Министерство образования и науки РФ
Новосибирский Государственный Технический Университет
Кафедра ПМт

Лабораторная работа №1

«Прямая градиентная процедура планирования»

по курсу «Математические методы планирования эксперимента»

Факультет:	ПМИ
Группа:	ПММ-81
Студенты:	Михайлов А.А.,
	Санина А.А.
Преподаватели:	Черникова О.С.,
	Чубич В.М.

Новосибирск 2013

1. Условие задачи

Исследовать метод D–оптимального планирования непрерывных оптимальных планов с помощью прямой градиентной процедуры.

2. Ход работы:

2.1. Постановка задачи

Пусть ν — возможное количество запусков системы, причём сигнал α_1 подаётся на вход системы k_1 раз, сигнал α_2 — k_2 раз и т.д., сигнал α_q — q раз.

Под *непрерывным нормированным планом* ξ условимся понимать совокупность величин

$$\xi = \begin{pmatrix} \alpha_1, & \alpha_2, & \dots, & \alpha_q \\ p_1, & p_2, & \dots, & p_q \end{pmatrix}, \ p_i \geq 0, \ \sum_{i=1}^{q} p_i = 1, \ \alpha_i \in \Omega_\alpha, \ i = \overline{1, q}. \tag{1}$$

В общем случае непрерывный нормированный план ξ соответствует вероятностной мере $\xi\left(d\alpha\right)$, заданной на области Ω_α и удовлетворяющей условиям неотрицательности и нормировки:

$$\int\limits_{\Omega_\alpha} \xi\left(d\alpha\right) = 1, \ \xi\left(d\alpha\right) \geq 0, \ \alpha \in \Omega_\alpha. \tag{2}$$

При этом *нормированная информационная матрица* плана определяется соотношением

$$M\left(\xi\right) = \int\limits_{\Omega_\alpha} M\left(\alpha\right) \xi\left(d\alpha\right). \tag{3}$$

Для плана (1) интеграл в (3) переходит в сумму, т.е.

$$M\left(\xi\right) = \sum_{i=1}^{q} p_i \, M\left(\alpha_i\right), \tag{4}$$

где $M\left(\alpha_i\right)$ — информационные матрицы точек спектра плана. Планирование оптимальных экспериментов опирается на критерии оптимальности планов.

2.2. Алгоритм метода D–оптимального планирования непрерывных оптимальных планов с помощью прямой градиентной процедуры

Критерий D -оптимальности. План называется D-оптимальным, если

$$\xi^* = \arg\max_{\xi \in \Omega_\xi} \det\left(M\left(\xi\right)\right) = \arg\min_{\xi \in \Omega_\xi} \det\left(D\left(\xi\right)\right) \tag{5}$$

где $D\left(\xi\right) = M^{-1}\left(\xi\right)$ — *дисперсионная матрица плана.*

Прямая градиентная процедура синтеза непрерывных оптимальных планов.

Оптимизационная задача

$$\xi^* = \arg\min_{\xi \in \Omega_\xi} X\left[M\left(\xi\right)\right]. \tag{6}$$

1) Зададим начальный невырожденный план

$$\xi = \begin{pmatrix} \alpha_1^0, & \alpha_2^0, & \dots, & \alpha_q^0 \\ p_1^0, & p_2^0, & \dots, & p_q^0 \end{pmatrix}, \ p_i^0 = \frac{1}{q}, \ \alpha_i \in \Omega_\alpha, \ i = \overline{1, q}. \tag{7}$$

в котором $q = \frac{s(s+1)}{2} + 1$. Вычислим информационные матрицы $M\left(\alpha_i^0\right)$ одноточечных планов для $i = \overline{1, q}$ и по формуле (4) информационную матрицу всего плана ξ_0. Положим $l = 0$.

2) Считая веса $p_1^l, p_2^l, \dots, p_q^l$ фиксированными, для задачи

$$X\left[M\left(\xi\right)\right] \to \min_{\alpha_1^l, \dots, \alpha_q^l}, \ \alpha_i^l \in \Omega_\alpha, \ i = \overline{1, q}, \tag{8}$$

выполним одну итерацию метода проекции градиента:

$$\breve{A}^{l+1} = \pi_{\Omega_{\breve{A}}} \left\{ \breve{A}^l - \rho_l' \nabla_{\breve{A}} X\left[M\left(\xi_l\right)\right] \right\}, \tag{9}$$

где

- $\breve{A}^T = \left(\alpha_1^T,\, \alpha_2^T,\, \ldots,\, \alpha_q^T\right);$
- $\pi_{\Omega_{\breve{A}}}(z)$ — проекция точки на множество $\Omega_{\breve{A}}$;
- $\rho_l^{'} \geq 0$ — длина шага.

Для сигналов, ограниченных по мощности или по амплитуде, когда $\Omega_{\breve{A}}$ — соответственно координатный шар или параллелепипед, известны явные выражения для проекции:

- если $\Omega_{\breve{A}} = \left\{\breve{A} \in R^k \mid \left\|\breve{A}\right\| \leq C\right\}$, то $\pi_{\Omega_{\breve{A}}}(z) = \frac{z}{\|z\|}C$;
- если $\Omega_{\breve{A}} = \left\{\breve{A} \in R^k \mid \beta_j \leq \breve{A}_j \leq \gamma_j,\ j = \overline{1,\, k}\right\}$, то

$$\left[\pi_{\Omega_{\breve{A}}}(z)\right] = \begin{cases} \beta_j, & z_j < \beta_j; \\ z_j, & \beta_j \leq z_j \leq \gamma_j; \\ \gamma_j, & z_j > \gamma_j. \end{cases}$$

Далее составим план

$$\xi = \begin{pmatrix} \alpha_1^{l+1}, & \alpha_2^{l+1}, & \ldots, & \alpha_q^{l+1} \\ p_1^l, & p_2^l, & \ldots, & p_q^l \end{pmatrix},$$

где α_i^{l+1} — точки найденные на шаге 2. Вычислим $M\left(\alpha_i^{l+1}\right)$, $i = \overline{1,\, q}$.

3) Зафиксируем точки спектра полученного плана и для задачи

$$X\left[M\left(\breve{\xi}_l\right)\right] \to \min_{p_1^l,\, \ldots,\, p_q^l},\ p_i^l \geq 0,\ \sum_{i=1}^{q} p_i^l = 1,\ i = \overline{1,\, q},$$

выполним одну итерацию метода проекции градиента Розена:

$$\tilde{p}^{l+1} = \tilde{p}^l - \rho_l^{''} \nabla_{\tilde{p}} X\left[M\left(\tilde{\xi}_l\right)\right],$$

где $\widetilde{p} = (p_1,\, p_2,\, \ldots,\, p_q)$, $\rho_l^{''}$ — длина шага, P — матрица оператора проектирования. Составим план

$$\xi_{l+1} = \begin{pmatrix} \alpha_1^{l+1}, & \alpha_2^{l+1}, & \ldots, & \alpha_q^{l+1} \\ p_1^{l+1}, & p_2^{l+1}, & \ldots, & p_q^{l+1} \end{pmatrix}.$$

4) Если выполняется неравенство

$$\sum_{i=1}^{q} \left[\left\|\alpha_i^{l+1} - \alpha_i^l\right\|^2 + \left(p_i^{l+1} - p_i^l\right)^2\right] \leq \delta,$$

где δ — малое положительное число, перейдём на шаг 5. В противном случае для повторим шаги 2 и 3.

5) Проверим необходимое условие оптимальности плана.

$$\left|\mu\left(\alpha_i^{l+1},\, \xi_{l+1}\right) - \eta\right| \leq \delta,\ i = \overline{1,\, q}.$$

Соответствие значений параметров $X\left[M(\xi)\right]$, $\mu(\alpha,\, \xi)$, η прямой процедуры критерию D-оптимальности указано в табл. 1.

Если необходимое условие оптимальности выполняется, закончим процесс. В противном случае повторим всё сначала, скорректировав начальное приближение ξ_0.

Таблица 1. Парметры критерий D-оптимального плана

$X\left[M(\xi)\right]$	$\mu(\alpha,\, \xi)$	η
$-\ln\det\left(M(\xi)\right)$	$Sp\left[M^{-1}(\xi)\ M(\alpha)\right]$	s

2.3. Текст программы на языке Python

```python
# -*- coding: utf-8 -*-

import numpy as np
from numpy import linalg as la
from scipy.optimize import minimize

import pylab as pl


class Plan(object):
    def __init__(self, s, q=None):
        self.s = s                                          # Число регрессоров
        if q is None:
            self.q = int(0.5 * s * (s + 1) + 1)             # Число точек
        else:
            self.q = q
        self.A = np.matrix(np.ndarray((self.s, self.q)))    # Матрица плана: [s,q]
        self.p = np.matrix(np.ndarray((self.q, 1)))
        self.p[:, :] = 1.0 / self.q                         # Веса точек: вектор, или матрица [q,1]

    def set_bounds(self, bounds):
        self.bnds = bounds

    def _remove_ith_point(self, i):
        self.p = np.delete(self.p, i, 0)
        self.A = np.delete(self.A, i, 1)
        self.q -= 1

    def reduce_plan(self):
        epsilon = 1.0e-5
        i = 0
        while i < self.q:
            if self.p[i, 0] < epsilon:
                self._remove_ith_point(i)
            else:
                i += 1

    def partial_inf_matrix(self, i, f, A=None):
        if A is None:
            A = self.A
        return f(A[:, i]) * f(A[:, i]).transpose()

    def inf_matrix(self, f, A=None, p=None):
        if A is None:
            A = self.A
        if p is None:
            p = self.p
        q = self.q
        return sum(self.partial_inf_matrix(i, f, A) * p[i, 0] for i in xrange(q))

    def X(self, f, A=None, p=None):
        log = np.log
        det = la.det
        M   = self.inf_matrix(f, A, p)
        return -log(det(M))

    def mu(self, f):
        M = self.inf_matrix(f)
        return max(np.asscalar((M**(-1) * self.partial_inf_matrix(i, f)).trace()) for i in xrange(self.q))

    def eta(self, f):
        return self.s

    def __str__(self):
        return '%s\n%s' % (self.A, self.p)


def build_plan_dirgrad(f, xi, epsilon=1.0e-6):
    """Синтез оптимального плана с помощью прямой градиентной процедуры. D критерий.
    """
    exit_cond = np.inf
    iter = 0
    while exit_cond > epsilon:
        iter += 1
        detM = np.log(la.det(xi.inf_matrix(f)))
        mu   = xi.mu(f)
        print 'On iter %d:\n%s\n\nlog(det(M(xi))) = %.3lf\nmu(alpha,xi) = %.3lf\n==================' % (
            iter, xi, detM, mu)

        bnds = reduce(lambda a, b: a + b, [(xi.bnds[i], ) * xi.q for i in xrange(xi.s)])      # Границы для точек плана
        res = minimize(lambda A: xi.X(f, A=np.asmatrix(A).reshape((xi.s, xi.q))),
                       x0=np.squeeze(np.asarray(xi.A.reshape((1, xi.s * xi.q)))),
                       method='L-BFGS-B',
                       bounds=bnds)
        A_new = np.matrix(res['x']).reshape((xi.s, xi.q))
        xi.A[:, :] = A_new

        bnds = ((0.0, 1.0), ) * xi.q                           # Границы для весов p: [0.0 .. 1.0]
        cons = ({'type': 'eq', 'fun': lambda p: sum(p) - 1},)  # Нормированность суммы весов
        res = minimize(lambda p: xi.X(f, p=np.asmatrix(p).reshape((xi.q, 1))),
                       x0=np.squeeze(np.asarray(xi.p)),
```

```
                             method='SLSQP',
                             bounds=bnds,
                             constraints=cons)
            p_new = np.matrix(res['x']).reshape((xi.q, 1))

            exit_cond = sum(np.linalg.norm(A_new[:, i] - xi.A[:, i])**2 + (p_new[i, 0] - xi.p[i, 0])**2 for i in xrange(xi.q))
            xi.p[:, :] = p_new
        detM = np.log(la.det(xi.inf_matrix(f)))
        mu   = xi.mu(f)
        print 'Finally:\n%s\n\nlog(det(M(xi))) = %.3lf\nmu(alpha, xi) = %.3lf\n====================' % (
            xi, detM, mu)

        xi.reduce_plan()
        detM = np.log(la.det(xi.inf_matrix(f)))
        mu   = xi.mu(f)
        print 'After reduction:\n%s\n\nlog(det(M(xi))) = %.3lf\nmu(alpha, xi) = %.3lf\n====================' % (
            xi, detM, mu)
        return xi


def build_plan_dirscan(f, xi0):
    base = np.log(la.det(xi0.inf_matrix(f)))
    print 'Base plan: \n%s\nlog(det(M))=%.3lf\n====================\n' % (xi0, base)

    xi = Plan(xi0.s, xi0.q + 1)
    xi.set_bounds(xi0.bnds)
    xi.A[:, :-1] = xi0.A[:, :]
    xi.A[:,  -1] = 0
    xi.p[:-1, :] = xi0.p * (float(xi0.q) / (xi0.q + 1))
    xi.p[ -1, :] = 1.0 / (xi0.q + 1)

    def variate_point(f, xi, i, x, y):
        xi.A[0, i] = x
        xi.A[1, i] = y
        return la.det(xi.inf_matrix(f))

    func = lambda x, y: variate_point(f, xi, -1, x, y) - base
    dx = (xi.bnds[0][1] - xi.bnds[1][0]) / 100
    dy = (xi.bnds[1][1] - xi.bnds[1][0]) / 100
    x = np.arange(xi.bnds[0][0], xi.bnds[0][1] + dx, dx)
    y = np.arange(xi.bnds[1][0], xi.bnds[1][1] + dy, dy)
    X, Y = pl.meshgrid(x, y)
    Z = np.array([[func(xv, yv) for xv in x] for yv in y])
    pl.pcolor(X, Y, Z, cmap = pl.cm.PuBu)

    pl.xlim(xi.bnds[0])
    pl.ylim(xi.bnds[1])
#    pl.colorbar()
#    pl.title("Влияние положения точек плана на величину потерь информации det(M) = %.3lf by adding a point" % base)

    ij = np.unravel_index(Z.argmax(), Z.shape)
    xi.A[0, -1] = X.item(ij)
    xi.A[1, -1] = Y.item(ij)
    print 'Found plan: \n%s\nlog(det(M))=%.3lf\n====================\n' % (xi, np.log(la.det(xi.inf_matrix(f))))

    pl.show()

    return xi


def main():
    s = 2
    q = 2
    f = lambda alpha: np.matrix([[np.cos(alpha[0,0])],
                                 [np.sin(alpha[1,0])]])
    x0 = -5; x1 = 5
    A = (x1 - x0) * np.random.random((s, q)) + x0      # starting with random plan
    # A = [[-5, 5], [-5, 5]]
    xi = Plan(s, q)
    xi.A[:, :] = A
    xi.set_bounds(((-5.0, 5.0), (-5.0, 5.0)))
    build_plan_dirgrad(f, xi)
    print 'Checking solution for being optimal (less is better): [%.2lf]' % np.abs(xi.mu(f) - xi.eta(f))

    build_plan_dirscan(f, xi)


if __name__ == '__main__':
    np.set_printoptions(precision=3)
    main()
```

2.4. Результат работы программы

2.4.1. $f(a, b, x, y) = ax + by$

1) 2-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-0.737  0.439]
 [-3.672  1.339]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = -2.329
mu(alpha,xi) = 2.000
=================
Finally:
[[ 5.  5.]
 [-5.  5.]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = 6.438
mu(alpha,xi) = 2.000
=================
After reduction:
[[ 5.  5.]
 [-5.  5.]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = 6.438
mu(alpha,xi) = 2.000
=================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ 5.  5.]
 [-5.  5.]]
[[ 0.5]
 [ 0.5]]
log(det(M))=6.438
================

Found plan:
[[ 5.  5. -5.]
 [-5.  5. -5.]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]
log(det(M))=6.320
================
```

2) 3-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[ 2.057 -2.942  2.181]
 [ 3.438  3.69   3.34 ]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]

log(det(M(xi))) = 4.254
mu(alpha,xi) = 2.998
=================
On iter 2:
[[ 5. -5.  5.]
 [ 5.  5.  5.]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 6.438
mu(alpha,xi) = 2.000
=================
Finally:
[[ 5. -5.  5.]
 [ 5.  5.  5.]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 6.438
mu(alpha,xi) = 2.000
=================
After reduction:
[[ 5. -5.  5.]
 [ 5.  5.  5.]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 6.438
mu(alpha,xi) = 2.000
=================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ 5. -5.  5.]
 [ 5.  5.  5.]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]
log(det(M))=6.438
================

Found plan:
[[ 5. -5.  5.  5.]
 [ 5.  5.  5. -5.]]
[[ 0.188]
 [ 0.375]
 [ 0.188]
 [ 0.25 ]]
log(det(M))=6.373
================
```

3) 4-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[ 3.216 -3.381 -4.342  0.061]
 [-1.973 -2.626 -1.501 -1.705]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]

log(det(M(xi))) = 3.569
mu(alpha, xi) = 3.075
==============================
Finally:
[[ 5. -5. -5.  5.]
 [-5. -5. -5. -5.]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]

log(det(M(xi))) = 6.438
mu(alpha, xi) = 2.000
==============================
After reduction:
[[ 5. -5. -5.  5.]
 [-5. -5. -5. -5.]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]

log(det(M(xi))) = 6.438
mu(alpha, xi) = 2.000
==============================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ 5. -5. -5.  5.]
 [-5. -5. -5. -5.]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]
log(det(M))=6.438
============================

Found plan:
[[ 5. -5. -5.  5. -5.]
 [-5. -5. -5. -5. -5.]]
[[ 0.2]
 [ 0.2]
 [ 0.2]
 [ 0.2]
 [ 0.2]]
log(det(M))=6.397
============================
```

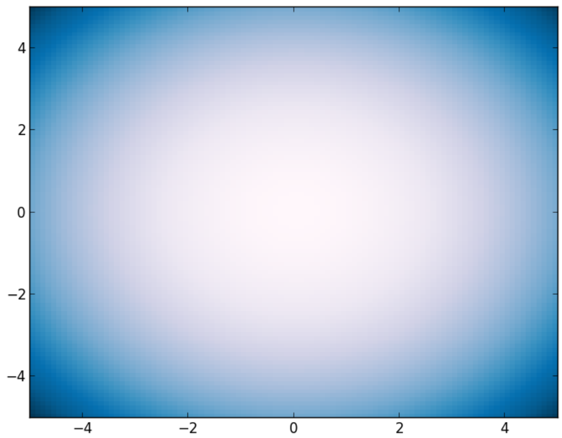

Рис. 1. Влияние положения точек плана на величину потерь информации

2.4.2. $f(a, b, x, y) = a\left(25 - x^2\right) + b\left(25 - y^2\right)$

1) 2-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[ 4.026  3.563]
 [ 1.152  0.392]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = 13.458
mu(alpha, xi) = 2.000
==============================
Finally:
[[  5.000e+00   1.931e-05]
 [ -5.000e+00   6.833e-01]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = 18.656
mu(alpha, xi) = 2.000
==============================
```

```
After reduction:
[[  5.000e+00   1.931e-05]
 [ -5.000e+00   6.833e-01]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = 18.656
mu(alpha, xi) = 2.000
==================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[  5.000e+00   1.931e-05]
 [ -5.000e+00   6.833e-01]]
[[ 0.5]
 [ 0.5]]
log(det(M))=18.656
=================

Found plan:
[[  5.000e+00   1.931e-05  -1.776e-14]
 [ -5.000e+00   6.833e-01  -5.000e+00]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]
log(det(M))=18.596
=================
```

2) 3-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-4.667  -2.21    4.196]
 [-4.045   0.462  -0.119]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]

log(det(M(xi))) = 17.360
mu(alpha, xi) = 2.829
==================
On iter 2:
[[ -5.000e+00   7.874e-08   5.000e+00]
 [ -5.000e+00   6.376e-01  -5.000e+00]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 18.656
mu(alpha, xi) = 2.000
==================
Finally:
[[ -5.000e+00   7.874e-08   5.000e+00]
 [ -5.000e+00   6.376e-01  -5.000e+00]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 18.656
mu(alpha, xi) = 2.000
==================
After reduction:
[[ -5.000e+00   7.874e-08   5.000e+00]
 [ -5.000e+00   6.376e-01  -5.000e+00]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = 18.656
mu(alpha, xi) = 2.000
==================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ -5.000e+00   7.874e-08   5.000e+00]
 [ -5.000e+00   6.376e-01  -5.000e+00]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]
log(det(M))=18.656
=================

Found plan:
[[ -5.000e+00   7.874e-08   5.000e+00  -1.776e-14]
 [ -5.000e+00   6.376e-01  -5.000e+00  -5.000e+00]]
[[ 0.188]
 [ 0.375]
 [ 0.188]
 [ 0.25 ]]
log(det(M))=18.637
=================
```

3) 4-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-0.361   3.514  -4.451   3.275]
 [-1.557  -3.027   4.628   4.227]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]

log(det(M(xi))) = 16.368
mu(alpha, xi) = 3.064
==================
On iter 2:
[[ -9.320e-06   5.000e+00  -5.000e+00  -1.473e-06]
 [ -5.000e+00  -5.000e+00  -5.000e+00   5.000e+00]]
[[ 0.271]
 [ 0.229]
 [ 0.229]
 [ 0.271]]

log(det(M(xi))) = 18.737
mu(alpha, xi) = 2.000
==================
```

```
Finally:
[[ -9.320e-06   5.000e+00  -5.000e+00  -1.473e-06]
 [ -5.000e+00  -5.000e+00  -5.000e+00   5.000e+00]]
[[ 0.271]
 [ 0.229]
 [ 0.229]
 [ 0.271]]

log(det(M(xi))) = 18.737
mu(alpha,xi) = 2.000
===================
After reduction:
[[ -9.320e-06   5.000e+00  -5.000e+00  -1.473e-06]
 [ -5.000e+00  -5.000e+00  -5.000e+00   5.000e+00]]
[[ 0.271]
 [ 0.229]
 [ 0.229]
 [ 0.271]]

log(det(M(xi))) = 18.737
mu(alpha,xi) = 2.000
===================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ -9.320e-06   5.000e+00  -5.000e+00  -1.473e-06]
 [ -5.000e+00  -5.000e+00  -5.000e+00   5.000e+00]]
[[ 0.271]
 [ 0.229]
 [ 0.229]
 [ 0.271]]
log(det(M))=18.737
==================

Found plan:
[[ -9.320e-06   5.000e+00  -5.000e+00  -1.473e-06  -1.776e-14]
 [ -5.000e+00  -5.000e+00  -5.000e+00   5.000e+00   5.000e+00]]
[[ 0.217]
 [ 0.183]
 [ 0.183]
 [ 0.217]
 [ 0.2  ]]
log(det(M))=18.696
==================
```

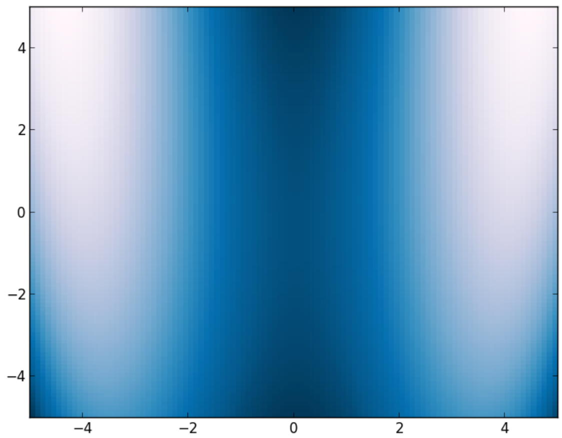

Рис. 2. Влияние положения точек плана на величину потерь информации

2.4.3. $f\left(a,\, b,\, x,\, y\right) = a \sin\left(x\right) + b \sin\left(y\right)$

1) 2-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-4.934   4.679]
 [-0.074  -2.263]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = -3.354
mu(alpha,xi) = 2.000
===================
Finally:
[[ -4.712e+00   4.712e+00]
 [ -6.798e-06   1.834e-06]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
===================
After reduction:
[[ -4.712e+00   4.712e+00]
 [ -6.798e-06   1.834e-06]]
[[ 0.5]
 [ 0.5]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
===================
```

```
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ -4.712e+00   4.712e+00]
 [ -6.798e-06   1.834e-06]]
[[ 0.5]
 [ 0.5]]
log(det(M))=-0.000
================

Found plan:
[[ -4.712e+00   4.712e+00  -4.700e+00]
 [ -6.798e-06   1.834e-06  -1.776e-14]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]
log(det(M))=-0.118
================
```

2) 3-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-2.233   4.14    4.19 ]
 [-4.719   4.182   0.667]]
[[ 0.333]
 [ 0.333]
 [ 0.333]]

log(det(M(xi))) = -1.639
mu(alpha,xi) = 2.719
================
On iter 2:
[[ -1.571e+00   4.712e+00   4.712e+00]
 [ -3.142e+00   3.142e+00   2.256e-06]]
[[ 0.25]
 [ 0.25]
 [ 0.5 ]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
================
Finally:
[[ -1.571e+00   4.712e+00   4.712e+00]
 [ -3.142e+00   3.142e+00   2.256e-06]]
[[ 0.25]
 [ 0.25]
 [ 0.5 ]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
================
After reduction:
[[ -1.571e+00   4.712e+00   4.712e+00]
 [ -3.142e+00   3.142e+00   2.256e-06]]
[[ 0.25]
 [ 0.25]
 [ 0.5 ]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[ -1.571e+00   4.712e+00   4.712e+00]
 [ -3.142e+00   3.142e+00   2.256e-06]]
[[ 0.25]
 [ 0.25]
 [ 0.5 ]]
log(det(M))=-0.000
================

Found plan:
[[ -1.571e+00   4.712e+00   4.712e+00   4.700e+00]
 [ -3.142e+00   3.142e+00   2.256e-06  -1.776e-14]]
[[ 0.188]
 [ 0.188]
 [ 0.375]
 [ 0.25 ]]
log(det(M))=-0.065
================
```

3) 4-х точечный исходный план

```
C:\Python27\python.exe "F:/Documents/PyCharm projects/Test/main.py"
On iter 1:
[[-4.28   -2.315   0.308   1.677]
 [-3.502   2.456   4.709  -3.274]]
[[ 0.25]
 [ 0.25]
 [ 0.25]
 [ 0.25]]

log(det(M(xi))) = -1.290
mu(alpha,xi) = 3.848
================
On iter 2:
[[-4.712  -1.571   1.571   1.571]
 [-3.142   3.142   5.     -3.142]]
[[  2.500e-01]
 [  5.000e-01]
 [  1.950e-16]
 [  2.500e-01]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
================
Finally:
[[-4.712  -1.571   1.571   1.571]
 [-3.142   3.142   5.     -3.142]]
[[  2.500e-01]
 [  5.000e-01]
 [  1.950e-16]
 [  2.500e-01]]

log(det(M(xi))) = -0.000
mu(alpha,xi) = 2.000
================
```

```
After reduction:
[[-4.712  -1.571   1.571]
 [-3.142   3.142  -3.142]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]

log(det(M(xi))) = -0.000
mu(alpha, xi) = 2.000
=============================
Checking solution for being optimal (less is better): [0.00]
Base plan:
[[-4.712  -1.571   1.571]
 [-3.142   3.142  -3.142]]
[[ 0.25]
 [ 0.5 ]
 [ 0.25]]
log(det(M))=-0.000
=============================

Found plan:
[[ -4.712e+00  -1.571e+00   1.571e+00  -4.700e+00]
 [ -3.142e+00   3.142e+00  -3.142e+00  -1.776e-14]]
[[ 0.188]
 [ 0.375]
 [ 0.188]
 [ 0.25 ]]
log(det(M))=-0.065
=============================
```

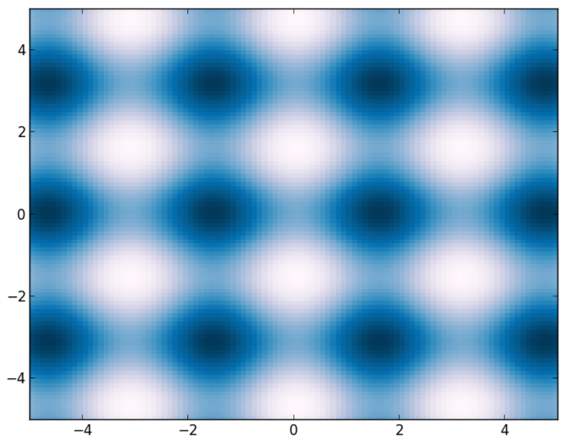

Рис. 3. Влияние положения точек плана на величину потерь информации

3. Вывод

В ходе лабораторной работы была проведено исследование метода D–оптимального планирования непрерывных оптимальных планов с помощью прямой градиентной процедуры.