

Одна из самых популярных постановок, является задача о поиске наилучшего приближения заданной матрицы матрицей малого ранга:

$$\min_X rk(X) \\ X_{i,j} = Y_{i,j}, \quad (i, j) \in E$$

Как известно, в общем случае эта задача является NP-трудной.

Для того, чтобы обойти это припятствие ранг матрицы аппроксимируется той или иной выпуклой функцией от матрицы X .

Опять же стандартным выбором является переход к постановке задачи с использованием 1-й нормы Шаттена (она же trace norm).

RegMC problem

$$\min_X \|X\|_* \\ X_{i,j} = Y_{i,j}, \quad (i, j) \in E$$

Здесь $X_* = \sum \sigma_i(X)$.

1) Найдем аналитическую запись для градиента сглаженной версии целевого функционала. Для этого найдем двойственную функцию для trace norm:

$$f^*(Y) = \sup_X \langle X, Y \rangle_F - \|X\|_{tr}$$

Докажем следующее неравенство:

$$\langle X, Y \rangle_* \leq \|X\|_{tr} \cdot \|Y\|_2$$

Чтобы это сделать воспользуемся следующим фактом, что $\forall (i, j) |A_{i,j}| \leq \|A\|_2$. Тогда, используя SVD разложение $X = VDW$ и свойства следа, получим:

$$\begin{aligned} \langle X, Y \rangle &= \text{Tr}(Y^T X) = \text{Tr}(Y^T V D W) = \text{Tr}(W Y^T V D) = \text{Tr}((V Y W)^T D) \\ &= \sum_{k=1}^M \sum_{j=1}^N (V Y W)_{kj}^T D_{jk} = \sum_{k=1}^{\min M, N} (V Y W)_{kk}^T \sigma_k(X) \leq \|V Y W\|_2 \sum_{k=1}^{\min M, N} \sigma_k(X) \\ &= \|V Y W\|_2 \|X\|_{tr} = \|Y\|_2 \|X\|_{tr}. \end{aligned}$$

В соответствии с этим фактом, получим:

$$f^*(Y) = \sup_X \langle X, Y \rangle_F - \|X\|_{tr} \leq \sup_X \|X\|_{tr} \cdot \|Y\|_2 - \|X\|_{tr} = \sup_X (1 - \|Y\|_2) \cdot \|X\|_{tr}$$

Так как $\|X\|_{tr} \geq 0$, то:

$$f^*(Y) = \begin{cases} 0 & \|Y\|_2 \leq 1 \\ \infty & \text{else} \end{cases}$$

Докажем, что данный супремум достижим. В случае, если $\|Y\|_2 \leq 1$, то корректность очевидна, так как можно взять нулевую матрицу и получить 0. В случае, если $\|Y\|_2 \geq 1$, то возьмем X , у которого первое сингулярное число равно $\alpha \cdot \sigma_Y$. Тогда $\sup_X \langle X, Y \rangle_F - \|X\|_{tr} = \sup_\alpha \alpha \cdot \sigma_Y^2 - \alpha \cdot \sigma_Y = \infty$ (если устремить $\alpha \rightarrow \infty$).

Используем полученный результат, для построения сглаженной целевой функции.

Согласно описанию, сглаженная функция имеет следующий вид:

$$f_\mu(x) = \max_u \langle Ax, u \rangle - \phi(u) - \mu \cdot d(u)$$

В качестве $\phi(u)$ возьмем полученную двойственную функцию, в качестве прокс функции, возьмем функцию из описания задачи.

Тогда:

$$f_\mu(x) = \max_{y: \|y\|_2 \leq 1} \langle X, Y \rangle_F - \mu/2 \|Y\|_F^2$$

Согласно описанию, градиент такой функции:

$$\nabla f_\mu(x) = \arg \max_{y: \|y\|_2 \leq 1} \langle X, Y \rangle_F - \mu/2 \|Y\|_F^2$$

Выделим полный квадрат:

$$\langle X, Y \rangle_F - \mu/2 \|Y\|_F^2 = \langle X, Y \rangle_F - \mu/2 \|Y\|_F^2 - 1/\mu \cdot \langle X, X \rangle_F + 1/\mu \cdot \langle X, X \rangle_F$$

Тогда:

$$\begin{aligned} \max_y \langle X, Y \rangle_F - \mu/2 \|Y\|_F^2 - 1/\mu \cdot \langle X, X \rangle_F + 1/\mu \cdot \langle X, X \rangle_F &= \min_y - \langle X, Y \rangle_F + \mu/2 \|Y\|_F^2 + 1/\mu \cdot \langle X, X \rangle_F - 1/\mu \cdot \langle X, X \rangle_F \\ &= \min_y \|1/\sqrt{\mu} X - \sqrt{\mu/2} Y\|_F^2 - 1/\mu \cdot \langle X, X \rangle_F = \min_y \mu/2 \cdot \|1/\mu X - Y\|_F^2 - 1/\mu \cdot \langle X, X \rangle_F \end{aligned}$$

Заметим, что только $\|1/\mu X - Y\|_F^2$ влияет на достижение минимума функции. А это слогасное эквивалентно задаче минимизации матрицей наименьшего ранга. Можно заметить, что минимум достигим тогда, когда Y сингулярное разложение совпадает. Но так как у нас стоит ограничение на спектральную норму, то сингулярное число выбирается как $\min(\sigma_X/\mu, 1)$. Используя такую матрицу, можно достигнуть минимума. Таким образом:

$$\nabla f_\mu(x) = \sum_i \min(\sigma_{x,i}/\mu, 1) \cdot v_i u_i^T$$

2) Согласно полученной сглаженной функции, построим быстрый градиентный спуск. Для начала определим константу липшца L . Согласно теории и полученным расчетам, получаем:

$$L = \frac{1}{(\mu)}$$

Тогда:

$$y_k = \operatorname{argmin}_y \langle \nabla f(x), y - x \rangle + \frac{1}{2} \cdot L \cdot \|y - x\|_F^2$$

$$y_k = \frac{1}{\mu} \cdot X - \nabla f(x)$$

$$z_k = \operatorname{argmin}_x \frac{L}{\sigma} \cdot d(x) + \sum_{i=1}^k \frac{i+1}{2} \cdot [f(x_i) + \langle \nabla f(x_i), x - x_i \rangle]$$

$$z_k = -\mu \sum_{i=1}^k \frac{i+1}{2} \cdot \nabla f(x_i)$$

$$x_k = \frac{2}{k+3} \cdot z_k \frac{k+1}{k+3} \cdot y_k$$

Для реализации полученной схемы импортируем необходимые библиотеки и реализуем функции необходимые для работы метода и решение дальнейших задач.

```
In [1]: 1 import numpy as np
2 from PIL import Image, ImageDraw
3 import matplotlib.pyplot as plt
4 from IPython.display import clear_output
5 from sklearn.datasets import fetch_lfw_people
6 from collections import defaultdict
```

Реализуем функцию, которая заполняет матрицу по указанным индексам определенными значениями.

```
In [2]: 1 def fillNA(X, ind, y):
2     X_copy = X.copy()
3     X_copy = X_copy.reshape(-1)
4     X_copy[ind] = y
5     return X_copy.reshape(X.shape)
```

Реализуем функцию, которая "выбивает" пиксели из картинки. На вход берется картинка, и отношение выбитых пикселей. Функция будет возвращать индексы пикселей, которые не были выбиты, индексы пикселей, которые были выбиты и значение пикселей, которые не были выбиты.

```
In [108]: 1 def beat_img(img, p=0.8):
2     A_copy = img.copy().reshape(-1)
3     rnd = np.random.choice(list(range(A_copy.shape[0])), int((1-p)*A_copy.shape[0]), replace=False)
4     del_ind = np.array(list(set(list(range(A_copy.shape[0])) - set(rnd))))
5     return rnd, A_copy[rnd], del_ind
```

Реализуем описанную схему.

```

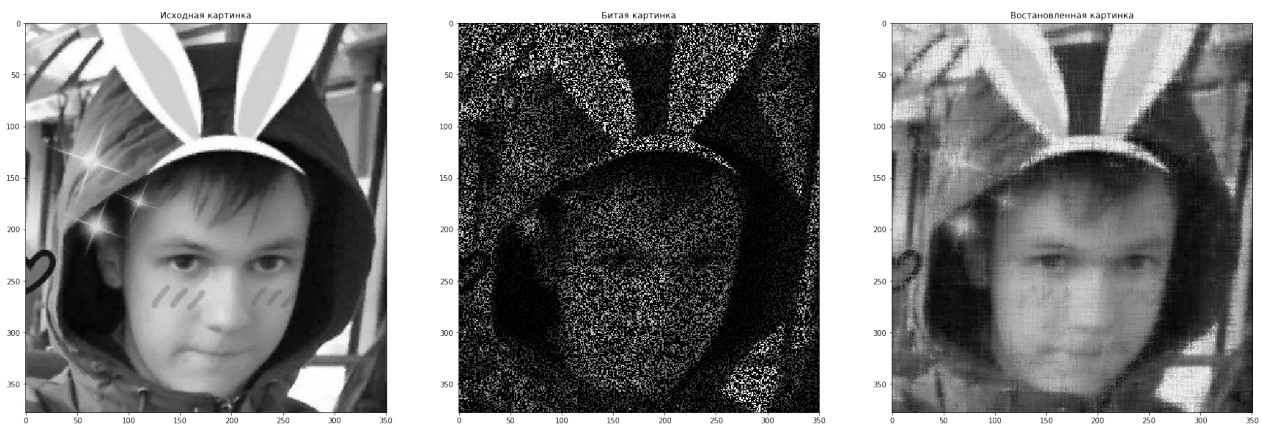
In [109]: 1 #градиент функции
2 def gradX(X, mu):
3     #SVD разложение матрицы
4     S, V, D = np.linalg.svd(X)
5     h = lambda x, mu : x/mu if mu >= x else 1
6     res = np.zeros(X.shape)
7     #подсчет градиента
8     for i in range(V.shape[0]):
9         res += h(V[i],mu)*np.outer(S[:,i],D.T[:,i])
10    return res
11 #минимизируемая функция
12 def fu(X, mu):
13     #SVD разложение матрицы
14     S, V, D = np.linalg.svd(X)
15     h = lambda x, mu : x**2/mu/2 if mu >= x else x - mu/2
16     res = 0
17     #подсчет значения функции
18     for i in range(V.shape[0]):
19         res += h(V[i],mu)
20    return res
21 #градиентный спуск
22 def fast_grad_des(X0, index_full, y, mu, e = 0.0001, max_iter = 1000):
23     #инициализация параметров
24     X = X0.copy()
25     k = 0
26     acc_gradX = 0
27     #реализация схемы
28     while(True):
29         new_grad = gradX(X, mu)
30         f1 = fu(X, mu)
31         acc_gradX = acc_gradX + (k+1)/2*new_grad
32         y = 1/mu*X - new_grad
33         z = -mu*acc_gradX
34         X = 2/(k+3)*z + (k+1)/(k+3)*y
35         X = fillNA(X, index_full, y_true)
36         f2 = fu(X, mu)
37         #критерий останова
38         if (abs(f1 - f2) < e):
39             return X, k
40         grad = new_grad
41         k += 1
42         if(k >= max_iter):
43             print('Достигнуто максимальное количество итераций')
44             print('GD',abs(f1 - f2))
45             return X, k

```

Проверим работаспособность описанной схемы. Для этого возьмем μ равное 1 и произвольную картинку (сделав ее черно-белой) и выйдем из нее 70% пикселей и постараемся восстановить выбитые пиксели полученным методом.

```
In [110]: 1 from IPython.display import clear_output
2 image = Image.open('Screenshot.jpg')
3 img = image.convert('L')
4 mu = 1
5 A = np.asarray(img, dtype=np.float32).copy()
6 A = 1/255 * A
7 index_full, y_true, index_del = beat_img(A, 0.7)
8 true_img = fillNA(np.zeros(A.shape), index_full, y_true)
9 X = np.random.random(A.shape)
10 X = fillNA(X, index_full, y_true)
11 X, k = fast_grad_des(X, index_full, y_true, mu)
12 plt.figure(figsize = (30,10))
13 plt.subplot(1,3,1)
14 plt.imshow(A, cmap = 'gray')
15 plt.title('Исходная картинка')
16 plt.subplot(1,3,2)
17 plt.imshow(fillNA(A, index_del, 0), cmap = 'gray')
18 plt.title('Битая картинка')
19 plt.subplot(1,3,3)
20 plt.imshow(X, cmap = 'gray')
21 plt.title('Восстановленная картинка')
```

Out[110]: Text(0.5, 1.0, 'Восстановленная картинка')



Сравнив полученную картинку, можно сделать вывод, что метод работает

Для того, что-бы выполнить проксимальный градиентный спуск, запишем эквивалентную задачу:

$$\min_X \frac{1}{2} \cdot \|P(A) - P(X)\|_F^2 + \lambda \|X\|_{lr}$$

$$P(X_{i,j}) = \begin{cases} 0 & (i,j) \notin \Omega \\ X_{i,j} & else \end{cases}$$

Тогда, задачу можно записать как:

$$\min_X f(x) + h(x)f(x) = \frac{1}{2} \cdot \|P(A) - P(X)\|_F^2 h(x) = \lambda \|X\|_{lr}$$

Тогда :

$$\nabla f(x) = P(A) - P(X) \text{Prox}_{h,t}(Y) = \operatorname{argmin}_x \frac{1}{2t} \|X - Y\|_F^2 + \lambda \|X\|_{lr}$$

Докажем, что $\text{Prox}_{h,t}(Y) = U \Sigma_{\lambda} V^T$, где $\Sigma_{\lambda,i,i} = \max([0, \sigma_Y - \lambda])$, а U и V матрицы при сингулярном разложении. Согласно условию, $\text{Prox}_{h,t}(Y) = Z$, тогда когда:

$$0 \in Z - B + \lambda \cdot t \cdot \delta \|Z\|_{lr}$$

Тогда, если $Z = U \Sigma V^T$, то верно следующее:

$$\delta \|Z\|_{lr} = \{UV^T + \|W\| : \|W\| \leq 1, U^T W = 0, W V = 0\}$$

Подставим $Z = U \Sigma_{\lambda} V^T$ и получим, что условие истинно. Тогда, шаг проксимального градиентного спуска имеет вид:

$$X_{k+1} = \text{Prox}_{h,t}(X + t(P(A) - P(X)))$$

Так как $\frac{1}{2} \cdot \|P(A) - P(X)\|_F^2$ имеет липщевый градиент с параметром 1, то шаг можно переписать как:

$$X_{k+1} = \text{Prox}_{h,t}(X + (P(A) - P(X)))$$

```

In [111]: 1 #прокс функция
2 def Prox(alp, X):
3     #svd разложение
4     S, V, D = np.linalg.svd(X)
5     res = np.zeros(X.shape)
6     for i in range(V.shape[0]):
7         res += max(V[i] - alp, 0)*np.outer(S[:,i],D.T[:,i])
8     return res
9 #исходная функция
10 def f(alp, X, A):
11     f1 = np.linalg.norm(fillNA(A, index_del, 0) - fillNA(X, index_del, 0), 'fro')**2
12     f2 = np.linalg.norm(X, 'nuc')
13     return f1 + alp*f2
14 #проксимальный градиентный спуск
15 def prox_grad_des(X0, alp, index_del, e = 0.0001, max_iter = 1000):
16     #инициализация параметров
17     X = X0.copy()
18     k = 0
19     #реализация схемы
20     while(True):
21         #print(k)
22         f1 = f(alp, X, A)
23         X = Prox(alp, X + fillNA(A, index_del, 0) - fillNA(X, index_del, 0))
24         k+=1
25         f2 = f(alp, X, A)
26         if(abs(f1 - f2) < e):
27             return X, k
28         #print(abs(f1 - f2))
29         if(k >= max_iter):
30             print('Достигнуто максимальное число итераций')
31             print('PD',abs(f1 - f2))
32             return X, k
33

```

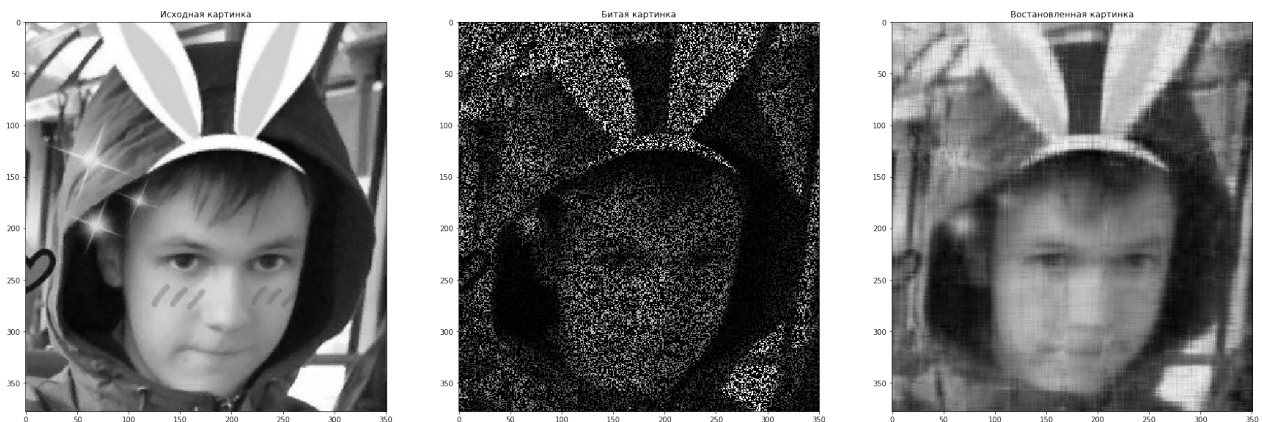
Реализуем полученный метод и проверим его работу способность как у преведущего метода.

```

In [112]: 1 X = np.random.random(A.shape)
2 alp = 1
3 X, k = prox_grad_des(X, alp, index_del, max_iter = 150)
4 plt.figure(figsize = (30,10))
5 plt.subplot(1,3,1)
6 plt.imshow(A, cmap = 'gray')
7 plt.title('Исходная картинка')
8 plt.subplot(1,3,2)
9 plt.imshow(fillNA(A, index_del, 0), cmap = 'gray')
10 plt.title('Битая картинка')
11 plt.subplot(1,3,3)
12 plt.imshow(X, cmap = 'gray')
13 plt.title('Восстановленная картинка')

```

Out[112]: Text(0.5, 1.0, 'Восстановленная картинка')



Сравнив полученную картинку, можно сделать вывод, что метод работает

Сделаем так, что каждый из каналов цветов будет восстанавливаться по отдельности. Тогда можно восстановить и цветное изображение. Так же построим график невязки для каждого из каналов на каждой итерации и сравним скорость сходимости этих методов. Так же построим график функции потерь по каждому пикселю.

Для этого слегка перепишем полученные нами методы


```

In [102]: 1 def error_img(X, A):
2     return ((X - A)**2).sum().sum()
3
4 def fast_grad_des(X0, C, index_full, y, mu, e = 0.0001, max_iter = 1000):
5     #инициализация параметров
6     X = X0.copy()
7     k = 0
8     acc_gradX = 0
9     eps = []
10    error = []
11    #реализация схемы
12    while(True):
13        new_grad = gradX(X, mu)
14        f1 = fu(X, mu)
15        acc_gradX = acc_gradX + (k+1)/2*new_grad
16        y = 1/mu*X - new_grad
17        z = -mu*acc_gradX
18        X = 2/(k+3)*z + (k+1)/(k+3)*y
19        X = fillNA(X, index_full, y_true)
20        f2 = fu(X, mu)
21        #критерий останова
22        eps.append(abs(f1 - f2))
23        error.append(error_img(X, C))
24        if (abs(f1 - f2) < e):
25            return X, k, eps, error
26        grad = new_grad
27        k += 1
28        if(k >= max_iter):
29            print('Достигнуто максимальное количество итераций')
30            print('GD',abs(f1 - f2))
31            return X, k, eps, error
32
33 def prox_grad_des(X0, C, alp, index_del,e = 0.0001, max_iter = 1000):
34     #инициализация параметров
35     X = X0.copy()
36     k = 0
37     eps = []
38     error = []
39     #реализация схемы
40     while(True):
41         f1 = f(alp, X, C)
42         X = Prox(alp, X + fillNA(C, index_del, 0)- fillNA(X, index_del, 0))
43         k+=1
44         f2 = f(alp, X, C)
45         eps.append(abs(f1 - f2))
46         error.append(error_img(X, C))
47         if(abs(f1 - f2) < e):
48             return X, k, eps, error
49         #print(abs(f1 - f2))
50         if(k >= max_iter):
51             print('Достигнуто максимальное число итераций')
52             print('PD',abs(f1 - f2))
53             return X, k, eps, error
54
55 def error_img(X, A):
56     return ((X - A)**2).sum().sum()
57
58 def retake(X, ind):
59     X_copy = X.copy().reshape(-1)
60     return X_copy[ind]
61
62 def fast_grad_des(X0, C, index_full, y, mu, e = 0.0001, max_iter = 1000):
63     #инициализация параметров
64     X = X0.copy()
65     k = 0
66     acc_gradX = 0
67     eps = []
68     error = []
69     #реализация схемы
70     while(True):
71         new_grad = gradX(X, mu)
72         f1 = fu(X, mu)
73         acc_gradX = acc_gradX + (k+1)/2*new_grad
74         y = 1/mu*X - new_grad
75         z = -mu*acc_gradX
76         X = 2/(k+3)*z + (k+1)/(k+3)*y
77         X = fillNA(X, index_full, y_true)
78         f2 = fu(X, mu)
79         eps.append(abs(f1 - f2))
80         error.append(error_img(X, C))
81         #критерий останова

```

```
82     if (abs(f1 - f2) < e):
83         return X, k, eps, error
84     grad = new_grad
85     k += 1
86     if(k >= max_iter):
87         print('Достигнуто максимальное количество итераций')
88         print('GD',abs(f1 - f2))
89         return X, k, eps, error
```



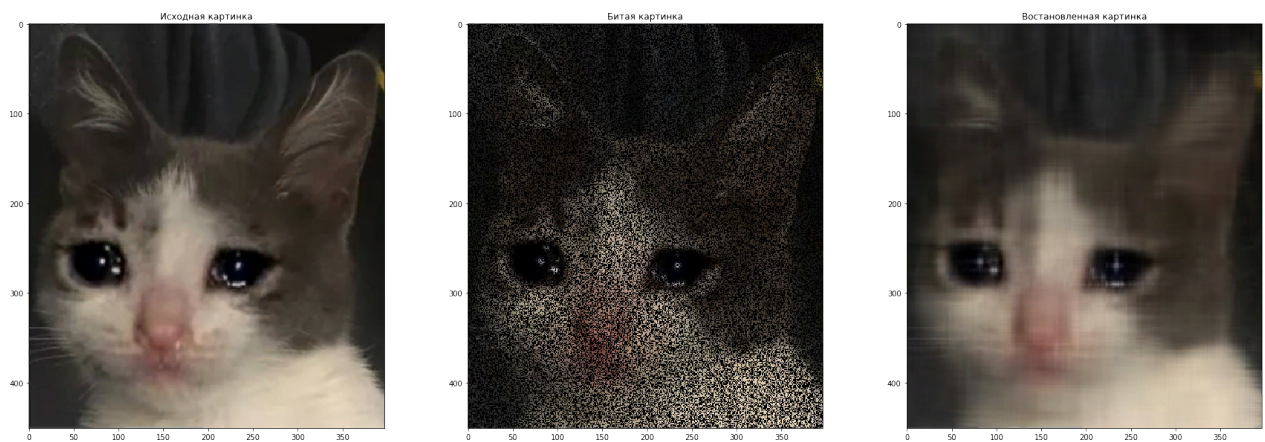
```

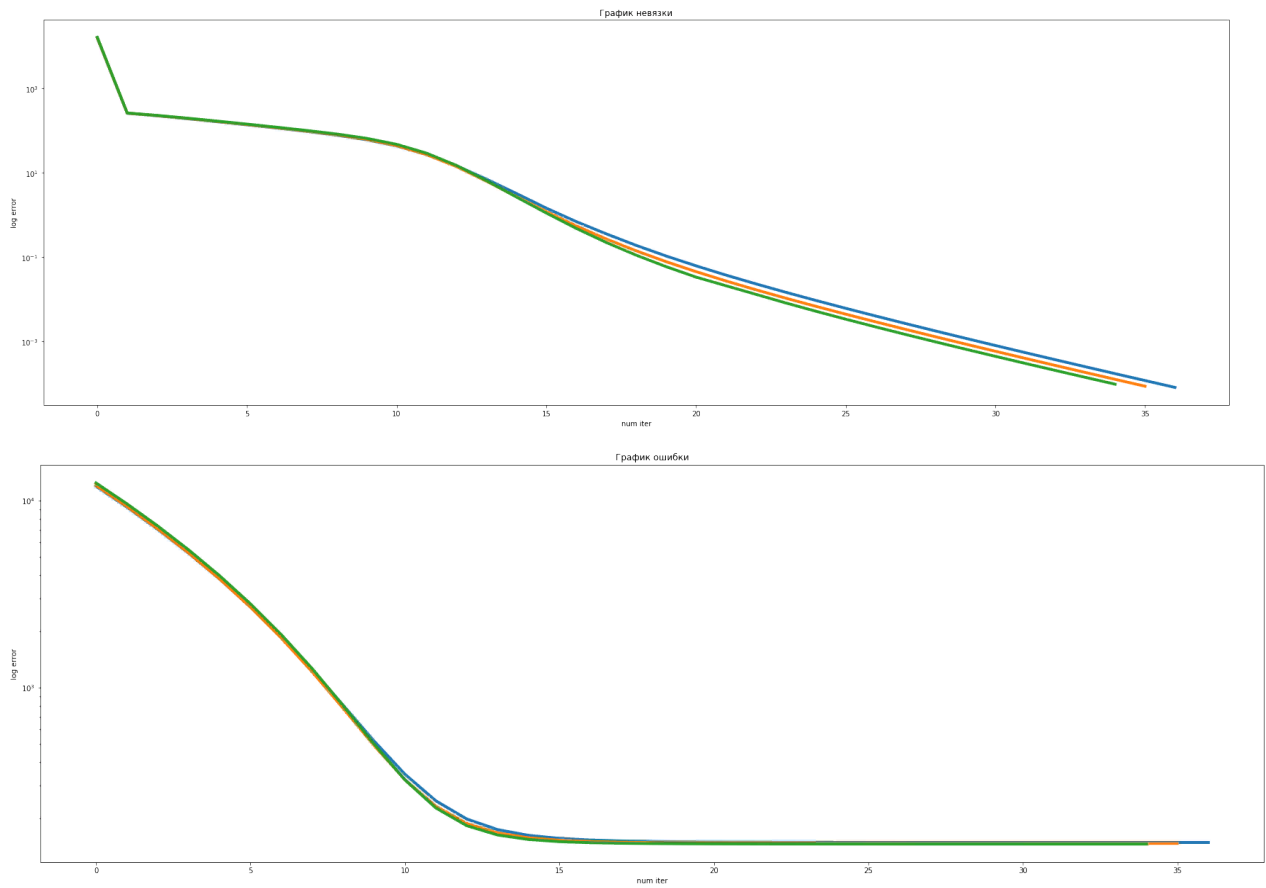
In [105]: 1 image = Image.open('cat.jpg')
2 alp = 1
3 A = np.asarray(image, dtype=np.float32).copy()
4 A = A/255
5 X = np.random.random(A.shape)
6 R, G, B = A[:, :, 0], A[:, :, 1], A[:, :, 2]
7 res, eps_chanel, error_chanel = [], [], []
8 index_full, _, index_del = beat_img(R, 0.5)
9
10 del_img = []
11 for chanel in [R, G, B]:
12     del_img.append(fillNA(chanel, index_del, np.zeros(len(index_del))))
13 del_img = f_rec(del_img)
14
15 for chanel in [R, G, B]:
16     X = np.random.random(chanel.shape)
17     X, k, eps, error = prox_grad_des(X, chanel, alp, index_del, max_iter = 150)
18     res.append(X)
19     eps_chanel.append(eps)
20     error_chanel.append(error)
21
22 rec_img = f_rec(res)
23
24 plt.figure(figsize = (30,10))
25 plt.subplot(1,3,1)
26 plt.imshow(A)
27 plt.title('Исходная картинка')
28 plt.subplot(1,3,2)
29 plt.imshow(del_img)
30 plt.title('Битая картинка')
31 plt.subplot(1,3,3)
32 plt.imshow(rec_img)
33 plt.title('Восстановленная картинка')
34 plt.figure(figsize = (30,10))
35 for e in eps_chanel:
36     plt.plot(list(range(len(e))), e, linewidth = 4)
37 plt.title('График невязки')
38 plt.yscale('log')
39 plt.xlabel('num iter')
40 plt.ylabel('log error')
41 fig = plt.figure(figsize = (30,10))
42 for e in error_chanel:
43     plt.plot(list(range(len(e))), e, linewidth = 4)
44 plt.title('График ошибки')
45 plt.yscale('log')
46 plt.xlabel('num iter')
47 plt.ylabel('log error')

```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Out[105]: Text(0, 0.5, 'log error')





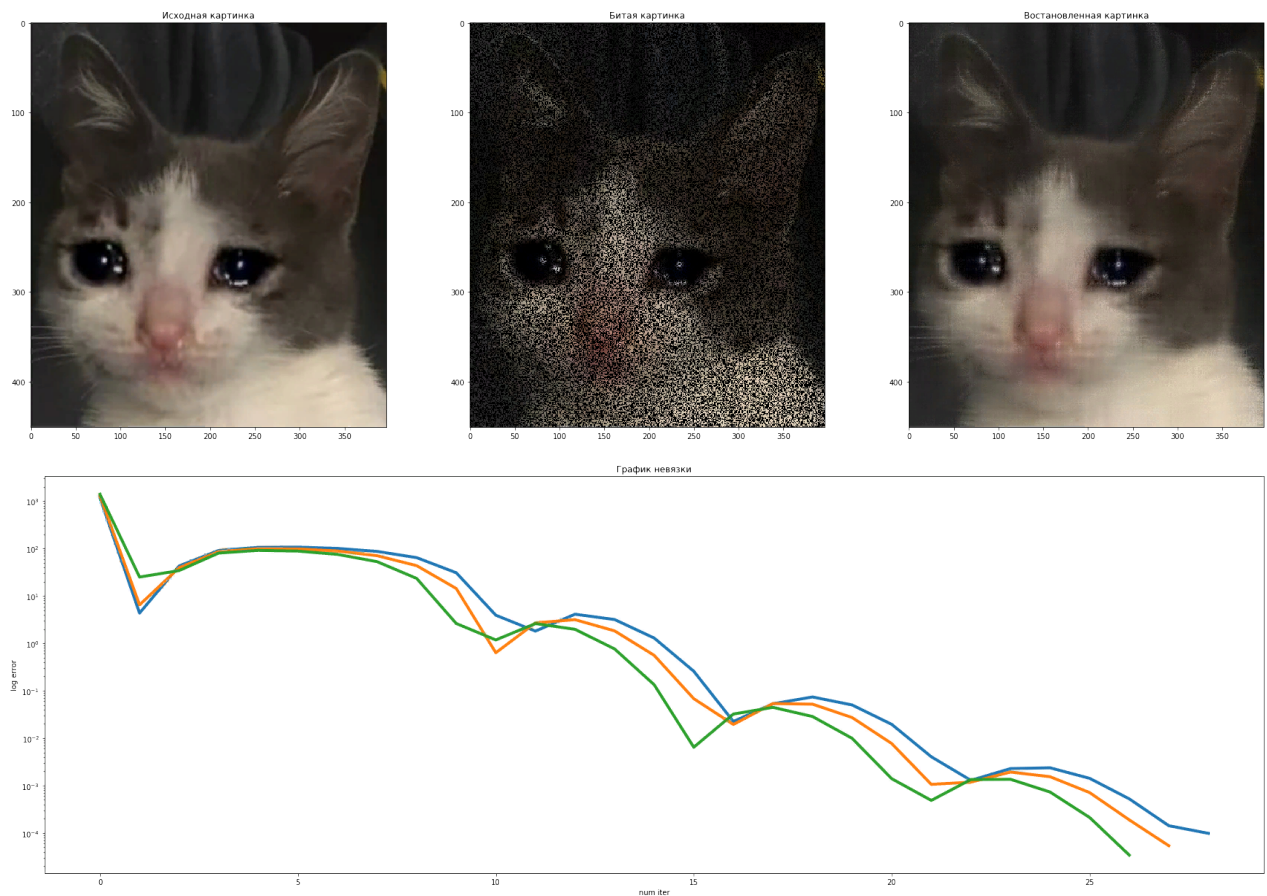
```

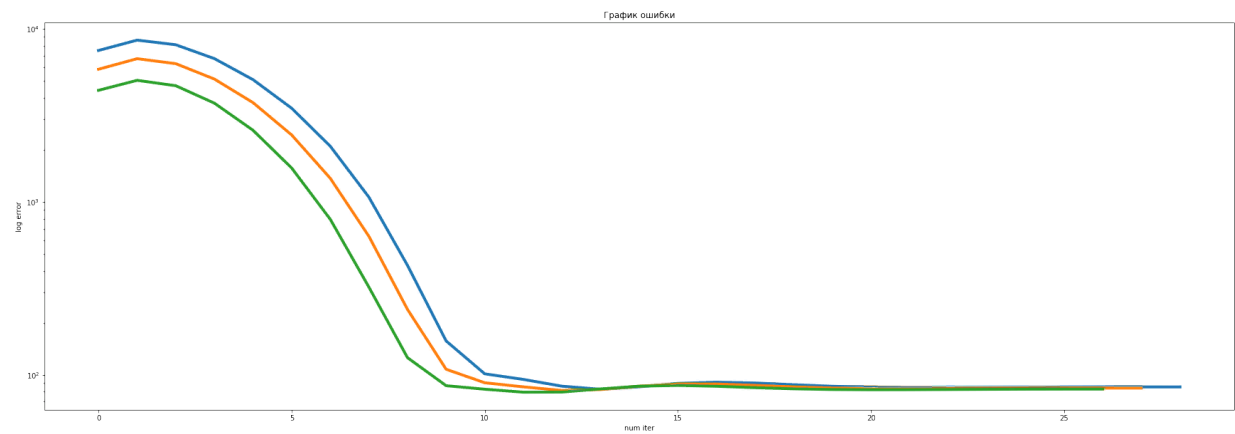
In [106]: 1 mu = 1
2 res, eps_chanel, error_chanel = [], [], []
3 for chanel in [R, G, B]:
4     X = np.random.random(chanel.shape)
5     y_true = retake(chanel, index_full)
6     X, k, eps, error = fast_grad_des(X, chanel, index_full, y_true, mu)
7     res.append(X)
8     eps_chanel.append(eps)
9     error_chanel.append(error)
10
11 rec_img = f_rec(res)
12 plt.figure(figsize = (30,10))
13 plt.subplot(1,3,1)
14 plt.imshow(A)
15 plt.title('Исходная картинка')
16 plt.subplot(1,3,2)
17 plt.imshow(del_img)
18 plt.title('Битая картинка')
19 plt.subplot(1,3,3)
20 plt.imshow(rec_img)
21 plt.title('Восстановленная картинка')
22 plt.figure(figsize = (30,10))
23 for e in eps_chanel:
24     plt.plot(list(range(len(e))), e, linewidth = 4)
25 plt.title('График невязки')
26 plt.yscale('log')
27 plt.xlabel('num iter')
28 plt.ylabel('log error')
29 fig = plt.figure(figsize = (30,10))
30 for e in error_chanel:
31     plt.plot(list(range(len(e))), e, linewidth = 4)
32 plt.title('График ошибки')
33 plt.yscale('log')
34 plt.xlabel('num iter')
35 plt.ylabel('log error')

```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Out[106]: Text(0, 0.5, 'log error')





In []: ▶

1