Chapter 9. Dynamical System

9.1. Differentiation of vector-(matrix-)valued functions

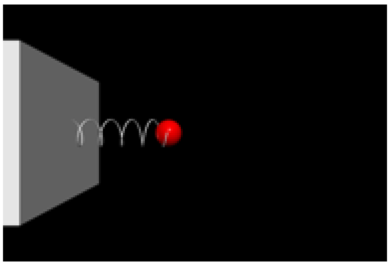
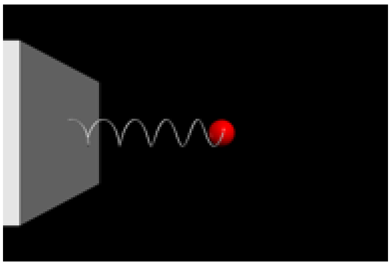
Empty

9.2. Newton's equation of motion

Programs: newton.py

In [1]:

```
from vpython import *

Ball = sphere(color=color.red)
Wall = box(pos=vec(-10, 0, 0), length=1, width=10, height=10)
Spring = helix(pos=vec(-10, 0, 0), length=10)
dt, x, v = 0.01, 2.0, 0.0
while True:
    rate(1 / dt)
    dx, dv = v * dt, -x * dt
    x, v = x + dx, v + dv
    Ball.pos.x, Spring.length = x, 10 + x
```

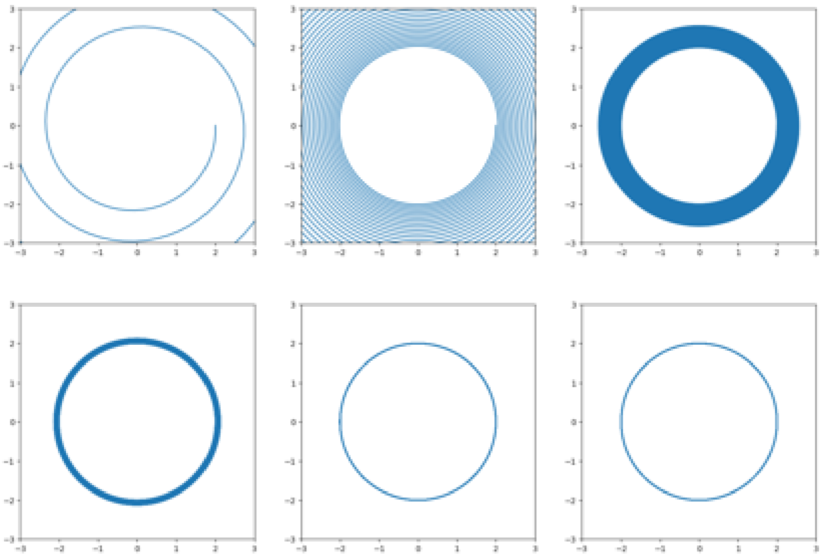



Programs: newton2.py

In [1]:

```
from numpy import arange
import matplotlib.pyplot as plt


def taylor_1st(x, v, dt):
    dx = v * dt
    dv = -x * dt
    return x + dx, v + dv


def taylor_2nd(x, v, dt):
    dx = v * dt - x / 2 * dt ** 2
    dv = -x * dt - v / 2 * dt ** 2
    return x + dx, v + dv
```

```
fig = plt.figure(figsize=(18,6))
update = taylor_1st  # taylor_2nd
for dt, pos in [(0.1, 131),  (0.01, 132), (0.001, 133)]:
    plt.subplot(pos)
    plt.axis('scaled'), plt.xlim(-3.0, 3.0), plt.ylim(-3.0, 3.0)
    path = [(2.0, 0.0)]  # (x, v)
    for t in arange(0, 500, dt):
        x, v = path[-1]
        path.append(update(x, v, dt))
    plt.plot(*zip(*path))
```



Untitled.ipynb

In [1]:
```
from sympy import *
A = Matrix([[0, 1], [-1, 0]])
A.diagonalize()
```

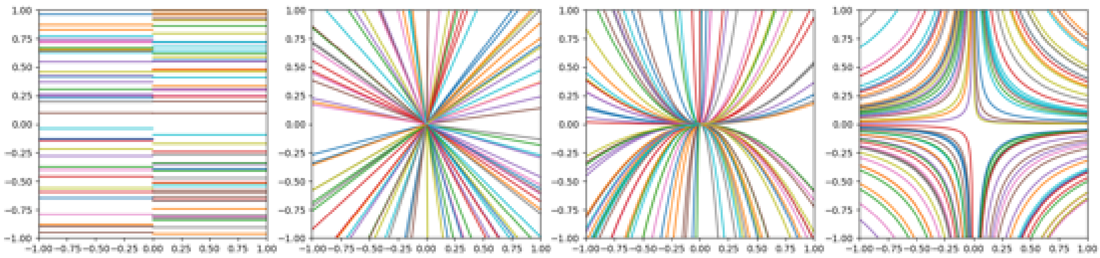
Out[1]:
```
(Matrix([
 [I, -I],
 [1,  1]]),
 Matrix([
 [-I, 0],
 [ 0, I]]))
```

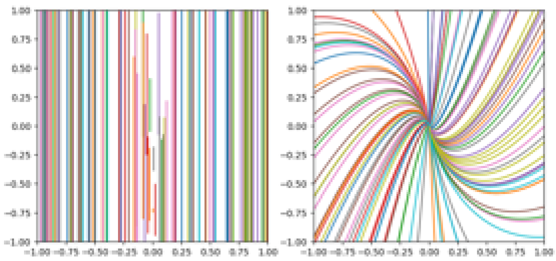
9.3. Linear differential equation

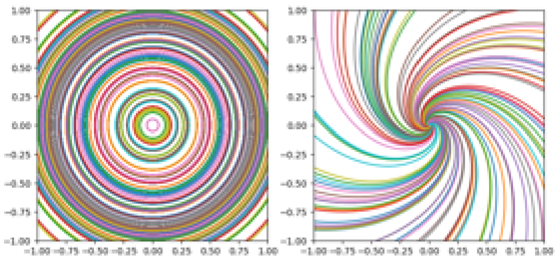
Program: phasesp.py

In [1]:
```
from numpy import array, arange, exp, sin, cos
from numpy.random import uniform
import matplotlib.pylab as plt


def B1(lmd1, lmd2):
    return lambda t: array([[exp(lmd1 * t), 0],
                            [0, exp(lmd2 * t)]])
```

```
def B2(lmd):
    return lambda t: exp(lmd * t) * array([[1, 0], [t, 1]])


def B3(a, b):
    return lambda t: exp(a * t) * array([
        [ cos(b * t), sin(b * t)],
        [-sin(b * t), cos(b * t)]])

# B1(lmd1, lmd2), B2(lmd), B3(a, b)
#
fig = plt.figure(figsize=(20, 5))
pos = 140
for B in [B1(1, 0), B1(1, 1), B1(1, 2), B1(1, -1)]:
#
#fig = plt.figure(figsize=(10, 5))
#pos = 120
#for B in [B2(0), B2(1)]:
#
#fig = plt.figure(figsize=(10, 5))
#pos = 120
#for B in [B3(0, 1), B3(1, 1)]:
    pos += 1
    V = uniform(-1, 1, (100, 2))
    T = arange(-10, 10, 0.01)
    plt.subplot(pos)
    plt.axis('scaled'), plt.xlim(-1, 1), plt.ylim(-1, 1)
    [plt.plot(*zip(*[B(t).dot(v) for t in T])) for v in V]
```

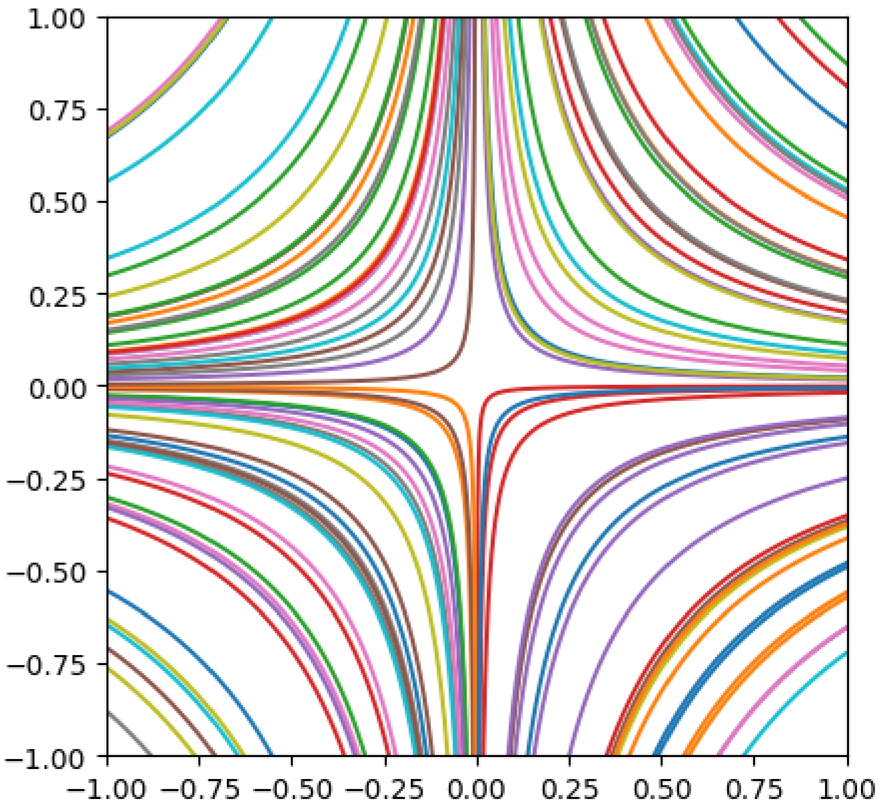






Untitled.ipynb

In [1]:
```
from numpy import array, arange, exp, sin, cos
from numpy.random import uniform
import matplotlib.pylab as plt

def B1(lmd1, lmd2):
    return lambda t: array([[exp(lmd1 * t), 0],
                            [0, exp(lmd2 * t)]])

def B2(lmd):
    return lambda t: exp(lmd * t) * array([[1, 0], [t, 1]])

def B3(a, b):
    return lambda t: exp(a * t) * array([
        [ cos(b * t), sin(b * t)], [-sin(b * t), cos(b * t)]])

B = B1(1, -1)  # B1(lmd1, lmd2), B2(lmd), B3(a, b)
V = uniform(-1, 1, (100, 2))
T = arange(-10, 10, 0.01)
plt.axis('scaled'), plt.xlim(-1, 1), plt.ylim(-1, 1)
[plt.plot(*zip(*[B(t).dot(v) for t in T])) for v in V]
plt.show()
```



Untitled1.ipynb

In [1]:
```
import sympy as sp
import numpy as np
A = [[1, 2], [2, 1]]
sp.exp(sp.Matrix(A))
```

Out[1]:

$$\begin{bmatrix} \frac{1}{2e} + \frac{e^3}{2} & -\frac{1}{2e} + \frac{e^3}{2} \\ -\frac{1}{2e} + \frac{e^3}{2} & \frac{1}{2e} + \frac{e^3}{2} \end{bmatrix}$$

In [2]:
```
sp.exp(sp.Matrix(A)).evalf()
```

Out[2]:

$$\begin{bmatrix} 10.2267081821796 & 9.85882874100811 \\ 9.85882874100811 & 10.2267081821796 \end{bmatrix}$$

In [3]:
```
np.exp(np.matrix(A))
```

Out[3]:
```
matrix([[2.71828183, 7.3890561 ],
        [7.3890561 , 2.71828183]])
```

In [4]:
```
sp.Matrix(A)**2
```

Out[4]:

$$\begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

In [5]:
```
np.matrix(A)**2
```

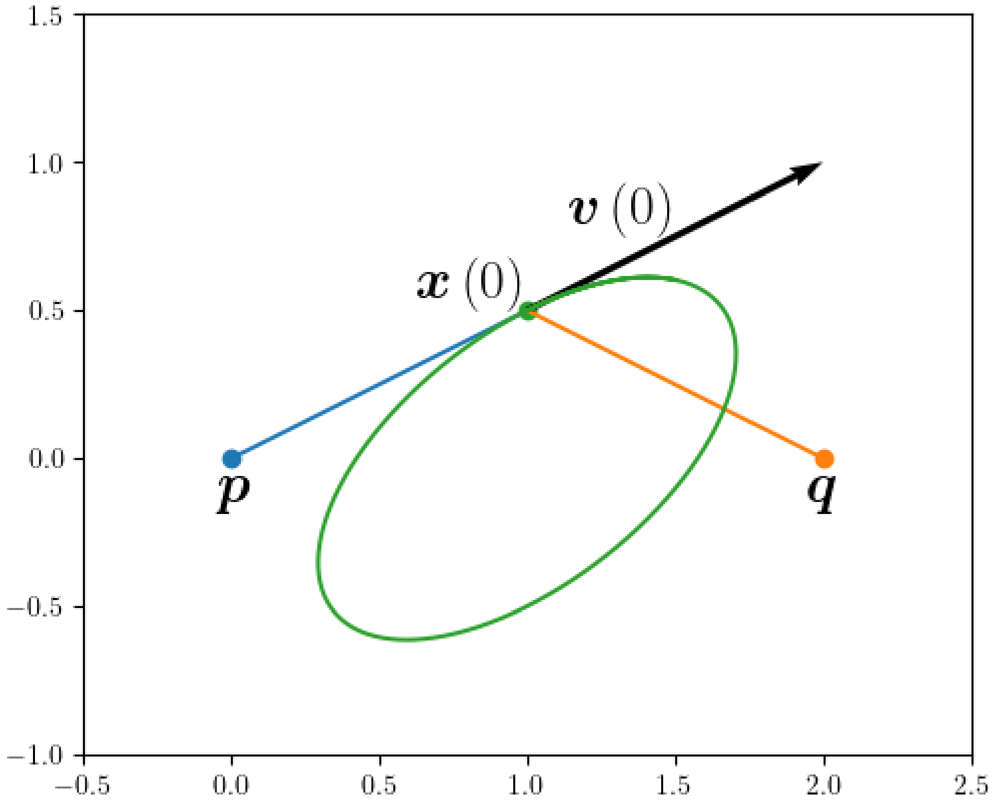
Out[5]:
```
matrix([[5, 4],
        [4, 5]])
```

Program: exercise.py

In [1]:
```
from numpy import *
import matplotlib.pyplot as plt

plt.rc('text', usetex=True)
plt.rcParams["text.latex.preamble"] = r'''
\usepackage{amssymb}
\usepackage{amsmath}
\usepackage{stmaryrd}
\newcommand{\vv}[1]{\ensuremath{\boldsymbol{#1}}}
'''

p = array((0, 0))
q = array((2, 0))
t = linspace(0, 5, 1000)
x1 = 1 + sin(sqrt(2)*t)/sqrt(2)
x2 = (sqrt(2)*cos(sqrt(2)*t) + sin(sqrt(2)*t)) / (2*sqrt(2))
v1 = cos(sqrt(2*t))
v2 = (-sqrt(2)*sin(sqrt(2)*t) + cos(sqrt(2)*t)) / 2

plt.axis('scaled'), plt.xlim(-0.5,2.5), plt.ylim(-1,1.5)
plt.scatter(0, 0)
plt.scatter(2, 0)
plt.plot([0, x1[0]], [0, x2[0]])
plt.plot([2, x1[0]], [0, x2[0]])
plt.text(0, -0.02, r'$\vv{p}$', fontsize = 20,
         verticalalignment ='top', horizontalalignment ='center')
plt.text(2, -0.02, r'$\vv{q}$', fontsize = 20,
         verticalalignment ='top', horizontalalignment ='center')
plt.text(x1[0], x2[0], r'$\vv{x}\left(0\right)$', fontsize = 20,
         verticalalignment ='bottom', horizontalalignment ='right')
```

```
plt.plot(x1, x2)
plt.scatter(x1[0], x2[0])
plt.quiver(x1[0],  x2[0], v1[0], v2[0], units='xy', scale=1)
plt.text(x1[0]+v1[0]/2, x2[0]+v2[0]/2, r'$\vv{v}\left(0\right)$', fontsiz
         verticalalignment ='bottom', horizontalalignment ='right')
```

Out[1]: Text(1.5, 0.75, '$\\vv{v}\\left(0\\right)$')



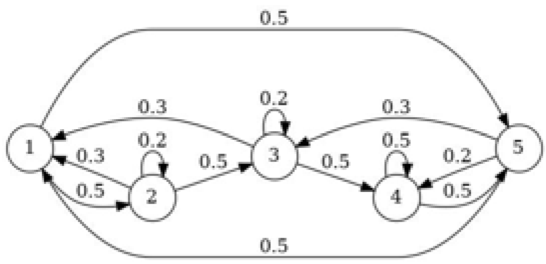
9.4. Stationary state of Markov chain

Program: graph.py

In [1]:
```
from graphviz import Digraph

A = [[  0, 0.3, 0.3,   0, 0.5],
     [0.5, 0.2,   0,   0,   0],
     [  0, 0.5, 0.2,   0, 0.3],
     [  0,   0, 0.5, 0.5, 0.2],
     [0.5,   0,   0, 0.5,   0]]

N = len(A)

G = Digraph(format='jpg')
G.body.extend(['rankdir=LR'])

G.attr('node', shape='circle')

for n in range(N):
    node = str(n + 1)
    G.node(node)

for m in range(N):
```

```
    for n in range(N):
        s = A[m][n]
        if s != 0:
            node1 = str(n + 1)
            node2 = str(m + 1)
            label = str(s)
            G.edge(node1, node2, label)

G.render('graph')
```

Out[1]: '9.4/graph.jpg'



Also generated a dot-file graph.

Install `dot2tex` if we need.

```
sudo apt install dot2pdf
```

Convert graph to graph.tex.

```
dot2tex graph
```

Edit graph.tex:

Before:

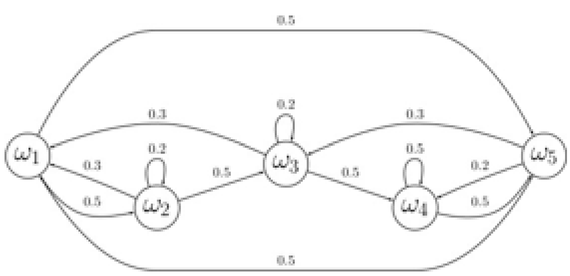
```
\node (1) at (18.0bp,84.0bp) [draw,circle] {1};
  \node (2) at (113.0bp,46.0bp) [draw,circle] {2};
  \node (5) at (398.0bp,84.0bp) [draw,circle] {5};
  \node (3) at (208.0bp,78.0bp) [draw,circle] {3};
  \node (4) at (303.0bp,46.0bp) [draw,circle] {4};
```

After:

```
\node (1) at (18.0bp,84.0bp) [draw,circle] {\huge$\omega_1$};
  \node (2) at (113.0bp,46.0bp) [draw,circle] {\huge$\omega_2$};
  \node (5) at (398.0bp,84.0bp) [draw,circle] {\huge$\omega_5$};
  \node (3) at (208.0bp,78.0bp) [draw,circle] {\huge$\omega_3$};
  \node (4) at (303.0bp,46.0bp) [draw,circle] {\huge$\omega_4$};
```

Then, typeset. We have graph.pdf.

```
pdfcrop graph.pdf
```

We have a croped pdf file  .

9.5. Markov random field

Program: gibbs.py

In [1]:

```
from numpy import random, exp, dot
from tkinter import Tk, Canvas


class Screen:
    def __init__(self, N, size=600):
        self.N = N
        self.unit = size // self.N
        tk = Tk()
        self.canvas = Canvas(tk, width=size, height=size)
        self.canvas.pack()
        self.pallet = ['white', 'black']
        self.matrix = [[self.pixel(i, j) for j in range(N)]
                       for i in range(N)]

    def pixel(self, i, j):
        rect = self.canvas.create_rectangle
        x0, x1 = i * self.unit, (i + 1) * self.unit
        y0, y1 = j * self.unit, (j + 1) * self.unit
        return rect(x0, y0, x1, y1)

    def update(self, X):
        config = self.canvas.itemconfigure
        for i in range(self.N):
            for j in range(self.N):
                c = self.pallet[X[i, j]]
                ij = self.matrix[i][j]
                config(ij, outline=c, fill=c)
        self.canvas.update()


def reverse(X, i, j):
    i0, i1 = i - 1, i + 1
    j0, j1 = j - 1, j + 1
    n, s, w, e = [X[i0, j], X[i1, j], X[i, j0], X[i, j1]]
    nw, ne, sw, se = [X[i0, j0], X[i0, j1], X[i1, j0], X[i1, j1]]

    a = X[i, j]
    b = 1 - 2 * a
    intr1 = b
    intr20 = b * sum([n, s, w, e])
    intr21 = b * sum([nw, ne, sw, se])
    intr3 = b * sum([n * ne, ne * e, e * n, e * se,
                     se * s, s * e, s * sw, sw * w,
                     w * s, w * nw, nw * n, n * w])
    intr4 = b * sum([n * ne * e, e * se * s,
                     s * sw * w, w * nw * n])
    return intr1, intr20, intr21, intr3, intr4


N = 100
#beta, J = 1.0, [-4.0, 1.0, 1.0, 0.0, 0.0]
#beta, J = 2.0, [0.0, 1.0, -1.0, 0.0, 0.0]
#beta, J = 4.0, [-2.0, 2.0, 0.0, -1.0, 2.0]
beta, J = 1.5, [-2.0, -1.0, 1.0, 1.0, -2.0]
```

```
scrn = Screen(N)
X = random.randint(0, 2, (N, N))
while True:
    for i in range(-1, N - 1):
        for j in range(-1, N - 1):
            S = reverse(X, i, j)
            p = exp(beta * dot(J, S))
            if random.uniform(0.0, 1.0) < p / (1 + p):
                X[i, j] = 1 - X[i, j]
    scrn.update(X)
```

fig. 9.9 All images on the screen with 3 pixels

fig9-9.py

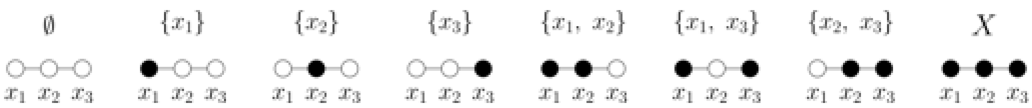


fig. 9.10 Examples of acceptance functions

fig9-10.py

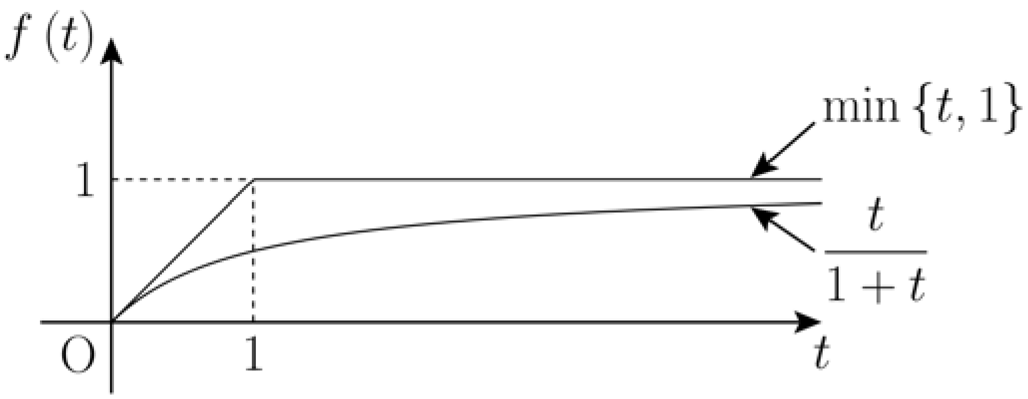


fig. 9.11 The spacial Markov property

fig9-11.py

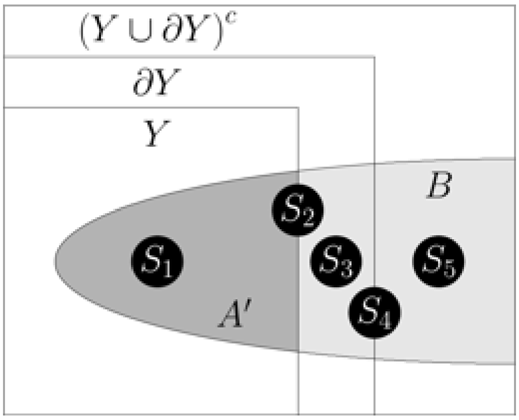


fig. 9.12 The 8-neighborgrig graph

fig9-12.py

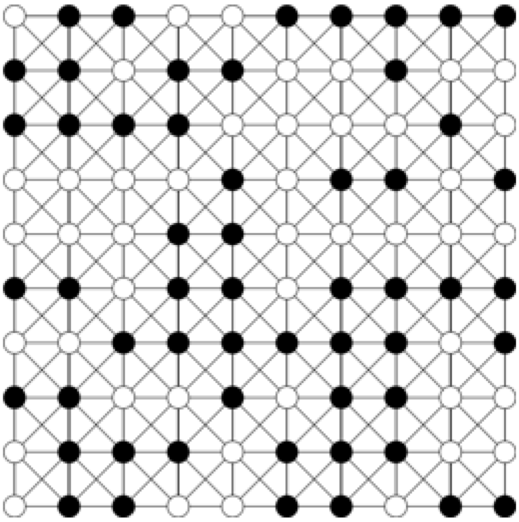


fig. 9.13 10 types of simplexes

fig9-13-1.py

fig9-13-20.py

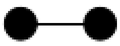


fig9-13-21.py

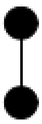


fig9-13-22.py

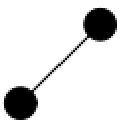


fig9-13-23.py

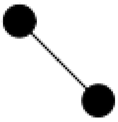


fig9-13-30.py

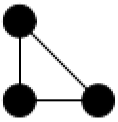


fig9-13-31.py

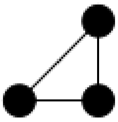


fig9-13-32.py

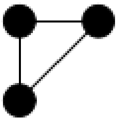


fig9-13-33.py

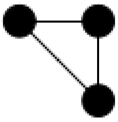
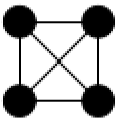
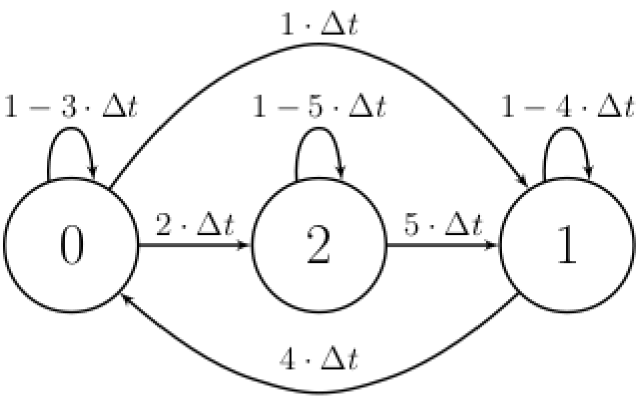


fig9-13-4.py



9.6. One-parameter semigroup and generator matrix



To draw such a graph containing $\LaTeX$ formulas, use the command `dot2tex` on Linux.

Install `dot2tex` is installed as follows:

```
sudo apt install dot2tex -y
```

Next, graph drawing instructions are written in the DOT language.

graph2.dot

```
digraph {
rankdir=LR
    node [fontname=ipag fontsize=24 shape=circle]
    0 [texlbl="\huge $0$"];
    1 [texlbl="\huge $1$"];
    2 [texlbl="\huge $2$"];
    0 -> 0 [label=" ", texlbl="\large $1-3\cdot\Delta
t$"];
    1 -> 0 [label=" ", texlbl="\large $4\cdot\Delta t$"];
    0 -> 1 [label=" ", texlbl="\large $1\cdot\Delta t$"];
    1 -> 1 [label=" ", texlbl="\large $1-4\cdot\Delta
t$"];
    2 -> 1 [label=" ", texlbl="\large $5\cdot\Delta t$"];
    0 -> 2 [label=" ", texlbl="\large $2\cdot\Delta t$"];
    2 -> 2 [label=" ", texlbl="\large $1-5\cdot\Delta
t$"];
}
```

Convert this to the $\LaTeX$ source file using `dot2tex`.

```
dot2tex graph2.dot > graph2.tex
```

Finally, typeset it.

```
pdflatex graph2.tex
```

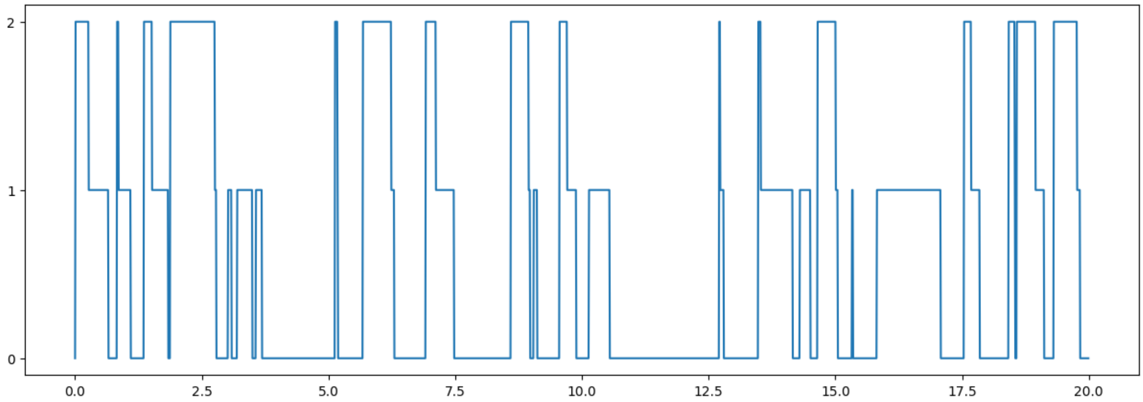
If necessary, use GIMP to remove margins or convert to other image formats.

Program: semigroup1.py**fig. 9.13 10 types of simplexes**

In [1]:
```
from numpy import arange, array, eye, exp
from numpy.random import choice, seed
from numpy.linalg import eig
import matplotlib.pyplot as plt

seed(2020)
dt, tmax = 0.01, 1000
T = arange(0.0, tmax, dt)
G = array([[-3,  4,  0], [ 1, -4,  5], [ 2,  0, -5]])
v = eig(G)[1][:, 0]
print(v / sum(v))
dP = eye(3) + G * dt

X = [0]
S = [[dt], [], []]
for t in T:
```

```
    x = X[-1]
    y = choice(3, p=dP[:, x])
    if x == y:
        S[x][-1] += dt
    else:
        S[y].append(dt)
    X.append(y)


plt.figure(figsize=(15, 5))

plt.plot(T[:2000], X[:2000])
plt.yticks(range(3))
plt.show()
```

```
[0.46511628-0.j 0.34883721-0.j 0.18604651-0.j]
```

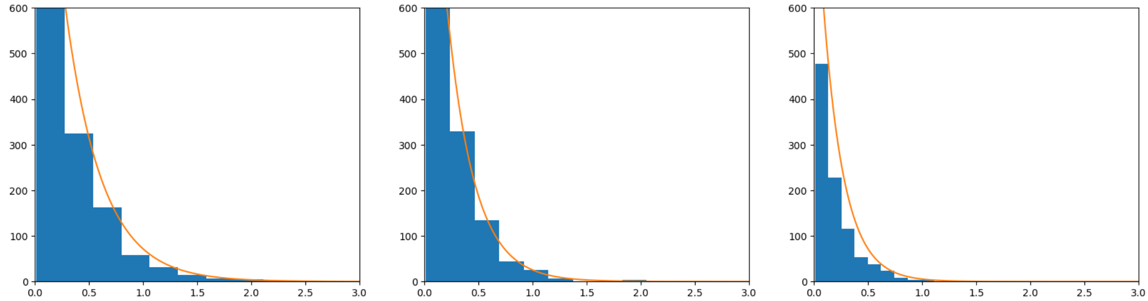


Program: semigroup2.py

In [1]:
```
from numpy import arange, array, eye, exp
from numpy.random import choice, seed
from numpy.linalg import eig
import matplotlib.pyplot as plt

seed(2020)
dt, tmax = 0.01, 1000
T = arange(0.0, tmax, dt)
G = array([[-3,  4,  0], [ 1, -4,  5], [ 2,  0, -5]])
v = eig(G)[1][:, 0]
print(v / sum(v))
dP = eye(3) + G * dt

X = [0]
S = [[dt], [], []]
for t in T:
    x = X[-1]
    y = choice(3, p=dP[:, x])
    if x == y:
        S[x][-1] += dt
    else:
        S[y].append(dt)
    X.append(y)


fig, axs = plt.subplots(1, 3, figsize=(20, 5))
for x in range(3):
```

```
    s, n = sum(S[x]), len(S[x])
    print(s / tmax)
    m = s / n
    axs[x].hist(S[x], bins=10)
    t = arange(0, 3, 0.01)
    axs[x].plot(t, exp(-t / m) / m * s)
    axs[x].set_xlim(0, 3), axs[x].set_ylim(0, 600)
```

```
[0.46511628-0.j 0.34883721-0.j 0.18604651-0.j]
0.4690599999999994
0.3469999999999995
0.18395000000000028
```



In [ ]: