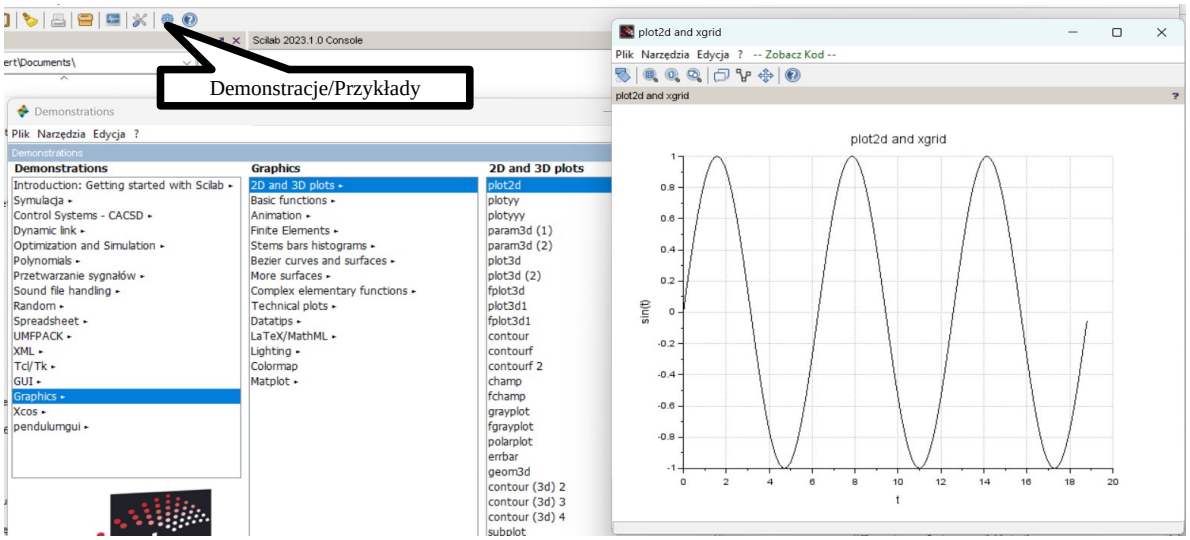
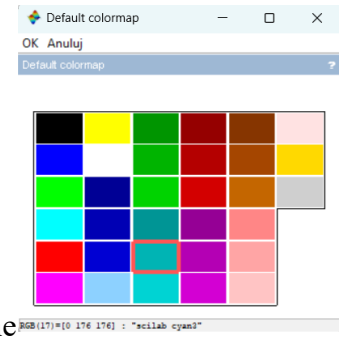
Laboratorium 3.

Elementy graficzne - wykresy.

SCILAB umożliwia tworzenie wykresów 2-wymiarowych, trójwymiarowych, w trybie subplot z zastosowaniem różnych specyficznych typów wykresów. Utworzone wykresy mogą być edytowane w zakresie opcji dotyczących wyglądu wykresu (kolory, typy linii, tytuły, znaczniki osi itp.) a także powiększane i obracane (wykresy 3D). Możliwości w zakresie tworzenia wykresów są obszernie prezentowane w *Demonstracje/Przykłady.*



Podczas tworzenia wykresów, przydatne może być korzystanie z kolorów. Dostępne standardowe kolory możemy sprawdzić poleceniem *getcolor()*. W wywołanym oknie zobaczymy paletę dostępnych kolorów standardowych. Klikając „kafelek" z kolorem, w dolnej części okna zobaczymy numer koloru, nazwę, oraz kompozycję RGB.



Wykresy dwuwymiarowe – podstawy:

Wykresy są tworzone w dodatkowym oknie nazywanym oknem graficznym. W SCILAB okno graficzne i jego zawartość (np. wykres) mają strukturę hierarchiczną. W oknie graficznym jest umieszczany wykres zawierający co najmniej jedną oś współrzędnych, z kolei oś/osie zawiera elementy podrzędne takie jak limie wykresu czy zestaw parametrów sterujących wyglądem obszaru wykresu (np. kolory tła oraz osi, położenie osi, rozmieszczenie , . Linia wykresu zawiera z kolei zestaw parametrów sterujących jej wyglądem (styl linii lub znacznika, kolor).

Podstawowym poleceniem jest *plot2d*.

plot2d(argument)

Można je wywoływać z argumentami różnych typów.

A) Jeżeli argumentem jest wektor to na osi *Ox* umieszczane zostaną numery współrzędnych wektora (pozycje kolejnych elementów wektora), na osi Oy natomiast współrzędne tego wektora.

Przykład:

Chcemy zbadać zachowanie ciągu $a_n = sin(n^2)$. dla 1<= n <=10000

Tworzymy wektor z kolejnymi elementami ciągu:

```
--> A=sin([1:10000].^2);
```

W wektorze wierszowym uzyskujemy 10000 początkowych wyrazów badanego ciągu.

Chcąc narysować wykres fragmentu tego ciągu (np. wyrazy od 801-go do 1000-go) stosujemy polecenie:

```
--> plot2d(A(801:1000))
```

Wyrażenie *A(801:1000)* indeksuje elementy wektora *A* , od elementu z pozycji 801 do elementu z pozycji 1000 co daje łącznie 200 elementów. Z tego powodu na osi *Ox* są liczby w zakresie do 200 co może być mylące. Aby uzyskać prawidłowe skalowanie osi *Ox,* odpowiadające pozycjom na jakich w wektorze *A* znajdują się liczby wzięte do utworzenia wykresu należy dodatkowo podać wektor zawierający liczby do oznaczenia osi *Ox*:

```
--> plot2d(801:1000,A(801:1000))
```

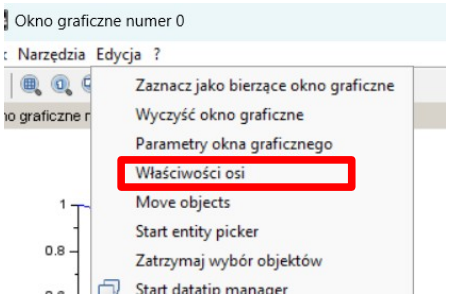
– zastosowana tu składnia zostanie opisana w punkcie C)

UWAGA:
Jeżeli powyższe polecenie rysowania wykresu zostanie wywołane przy otwartym oknie graficznym (z wykresem utworzonym wcześniej) to nowy wykres zostanie dorysowany do już istniejącego. Jest to standardowe ustawienie programu SCILAB przydatne gdy chcemy rysować wiele wykresów w jednym układzie współrzędnych. Jeśli chcemy uzyskać tylko wykres z prawidłowo wyskalowaną osią *Ox* należy najpierw zamknąć okno graficzne z poprzednim wykresem.
Można też zmienić standardowe ustawienie powodując, że nowy wykres nie będzie 'dorysowywany' tylko zastąpi poprzedni: *gda().auto_clear = "on"*.

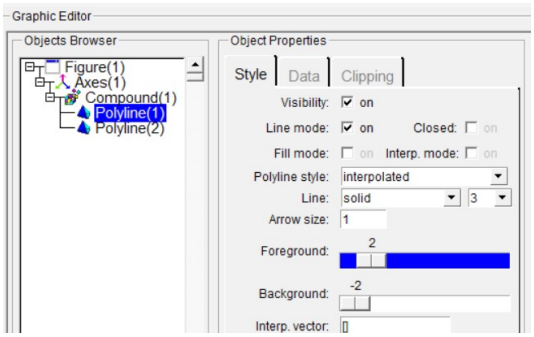
B) Jeżeli argumentem polecenia *plot2d* jest macierz to na osi *Ox* umieszczane zostaną numery wierszy macierzy, na osi *Oy* natomiast współrzędne będące wartościami w poszczególnych wierszach macierzy.

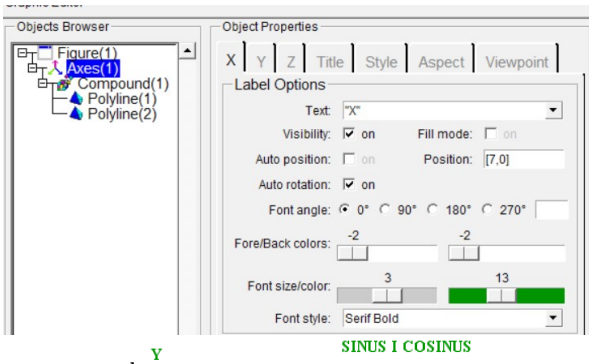
Przykład:
Chcemy narysować wykresy funkcji trygonometrycznych sin, cos, w zakresie 0-2pi.
--> X=linspace(0,2*%pi,201) – wektor 201-liczb w zakresie 0-2pi rozłożonych liniowo.
--> Y=[sin(X); cos(X)]' – dwukolumnowa macierz z wartościami funkcji sin, cos.
--> plot2d(Y) – rysownie wykresów.
Jak można zauważyć oś *Ox* jest skalowana wg liczby wierszy w macierzy *Y* (0-201).

C) Jeżeli argumentem są dwa wektory {*plot2d(wektorX,wektorY)*} - (*wektorX* oraz *wektorY* muszą mieć taką samą długość) to na osiach *Ox* oraz *Oy* umieszczane są współrzędne (wartości) z odpowiednich pozycji tych wektorów. Polecenie to nadaje się do rysowania tradycyjnych wykresów ze skalowaniem osi *Ox*.
--> plot2d(X,Y) – odpowiednie skalowanie na osi *Ox*. Wartości z wektora X skalują oś *Ox*.
Uzyskany wykres jest dość 'surowy' i jeśli miałby być wykorzystany np. w prezentacji wymaga dopracowania. Poddajmy go edycji aby poprawić wygląd. Jeśli chcemy edytować utworzony wykres wybieramy w menu okna wykresu polecenia *Edycja-Właściwości osi* . W nowo otwartym oknie możemy zmieniać wszystkie parametry odpowiedzialne za wygląd wykresu.

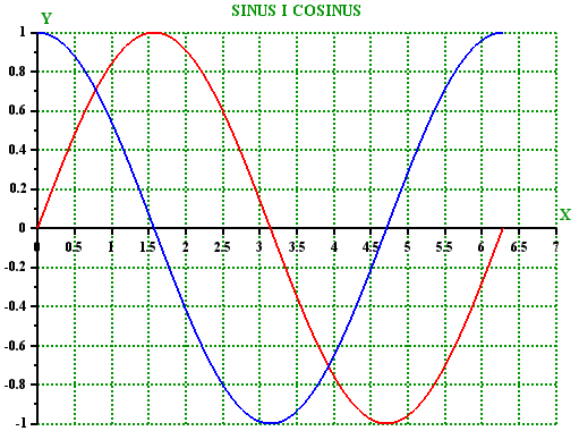


- Aby zmienić kolor linii wykresów rozwijamy *Axes-Compound* i zaznaczamy *Polyline (1)* lub *Polyline (2)*.
- Aby zmienić kolor linii ustawiamy suwak *Foreground* w pozycję 2 dla koloru niebieskiego lub 5 dla koloru czerwonego.
- Typ linii pozostawiamy *solid*, grubość linii zmieniamy na 3 (rysunek po lewej).
- Zaznaczając *Axes(1)* możemy wprowadzić podpisy poszczególnych osi X, Y, Z oraz w zakładce *Title* podpis całego wykresu *(rysunek po prawej)*. Dla osi X, Y wybieramy *Font size 3, Font color 13, Grid color 13*. W polach *Text* wstawić podpisy osi: *'X', 'Y'*. Wyłączyć opcję *Auto position* i wpisać pozycję tytułów osi: dla osi X *[7,0]* , dla osi Y *[0,1]*. Dla osi X wybrać *Location middle,* dla osi Y *Location left*.
- W zakładce *Title* w polu *Text* wpisać tytuł wykresu *'SINUS I COSINUS'*, wybrać *Font size 3, Font color 13.*





Możliwości edycji wykresu jest znacznie więcej.
Np w zakładce *Style* można zmieniać rozmiar, kolor oraz rodzaj czcionki dla znaczników osi, kolor osi, kolor tła wykresu i grubość linii siatki. Należy samodzielnie przetestować pozostałe dostępne opcje edycji. Uzyskany rysunek powinien wyglądać jak obok. Wszystkie opcje edycyjne wykresu mogą być również podane w formie poleceń



tekstowych. Ma to znaczenie szczególnie jeśli wykres w końcowej postaci ma być efektem działania skryptu i nie będzie podlegał dodatkowej 'ręcznej' edycji. Więcej informacji o opcjach polecenia można znaleźć w pomocy (-->help plot2d)
W tym miejscu podamy sposób wywoływania opcji dotyczących koloru linii wykresu oraz zastosowania znaczników punktów wykresu. Jeśli zastosujemy składnię : *plot2d(x,y,style)* to parametr *style* określa odpowiednie kolory linii lub typy znaczników.
Jeśli parametr *style* jest dodatnią liczbą całkowitą, wykres jest rysowany linią ciągłą w kolorze zależnym od wartości tej liczby. Liczby odpowiadające kolorom znajdziemy używając polecenia *getcolor().*
Jeśli parametr *style* jest ujemną liczbą całkowitą lub zerem, podane punkty wykresu są rysowane za pomocą znaczników. Typ znacznika zależy od wartości bezwzględnej parametru *style.* Poniżej podano znaczniki i przypisane im liczby (należy zastosować ujemne wartości).


```
-->plot2d(X,Y(:,1),2)
-->plot2d(X,Y(:,1),5)
-->plot2d(X,Y(:,1),-3)
-->plot2d(X,Y(:,1),-5)
```

Aby utworzyć nowe (kolejne) okno graficzne do rysowania wykresu, należy użyć polecenia *scf( )*. Jeżeli w nawiasie polecenia *scf( )* podamy liczbę to będzie ona numerem utworzonego okna, jeśli nawias pozostanie pusty to program przydzieli dla nowego okna graficznego kolejny numer.

W SCILAB dostępne jest również polecenie *plot()* zapożyczone z Matlab. Opcje polecenia *plot()* są również takie same jak w Matlab. Dla wielu, składnia polecenia *plot* może być bardziej przyjazna ponieważ używa oznaczeń literowych i symboli zamiast cyfr.
Podstawowa składnia polecenia: *plot(x,y,LineSpec)*
Parametr *LineSpec* określa styl linii, styl znacznika i kolor wykreślanych linii. Może być w postaci jednego lub dwóch znaków. Należy pamiętać, że parametr *LineSpec* powinien być zapisany w apostrofach.
Oznaczenia typu linii:
- linia ciągła, -- linia kreskowa, : linia kropkowa, -. linia kropka-kreska.
Oznaczenia typu znacznika:
+ o * . x s(kwadrat) d(diament) ∧ ∨ > < p(pentagram).
Oznaczenia koloru: k-czarny, w-biały, r-czerwony, g-zielony, c-niebieski, c-cyjan, m-magneta, y-żółty.
-->plot(X,Y,'r—') -->plot(X,Y,'bs')

Dodatkową funkcjonalnością jaką posiada polecenie *Plot* (a nie posiada polecenie *Plot2d*) jest możliwość wykreślenia wykresu funkcji wbudowanej (np. sin, cos) oraz funkcji utworzonych przez użytkownika poprzez ich zaprogramowanie.
Podstawowa składnia takiego polecenia: plot(x, func) , gdzie *x* jest wektorem lub macierzą argumentów wejściowych funkcji, *func* jest standardową nazwą funkcji wbudowanej lub nazwą nadana przez użytkownika podczas programowania funkcji użytkownika.
Przykład:
Sporządzić wykresy funkcji sin(x) oraz cos(x) dla x w zakresie < 7 , 7 >
W pierwszym kroku tworzymy wektor argumentów funkcji:
-->x=-7:0.1:7;
następne wywołujemy polecenia kreślenia wykresów wbudowanych funkcji trygonometrycznych:
-->plot(x,sin,'r—') -->plot(x,cos,'b:')
Zdefiniujmy funkcję użytkownika o nazwie *gauss* obliczającą wartości rozkładu normalnego. Wykorzystamy do tego celu polecenie *deff* umożliwiające definicję własnej funkcji w tzw. trybie in-line. (Szerzej na temat tworzenia własnych funkcji będzie później, w części dotyczącej instrukcji programistycznych).
-->deff('y = gauss(x)' , 'y = 1.5*exp(-(x/2-1).^2)-1');
Następnie możemy narysować wykres tak zdefiniowanej funkcji:
-->plot(x,gauss,'m*--')
Proszę zwrócić uwagę na zastosowane parametry sterujące wyglądem uzyskiwanych wykresów (rodzaj linii, kolor, rodzaj markera).

W wielu opracowaniach często zachodzi konieczność prezentowania danych na wykresach typu histogram. Tego typu wykresy są graficzna prezentacją jednowymiarowych danych (zazwyczaj w postaci prostokątów/słupków. Scilab oferuje funkcje do tworzenia wykresów histogramowych o wuglądzie ‘płaskim’ oraz trójwymiarowum. Mają tu zastosowanie funkcje: *histplot(), bar(), barh(), bar3d()*. Działanie wymienionych funkcji przedstawione zostaną na przykładach. Aby zapoznać się ze szczegółami składni skorzystaj z pomocy programu.

Przykłady:

Tworzymy wektor z danymi do zaprezentowania na histogramie (wektor z 1000 liczb losowych z rozkładem normalnym).

```
-->dane=rand(1,1000,'normal');
```

Tworzymy histogram dla 20-u przedziałów danych

```
-->histplot(20,dane)
```

Tworzymy prezentacje słupkową pierwszych 10-u elementów wektora *dane*

```
-->bar(dane(1:10))
-->clf
-->barh(dane(1:10))
```

Tworzymy macierz losową 6x3 z danymi do zaprezentowania na trójwymiarowym wykresie słupkowym.

```
-->dane3d=5*rand(6,3)
```

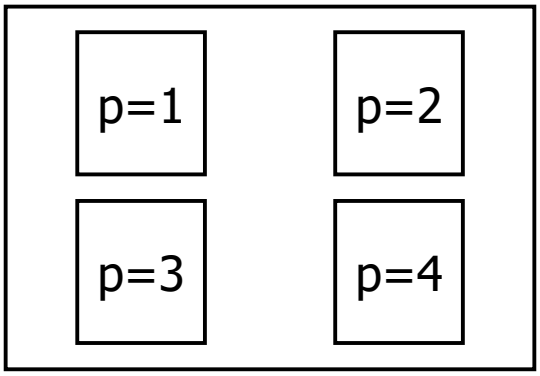
Tworzymy prezentacje słupkową macierzy *dane3d*

```
-->bar3d(dane3d)
```

Rysowanie wykresów na wielu układach współrzędnych w jednym oknie graficznym – polecenie *subplot*

Polecenie *subplot(m,n,p)* dzieli okno graficzne na macierz m x n sub-okien oraz wyznacza sub-okno o numerze p do rysowania bieżącego wykresu. Umożliwia to tworzenie wielu wykresów z różnymi układami współrzędnych w jednym oknie graficznym. Sub-okna są zliczane wierszowo.

Na rysunku poniżej przedstawiono układ sub-okien 2x2 (m=2, n=2) z oznaczoną numeracją sub-okien.



Np. polecenie *subplot(2,2,3)* spowoduje aktywowanie sub-okna o numerze 3. Polecenie *plot2d/plot* spowoduje narysowanie wykresu w aktywnym sub-oknie. Poniżej przykład rysowania wykresów w sub-oknach z pokazaniem innych typów wykresów 2d.

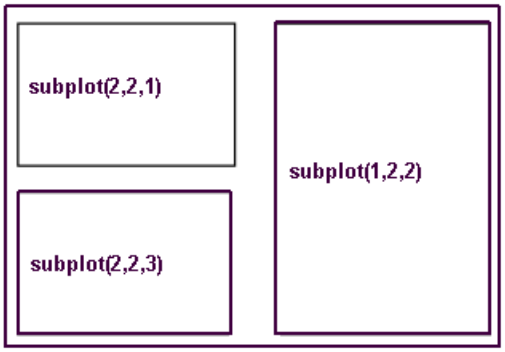
Przykłady:

```
--> x1=linspace(-3,3,101)
--> y1=exp(-(x1.*x1))
--> subplot(2,2,1)
--> plot2d(x1,y1) normalny wykres
--> subplot(2,2,2)
--> plot2d2(x1,y1,3) wykres stały pomiędzy punktami (schodkowy), kolor 3-zielony
--> subplot(2,2,3)
--> plot2d3(x1,y1,5) wykres słupkowy, kolor 5-czerwony
--> subplot(2,2,4)
--> plot2d4(x1,y1,2) wykres „strzałkowy”, kolor 2-niebieski
```

Kolejność aktywowania sub-okien może być dowolna. Pierwsze wydane polecenie *subplot* spowoduje otwarcie okna graficznego i aktywowanie w otwartym oknie obszaru odpowiadającego parametrowi *p.* Kolejne polecenia *subplot* aktywują odpowiednie obszary we wcześniej otwartym oknie graficznym. W jednym oknie graficznym można aktywować obszary z różnych okładów sub-okien (rysunek poniżej).

Przykład:

```
--> x=linspace(-3,3,40);
--> y=linspace(-5,5,40);
--> [X,Y]=meshgrid(x,y);
--> Z=(2*X.^2-Y.^2).*exp(-X.^2-0.5*Y.^2);
--> subplot(221)
--> surf(X,Y,Z)  // o wykresach 3d będzie w dalszej części
```



```
--> subplot(223)
--> contour(x,y,Z,10)
--> subplot(122)
--> plot2d(x,Z)
```
Scilab oferuje również inną funkcję do tworzenia sub-okien okna graficznego: *xsetech().* Skorzystaj z pomocy aby uzyskać więcej informacji.
Uwaga: zamiast polecenia plot2d można stosować polecenie plot.

Wykresy trójwymiarowe
Podstawowym poleceniem jest: *plot3d(wektorX,wektorY,wartości),*
wektor *wektorX* ma długość n_x , *wektorY* ma długość n_y (liczby n_x , n_y nie muszą być równe), *wartości* jest macierzą o rozmiarze $n_x \times n_y$. Składnia polecenia *plot3d()* jest podobna do składni *plot2d()* . Aby poznać szczegóły dotyczące parametrów funkcji *plot3d()* należy skorzystać z pomocy programu.
Przykład:
Sporządzić wykres funkcji f (x , y)= $x^2 + y^2$ w zakresie zmiennych x,y: [−3,3] .
Tworzymy wektory x, y z wartościami zmiennych;
```
--> x=linspace(-3,3,51)
--> y=linspace(-3,3,61)
```
Tworzymy macierze *xx* oraz *yy* poprzez replikowanie wektorów *x* i *y* korzystając z polecenia *ndgrid*.
```
--> [xx,yy]=ndgrid(x,y);
```
Macierz *xx* powstaje poprzez replikowanie wektora *x* w n_y-kolumnach a macierz *yy* powstaje poprzez replikowanie wektora *y* w n_x wierszach.
Następnie obliczamy wartości funkcji:
```
--> z=xx.*xx+yy.*yy;
```
i rysujemy wykres:
```
--> plot3d(x,y,z)
```

Innym sposobem tworzenia wykresów trójwymiarowych jest zastosowanie funkcji *surf( )* .
Przykład:
Sporządzić wykres funkcji f (x , y)=$(2*x^2 – y^2)*\exp(-x^2 – 0.5*y^2)$, w zakresie zmiennych x[−4,4] , y[-3,3].
Tworzymy wektory x, y z wartościami zmiennych;
```
--> x=linspace(-4,4,40)
--> y=linspace(-3,3,40)
```
Tworzymy macierze *xx* oraz *yy* poprzez replikowanie wektorów *x* i *y* korzystając z polecenia *ndgrid* lub *meshgrid*
```
--> [X,Y]=ndgrid(x,y);
--> [Xm,Ym]=meshgrid(x,y);
```
Następnie obliczamy wartości funkcji:
```
--> Z=(2*X.^2-Y.^2).*exp(-X.^2-0.5*Y.^2);
--> Zm=(2*X.^2-Y.^2).*exp(-X.^2-0.5*Y.^2);
```
i rysujemy wykres w dwóch oddzielnych oknach graficznych:
```
-->surf(X,Y,Z)
-->scf()
-->surf(Xm,Ym,Zm)
```

Wykresy krzywych w przestrzeni 3D.
Do wykreślania pojedynczej krzywej w przestrzeni trójwymiarowej służy funkcja *param3d(x,y,z),* gdzie x, y, z są wektorami o takim samym rozmiarze. Elementy wektorów z tych samych pozycji opisują punkty w przestrzeni (są współrzędnymi punktów w przestrzeni).
Przykład: rysowanie linii śrubowej.
```
--> t=linspace(0,4*%pi,101);   --> x=2*cos(t);   --> y=2*sin(t);   --> z=4*t;
--> param3d(x,y,z)
--> param3d(x,y,zeros(z))
```

Podstawy animacji.
W zadaniach programowania inżynierskiego często prowadzone są obliczenia dotyczące ruchu układu mechanicznego. Może okazać się przydatna możliwość weryfikacji/wizualizacji ruchu za pomocą animacji. W środowisku SCILAB możliwe jest tworzenie prostych animacji wizualizujących przeprowadzone obliczenia dotyczące ruchu. Realizacja animacji polega na cyklicznym wyświetlaniu statycznych wykresów/rysunków z odpowiednią częstotliwością. Obiekty, których ruch ma być wizualizowany są na kolejnych rysunkach przedstawiane w kolejnych położeniach. Każdy kolejny rysunek z nowym położeniem obiektu powinien zastępować rysunek z położeniem wcześniejszym. Wstawianie kolejnych rysunków odbywa się w zdefiniowanej pętli z zastosowaniem odpowiedniego opóźnienia. Tworzenie animacji polega więc na generowaniu serii elementów graficznych, po jednym dla każdej klatki animacji. Klatki mogą być wyświetlane w czasie rzeczywistym na ekranie lub zapisywane jako pliki graficzne, które można później zmontować w plik wideo. Istnieje kilka zasad, których przestrzeganie zapewnia generowanie płynnych animacji.
Do prześledzenia zasad tworzenia posłużymy się prostym przykładem wizualizacji ruchu punktu przemieszczającego się po okręgu o zadanym promieniu. Przykład realizujemy tworząc skrypt w oknie edytora. Do wywołania okna edytora służy polecenie ***edit*** wpisane w konsoli Scilab.
Najpierw należy wprowadzić podstawowe dane:

```
clf()       //zamykanie okna graficzne z poprzedniego uruchomienia animacji
alfa=linspace(0,4*%pi,360); //wektor odpowiadający zakresowi ruchu (4*%pi) oraz liczbie
                            klatek animacji (360)
R=5;        //promień okręgu po którym ma poruszać się punkt
xs=3;       //współrzędna środka okręgu
ys=6;       //współrzędna środka okręgu
x=xs+R*cos(alfa); //współrzędne punktów okręgu
y=ys+R*sin(alfa); //współrzędne punktów okręgu
plot(x,y,'.')     // rysowanie wykresu okręgu
```

Po wpisaniu polecenia ***plot*** i uruchomieniu skryptu uzyskamy rysunek okręgu złożonego z 360 punktów. Program automatycznie skaluje osie dostosowując je do zakresów danych wejściowych co powoduje, że rysunek bardziej przypomina elipsę niż okrąg. Aby uzyskać właściwy widok okręgu należy wymusić izometryczny układ osi współrzędnych.
W tym celu do powyższego skryptu dopisujemy:

```
gca().isoview='on'
```

Uzyskaliśmy rysunek proporcjonalnego okręgu. Nie jest to jednak animacja. Aby uzyskać animację ruchu punktu po okręgu, każdy punkt powinien pojawiać się oddzielnie. Chcemy więc utworzyć 360 rysunków punktów pojawiających się jeden po drugim. W tym celu należy zastosować polecenie pętli ***for***. Przed utworzeniem pętli należy usunąć lub wyłączyć linię z poleceniem ***plot***.

```
for i=1:360                  //początek pętli for
      plot(x(i),y(i),'.');   //rysowanie kolejnych punktów na okręgu
end                          //koniec pętli for
```

Po uruchomieniu skryptu, na rysunku pojawiają się kolejne punkty. Jednak zakresy osi są automatycznie skalowane przez co okno graficzne zmienia rozmiar. Aby temu zapobiec ustalamy zakresy osi wpisując wewnątrz pętli ***for*** (po poleceniu ***plot***) polecenie określające zakres osi *x* na <3:9> oraz zakres osi *y* na <0:12>:

```
gca().data_bounds=[-3,0;9,12];
```

W efekcie zakresy osi pozostają stałe i okno graficzne nie zmienia rozmiaru. Ciągle jednak widoczne są wszystkie rysowane punkty. Aby uzyskać animację ruchu punktu, przed narysowaniem danego punktu, poprzednio narysowany punkt powinien zostać usunięty z rysunku. W tym celu wstawimy polecenie ***clf()*** (przed poleceniem ***plot***) powodujące usuwanie poprzedniego rysunku przed wstawieniem rysunku z nowym punktem. Uzyskany efekt nie jest zadowalający. Na ekranie widzimy ‘migający bałagan’. Dzieje się tak,

ponieważ polecenie ***clf()*** powoduje wyczyszczenie okna graficznego i widzimy, że staje się ono puste a następnie, gdy polecenie *plot* powoduje rysowanie nowego okna graficznego, widzimy pojawiający się na chwilę punkt. Rezultatem jest irytujące "migotanie wykresu". Tak naprawdę chcemy, aby podczas oglądania klatki z danym punktem, klatka z kolejnym punktem była generowana ‘w tle’ w sposób dla nas niewidoczny z wykorzystaniem pamięci komputera. Następnie chcemy, aby nowa klatka "natychmiast" zastąpiła starą. Wymaga to dwóch segmentów pamięci lub "buforowania wideo". Jeden z nich przechowuje aktualnie widoczną klatkę. Drugi: bufor "tła" to miejsce, w którym komputer generuje następną klatkę. Gdy kolejna klatka jest gotowa, komputer szybko kopiuje zawartość bufora tła do widocznego bufora i na ekran komputera. Scilab udostępnia dwa polecenia do implementacji podwójnego buforowania: ***drawlater()*** *i* ***drawnow()****,* których zastosowanie jest bardzo łatwe. Wstawiamy ***drawlater()*** na początku pętli ***for*** – przed poleceniem ***clf()*** oraz wstawiamy polecenie ***drawnow()*** na końcu pętli ***for*** – przed poleceniem ***end***. Możemy powiększyć rozmiar poruszającego się punktu zmieniając właściwości określające sposób jego wyświetlania. W tym celu należy zmodyfikować polecenie ***plot***.

```
plot(x(i),y(i),'.','MarkerSize',20,'MarkerEdgeColor','blue','MarkerFaceColor','blue');
```

Uzyskano animację ruchu punktu po okręgu. Pomiędzy poleceniami ***drawlater()*** i ***drawnow()*** możemy modyfikować wykres w dowolny sposób - dodając etykiety, tytuły, zmieniając mapy kolorów dodając siatkę itp. Szybkość aktualizacji klatek zależy od czasu potrzebnego na wygenerowanie nowej klatki wynikającego z ilości działań i wydajności komputera. Możemy wydłużyć czas pomiędzy kolejnymi klatkami stosując np. polecenie ***sleep***.
Końcowy skrypt ma postać:

```
clf()
alfa=linspace(0,4*%pi,300);
R=5;
xs=3;
ys=6;

x=xs+R*cos(alfa);
y=ys+R*sin(alfa);

for i=1:300
   drawlater()
    clf()
    plot(x(i),y(i),'.','MarkerSize',20,'MarkerEdgeColor', 'blue', 'MarkerFaceColor', 'blue');
    gca().data_bounds=[-3,0;9,12];
    gca().isoview='on'
   drawnow()
// sleep(50)
end
```

Zadanie: zmodyfikować skrypt tak aby wraz z ruchem punktu był widoczny ruch promienia okręgu, który łączy poruszający się punkt ze środkiem okręgu.
Dodać interakcję skryptu z użytkownikiem umożliwiającą wybór współrzędnych środka okręgu, promienia okręgu, liczby obrotów zakreślanych przez poruszający się punkt.