Случайные процессы. Практическое задание 6

- Дедлайн **26 ноября 23:59** (13 дней на выполнение).
- Внимательно прочтите правила оформления. Задания, оформленные не по правилам, могут быть проигнорированы.
- В коде могут встречаться пропуски, которые обычно обозначаются так: <пояснение>

Для выполнения задания потребуются следующие библиотеки: hmmlearn, librosa. Следующими командами можно их поставить (Ubuntu):

sudo pip3 install hmmlearn

sudo pip3 install librosa

В случае возникновения проблем пишите на почту.

---

1. Скрытые марковские модели

Теоретический материал по скрытым марковским моделям будет рассказан на лекции 14 ноября (в день выдачи задания). Реализация методов является полезной, но технически сложной, поэтому мы воспользуемся готовой реализацией `hmmlearn`. Документация http://hmmlearn.readthedocs.io/ (http://hmmlearn.readthedocs.io/). Интерфейс данной библиотеки максимально близок к библиотеке `scikit-learn`, которая в следующем семестре у вас будет основной библиотекой в курсе машинного обучения.

Все необходимые комментарии по интерфейсу библиотеки `hmmlearn` приведены в коде далее. Следуйте указаниям.

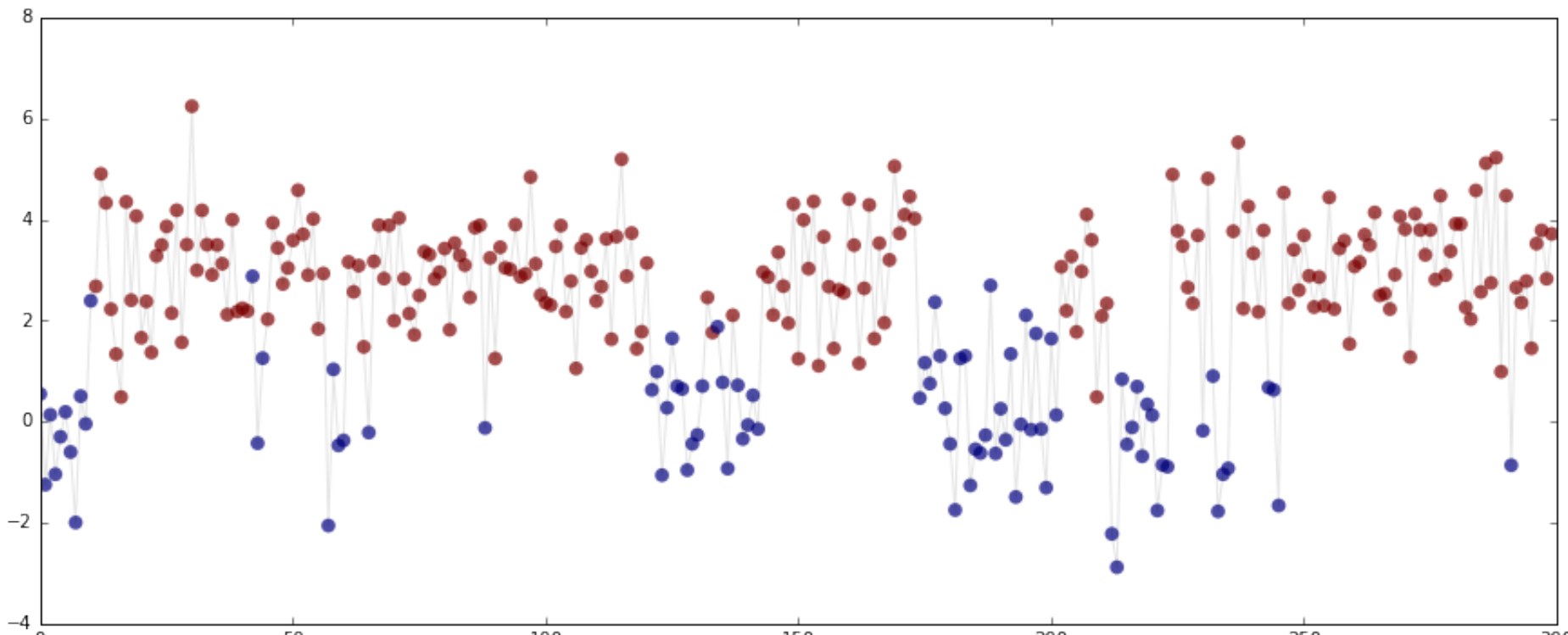
In [5]:
```
import numpy as np
import sys
from hmmlearn import hmm

import matplotlib.pyplot as plt
%matplotlib inline
```

Зададим некоторую скрытую марковскую модель

In [6]:
```
# Объявление скрытой марковской модели с двумя скрытыми состояниями,
# в которой предполагается, что каждое состояние может генерировать
# гауссовский случайный вектор с произвольной матрицей ковариаций.
# Используется метод Витерби.
# Поставьте 'map', чтобы использовать метод forward-backward.
model = hmm.GaussianHMM(n_components=2, covariance_type='full',
                        algorithm='viterbi')

# Параметры марковской цепи - начальное состояние и матрица переходных вероятностей
model.startprob_ = np.array([0.6, 0.4])
model.transmat_ = np.array([[0.9, 0.1],
                            [0.07, 0.93]])

# Параметры условных распределений Y_j при условии X_j - вектора средних и
# матрица ковариаций по количеству состояний. Поскольку в данном случае
# распределения одномерные, ниже записаны два вектора размерности 1
# и две матрицы размерности 1x1
model.means_ = np.array([[0.0], [3.0]])
model.covars_ = np.array([[[1]], [[1]]])
```

Сгенерируем некоторую последовательность с помощью определенной выше модели.

In [7]:
```
size = 300
Y, X = model.sample(size)  # Y наблюдаемы, X скрытые

plt.figure(figsize=(15, 6))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=np.array(X), lw=0, s=60, alpha=0.7)
plt.xlim((0, size))
plt.show()
```

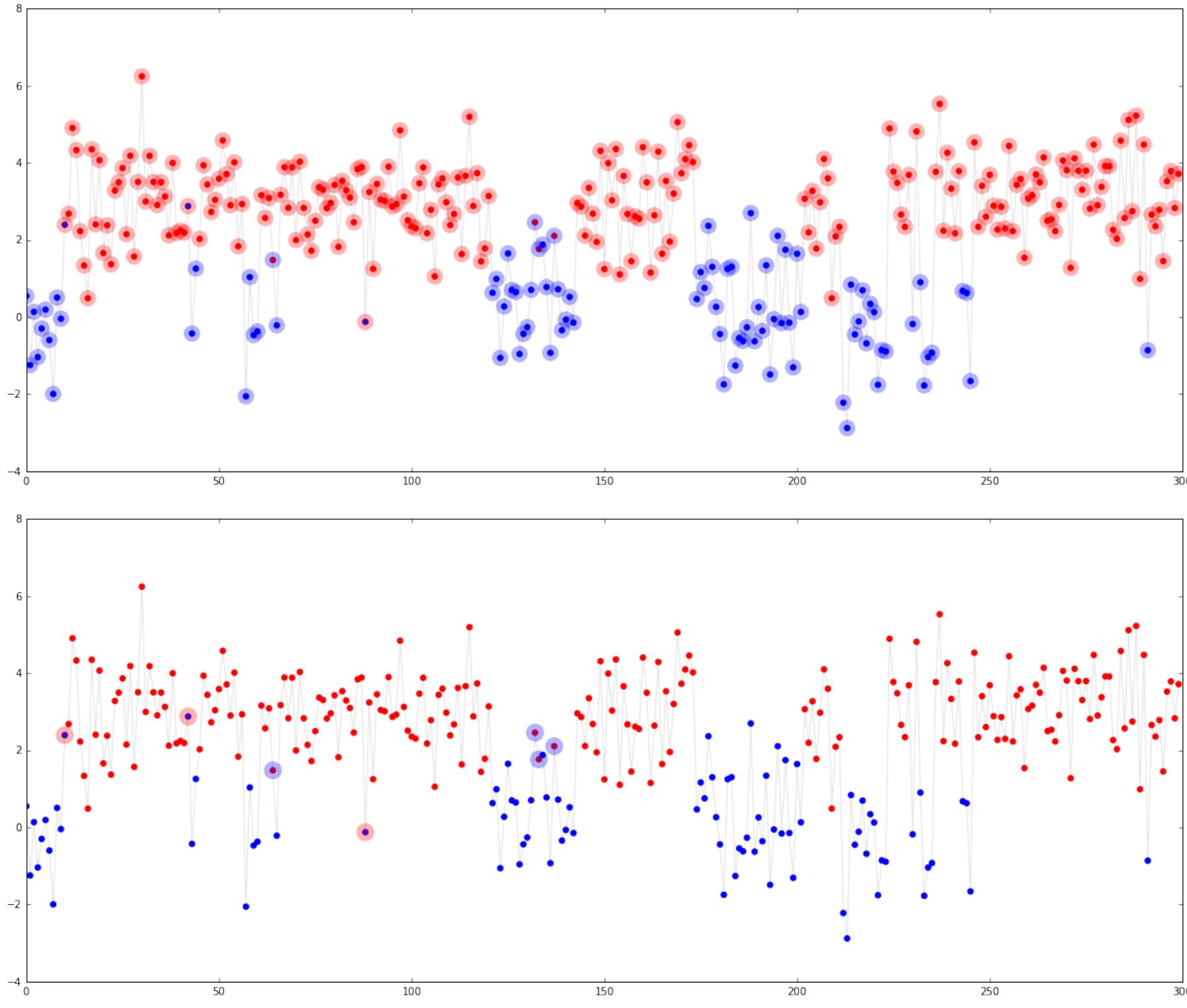


На основе сгенерированной выше последовательности оценим параметры ("обучим") скрытой марковской модели и значения скрытых состояний.

In [8]:
```
# Объявление скрытой марковской модели, в которой при оценке параметров
# будет производиться не более n_iter итераций EM-алгоритма.
remodel = hmm.GaussianHMM(n_components=2, covariance_type="full",
                          n_iter=100, algorithm='viterbi')

# Оценка параметров ("обучение")
remodel.fit(Y)

# Оценка ("предсказание") значений скрытых состояний
X_predicted = remodel.predict(Y)
```

Теперь изобразим полученные результаты. На обоих графиках непрозразными маленькими кружочками отмечена исходная последовательность. Полупрозрачными большими кружочками отмечены оценки значений скрытых состояний. На первом графике отмечены все такие точки, на втором только те из них, оценка значения скрытого состояния которых получилась неправильно.

In [9]:
```
colors = np.array(['blue', 'red'])

# Состояния определяются с точностью до их перестановки.
# При необходимости меняем местами состояния
if (X != X_predicted).sum() > size / 2:
    X_predicted = 1 - X_predicted

plt.figure(figsize=(20, 8))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X)],
            lw=0, s=40, alpha=1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X_predicted)],
            lw=0, s=250, alpha=0.3)
plt.xlim((0, size))
plt.show()

plt.figure(figsize=(20, 8))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X)],
            lw=0, s=40, alpha=1)
plt.scatter(np.arange(size)[X != X_predicted], Y[:, 0][X != X_predicted],
            c=colors[np.array(X_predicted)[X != X_predicted]],
            lw=0, s=300, alpha=0.3)
plt.xlim((0, size))
plt.show()
```

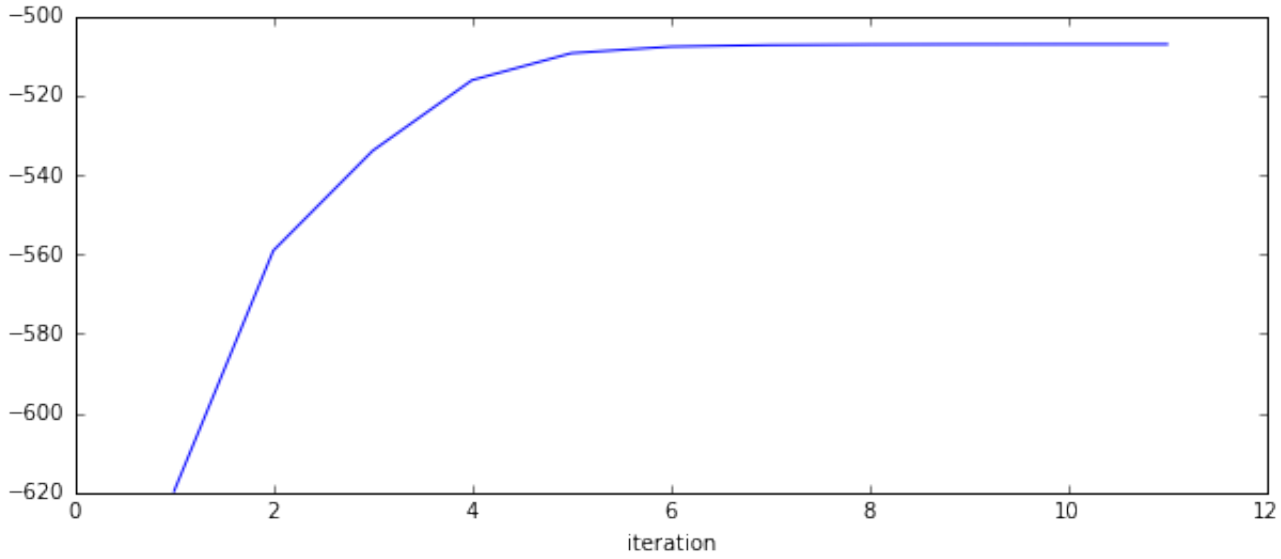


Как понять, что EM-алгоритм сошелся? Для этого нужно посчитать значение некоторого функционала, который является чересчур громоздким. Поэтому мы всего лишь посмотрим на его значения. Данная функциональность в библиотеке реализованна слишком странно. Следуйте коммантариям.

In [10]:
```
saved_strerr = sys.stderr  # сохраним в переменную поток вывода ошибок
sys.stderr = open('est_values.txt', 'w')  # и перенаправим его в файл

# ============
# Для вывода значений функционала нужно поставить параметр verbose
remodel = hmm.GaussianHMM(n_components=2, covariance_type="full",
                          n_iter=100, verbose=True)
remodel.fit(Y)
X_predicted = remodel.predict(Y)
# ============

# Возвращаем все, как было
sys.stderr = saved_strerr
```

Теперь можно загрузить значения и построить график

In [11]:
```
values = np.loadtxt('./est_values.txt')

plt.figure(figsize=(10, 4))
plt.plot(values[:, 0], values[:, 1])
plt.xlabel('iteration')
plt.show()
```

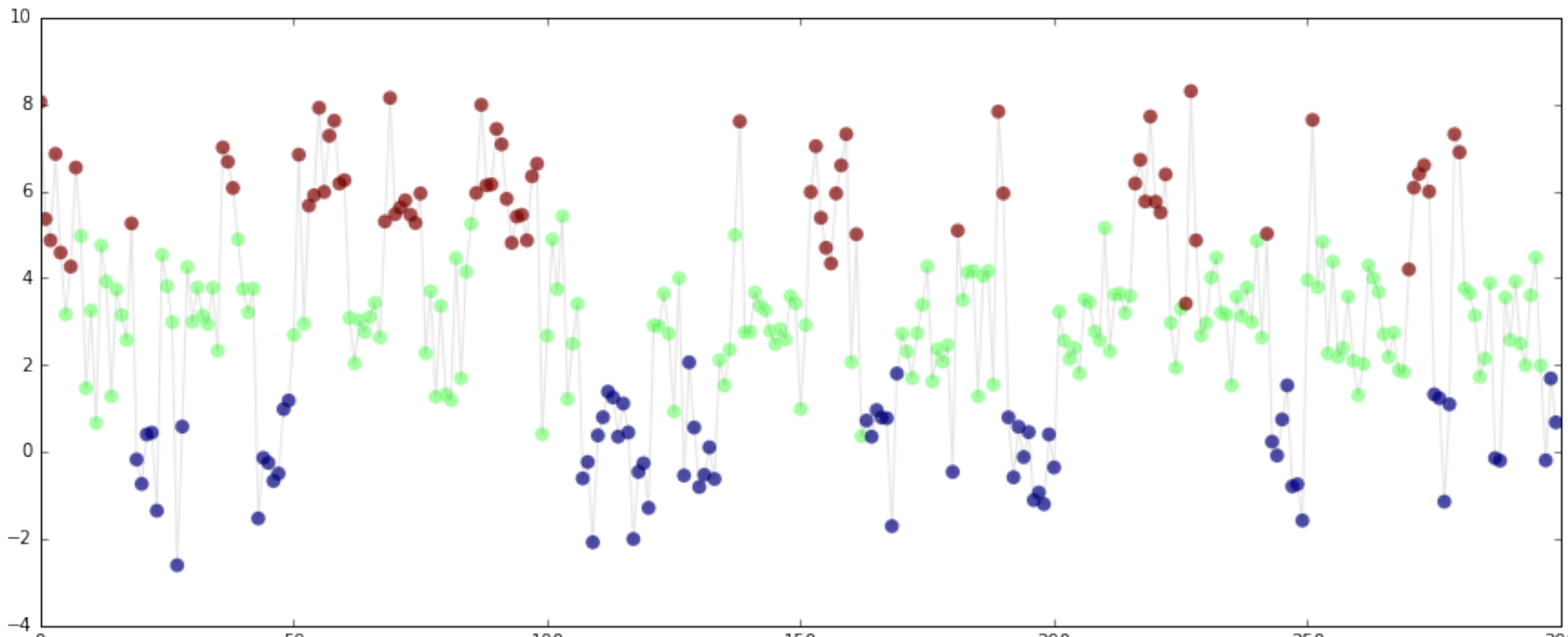


Выполните те же операции для следующих двух случаев

- скрытая марковская цепь имеет три скрытых состояния;
- распределение $Y_j \mid$ при условии $X_j \mid$ является двумерным гауссовским.

In [45]:
```
# Объявление скрытой марковской модели с ТРЕМЯ скрытыми состояниями,
# в которой предполагается, что каждое состояние может генерировать
# двумерный гауссовский случайный вектор с произвольной матрицей ковариаций.
# Используется метод Витерби.
# Поставьте 'map', чтобы использовать метод forward-backward.
model = hmm.GaussianHMM(n_components=3, covariance_type='full',
                        algorithm='viterbi')

# Параметры марковской цепи - начальное состояние и матрица переходных вероятностей
model.startprob_ = np.array([0.3, 0.3, 0.4])
model.transmat_ = np.array([[0.9, 0.05,0.05],
                            [0.1, 0.8 , 0.1],
                            [0.1, 0.2, 0.7]])

# Параметры условных распределений Y_j при условии X_j - вектора средних и
# матрица ковариаций по количеству состояний. Поскольку в данном случае
# распределения одномерные, ниже записаны три вектора размерности 2
# и три матрицы размерности 2x2
model.means_ = np.array([[0.0,0.0], [3.0,3.0] , [6.0,6.0]])
model.covars_ = np.array([[[1,0],[0,1]], [[1,0],[0,1]], [[1,0],[0,1]]])
```

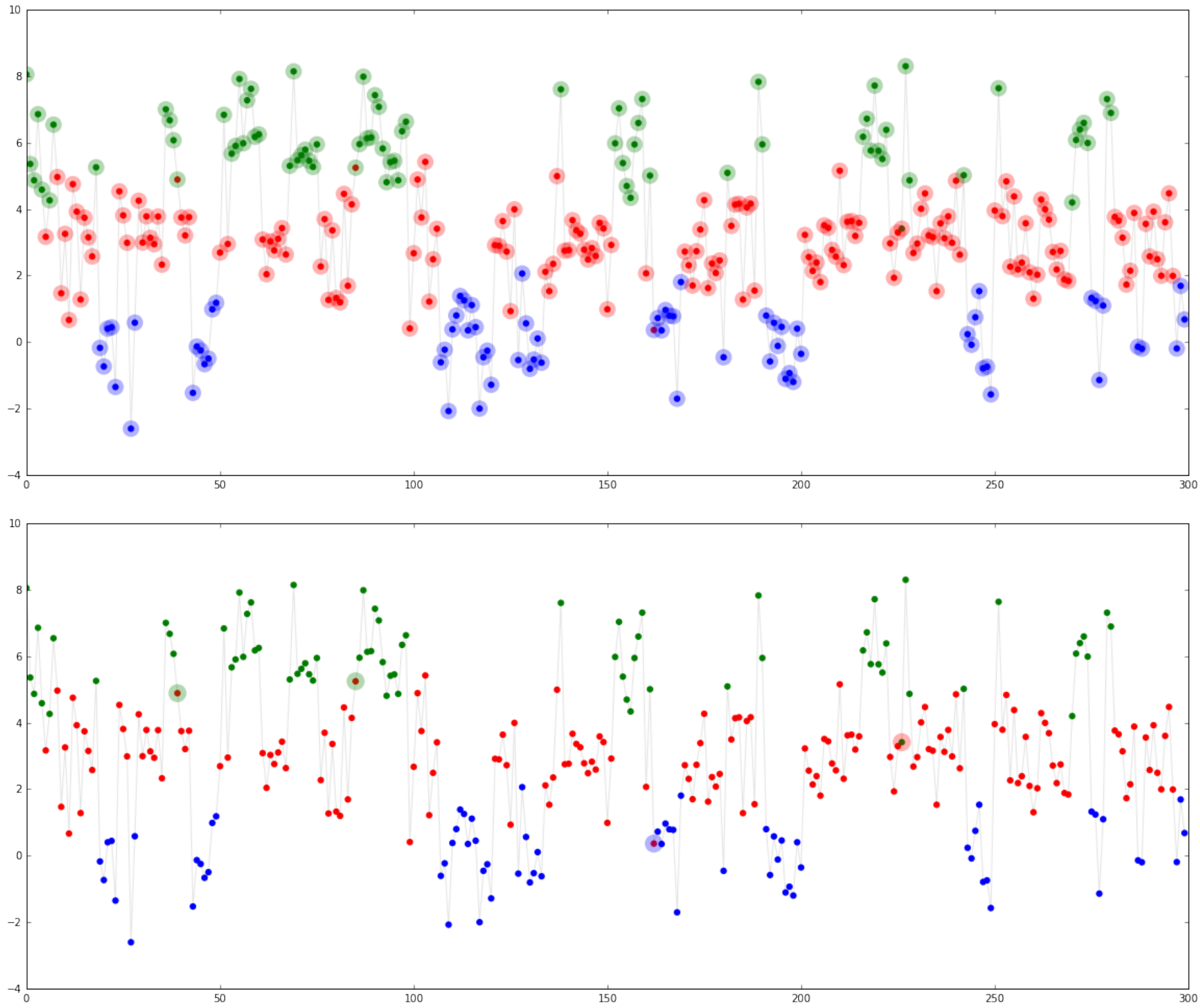
In [46]:
```
size = 300
Y, X = model.sample(size)  # Y наблюдаемы, X скрытые

plt.figure(figsize=(15, 6))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=np.array(X), lw=0, s=60, alpha=0.7)
plt.xlim((0, size))
plt.show()
```



In [47]:
```
# Объявление скрытой марковской модели, в которой при оценке параметров
# будет производиться не более n_iter итераций EM-алгоритма.
remodel = hmm.GaussianHMM(n_components=3, covariance_type="full",
                          n_iter=100, algorithm='viterbi')

# Оценка параметров ("обучение")
remodel.fit(Y)

# Оценка ("предсказание") значений скрытых состояний
X_predicted = remodel.predict(Y)
```

In [48]:
```
def find_permutation(X,X_predicted):
    perm = [[0,1,2], [0,2,1], [2,0,1], [1,0,2],[1,2,0],[2,1,0]] # all permutaions
    error = 1e9
    res = []
    for p in perm:
        cur_x = [p[x] for x in X_predicted]
        curerr = (X != cur_x).sum()
        if curerr < error:
            res = cur_x
            error = curerr

    return res
```

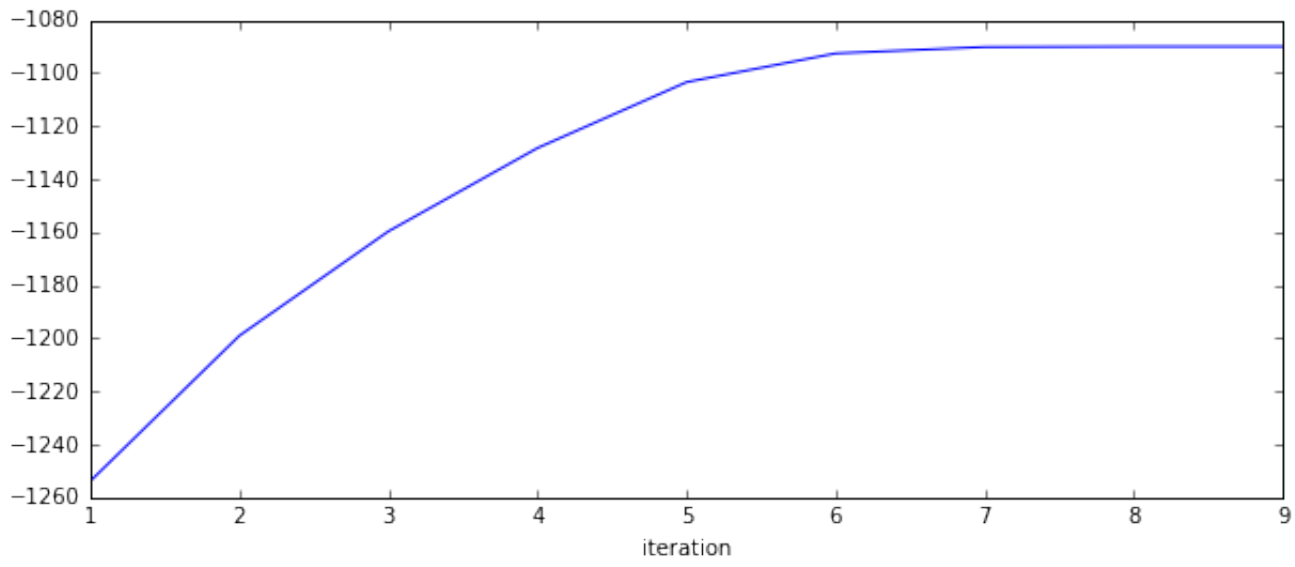
In [50]:
```
colors = np.array(['blue', 'red', 'green'])

#сделаем нужную перестановку минимизируя сумму ошибок
X_predicted = find_permutation(X,X_predicted)
err_rate = (X != X_predicted).sum() / len(X)
plt.figure(figsize=(20, 8))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X)],
            lw=0, s=40, alpha=1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X_predicted)],
            lw=0, s=250, alpha=0.3)
plt.xlim((0, size))
plt.show()

plt.figure(figsize=(20, 8))
plt.plot(np.arange(size), Y[:, 0], color='black', alpha=0.1)
plt.scatter(np.arange(size), Y[:, 0], c=colors[np.array(X)],
            lw=0, s=40, alpha=1)
plt.scatter(np.arange(size)[X != X_predicted], Y[:, 0][X != X_predicted],
            c=colors[np.array(X_predicted)[X != X_predicted]],
            lw=0, s=300, alpha=0.3)
plt.xlim((0, size))
plt.show()
print('error rate = ' + str(round(100*err_rate,1)) + '%')
```




```
error rate = 1.3%
```

In [42]:
```
saved_strerr = sys.stderr  # сохраним в переменную поток вывода ошибок
sys.stderr = open('est_values2.txt', 'w')  # и перенаправим его в файл

# ============
# Для вывода значений функционала нужно поставить параметр verbose
remodel = hmm.GaussianHMM(n_components=2, covariance_type="full",
                          n_iter=100, verbose=True)
remodel.fit(Y)
X_predicted = remodel.predict(Y)
# ============

# Возвращаем все, как было
sys.stderr = saved_strerr
```

In [43]:
```
values = np.loadtxt('./est_values.txt')

plt.figure(figsize=(10, 4))
plt.plot(values[:, 0], values[:, 1])
plt.xlabel('iteration')
plt.show()
```



Как видно, при малых значениях дисперсии и большой разницы матожиданий ($\sigma << |EY_i - EY_j|$) получается достичь хорошего эффекта, погрешность $\approx 1 - 2\%$, хорошая сходимость достигается уже после 7 - 9 итераций.

Кроме гауссовского случая в библиотеке реализовано два других

- `hmm.GMMHMM` --- распределение Y_j при условии X_j является смесью гауссовских распределений,
- `hmm.MultinomialHMM` --- распределение Y_j при условии X_j является дискретным.

2. Картинки

Допустим, у вас имеются спутниковые снимки, и вам нужно уметь автоматически производить их сегментацию, например, на следующие области: растительность, вода, иное. Давайте для этого разобьем изображение на прямоугольные блоки, которые будем размечать. Далее классификацию блоков можно производить независимо для каждого блока. Но в реальности между блоками есть некоторые зависимости. В частности, каждый блок сильно зависит от соседних. В качестве некоторого приближения можно предполагать между ними марковскую зависимость. Она не сможет полностью описать все зависимости, но будет лучше работать, чем если эти зависимости не учитывать вообще.

Однако марковская цепь является одномерным процессом, в то время как изображение двумерное. Чтобы применить к изображению скрытую марковскую модель, можно сделать некоторую цепочку из блоков, обходя их в некоторой последовательности. Правда, в таком случае будут потеряны еще некоторые зависимости, и сегментация будет сильно зависеть от способа обхода. Тогда можно произвести обход несколькими различными способами и результаты "объединить".

Задания:

(простой пример выполнения задания ниже)

1. Соберите спутниковые снимки, сделав скриншоты онлайн-карт. По каждому из них создайте примерную разметку на воду (закрасьте синим), растительность (закрасьте зеленым), остальное (закрасьте красным).
2. Каждое изображение разбейте на блоки, создав список блоков со всех снимков, используя некоторый метод обхода блоков.
3. Оцените самостоятельно все параметры скрытой марковской модели. Параметры условных плотностей можно оценить отдельно для каждого класса. Оцените также матрицу переходных вероятностей, каждый элемент которой оцените как долю переходов в одно состояние при условии того, что цепь находится в другом состоянии.

4. Загрузите новый спутниковый снимок без разметки и примените к нему полученную скрытую марковскую модель для предсказания скрытых состояний. Постройте визуальную разметку.

При желании вы можете выполнять аналогичное задание для других типов изображений, описав четко то, что вы делаете.

Пример выполнения:

Ниже определены некоторые вспомогательные магические функции для разрезания изображения и его обратной склейки, а так же функция для подсчета ОМП многомерного нормального распределения.

In [ ]:
```
def cut_image(image, n, m):
    ''' Разрезает изображение на блоки размера n на m, вытягивая их в вектор'''

    N, M, K = image.shape  # высота, ширина, глубина (3 для RGG, 4 для RGBa)

              # ставим высоту последним измерением и разбиваем по нему
    X = image.transpose((1, 2, 0)).reshape((M, K, N / n, n)) \
              # ставим ширину последним разбиением и разбиваем по нему
             .transpose((1, 2, 3, 0)).reshape((K, N / n, n, M / m, m)) \
              # на первое место ставим два измерения, соответствующие блокам
                # объединяем блоковые размерности и все остальные
             .transpose((1, 3, 2, 4, 0)).reshape((N * M / (n * m), n * m * K))

    return X


def recat_image(X, N, M, K, n, m):
    ''' Собирает разрезанное ранее изображение из блоков n на m'''

    # Тут такая же магия
    return X.reshape((N / n, M / m, n, m, K)).transpose((1, 3, 4, 0, 2)) \
            .reshape((M / m, m, K, N)).transpose((3, 2, 0, 1)) \
            .reshape((N, K, M)).transpose((0, 2, 1))


def est_gauss(X):
    ''' Оценка параметров многомерного гауссовского распределения с помощью ОМП
    '''

    mean = X.mean(axis=0)
    X_centered = np.matrix(X - mean[np.newaxis, :])
    cov_matrix = np.array(X_centered.T * X_centered / X.shape[0])
    return mean, cov_matrix
```

Загрузим карту и разметку (только одну, а вам нужно много снимков и разными способами обхода)

In [ ]:
```
image = plt.imread('map.png')
image = image[:550, :1225, :]  # обрезка краев, для целочисленного деления
Y = cut_image(image, 25, 25)   # при разбиении на блоки

target = plt.imread('map_target.png')
target = target[:550, :1225, :]
target = cut_image(target, 25, 25)
X = (target.sum(axis=1) > target.shape[1] / 2).astype(int)  # бинаризация меток
```

Оценим все параметры и зададим скрытую марковскую модель

In [ ]:
```
means, covars = [0, 0], [0, 0]
means[0], covars[0] = est_gauss(Y[X == 0])
means[1], covars[1] = est_gauss(Y[X == 1])
means = np.array(means)
covars = np.array(covars)

transmat = np.zeros((2, 2))
for i in range(2):
    for j in range(2):
        transmat[i, j] = <оцените вероятность перейти из состояния i в состояние j>

model = hmm.GaussianHMM(n_components=2, covariance_type="full")
model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = transmat
model.means_ = means
model.covars_ = covars
```

Загружаем тестовое изображение и разбиваем его на блоки

In [ ]:
```
test_image = plt.imread('map1.png')
test_image = test_image[:550, :1225, :]
test_Y = cut_image(test_image, 25, 25)
```

Оценим для него значения скрытых состояний и создадим необходимый фон.

In [ ]:
```
# Оценка значений скрытых состояний
# Может вылететь исключение о вырожденной матрице ковариаций.
# В таком случае можно матрицу регуляризовать --- добавить к диагональным
# элементам небольное число так, чтобы детерминант перестал быть слишком маленьким
predicted = model.predict(test_Y)

# Создаем фоновую картинку того же размера
background = recat_image(np.zeros_like(test_Y) + predicted[:, np.newaxis],
                         550, 1225, 3, 25, 25)

# Добавляем к ней 4-й альфа-канал, который отвечает за прозрачность
background_alpha = np.zeros((background.shape[0], background.shape[1], 4))
# RGB копируем, сейчас там либо 000, либо 111
background_alpha[:, :, :3] = background
# полупрозрачность
background_alpha[:, :, 3] = 0.5
# зеленая компонента обратно красной
background_alpha[:, :, 1] = 1 - background_alpha[:, :, 1]
# синию зануляем полностью
background_alpha[:, :, 2] = 0
```

Теперь можно изобразить исходную картинку и ее же с наложением фона

In [ ]:
```
plt.figure(figsize=(20, 10))
plt.imshow(test_image)
plt.axis('off')
plt.show()

plt.figure(figsize=(20, 10))
plt.imshow(test_image)
plt.imshow(background_alpha)
plt.axis('off')
plt.show()
```

Вспомним, что оценка траекторий определяется с помощью максимизации апостериорных вероятностей. Поэтому вместо строгой классификации можно использовать нестрогую --- относить каждый блок к каждому классу с некоторой вероятностью. По таким вероятностям можно сделать соответствующий фон для изображения. Сделайте это.

In [ ]:
```
# Оценка вероятностей
predicted_proba = model.predict_proba(test_Y)

<Определение background_alpha>
<Картинки>
```

Музыка

В отличие от изображений, в музыкальных треках есть четкая компонента времени.

Некоторый вспомогательный код.

In [24]:
```
import librosa
from IPython import display

def Audio(url):
    return display.HTML("<center><audio controls><source src='{}'" +
                        "type=\"audio/wav\"></audio>".format(url))
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-24-3c057cda9f5e> in <module>()
----> 1 import librosa
      2 from IPython import display
      3
      4 def Audio(url):
      5     return display.HTML("<center><audio controls><source src='{}'" +

ImportError: No module named 'librosa'
```

Загрузите сюда некоторый музыкальный трек (или свой голос). Тут же вы его можете послушать. Да, теперь можно выполнять

задание, и тут же в питон-ноутбуке слушать музыку. До чего техника дошла!

In [ ]:
```
sound_file = "./track.mp3"
Audio(url=sound_file)
```

Изобразим графики амплитуды для разной sampling rate (sr).

In [ ]:
```
for sr in [500, 1000, 5000, 10000]:
    plt.figure(figsize=(20, 4))
    y, sr = librosa.load(sound_file, sr=sr)
    plt.plot(y[:1000000:10])
    plt.ylabel("amplitude")
    plt.xlabel("time")
    plt.title('sampling rate = {}'.format(sr))
    plt.show()
```

Изобразим спектрограмму

In [ ]:
```
sr = 1500

# Если ваш ноутбук не справляется с чем-то, уменьшите параметр n_mels
S = librosa.feature.melspectrogram(y, sr=sr, n_mels=30)
log_S = librosa.logamplitude(S, ref_power=np.max)

plt.figure(figsize=(20,4))
librosa.display.specshow(log_S, sr=sr, x_axis='time',
                         y_axis='mel', cmap='hot')
plt.title('power spectrogram')
plt.colorbar(format='%+02.0f dB')
plt.tight_layout()
```

Если выполнить операцию транспонирования, то мы получим некоторый многомерный случайный процесс. К нему уже можно применить скрытую марковскую модель.

In [ ]:
```
Y = log_S.T
```

По данным наблюдениям оцените параметры скрытой марковской модели (см. примеры выше). За сколько итераций сошелся EM-алгоритм? Оцените значения скрытых состояний. На какие по смыслу состояния модель разделила ваш трек? Возможно, для этого придется еще раз послушать трек. Примените эту модель так же к другим трекам. Сделайте выводы.

Баллы за задание:

- **2 балла** за первую часть.
- по **5 баллов** максимум за вторую и третью части. Максимум ставится только при наличии интересных результатов.