Stærðfræði og reiknifræði – Skilaverkefni 10

Þetta er fyrsta skilaverkefnið með nýju sniði, tímadæmar falla niður og þarmeð tímadæmi en í staðin verða skilaverkefnin tvískipt: undirbúningsdæmi (merkt U10-a,b,c...) og skiladæmi (merkt S10-A,B,C). Lausnir verða birtar á undirbúningsdæmunum á venjulegum tíma, á fimmtudögum.

Eindregið er mælt með að þið vinnið verkefnið með því að leysa undirbúningsdæmin fyrst, áður en þið kíkið á lausnir þeirra og haldið áfram með skiladæmin. Athugið að í sumum reitum eru ein eða tvær fyrstu skipanirnar komnar til að hjálpa ykkur af stað.

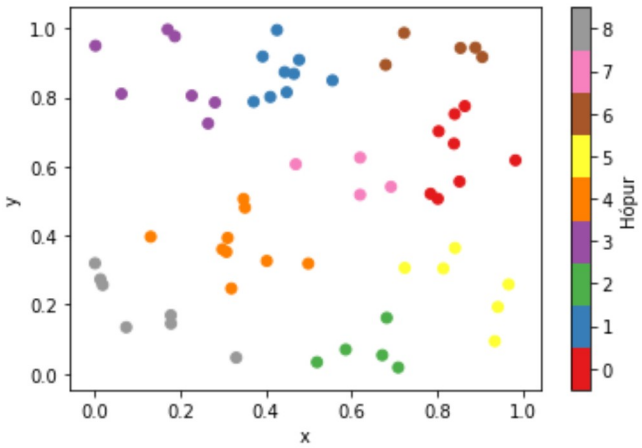
Ef þið viljið vinna saman gætuð þið prófað whereby.com (http://whereby.com).

In [1]:
```
#BYRJA -- Keyrið til að frumstilla
import numpy as np, numpy.random as npr
import scipy.stats as stat, statsmodels.api as sm, numpy.linalg as la
import matplotlib.pyplot as plt, matplotlib as mpl, struct, gzip
import matplotlib.colors as clrs
from scipy.cluster.vq import kmeans, whiten, vq
from urllib.request import urlopen
%matplotlib inline
from_list = clrs.LinearSegmentedColormap.from_list
def qcmap(n): return from_list("",plt.get_cmap('Set1')(range(0,n)),n)
plt.rc('axes', axisbelow=True);
# disp(x,y...) skrifar x,y... með 3 aukastöfum
def disp(*args): print(*(f'{a:.3f}' if isinstance(a,float) else a for a in args))
np.set_printoptions(precision=3, floatmode='fixed', suppress=True, linewidth=150)
mpl.rc('image',cmap='binary')
```

A. Flokkun með k-means

Um colorbar. Aftast í grein 5.1.6 (https://cs.hi.is/strei/kafli05.html#k-means-reikniriti) er sýnt hvernig hægt er að flokka slembipunkta með k-means og sýna hópana með aðgreinanlegum litum með `plt.scatter` . Byrjunarreiturinn að ofan skilgreinir fallið `qcmap` sem er notað í greininni. Í dæmum S4-2 og S6-C kynntumst við skipuninni `plt.colorbar` sem bætir merktri litastiku hægra megin við graf. Hér er lausn á Æfingu b í grein 5.1.6 þar sem búið er að bæta við slíkri stiku (keyrið hana):

In [2]:
```
# k-means og scatter með colorbar
X = npr.rand(60,2)
(x,y) = X.T
k = 9
(cb,d) = kmeans(X,k)
(code,dvec) = vq(X,cb)
plt.scatter(x, y, c=code, cmap=qcmap(k))
plt.xlabel('x'), plt.ylabel('y')
plt.colorbar(ticks=range(k), label='Hópur')
plt.clim(-0.5,k-0.5)
```

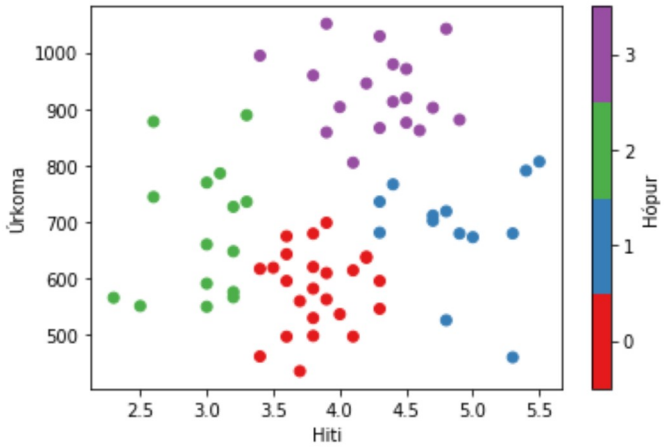


U10-a. Flokkun veðurgagna með k-means

1. Lesið skrána http://cs.hi.is/strei/hiti-urkoma.txt (http://cs.hi.is/strei/hiti-urkoma.txt) inn í þrjá vigra: ár, hiti, úrkoma (notið `urlopen` og `np.loadtxt`). Búið til n x 2 fylki úr hita og úrkomu (munið eftir `np.c_[]`), staðlið það með `whiten` og flokkið í fjóra hópa með `kmeans` og `vq` eins og í reitnum að ofan. Teiknið og bætið við colorbar.

In [15]:
```
# ua1
f = urlopen('http://cs.hi.is/strei/hiti-urkoma.txt')
(ar, hiti, urkoma) = np.loadtxt(f).T
X = np.c_[hiti,urkoma]
X = whiten(X)

k = 4
[cb,d] = kmeans(X,k)
[code,dvec] = vq(X,cb)
plt.scatter(hiti, úrkoma, c=code, cmap=qcmap(k))
plt.xlabel('Hiti'), plt.ylabel('Úrkoma')
plt.colorbar(ticks=range(k), label='Hópur')
plt.clim(-0.5,k-0.5)
```



S10-A. Flokkun Wikipediugreina

Í þessu verkefni á að flokka Wikipediugreinar með *k-means*. Unnið er með sömu gögn og í S7-C. Hér er fyrst reitur með falli sem nær í skrána ef þarf, les hana inn, og skilar 1000 staka vigri `orð` með orðunum, 300-staka vigri `grein` með titlum greinanna og 300 $\times$ 1000 fylki `X` með orðtíðni.

In [26]:
```
from os.path import exists
def lesawiki():
    skrá = 'wikitidni.npz'
    if not exists(skrá):
        print('næ í skrá...')
        fin = urlopen('https://cs.hi.is/strei/' + skrá)
        fout = open(skrá,'wb')
        fout.write(fin.read());
    data = np.load(skrá)
    data.allow_pickle = True
    orð = data["dictionary"]
    grein = data["article_titles"]
    X = data["article_histograms"]
    return orð,grein,X
```

1. Lesið gögnin og flokkið þau í $k = 2, 3,...,10$ hópa. Finnið gildi markfallsins fyrir hvern hópafjölda og teiknið graf af niðurstöðunni (merkið ása o.s.frv.). Við mundum búast við að fá umtalsvert lægra gildi fyrir $k \geq 5$ því vitum að greinarnar fjalla um 5 mismunandi efnisflokka. Rætist sú spá?

In [57]:
```
#A1
npr.seed(7)

tal = [None]*9

for i in range(2, 10):
    (orð, grein, X) = lesawiki()
    k = i
    [cb,d] = kmeans(X,k)
    [code,dvec] = vq(X,cb)
    tal[i-2] = X




(x,y) = cb.T
plt.plot( 'x', 'y', data=tal, linestyle='-', marker='o')

plt.xlabel('bottom'), plt.ylabel('top')

plt.show()

### Klára seinna
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-57-cf52d1884cb4> in <module>
     14
     15
---> 16 (x,y) = cb.T
     17 plt.plot( 'x', 'y', data=tal, linestyle='-', marker='o')
     18

ValueError: too many values to unpack (expected 2)
```

1. Finnið hóp hverrar greinar með `vq` þegar $k = 5$. Skrifið út töflu með hópnúmeri, fjarlægð frá miðpunkti, og efnisflokki fyrir 15. hverja grein. Taflan á að byrja svona (þið gætuð samt fengið önnur hópnúmer):

```
HÓPUR  FJARL.MIÐP.   EFNISFLOKKUR
  3       0.88       A Bar at the Folies-Bergère
  0       0.87       Atmosphere of Earth
```

   Efnisflokkarnir eru: *listir, fjarskipti, veðurfræði, pokemon, SÞ-stofnanir*. Hvert er hópnúmer hvers efnisflokks.

In [ ]:
```
#A2
npr.seed(7)
(cb,d) = kmeans(X,5)
```

1. Ákvarðið og skrifið töflu yfir hópnúmer, fjölda greina í hópi og efnisflokk þegar $k = 5$. Hafið fyrirsagnir yfir dálkum töflunnar.

In [ ]:
```
#A3
efnisflokkar = ["Veðurfræði",...
```

1. Finnið meðalfjarlægð greina hvers hóps frá miðpunkti hópsins (fyrir hóp nr. 0 a að koma út 0.939).

In [ ]:
```
#A4
```

B. MNIST gagnasafnið

MNIST (M = *Modified* og NIST = *National Institute of Standards and Technology*), er safn af myndum af alls 70000 handskrifuðum tölustöfum, sem skiptast í 60000 þjálfunarmyndir (*training images*) og 10000 prófunarmyndir (*test images*). Þetta safn er mikið notað til að prófa og þróa ýmis myndvinnslukerfi. Eitt af því sem algengast er að gera með safnið er að láta kerfin þekkja tölustafina með sem lægstri villutíðni. Ástæðan fyrir skiptingunni í *train* og *test* er að geta þjálfað kerfi og prófað það í framhaldinu með gögnum sem ekki hafa komið við sögu í þjálfuninni.

Náð í gögnin: Gögnin eru í fjórum skrám sem ég náði í beint frá heimasíðu safnsins (http://yann.lecun.com/exdb/mnist/). Þær eru á heimasvæðinu cs.hi.is/strei og heita

```
mnist-train.gz
mnist-train-label.gz
mnist-test.gz
mnist-test-label.gz
```

Hér er reitur sem afritar skrárnar yfir í núverandi svæði (það tekur nokkrar sekúndur að keyra hann):

In [68]:
```
from os.path import exists
def náiskrá(skrá):
    """Nær í skrá frá cs.hi.is/strei"""
    if exists(skrá): return
    fin = urlopen('https://cs.hi.is/strei/' + skrá)
    fout = open(skrá,'wb')
    fout.write(fin.read());

for f in ['train','train-label','test','test-label']:
    náiskrá('mnist-' + f + '.gz')
```

Gögn lesin inn: Skrárnar eru á svonefndu idx-gzip-sniði og í næsta reit er skilgreint fall sem getur lesið þær.

In [69]:
```
def lesa_igz(skráarnafn):
    "Skilar fylki með gögnum úr skrá með idx-gzip-sniði"

    with gzip.open(skráarnafn, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)
```

U10-b. Náð í gagnasafn og það skoðað

1. **MNIST á Wikipedíu:** Byrjið á að skoða lýsingu á safninu (https://en.wikipedia.org/wiki/MNIST_database) á Wikipediu. Stækkið m.a. myndina með sýnishornunum, finnið út stærð hverrar myndar í dílum (*pixels*) og hve lágri villutíðni menn hafa náð.

```
#ub1
Stærð í dílum: 28 x 28
Besta villutíðni:
```

1. **Lesið gögn:** Náið í skrárnar og lesið `mnist-train.gz` og `mnist-train-label.gz` inn í fylki `train` og `label` og hinar tvær inn í `test` og `tstlbl`. Skrifið út stærðir fylkjanna (með `np.shape`), stök nr. 0–9 í `label` og stak nr. 0 í `train` (með `disp`).

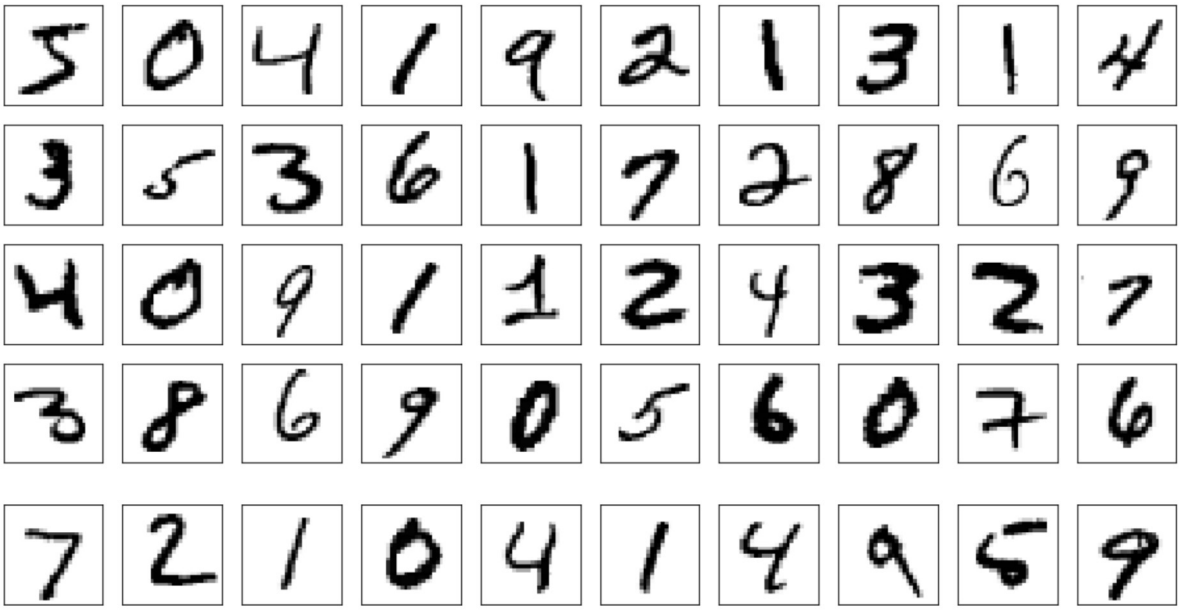
In [71]:
```
#ub2
train = lesa_igz('mnist-train.gz')
#ub2
train = lesa_igz('mnist-train.gz')
test = lesa_igz('mnist-test.gz')
label = lesa_igz('mnist-train-label.gz')
tstlbl = lesa_igz('mnist-test-label.gz')
print(np.shape(train), np.shape(test))
print(np.shape(label), np.shape(tstlbl))
print(label[:10])
disp(train[0])
```

```
(60000, 28, 28) (10000, 28, 28)
(60000,) (10000,)
[5 0 4 1 9 2 1 3 1 4]
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  2
6 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 17
2 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  8
2  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  2
7   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 18
7   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 24
9  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 20
7   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 18
2   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0]]
```

1. **Skoðið gögn:** Í næsta reit er fall til að sýna nokkrar tölustafamyndir í einu. Hægt er að kalla á þetta fall með `sýnamyndir(listi)` þar sem listi er listi eða vigur af myndum (t.d. `sýnamyndir(train[0:4])`) og þá teiknar það myndirnar í samsettri mynd (*figure*), 10 myndir í hverri línu. Teiknið fyrstu 40 myndirnar í `train` og fyrstu 10 í `test` .

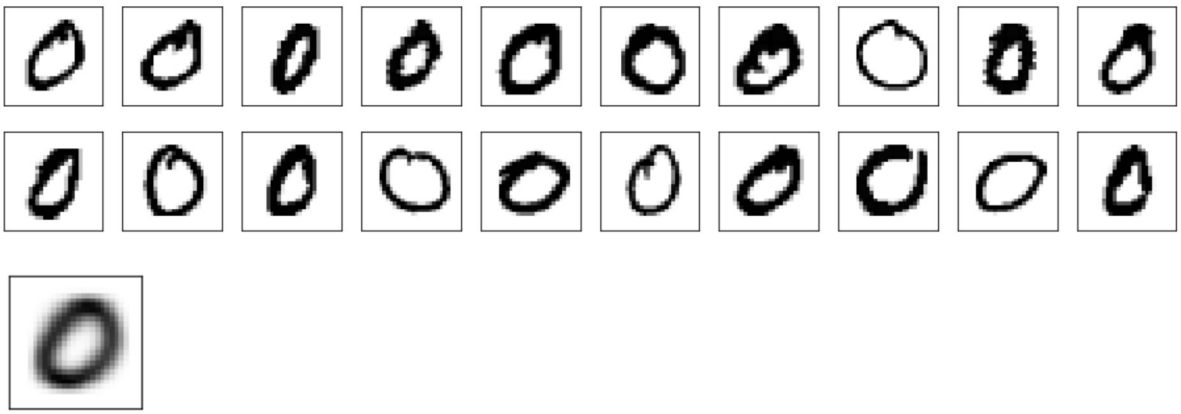
In [73]:
```
#ub3
def sýnamyndir(list, n=10, width=16):
    """Birtir myndir list[0], list[1]... í samsettri mynd, n í hverri línu"""
    list = np.reshape(list, (-1, 28, 28))
    fjöldi = len(list)
    m = (fjöldi-1)//n + 1
    plt.figure(figsize=(width, width*m/n))
    for i in range(fjöldi):
        plt.subplot(m, n, i+1)
        plt.imshow(list[i])
        plt.xticks([]); plt.yticks([])

sýnamyndir(train[:40])
sýnamyndir(test[:10])
```



1. **Meðaltölustafur:** Ein leið til að búa til reiknirit til að þekkja tölustafi er að byrja með **miðpunkt** (*centroid*) fyrir hvern tölustaf, sem fæst með því að reikna meðaltal af öllum myndum af stafnum, í anda þess sem gert er í *k-means*. Með því að skrifa t.d. `A = train[label==0]` fær maður í `A` allar myndir af núllum (þau hafa label 0) og svo má finna miðpunktinn með `m0 = np.mean(A,0)` (*mean* yfir vídd 0). Teiknið 20 fyrstu núllin í `train` og svo meðalnúllið.

In [62]:
```
#ub4
A = train[label==0]
m0 = np.mean(A,0)
sýnamyndir(A[:20])
sýnamyndir(m0)
```



1. **Fylki breytt í vigur** Hér fylgir fall sem breytir hverri tölustafamynd í vigur af taginu *float*. Það gerir kleyft að nota `kmeans` - og `vq` -föllin á MNIST-gögnin. Skoðið gagnatag og stærð fyrstu 5 tölustafanna í `train` , með því setja `t5 = train[:5]` og skoða `t5.type` og `t5.shape` . Beitið svo `vectorize` á `t5` og finnið aftur gagnatag þess og stærð.

In [70]:
```
#ub5
def vectorize(x):
    """Breytir hverri mynd í x í float vigur með stökum í [0,1]
    Ef shape(x) er (k,28,28) er skilað fylki með k línum
    og 784 dálkum (28*28=784)"""
    if np.ndim(x) == 2: x = np.reshape(x, (1,28,28))
    return np.reshape(x, (len(x),28*28)).astype(float)/255
```

1. **Fjarlægðir frá meðaltölustaf:** Sendið öll þjálfunargögnin og líka meðalnúllið í gegn um vectorize (t.d. með `X = vectorize(train)` og `y0 = vectorize(m0)` . Þá er hægt að mæla fjarlægð milli hvers tölustafs og meðalnúllsins með því að nota `la.norm` , t.d.

```
d0 = la.norm(X[0] - y0)
d1 = la.norm(X[1] - y0)
d0,d1
```

sem ætti að gefa mun hærri tölu í `d0` en í `d1` (því fyrsti tölustafurinn í train er "5" en sá næsti er "0", sbr. dæmi b3). Finnið fjarlægð meðalnúllsins frá fyrstu 40 þjálfunartölustöfunum og notið myndina sem fékkst í b3 til að athuga hvort núllin séu nær meðalnúllinu en hinnir tölustafirnir.

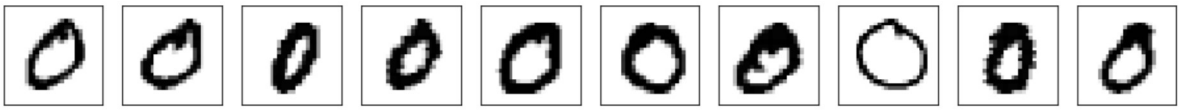
In [64]:
```
#ub6
y0 = vectorize(m0)
X = vectorize(train)
d0 = la.norm(X[0] - y0)
d1 = la.norm(X[1] - y0)
d = np.zeros(40)
for i in range(40):
    d = la.norm(X[i] - y0)
    if d<7.8: disp(i,d,'*')
    else: disp(i,d)
```

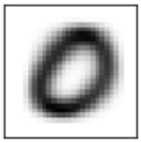
```
0 8.133
1 5.266 *
2 9.199
3 9.187
4 8.721
5 8.321
6 9.192
7 8.437
8 8.748
9 8.934
10 8.452
11 8.870
12 9.562
13 8.253
14 8.786
15 8.372
16 8.326
17 8.695
18 7.843
19 8.800
20 10.174
21 5.922 *
22 8.576
23 9.157
24 8.210
25 8.572
26 8.656
27 9.274
28 9.764
29 8.858
30 9.502
31 9.196
32 8.034
33 8.891
34 7.687 *
35 8.746
36 9.198
37 6.644 *
38 9.100
39 8.077
```

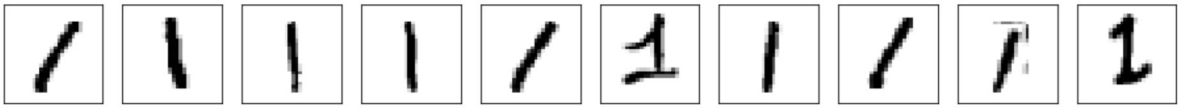
S10-B. Tölustafir þekktir

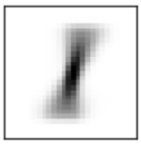
1. **Meðaltölustafir:** Lesið inn þjálfunargögnin (keyrið reit b2). Búið til lista `M` með 10 myndum, þannig að `M[i]` sé meðaltalsmynd af tölustafnum `i` (sbr. b4). Teiknið myndirnar með `sýnamyndir`.
   Ath: Það eru margar leiðir til að búa til M, t.d. append fallið sbr. grein 1.6.4 (https://cs.hi.is/strei/kafli01.html?highlight=comprehension#helstu-foll-sem-aeins-duga-a-lista), comprehension sbr. grein 1.6.9 (https://cs.hi.is/strei/kafli01.html?highlight=comprehension#yfirgrip-comprehension), og svo má líka byrja á að nota `np.zeros` til að búa til fylki af stærð (10,28,28).

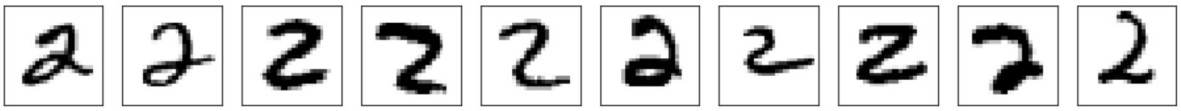
In [83]:
```
#B1
for i in range(0,10):
    A = X[label==i]
    m0 = np.mean(A,0)
    sýnamyndir(A[:10])
    sýnamyndir(m0)
```

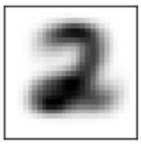


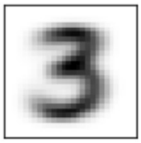


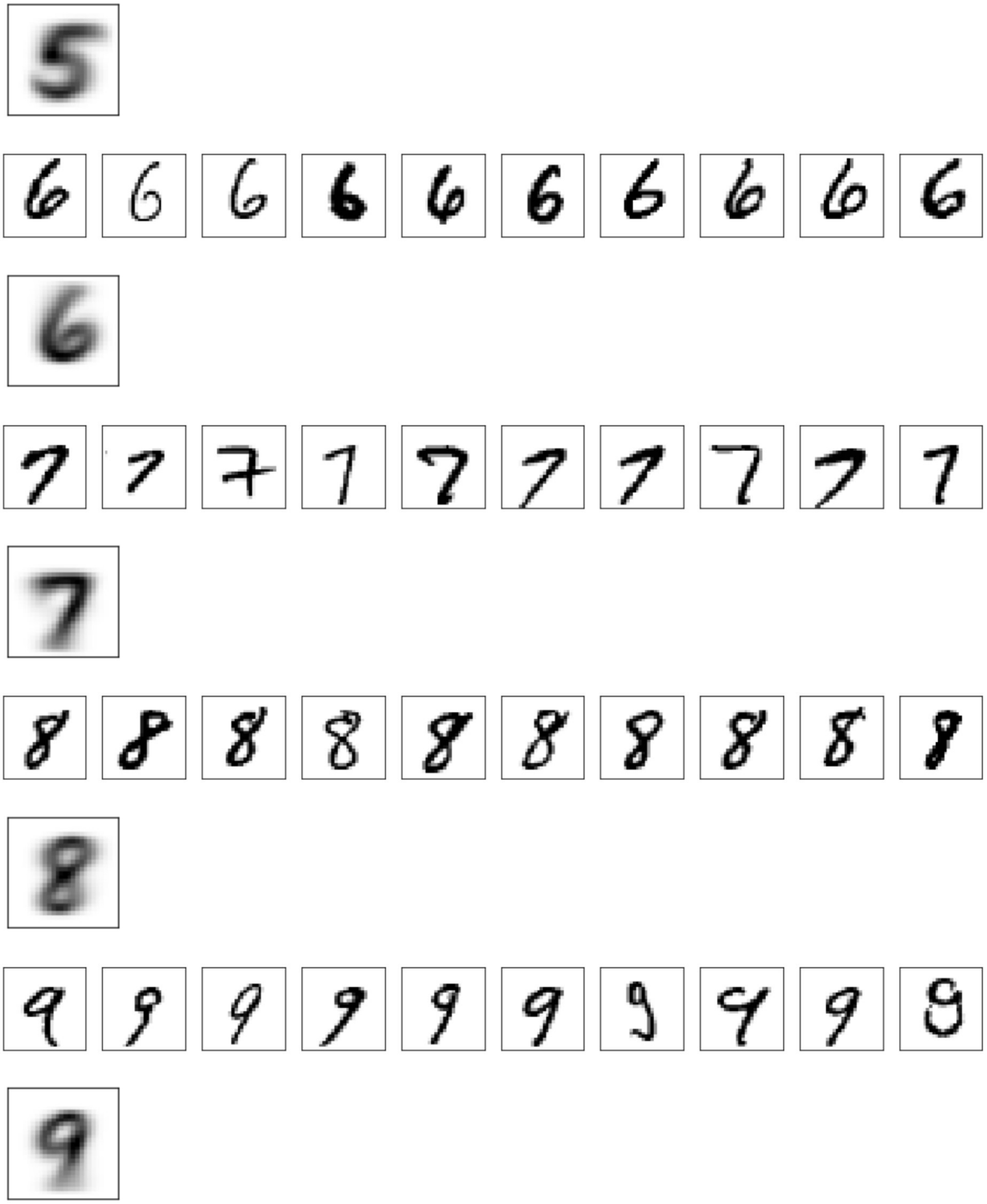












1. **Flokkur fyrstu fimmunnar:** Búið til `X` með því að breyta hverjum tölustaf í `train`-gögnunum í vigur með `vectorize`, og sömuleiðis `cb` úr `M`-listanum. Fylkið sem búið er til úr `M` mun gegna hlutverki *code-book* í næsta lið. Í svarreitnum er búið að búa til bæði fylki `X` úr `train`-fylkinu og fylki `cb` úr `M`-listanum. Hvaða víddir hafa X og cb? Finnið svo fjarlægð `X[0]` frá öllum myndunum í `M` með `la.norm(X[0] - cb, axis=1)` og skoðið hvaða mynd er næst `X[0]`. Passar niðurstaðan?

In [ ]:
```
#B2
X = vectorize(train)
cb = vectorize(M)
```

1. **Flokkun með vq:** Notið nú `vq`-fallið til að flokka test-gögnin, þannig að hver mynd sé sett í flokk með þeim miðpunkti sem er næstur henni miðað við Evklíðska fjarlægð (þ.e. normið af mismun vigranna sem svara til myndarinnar og miðpunktsins). Ákvarðið hve stórt hlutfall er rangt flokkað (ef `code` er flokkunin sem kemur út úr `vq` þá gefur `sum(code != tstlbl)` fjölda mynda sem flokkast rangt). Teiknið líka úrval rangt flokkaðra mynda (þær fást með `test[code != tstlbl]`)

In [ ]:
```
#B3
```

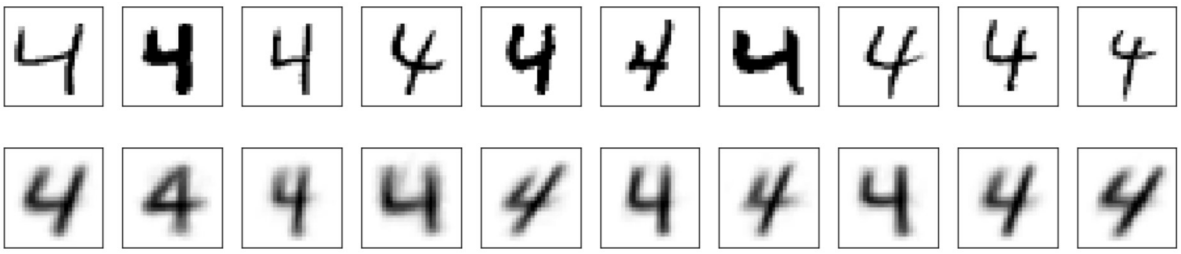
C. Mismunandi gerðir af hverjum tölustaf

Einn möguleiki til að ná betri árangri við flokkunina er að átta sig á að það eru nokkrir flokkar af hverjum tölustaf, T.d. 1 með hallandi striki efst og án, 7 með striki og án, 9 með beinu striki og bognu, 4 sem lokast efst eða ekki. Einnig eru lóðréttar tölur og tölur sem hallast til hægri. Til að finna þessa flokka má nota *k-means* aðferð á allar myndir af hverjum staf. Í framhaldi má búa til nýja "code-book" með öllum flokkunum fyrir allar tölurnar.

U10-c. Mismunandi fjarkar

1. Náið í alla fjarkana með `X4 = X[label==4]`. Til að `kmeans` verði ekki of lengi að keyra má fækka fjörkunum t.d. með `X4 = np.copy(X4[::5])`. Notið svo `kmeans` til að finna 10 fjarkaflokka og teiknið miðpunktana sem finnast.

In [66]:
```
#uc1
X4 = X[label==4]
X4 = np.copy(X4[::5])
sýnamyndir(X4[:10])
k = 10
[cb,d] = kmeans(X4,k)
sýnamyndir(cb)
```



1. Búið til fall `undirflokkar` úr skipununum í lið 1 sem býr til *code-book* fyrir tölustafinn i með 10 flokkum. T.d. mundi `cb9 = undirflokkar(9)` skila fylki með 10 línum og 784 dálkum með allskonar mismunandi útgáfum af tölustafnum 9.

In [67]:
```
#uc2
def undirflokkar(t):
    Xt = X[label==t]
    Xt = np.copy(Xt[::5])
    k = 10
    [cb,d] = kmeans(Xt,k)
    return cb
cb9 = undirflokkar(9)
print(cb9.shape)
sýnamyndir(cb9)
```

```
(10, 784)
```

S10-C Betri flokkun á MNIST

1. Hér er reiknirit til að flokka alla 10 tölustafina í 10 undirflokka og búa til *kóðabók* með öllum 100 flokkunum sem út úr því koma.

```
cb100 := fylki með 0 línum og 28 x 28 dálkum (stærð (0,28,28))
fyrir i=0,...,9:
    skrifa 'bý til undirflokka fyrir ', i
    uf := undirflokkar(i)
    setjum uf neðst í cb100
```

Til að bæta uf neðan á cb100 má nota `np.r_[cb100, uf]` .

Þýðið þetta reiknirit yfir í Python og keyrið. Kallið á `sýnamyndir(cb100)` til að skoða niðurstöðuna.

In [92]:
```
#C1
def synamyndir(nr):
    for i in range(0,9):
        print('Bý til undirflokka fyrir', i)
        uf = undirflokkar(i)
        np.r_[nr, uf]

cb100 = np.zeros((28,28))
synamyndir(cb100)
```

```
Bý til undirflokka fyrir 0

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-92-d1b0808d0a69> in <module>
      7
      8 cb100 = np.zeros((28,28))
----> 9 synamyndir(cb100)

<ipython-input-92-d1b0808d0a69> in synamyndir(nr)
      4         print('Bý til undirflokka fyrir', i)
      5         uf = undirflokkar(i)
----> 6         np.r_[nr, uf]
      7
      8 cb100 = np.zeros((28,28))

d:\Users\Sigga\Anaconda3\lib\site-packages\numpy\lib\index_tricks.py in __getite
m__(self, key)
    402                 objs[k] = objs[k].astype(final_dtype)
    403
--> 404         res = self.concatenate(tuple(objs), axis=axis)
    405
    406         if matrix:

ValueError: all the input array dimensions except for the concatenation axis mus
t match exactly
```

1. Notið nú *kóðabókina* `cb100` úr lið C1 til að flokka allar myndirnar í prófunarmenginu. Ef við notum `vq` til þess fást flokkar á bilinu 0–99, flokkar 0-9 eru núllin, flokkar 10-19 eru ásarnir o.s.frv., svo það þarf að deila með 10 í flokkinn sem kemur úr `vq`, t.d. með `spáð = code//10`. Hvert verður hlutfall rangt flokkaðra mynda nú?

In [ ]:
```
#C2
```

D. Hvernig gekk

Skrifið örfá orð um hvernig gekk.

In [ ]:
```
Fluttningavika, gerði eins og ég gat en mjög lítið fókus til að virkilega lesa allt
í gegn.
- sos42
```