაა, ვიყენებთ ფორმულას:

$$velocity = 0.625 + 0.12d + 0.00025d^2$$

სიჩქარის მნიშვნელობები შეგვიძლია გამოვითვალოთ შემდეგი ბრძანების საშუალებით:

```
if d < 30
   velocity = 0.425 + 0.00175*d^2;
   else
   velocity = 0.625 + 0.12*d + 0.00025*d^2;
end
```

elseif ოპერატორი

elseif ოპერატორს მივმართავთ, იმისათვის რომ შევამოწმოთ რამდენიმე ლოგიკური გამოსახულება და გამოთვლები საჭირო მიმართულებით წარვმართოთ:

```
if   ლოგიკური გამოსახულება 1
       ბრძანებათა ჯგუფი A
elseif ლოგიკური გამოსახულება 2
       ბრძანებათა ჯგუფი B
elseif ლოგიკური გამოსახულება 3
       ბრძანებათა ჯგუფი C
end
```

გვაქვს ორი **elseif** ოპერატორი, შეგვიძლია უფრო მეტიც გამოვიყენოთ. თუ 'ლოგიკური გამოსახულება 1' ჭეშმარიტია, მხოლოდ A ჯგუფის ბრძანებები სრულდება. თუ 'ლოგიკური გამოსახულება 1' მცდარია და 'ლოგიკური გამოსახულება 2' ჭეშმარიტია, მხოლოდ ბრძანებათა ჯგუფი B სრულდება, თუ 'ლოგიკური გამოსახულება 1' და 'ლოგიკური გამოსახულება 2' მცდარია და 'ლოგიკური გამოსახულება 3' ჭეშმარიტია, მხოლოდ ბრძანებათა ჯგუფი C სრულდება. თუ ერთზე მეტი ლოგიკური გამოსახულებაა ჭეშმარიტი, პირველი ჭეშმარიტი გამოსახულება განსაზღვრავს, თუ ბრძანებათა რომელი ჯგუფი უნდა შესრულდეს. თუ არც ერთი ლოგიკური გამოსახულება არ არის ჭეშმარიტი, **if** სტრუქტურის არცერთი ბრძანება არ შესრულდება.

შეიძლება **if** სტრუქტურაში **else** და **elseif** ოპერატორები ერთდროულად გამოვიყენოთ:

```
if   ლოგიკური გამოსახულება 1
       ბრძანებათა ჯგუფი A
elseif ლოგიკური გამოსახულება 2
       ბრძანებათა ჯგუფი B
elseif ლოგიკური გამოსახულება 3
```

```
    ბრძანებათა ჯგუფი C
else
    ბრძანებათა ჯგუფი D
end
```

თუ არცერთი ლოგიკური გამისახულება არ არის ჭეშმარიტი, მაშინ შესრულდება ბრძანებათა ჯგუფი D. if-elseif სტრუქტურას ხშირად ამომრჩევ სტრუქტურასაც უწოდებენ, რადგანაც მოწმდება რამდენიმე პირობა.

სავარჯიშო

დაწერეთ MATLAB ბრძანებების მწკრივი, რომ შესრულდეს განსაზღვრული საფეხურები, დავუშვათ ცვლადები საკალარული სიდიდეებია.

1. თუ დრო მეტია, ვიდრე 50, დაუმატე მას 1.
2. როცა კვადრატული ფესვი ცვლადიდან poly ნაკლებია ვიდრე 0.001, დაბეჭდე მისი მნიშვნელობა
3. თუ სხვაობა volt_1 და volt_2 შორის მეტია, ვიდრე 2.0, დაბეჭდე მათი მნიშვნელობები
4. თუ den მნიშვნელობა ნაკლებია ვიდრე 0.003, მიანიჭეთ result –ს 0-ის ტოილ მნიშვნელობა, თუ არა, მიანიჭეთ result-ს მნიშვნელობა num/den
5. თუ x-ის ლოგარითმი ნატურალური ფუძით მეტია, ან ტოლია 10-ის, მიანიჭეთ time –ს 0-ის ტოილ მნიშვნელობა და გაზარდეთ count მნიშვნელობა
6. თუ dist ნაკლებია 50, და time მეტია 10, გაზარდეთ time მნიშვნელობა 2-ით, თუ არა, გაზარდეთ time 5-ით
7. თუ dist მეტია ან ტოლია 100, გაზარდეთ time 10-ით, თუ dist მოთავსებულია 50 და 100 შორის, გაზარდეთ time 1-ით. სხვა სემთხვევაში გაზარდეთ time 0.5-ით

for cikli

ციკლისათვის MATLAB აქვს ორი ოპერატორი **for** და **while**. ჯერ განვიხილავთ ციკლის ოპერატორს **- for**.

მას აქვს ზოგადი სახე:

```
for index = გამოსახულება
    ბრძანებათა ჯგუფი A
end
```

გამოსახულება შეიძლება იყოს მატრიცა, ვექტორი ან სკალარი. ბრძანებათა ჯგუფი სრულდება იმდენჯერ, რამდენი სვეტიცაა მატრიცაში. ინდექსი მიმდევრობით ღებულობს მატრიცის ელემენტების მნიშვნელობებს. **for** ოპერატორის გაცნობამდე განვიხილოთ მაგალითი:

დავუშვათ გვაქვს ვექტორი, რომელიც შეიცავს მანძილების მნიშვნელობებს, რომლებიც შეესაბამება საბაგირო გზის ვაგონის დაშორებას უახლოესი კოშკიდან. გვინდა მივიღოთ ვექტორი, რომელიც შეიცავს აჩქარების შესაბამის მნიშვნელობებს. თუ მანძილი ნაკლებია 30 ფუტზე, უნდა ვისრგებლოთ ფორმულით:

$$velocity = 0.425 + 0.00175d^2$$

თუ ვაგონი კოშკიდან უფრო შორსაა, ვიყენებთ ფორმულას:

$$velocity = 0.625 + 0.12d + 0.00025d^2$$

სიჩქარის ფორმულის არჩევა დამოკიდებულია d მაძილის მნიშვნელობაზე. იმისათვის, რომ d თითოეული მნიშვნელობისათვის ცალკ-ცალკე არ გამოვითვალოთ სიჩქარეები, ვიყენებთ **for** ოპერატორს:

დავუშვათ d ვექტორი შეიცავს 25 ელემენტს.

```
for k = 1:25
    if d(k) <= 30
        velocity = 0.425 + 0.00175*d(k)^2;
    else
        velocity = 0.625 + 0.12*d(k) + 0.00025*d(k)^2;
    end
end
```

შემდეგ ამოხსნაში დავუშვებთ, რომ d ვექტორის ზომა უცნობია. ამიტომ ვისრგებლებთ **length** ბრძანებით.

```
for k = 1:length(d)
    if d(k) <= 30
        velocity = 0.425 + 0.00175*d(k)^2;
    else
        velocity = 0.625 + 0.12*d(k) + 0.00025*d(k)^2;
    end
end
```

for ოპერატორით სარგებლობისას უნდა დავიცვათ შემდეგი წესები:

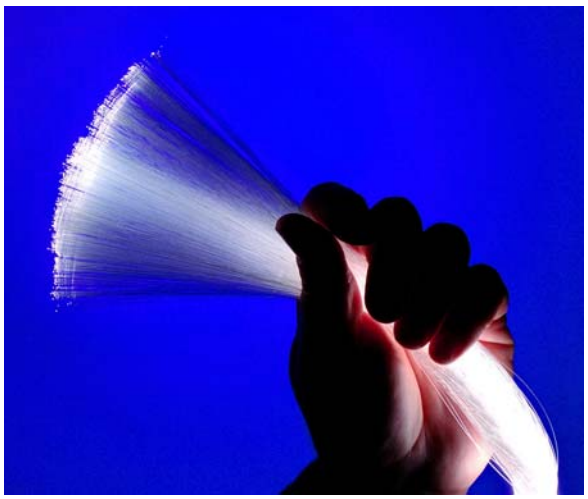
1. **for** ოპერატორის ინდექსი უნდა იყოს ცვლადი
2. თუ გამოსახულება ცარიელი მატრიცაა, ციკლის გაშვება არ მოხდება, პროგრამის კონტროლი გამოტოვებს ყველა ბრძანებას, რომელიც მოთავსებულია **for** და **end** შორის.
3. თუ გამოსახულება სკალარია, ციკლი მხოლოდ ერთხელ შესრულდება
4. თუ გამოსახულება სტრიქონი ვექტორია, ინდექსი მორიგეობით ღებულობს ვექტორის ელემენტების მნიშვნელობებს.
5. თუ გამოსახულება მატრიცაა, ინდექსი მიიღებს მნიშვნელობებს ჯერ პირველი სვეტის ელემენტებისას, შემდეგ მეორე სვეტის და ა. შ.
6. ციკლის ოპერატორის დასრულებისას ინდექსი ინარჩუნებს საბოლოოდ მიღებულ მნიშვნელობას
7. თუ გამოსახულების მატრიცის მისაღებად გამოყენებულია ორწერტილიანი ოპერატორი
   for k = საწყისი მნიშვნელობა : ნაზრდი : საბოლოო მნიშვნელობა
   მაშინ იმის დასადგენად, თუ რამდენჯერ შესრულდება ციკის ოპერატორით განსაზღვრული მოქმედებები, ვისარგებლებთ ფორმულით:
   floor((საბოლოო მნიშვნელობა – საწყისი მნიშვნელობა)/ნაზრდი)+1
   თუ ეს სიდიდე უარყოფითია, ციკლის გაშვება არ მოხდება.

განსაზღვრეთ რამდენჯერ შესრულდება ციკლის ოპერატორით განსაზღვრული მოქმედება თუ ინდექსის გამოსახულება შემდეგი სახისაა. შეამოწმეთ პასუხი MATLAB საშუალებით.

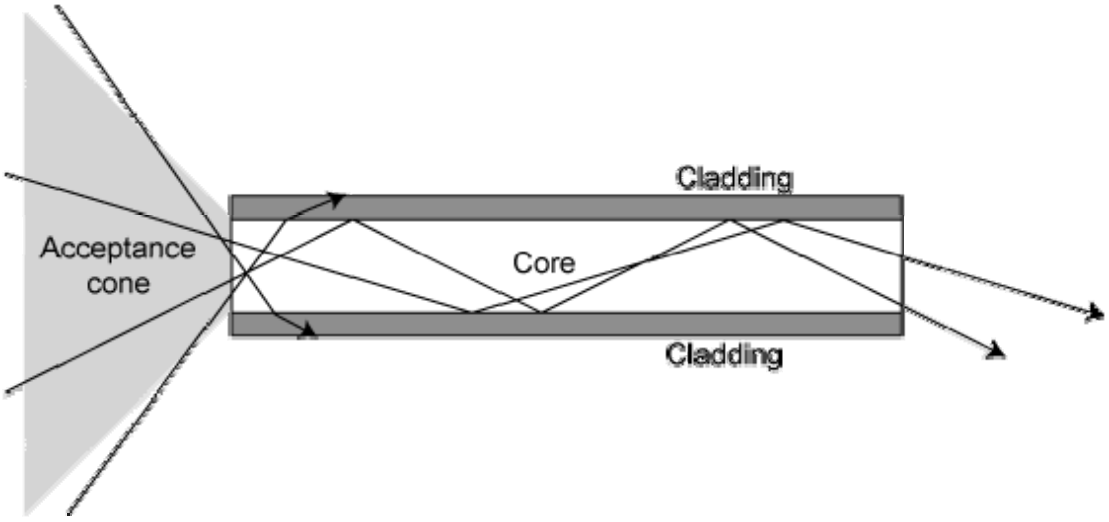
9. for k = 3:20
10. for count = -2:14
11. for k = -2:-1:-10
12. for time = 10:5
13. for index = 52:-12

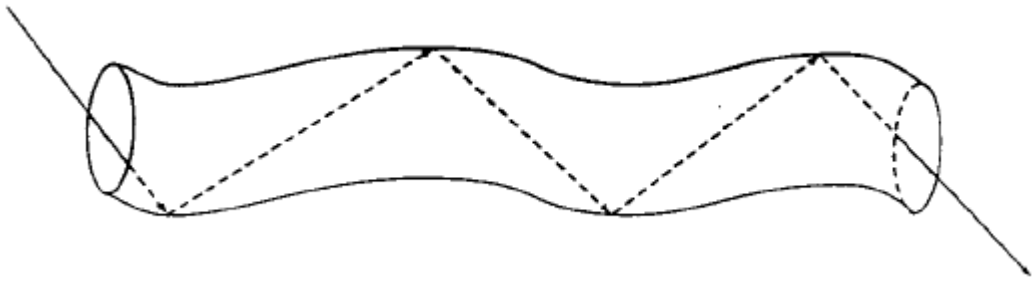
ციკლი შეგვიძლია შევწყვიტოთ ბრძანებით: **break.** ეს ბრძანება აუცილებლად დაგვჭირდება, თუ ციკლის ოპერატორში პირობა შეცდომითაა განსაზღვრული.

პრობლემა: ოპტიკური ბოჭკო



ნახ. 4.1 ოპტიკურ ბოჭკოთა კონა





ნახ. 4.2 ოპტიკური ბოჭკო

თუ სინათლე ეცემა მინის ან პლასტიკის გრძელ ღეროს, იგი მრავალგზის აირეკლება ღეროს კედლებიდან ვიდრე მეორე ბოლოს მოაღწევს. ეს საინტერესო ფენომენი შეიძლება გამოყენებული იქნას სინათლის ან სულაც გამონასახის გადასაცემად. თუ ავიღებთ ასეთი ღეროების კონას, სინათლე გავრცელდება მასში და მიაღწევს მეორე ბოლოს, როგორც ეს ნახ. 4.2 მოცემული.

ოპტიკური ბოჭკო არის ძალიან წვრილი მინის ან პლასტიკის ბოჭკო. ასეთი ბოჭკოების კონა შეიძლება გამოვიყენოთ როგორც სინათლის გადაცემის საშუალება. ოპტიკური ბოჭკო გამონასახის გადაცემის საშუალებასაც იძლევა, თუ ბოლოებზე სპეციალურ მოწყობილობას მივუერთებთ. რადგან ოპტიკური ბოჭკო ძალზე მოქნილია, ის შეგვიძლია გამოვიყენოთ როგორც გამონასახის გადაცემის საშუალება ისეთ ადგილებში, რომელიც სხვა ხელსაწყოებისათვის მიუწვდომელია, მაგალითად ენდოსკოპში. ენდოსკოპი ხელსაწყოა, რომელიც გამოიყენება პაციენტის ორგანიზმის გამოსაკვლევად. ორგანიზმში შეღწევა ხდება უმცირესი ჭრილობის საშუალებით. ოპტიკური ბოჭკო ლაზერული სხივის გადასაცემადაც გამოიყენება, რომელიც გამოიყენება არტერიების გასათავისუფლებად, თირკმელში ქვების დასამსხვრევად, კატარაქტის მოსაცილებლად და სხვა.

ბოჭკოებში შიდა არეკვლის ფენომენი აიხსნება შნელის კანონით. ოპტიკური ბოჭკო შედგება ორგვარი მასალისაგან – მასალა, რომლისგანაც შედგება ბოჭკო და მასალა, რომელიც გარს აკრავს მას. ჩვეულებრივ ბოჭკოს შემადგენელი მასალა უფრო მკვრივია, ვიდრე გარსი. როცა სინათლე ერთი გარემოდან განსხვავებული სიმკვრივის მქონე მეორე გარემოში გადადის, იგი გარდატყდება გამყოფ ზედაპირზე. გარდატეხის კუთხე დამოკიდებულია გარემოს გარდატეხის კოეფიციენტზე და დაცემის კუთხეზე. როცა სინათლე უფრო მკვრივიდან ნაკლებ მკვრივ გარემოში გადადის, დაეცემა რა ზედაპის მისი ნაწილი აირეკლება, ნაწილი კი გააღწევს მეორე გარემოში. სინათლის დაცემის კუთხეს, როცა იგი მთლიანად აირეკლება ზედაპირიდან კრიტიკული კუთხე ეწოდება. რადგანაც კრიტიკული კუთხე დამოკიდებულია ერთი გარემოს მეორის მიმართ გარდატეხის კოეფიციენტზე შეგვიძლია გამოვითვალოთ ეს კუთხე და განვსაზღვროთ მოცემული კუთხით დაცემული სინათლე გაივლის თუ არა ბოჭკოში. დავუშვათ n_2 გარემომცველი გარემოს გარდატეხის კოეფიციენტია, ხოლო n_1 თავად ბოჭკოსი, თუ $n_2 > n_1$ ბოჭკო სინათლეს არ გაატარებს. კრიტიკული კუთხე გამოითვლება ფორმულით:

$$\sin\theta_c = \frac{n_2}{n_1}$$

დაწერეთ MATLAB პროგრამა, რომელიც განსაზღვრავს გაივლის თუ არა სინათლე ოპტიკურ ბოჭკოში, რომელიც გარემოცულია განსხვავებული სიმკვრივის მქონე მასალით. დავუშვათ, გვაქვს ASCII მონაცემთა ფაილი სახელით indicies.dat, რომელიც შეიცავს

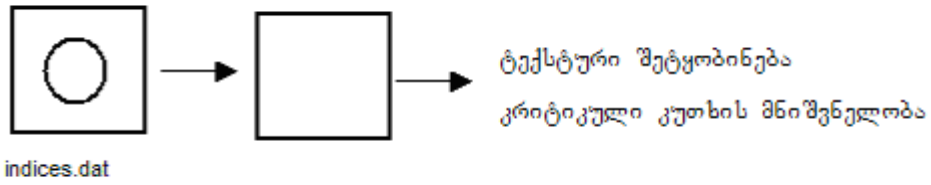
მონაცემებს ოპტიკური ბოჭკოსა და გარემომცველი გარემოს შესახებ. ფაილის ყოველი სტრიქონი შეიცავს ბოჭკოსა და გარსის მასალის გარდატეხის კოეფიციენტს. პროგრამამ უნდა განსაზღვროს გაატარებს თუ არა სინათლეს ასეთი მონაცემების მქონე გამტარი და თუ გაატარებს, როგორია კრიტიკული კუთხის მნიშვნელობა.

1. ამოცანის დასმა

განვსაზღვროთ, გაივლის თუ არა სინათლე მოცემულ გარემოში. განვსაზღვროთ დაცემის კუთხე, როცა სინათლე გაივლის ბოჭკოში.

2. INPUT/OUTPUT აღწერა

როგორც ნაჩვენებია 4.2 სურათზე, პროგრამის შესავალი მნიშვნელობაა მონაცემთა ფაილი, რომელიც შეიცავს პოტენციური სინათლის გამტარის გარდატეხის კოეფიციენტებს. გამოსავალი მნიშვნელობა კი ცნობა იმის შესახებ გაივლის თუ არა სინათლე მოცემული პარამეტრების მქონე გამტარში და კრიტიკული კუთხე.



ნახ. 4.3 I UT/OUTPUT დიაგრამა

3. სახელდახელო ამოხსნა

ჰაერის გარდატეხის კოეფიციენტია 1.0003, მინის – 1.5. თუ შევქმნით სინათლის გამტარს ჰაერით გარემოცული მინის ბოჭკოთი, კრიტიკული კუთხე შეიძლება გამოვითვალოთ ფორმულით:

$$\theta_c = \sin^{-1}\frac{n_2}{n_1} = \sin^{-1}\frac{1.0003}{1.5} = \sin^{-1}(0.66687) = 41.82^o$$

ასეთი გამტრი გაატარებს სინათლეს, რომელიც დაეცემა 41.82^o-ზე მეტი კუთხით.

4. MATLAB ამოხსნა

```
%
%     This program reads the indices of  refraction
%     for materials forming a light pipe. For each
%     pair of materials, we determine if the pipe
%     will transmit light, and at what angles of
%     incidence in degrees.
%
factor = 180/pi;
```

```
load indices.dat
n1 = indices(:,1);
n2 = indices(:,2);
for k = 1: length(n1)
    if  n2(k) > n1(k)
        fprintf('light is not transmitted for \n')
        fprintf('rod  index  %g  and  medium  index  %g  \n\n',
            n1(k), n2(k))
    else
        critical_angle = asin(n2(k)/n1(k))*factor;
        fprintf('light is not transmitted for \n')
        fprintf('rod  index  %g  and  medium  index  %g  \n\n',
            n1(k), n2(k))
        fprintf('for angles greater then %g  degrees \n\n',…
            critical_angle)
    end
end
```

5. შემოწმება

შევამოწმოთ ეს ამოცანა ფაილით რომლის შემცველობაა:

1.31	1.473
1.5	1.0003
1.49	1.33

მივიღებთ:

```
light is not transmitted for
rod index 1.31 and medium index 1.473
light is transmitted for
rod index 1.5 and medium index 1.0003
for angles greater then 41.8257  degrees
light is transmitted for
rod index 1.49 and medium index 1.33
for angles greater then 63.204  degrees
```

სცადეთ იპოვოთ მასალები, რომელთათვისაც დაცემის კრიტიკული კუთხე 45 მეტია.

while cikli

ეს ბრძანება მნიშვნელოვანია, როცა გვსურს პროგრამაში მოქმედებები განმეორდეს ვიდრე გარკვეული პირობა სრულდება. მისი ზოგადი ფორმაა:

while გამოსახულება
    ბრძანებათა ჯგუფი A
end

თუ გამოსახულება ჭეშმარიტია, A ჯგუფის ბრძანებები სრულდება, ამის შემდეგ მოწმდება პირობა ხელახლა. თუ პირობა კვლავ ჭეშმარიტია, ისევ სრულდება ბრძანებათა A ჯგუფი და ასე გრძელდება, ვიდრე პირობა არ გახდება მცდარი. ამის შემდეგ პროგრამა გადადის **end** –ის მომდევნო ბრძანებაზე. ცვლადები, რომლებიც მონაწილეობენ ბრძანებათა ჯგუფში, უნდა შეიცავდნენ გამოსახულებაში მონაწილე ცვლადებს. თუ ეს ასე არ არის,

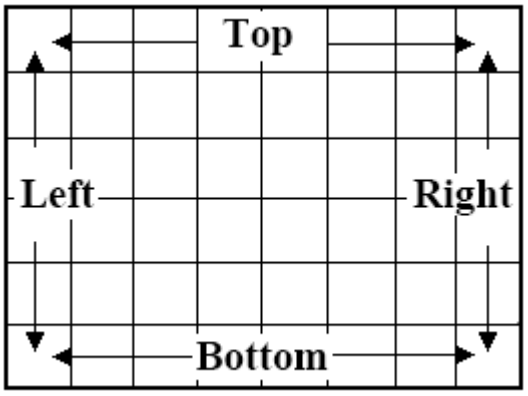
გამოსახულების მნიშვნელობა არ შეიცვლება. თუ გამოსახულება ყოველთვის ჭეშმარიტი რჩება(ან არის არანულოვანი რიცხვი), ციკლი უსასრულოდ გრძელდება. (გაიხსენეთ, რომ ციკლის შეწყვეტა შეგიძლიათ ბრძანებით **^c (ctrl+c).**

while ციკლის საილუსტრაციოდ განვიხილოთ ბრძანება, რომელიც ვექტორის ელემენტებს უმატებს ცვლადს sum, ვიდრე უარყოფითი რიცხვი არ შეხვდება.

```
sum = 0;
k = 1;
while x(k) >= 0 & k <= length(x)
    sum = sum + x(k);
    k = k + 1;
end
```

პრობლემა – ტემპერატურული წონასწორობა

ახალი მასალების გამოგონება და გაუმჯობესება საჭიროებს მათ გამოცდას მრავალი პარამეტრის მიხედვით. ერთ-ერთი მათგანია ტემპერატურის განაწილება. ამ ამოცანაში განვიხილავთ ტემპერატურის განაწილებას მეტალის თხელ ფირფიტაზე. ფირფიტის მასალა შექმნილია ისეთი მოწყობილობისათვის, რომელშიც მისმა ოთხივე კიდემ უნდა შეინარჩუნოს მუდმივი, ან იზოთერმული ტემპერატურა. ფირფიტის სხვა წერტილებში ტემპერატურა გამოითვლება მეზობელი წერტილების ტემპერატურის მიხედვით. წარმოვიდგინოთ, რომ ფირფიტა დაფარულია ბადით, მაშინ ფირფიტა შეგვიძლია განვიხილოთ, როგორც მატრიცა, რომლის ელემენტები წარმოადგენს ტემპერატურას შესაბამის უჯრედში. ნახ. 4.4 წარმოადგენს ამგვარი ბადით დაფარულ ფირფიტას. ჩავთვალოთ, რომ ბადის უჯრედის შიგნით ტემპერატურა მუდმივია. ფირფიტა დაყოფილია 6 × 9 უჯრედად. სულ ტემპერატურის 48 მნიშვნელობა გვაქვს მოცემული მატრიცის სახით.



ნახ. 4.4 მეტალის ფირფიტის დაყოფა

მოცემულია იზოთერმული ტემპერატურა ფირფიტის ოთხივე გვერდისთვის; დავუშვათ ფირფიტის გვერდითი და ზედა კიდეები ერთნაირი ტემპერატურისაა, ხოლო ქვედა კიდე განსხვავებულია. იგულისხმება, რომ დასაწყისისათვის ფირფიტის სხვა უჯრედების

ტემპერატურა 0-ის ტოლია. ყოველი შიდა წერტილისათვის ახალი ტემპერატურა გამოითვლება როგორც მისი მეზობელი წერტილების საშუალო შემდეგი განტოლებით:

	T_1	
T_4	T_0	T_2
	T_3	

$$T_0 = \frac{T_1 + T_2 + T_3 + T_4}{4}$$

შიდა წერტილებისათვის ახალი ტემპერატურების გამოთვლის შემდეგ შეფასდება სხვაობა ტემპერატურის ძველსა და ახალ მნიშვნელობას შორის. თუ ტემპერატურის ცვლილება მეტია, ვიდრე წინასწარ განსაზღვრულ სიდიდეზე – ფირფიტა ჯერ კიდევ არ არის თერმულ წონასწორობაში და აღწერილი პროცესი ისევ განმეორდება.

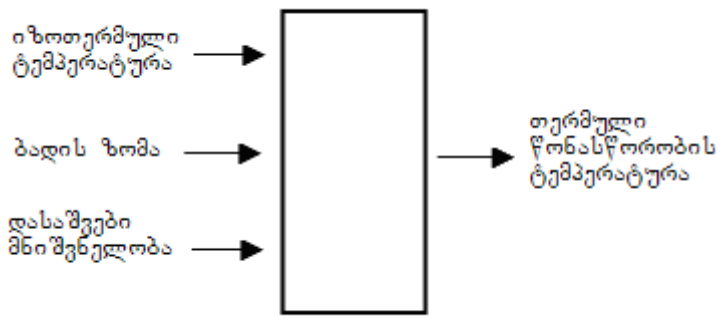
ვსარგებლობთ ორი მატრიცით ტემპერატურებისათვის; ერთი ძველი მნიშვნელობებისათვის, მეორე ახლად გამოთვლილი ტემპერატურისათვის. ორი მატრიცა იმისათვის გვჭირდება, რომ ვუშვებთ: ტემპერატურის ცვლილება ყველა ელემენტისათვის ერთდროულად ხდება. მაგალითად ვითვლით ტემპერატურას წერტილში (3,3). ახალი მნიშვნელობა გამოითვლება როგორც საშუალო მისი მეზობელი მნიშვნელობებისა (2,3), (3,2), (3,4) და (4,3). შემდეგ ვითვლით ტემპერატურას მდებარეობისათვის (3,4). ისევ ვითვლით საშუალოს, მაგრამ ჩვენ გვჭირდება (3,3) მდებარეობის ძველი და არა ახლად გამოთვლილი მნიშვნელობა, ამიტომ საჭიროა ტემპერატურის ახლადგამოთვლილი მნიშვნელობები ჩავწეროთ ახალი მატრიცის სახით.

ამრიგად, ვიყენებთ მატრიცას ძველი ტემპერატურებით იმისათვის, რომ გამოვითვალოთ ახალი ტემპერატურები და ვამოწმებთ თუნდაც ერთი სხვაობა ხომ არ აჭარბებს დასაშვებ მნიშვნელობას. როცა ტემპერატურის ცვლილება ფირფიტის ყოველი უჯრედისათვის, ე. ი ნებისმიერი სხვაობა ახალი და ძველი მატრიცის შესაბამის ელემენტებს შორის გახდება დასაშვებ მნიშვნელობაზე ნაკლები, ჩავთვლით, რომ თერმული წონასწორობა დამყარდა.

1. ამოცანის დასმა

განვსაზღვროთ თერმული წონასწორობის შესაბამისი სიდიდეები მეტალის ფირფიტისათვის იზოთერმული კიდეებით.

2. INPUT/OUTPUT აღწერა



ნახ. 4.5 I/O დიაგრამა

როგორც ნახაზზეა ნაჩვენები, პროგრამის შესავალი მნიშვნელობაა ფირფიტის ბადის ზომა, იზოთერმული ტემპერატურა და დასაშვები მნიშვნელობა.

3. სახელდახელო ამოხსნა

რომ დავრწმუნდეთ რამდენად კარგად გავიგეთ აღწერილი პროცესი, ამოვხსნათ პრობლება მარტივი შემთხვევისათვის. დავუშვათ მატრიცა შეიცავს 4 სტრიქონსა და 4 სვეტს. გვერდითი და ზედა კიდეების იზოთერმული ტემპერატურა 100 გრადუსია, ხოლო ქვედა კიდისათვის – 50 გრადუსი. დასაშვები სიდიდე ავიღოთ 40 გრადუსის ტოლი. ფირფიტის ელემენტების ტემპერატურის საწყისი მნიშვნელობები იქნება:

100 100 100 100
100 0 0 100
100 0 0 100
50 50 50 50

პირველი იტერაციის შემდეგ მივიღებთ;

100 100 100 100
100 50 50 100
100 37.5 037.5 100
50 50 50 50

მეორე იტერაციის შემდეგ მივიღებთ:

100 100 100 100
100 71.875 071.875 100
100 059.375 059.375 100
50 50 50 50

ამის შემდეგ ტემპერატურათა სხვაობა არ აღემატება 40 გრადუსს ფირფიტის არცერთი წერტილისათვის. ესე იგი თერმული წონაწორობა დამყარდა და მეორე იტერაციის შედეგად მიღებული ტემპერატურები შეესაბამება ტემპერატურულ წონასწორობას.

4. MATLAB ამოხსნა

```
%     This program initializes the temperatures in
%     metal plate and determines the equilibruim
%     temperatures based on atolerance value
%
nrows = input(' Enter number of rows');
ncols = input('Enter number of columns');
iso1 = input('Enter temperature for top and sides');
iso2 = input('Enter temperature for bottom');
tolerance = input('Enter equilibrium tolerance');
%     %
%     %     Initialize and print temperature matrix
%     %
old = zeros(nrows, ncols);
old(1,:) = iso1 + zeros(1, ncols);
```

```
old(:,1) = iso1 + zeros(nrows,1);
old(:,ncols) = iso1 + zeros(nrows,1);
old(nrows,:) = iso2 + zeros(1, ncols);
disp('Initial Temperatures');
disp(old)
new = old;
equilibrium = 0;
%    %
%    %     Update temperatures and test for equilibrium
%    %
while ~equilibrium
    for m = 2:nrows - 1
        for n = 2: ncols - 1
            new(m,n) = (old(m-1,n)+old(m,n-1)+...
            old(m,n+1)+old(m+1,n))/4;
        end
    end
    if all(new-old<=tolerance)
        equilibrium = 1;
        disp('Equilibrium Temperatures');
        disp(new)
    end
    old = new;
end
```

5. შემოწმება

თუ გავუშვებთ პროგრამას MATLAB –ში და მივაწოდებთ სათანადო მნიშვნელობებს, შედეგად მივიღებთ:

```
Enter number of rows  4
Enter number of columns  4
Enter temperature for top and sides  100
Enter temperature for bottom  50
Enter equilibrium tolerance  40
Initial Temperatures
   100   100   100   100
   100     0     0   100
   100     0     0   100
    50    50    50    50
Equilibrium Temperatures
  100.0000  100.0000  100.0000  100.0000
  100.0000   71.8750   71.8750  100.0000
  100.0000   59.3750   59.3750  100.0000
   50.0000   50.0000   50.0000   50.0000
```

ამ ამოცანის ამოხსნისას tolerance მნიშვნელოვანი ცვლადია. სცადეთ შეასრულოთ იგივე ამოცანა tolerance მცირე მნიშვნელობისათვის, მაგალითად 1 გრადუსი და დააკვირდით განსხვავებას შედეგად მიღებულ ტემპერატურული წონასწორობის სურათებს შორის. ასევე საინტერესო იქნება პროგრამა ისე შეცვალოთ, რომ დამატებით გვაძლევდეს იტერაციათა რაოდენობას, რომელიც აუცილებელია ტემპერატურული წონასწორობის დასამყარებლად.

ამ თავში განვიხილეთ პროგრამის კონტროლის სტრუქტურის 3 განსხვავებული ტიპის ოპერატორი. **if** ბრძანება, ოპერატორებთან **else** და **elseif** ერთად საშუალებას გვაძლევს გავუშვათ ბრძანებათა სხვადასხვა ჯგუფი, იმის მიხედვით, თუ როგორია ლოგიკური

გამოსახულება. **for** და **while** ბრძანებაები, რომელნიც ბრძანებთა ჯგუფის რამდენჯერმე განმეორების საშუალებას იძლევა. კონტროლის ეს სტრუქტურა აუცილებელია მრავალი საინჟინრო პრობლემის გადასაწყვეტად. განვიხილეთ ოპტიკურ ბოჭკოებთან და ტემპერატურულ წონასწორობასთან დაკავშირებული ამოცანები.

specsimboloebi

<	ნაკლებია
<=	ნაკლებია ან ტოლია
>	მეტია
>=	მეტია ან ტოლია
==	ტოლია
~=	არ უდრის
&	და
|	ან
~	არ

brZanebebi da funqciebi

all	განსაზღვრავს ჭეშმარიტია თუ არა ყველა მნიშვნელობა
any	განსაზღვრავს რომელიმე მნიშვნელობა მაინც თუ არის ჭეშმარიტი
break	შეწყვეტს პროგრამის მიერ გაშვებულ ციკლს
else	**if** სტრუქტურის ოპერატორი
elseif	**if** სტრუქტურის ოპერატორი
end	განსაზღვრავს კონტროლის სტრუქტურის დასასრულს
exist	განსაზღვრავს მოცემულ სამუშაო სივრცეში არის თუ არა მითითებული ცვლადი
find	პოულობს არანულოვან მნიშვნელობების ინდექსებს
finite	განსაზღვრავს სასრულია თუ არა სიდიდე
for	აწარმოებს ციკლის სტრუქტურას
if	ამოწმებს ლოგიკურ გამოსახულებას
isempty	განსაზღვრავს ცარიელია თუ არა მატრიცა
isnan	განსაზღვრავს განუზღვრელობაა თუ არა მოცემული სიდიდე
isstr	გაბსაზღვრავს ცვლადი სიმბოლოა თუ რიცხვითი გამოსახულება
strcmp	ადარებს ერთმანეთს ორ სტრიქონს
while	აყალიბებს ციკლის სტრუქტურას

ამოცანები

1 – 8 პერობლემა დაკავშირებულია ამ თავში განხილულ ამოცანებთან, 9 –23 შეეხება განსხვავებულ საინჟინრო პრობლემებს

ოპტიკური ბოჭკოები. ეს ამოცანები დაკავშირებულია გარდატეხის მაჩვენებელთა ცხრილთან, რომელიც მოცემული იყო ამ თავის შესაბამის ნაწილში.

1. დაბეჭდეთ ცხრილი, რომელიც შეიცავს ინფორმაციას ფაილიდან indices.dat შემდეგი ფორმატით:

refraction indices
Rod Index medium Index

2. დაუმატეთ 1 ამოცანაში აღწერილ ცხრილს სვეტი, რომელი შეიცავს კრიტიკულ კუთხეს გრადუსებში
3. დაუმატეთ 1 ამოცანაში აღწერილ ცხრილს სვეტი, რომელი შეიცავს კრიტიკულ კუთხეს რადიანებში
4. დაბეჭდეთ ინფორმაცია ფაილიდან indices.dat, რომელიც დააკავშირებს ისეთ მასალებს, რომლებიც შექმნიან სინათლის გამტარს. შემდეგი ფორმატით

Light Transmitting Pipes
Rod Index medium Index Angles of Incidense

5. შეცვალე 4 ამოცანით განსაზღვრული ცხრილი ისე, რომ კუთხე ნაცვლად გრადუსებისა, დაიბეჭდოს რადიანებში

თერმული წონასწორობა. ეს პრობლემები უკავშირდება ამ თავში განხილულ შესაბამის ამოცანას

6. ტემპერატურულ წონასწორობასთან დაკავშირებული პროგრამა შეცვალე ისე, რომ მეტალის ფირფიტის ოთხივე კიდეს განსხვავებული ტემპერატურეა ჰქონდეს
7. ტემპერატურულ წონასწორობასთან დაკავშირებული პროგრამა შეცვალე ისე, რომ დაბეჭდოს დამატებით იტერაციათა რაოდენობა, რომელიც საჭიროა თერმული წონასწორობის დასამყარებლად
8. ტემპერატურულ წონასწორობასთან დაკავშირებული პროგრამა შეცვალე ისე, რომ მეტალის ფირფიტის მხოლოდ ერთ კიდეს ჰქონდე ნულისაგან განსხვავებული ტემპერატურა. წერტილს შეიძლება ჰქონდეს ერთი, ორი ან სამი მეზობელი ტემპერატურა.

რაკეტის ტრაექტორია. ქარის პარამარტების გამოსაკვლევად შეიქმნა მცირე ზომის რაკეტა. ვიდრე ტესტირება დაიწყება, საჭიროა შესრულდეს რაკეტის ტრაექტორიის მოდელირება. ინჟინერთა გამოთვლით რაკეტის ტრაექტორია აღიწერება ფორმულით:

$$height = 60 + 2.13t^2 - 0.0013t^4 + 0.000034t^{4.751}$$

განტოლება გვაძლევს რაკეტის სიმაღლეს დედამიწის ზედაპირიდან დროის მოცემულ t მომენტში. სიმაღლის პირველი მნიშვნელობა (60) არის რაკეტის სიმაღლე, ანუ მანძილი დედამიწის ზედაპირიდან რაკეტის წვერომდე.

9. დაწერეთ პროგრამა, რომელიც გამოითვლის და დაბეჭდავს დროისა და რაკეტის სიმაღლის შესაბამისი მნიშნელობებს t = 0 დან იმ მომენტამდე, როცა რაკეტა დაეცემა დედამოწაზე. აიღეთ დროის ნაზრდი 2 წამი. თუ რაკეტა დედამიწაზე არ დაეცემა 100 წამის შემდეგ, შეწყვიტეთ პროგრამა.
10. 9 ამოცანაში აღწერილი პროგრამა შეცვალეთ ისე, რომ ცხრილის ნაცვლად დაბეჭდოს დროის მნიშვნელობა, როცა რაკეტა დედამიწაზე დაშვებას იწყებს და როცა იგი შეეხება დედამიწას.

საოპერაციო ძაფის დაფასოება. მედიცინაში ცოცხალი ქსოვილის გასაკერად ოპერაციის შემდეგ გამოიყენება სპეციალური ბოჭკო. მათი დაფასოებისას დიდი სიფრთხილეა საჭირო, რომ მტვერი და მიკრობები არ მოხვდეს პაკეტში. შეფუთვის შემდეგ ხდება პაკეტის დაბეჭდვა. შტამპი, რომელიც ლუქავს პაკეტს ცხელდება ელექტროგამათბობლით. იმისათვის, რომ დალუქვის პროცესი დამაკმაყოფილებლად ჩაითვალოს, შტამპმა უნდა შეინარჩუნოს საჭირო დონეზე განსაზღვრული ტემპერატურა, წნევა, რმლითაც ის პაკეტს აწვება და დროის ინტერვალი ორ მომდევნო ოპერაციას შორის. დროის პერიოდს ორ ურთირთმომდევნო კონტაქტს შორის (dwell time)- შეყოვნება ეწოდება. დავუშვათ დასაშვები მნიშვნელობების ინტერვალი დამაკმაყოფილებელი პროცესისათვის ასეთია:

ტემპერატურა	150-170^{0}C
წნევა	60-70 psi
შეყოვნება	2-2.5 წმ

11. მონაცემთა ფაილი suture.dat შეიცავს ინფორმაციას ერთი კვირის განმავლობაში წუნდებული სამედიცინო ძაფის პარტიის შესახებ. ყოველი სტრიქონი ფაილში შეიცავს პარტიის ნომერს, ტემპერატურას, წნევას, შეყოვნების პერიოდს დაწუნებული პარტიისათვის. ხარისხის მაკონტროლებელმა ინჟინერმა უნდა გააანალიზოს ეს ინფორმაცია, რათა შეაფასოს რამდენი პროცენტია წუნდებულ პაკეტებში გამოწვეული ტემპეტატურის, წნევის და შეყოვნების დეფექტის გამო. შესაძლოა ზოგიერთი პარტია წუნდებულია ორი სხვადასხვა ნიშნით, მაშინ იგი რეგისტრირდება ყველა შემთხვევაში ცალკ-ცალკე. დაწერეთ პროგრამა, რომელიც დაითვლის და დაბეჭდავს სხვადასხვა ტიპის წუნის პროცენტულ შედგენილობას.
12. 11 ამოცანაში აღწერილი პროგრამა შეცვალეთ ისე, რომ იგი აგრეთვე ბეჭდავდეს პარტიათა ნომერს წუნის თითოეული კატეგორიისათვის და პარტიათა საერთო რაოდენობას, რომელიც ამოღებული იქნა. ყურადღება მიაქციეთ, დაწუნებული პარტია საერთო რაოდენობაში მხოლოდ ერთხელ უნდა ფიგურირებდეს, თუმცა შესაძლოა რამდენჯერმე შეგვხვდეს სხვადასხვა კატეგორიით დაწუნებულ პარტიებში.
13. დაწერეთ პროგრამა, რომელიც წაიკითხავს ფაილს suture.dat, დარწმუნდით, რომ მონაცემებში მართლაც წუნდებული პარტიები მოყვა. თუ რომელიმე პარტია შეცდომითაა მოხვედრილი ფაილში, პროგრამამ დაგვიბეჭდოს სათანადო ცნობა ამის თაობაზე.

ტყის განახლება. ამოცანა ეხება პრობლემას ხეტყის დამუშავებაში. ტყის ფართობის რა ნაწილი უნდა დარჩეს გაუჩეხავი, რომ მოხდეს ტყის განახლება სასურველ პერიოდში. ითვლება, რომ გაჩეხილი ტყის განახლებას გარკვეული დრო სჭირდება ნიადაგისა და კლიმატის პირობების გათვალისწინებით. ტყის განახლების განტოლება გამოხატავს ამ დროს როგორც გაუჩეხავად დარჩენილი ტყის ფართობის და განახლების კოეფიციენტის (სიჩქარის) ფუნქციას. მაგალითად, ვთქვათ ხეტყის დამზადების შემდეგ 100 აკრი ტყე დარჩა გაუჩეხავი, ხოლო ტყის განახლების კოეფიციენტია 0.05, მაშინ პირველი წლის ბოლოს გვექნება 100+0.5 * 100, ანუ 105 აკრი განახლებული ტყე, მეორე წლის ბოლოს კი – 105 + 0.05 * 105=110.25 აკრი იქნება განახლებული ტყის საერთო ფართი.

14. დავუშვათ გვაქვს ტყე საერთო ფართით 14000 აკრი, სადაც დარჩენილი გაუჩეხავი ფართი 2500 აკრია და ტყის განახლების კოეფიციენტია 0.02. დაბეჭდეთ ცხრილი, რომელიც უჩვენებს განახლებული ტყის ფართს ყოველი წლის ბოლოს 20 წლის განმავლობაში.
15. 14 ამოცანაში აღწერილი პროგრამა შეცვალეთ ისე, რომ საშუალება გვქონდეს შევიყვანოთ მონაცემი წლების რაოდენობის შესახებ.

16. 14 ამოცანაში აღწერილი პროგრამა შეცვალეთ ისე, რომ საშუალება გვქონდეს შევიყვანოთ მონაცემი აღდგენილი ტყის ფართის შესახებ და პროგრამამ განსაზღვროს რამდენი წელი დასჭირდება აასეთი ფართის აღდგენას.

კრიტიკული ანალიზი. ეს არის ტექნიკა, რომელიც გამოიყენება პროექტის გეგმის განრიგის შესადგენად. ეს ინფორმაცია სასარგებლოა პროექტის შესრულებისათვის თვალყურის სადევნებლად. ასეთი ანალიზის ერთ-ერთი მეთოდი შემდეგში მდგომარეობს: დავყოთ პროექტი თანმიმდევრულ მოვლენებად. ყოველი მოვლენა დავყოთ რამდენიმე ქვეპუნქტად. ისე, რომ ერთი მოვლენა უნდა დასრულდეს, ვიდრე მეორე დაიწყება. მოვლენის შიგნით რამდენიმე ქვეპუნქტი შესაძლოა ერთდროულად დაიწყოს და მიმდინარეობდეს. დრო, რომელიც საჭიროა ერთი მოვლენის დასამთავრებლად დამოკიდებულია დღეების რაოდენობაზე, რომელიც საჭიროა მისი ყველაზე ხანგრძლივი ქვეპუნქტის შესასრულებლად. მსგავსად ამისა პროექტის დასრულების დრო ტოლია მოვლენათა დასრულებისათვის საჭირო დროის ჯამისა. დავუშვათ კრიტიკული ანალიზის ინფორმაცია პროექტისათვის ჩაწერილია ფაილში: path. dat. ამ ფაილის ყოველი სტრიქონი შეიცავს მოვლენის ნომერს, ქვეპუნქტის ნომერს და ამ ქვეპუნქტის შესრულებისათვის სჭირო დღეების რაოდენობას. მონაცემები ისეა დალაგებული, რომ ჯერ მოცემულია პირველი მოვლენის ყველა ქვეპუნქტი. ამას მოჰყვება მეორე მოვლენის ყველა ქვეპუნქტი და ა.შ.

17. დაწერეთ პროგრამა, რომელიც წაიკითხავს ინფორმაციას ფაილიდან და დაბეჭდავს პროექტის დასრულების განრიგს. ცხრილში მოცემული უნდა იყოს თითოეული მოვლენის ნომერი, ყველაზე ხანგრძლივი ქვეპუნქტის დასრულებისათვის საჭირო დღების რაოდენობა, პროექტის დასრულებისათვის საჭირო დრო დღეებში.
18. წინა ამოცანაში აღწერილი პროგრამა შეცვალე ისე, რომ პროგრამამ დაბეჭდოს მხოლოდ ის მოვლენა და ქვეპუნქტი, რომლის დასრულებისათვის საჭიროა 5 დღეზე მეტი.
19. შეცვალე პროგრამა ისე, რომ დაბეჭდოს მოვლენის ნომერი და მასში შემავალი ქვეპუნქტების რაოდენობა.

მონაცემები მიმღებიდან. დავუშვათ ფაილი სახელით **sensor.dat** შეიცავს ინფორმაციას, რომელიც შეგროვილია რამდენიმე სენსორიდან ერთდროულად. ყოველი სტრიქონი შეიცავს სხვადასხვა სენსორიდან აღებულ მონაცემებს. პირველ სტრიქონში გვაქვს მონაცემები აღებული t = 0.0 წმ-ზე, მეორეში - 1.0 წმ-ზე და ა. შ.

20. დაწერეთ პროგრამა, რომელიც კითხულობს მონაცემთა ფაილს და ბეჭდავს სენსორების რაოდენობას და და ინფორმაციას იმის შესახებ, რამდენი წამის განმავლობაში გროვდებოდა მონაცემები.
21. შეცვალე მე-20 ამოცანისათვის დაწერილი პროგრამა ისე, რომ ყველა მონაცემი, რომელიც აჭარბებს 10.0 გაუტოლდეს 10.0, ხოლო ყველა მონაცემი, რომელიც ნაკლებია –10, გაუტოლდეს –10.0
22. დაწერეთ პროგრამა, რომელიც ააგებს პირველ მიმღებზე შეგროვილი მონაცემების გრაფიკს დროის მიმართ.
23. დაწერეთ პროგრამა, რომელიც იპოვის ცხრილში იმ მონაცემის მდებარეობას (ინდექსს), რომლის მნიშვნელობაც სასრული სიდიდე არ არის
24. დაწერეთ პროგრამა, რომელიც განსაზღვრავს იმ მონაცემთა მდებარეობას, რომელთა მნიშვნელობაც მეტია 20-ზე
25. დაწერეთ პროგრამა, რომელიც დაბეჭდავს ისეთ მონაცემთა რაოდენობას, რომელთა მნიშვნელობაც 0-ის ტოლია.