Содержание

Задание

Решение системы нелинейных уравнений с помощью сочетания метода наискорейшего спуска и метода Ньютона.
Оценить скорость сходимости метода. На начальном этапе применить метод наискорейшего спуска, на завершающем – метод Ньютона.

Аннотация

В данной работе рассматриваются такие методы нахождения решения систем нелинейных уравнений, как метод простой итерации, методы градиентного и наискорейшего спусков и метод Ньютона.

Введение

Многие задачи физики и математики сводятся к решению систем нелинейных уравнений. При этом, очень часто для решения прикладных задач аналитическое решение не требуется, достаточно вычислить приближенное решение.

Для решения систем нелинейных уравнений нет универсального способа решения, поэтому при решении конкретной системы уравнений необходимо учитывать особенности данных уравнений. Для решения нелинейных систем не существует прямых методов, поэтому импользуются итерационные методы, основанные на существовании стационарной точки у сжимающего оператора, получающегося из системы.

Одним из таких методов и является рассматриваемый в данной работе метод Ньютона. Однако, в силу особой чувствительности этого метода к выбору начального приближения корня, на практике обычно используют более медленный, но менее чувствительный метод наискорейшего спуска на начальном этапе, получая лучшее приближение для метода Ньютона.

Определения и теоремы

Функционал — традиционное название для функции с областью определения в векторном пространстве и областью значений в множестве действительных чисел

Норма — неотрицательный невырожденный положительно однородный полуаддитивный функционал, определенный на линейном пространстве

Принцип сжимающих отображений

Пусть A — отображение полного метрического пространства (X, ρ_X) в себя. Пусть, кроме того $\forall x, y \in X$ выполнено следующее неравенство:

$$\rho(Ax, Ay) \leq q\rho(x, y), \tag{1}$$

где число $q \in (0, 1)$ и не зависит от x и y. Тогда существует единственная точка $z \in X$ такая, что

$$Az = z. \tag{2}$$

Такая точка z называется неподвижной.

Рассматривая в итерационном процессе x_n, как очередное приближение к стационарной точке, можно получить оценку

$$\rho(x_n, z) \leq \frac{q^n}{1-q}\rho(A(x_0), x_0) \tag{3}$$

Описание методов

Будем рассматривать систему нелинейных уравнений:

$$\begin{cases} f_1(x) &= 0 \\ f_2(x) &= 0 \\ \dots & \\ f_n(x) &= 0 \end{cases},$$

где $x = (x_1, x_2 \dots x_n)$
Общая проблема методов решения систем нелинейных уравнений заключается в их сугубо локальном характере сходимости. Это сильно затрудняет их применение в случаях, когда имеются проблемы с выбором начального приближения.
Для решения данной проблемы используют численные методы оптимизации, а именно, минимизации. Необходимо поставить задачу минимизации таким образом, чтобы её приближенное решение являлось решением исходной системы нелинейных уравнений. Для этого, можно, например, ввести функцию:

$$\Phi(x) = (f_1(x))^2 + (f_2(x))^2 + \ldots + (f_n(x))^2,$$

находя минимум которой, найдем и решение исходной системы.

Метод градиентного спуска

Из математического анализа известно, что функция растет быстрее всего в направлении своего градиента. Значит, оптимальным направлением движения для минимизации будет направление, противоположное градиенту в данной точке. То есть, для нахождения последующего приближения нужно выбирать точку, смещенную относительно предыдущего приближения на вектор антиградиента с неким коэффициентом, большим нуля.

$$x^{(k+1)} = x^{(k)} - \alpha \nabla \Phi(x^{(k)}), \tag{4}$$

где α, вообще говоря, зависит от текущего приближения, то есть $\alpha = \alpha_k$

Метод наискорейшего спуска

Итак, известно направление, в котором функция убывает быстрее всего. Однако, нужно еще определить, как далеко в этом направлении нужно искать следующее приближение. А оптимальным этот шаг будет, если

значение $\Phi(x^{(k+1)})$ минимальное из всех возможных в этом направлении. То есть

$$\alpha_k = \arg\min_{\alpha>0}(\Phi(x^{(k)} - \alpha\nabla\Phi(x^{(k)}))) \tag{5}$$

Сходимость этого метода линейная, что медленнее, чем, скажем, у метода Ньютона. Однако, как говорилось выше, метод Ньютона, как и другие, чувствителен к выбору начального приближения. Используя на начальном этапе метод наискорейшего спуска можно найти хорошее приближение для него.
Немного о реализации $argmin$. Использовать будем троичный поиск. Выбранный интервал разбивается на три равных интервала. Пусть, скажем, изначальный интервал был $[a, b]$. Тогда выбираются точки $m_1 = a + \frac{b-a}{3}$ и $m_2 = b - \frac{b-a}{3}$ и сравниваются значения в них. Допустим, ищется минимум. Если $f(m_1) > f(m_2)$, то $a = m_1$, в противном случае $b = m_2$. Алгоритм запускается заново с новыми параметрами. Так продолжается до тех пор, пока $||a - b|| > \epsilon$, где ϵ — один из параметров алгоритма — наименьшая длина отрезка, на котором ищется минимум,при достижении которой поиск останавливается и выбирается точка посередине такого отрезка. В итоге, можно достигнуть точности

$$||\arg\min_{\alpha}(f(\alpha)) - \frac{a+b}{2})|| \leq \frac{\epsilon}{2}.$$

Метод Ньютона

Если определено начальное приближение $x^{(0)}$, итерационный процесс нахождения решения системы методом Ньютона можно представить в виде

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}, \tag{6}$$

где значения $\Delta x^{(k)}$ определяются из решения системы линейных алгебраических уравнений, все коэффициенты которой выражаются через известное предыдущее приближение $x^{(k)}$. Вектор приращений

$$\Delta x^{(k)} = \begin{pmatrix} \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ \Delta x_3^{(k)} \\ \dots \\ \Delta x_n^{(k)} \end{pmatrix}$$

находится из решения уравнения

$$f(x^{(k)}) + J(x^{(k)})\Delta x^{(k)} = 0. \tag{7}$$

Здесь

$$J = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{pmatrix} \text{— Матрица Якоби}$$

первых производных вектор-функции $f(x)$. Выражая из (7) вектор приращений $\Delta x^{(k)}$ и подставляя его в (6), итерационный процесс нахождения решения можно записать в виде

$$x^{(k+1)} = x^{(k)} + [J(x^{(k)})]^{-1} f(x^{(k)}). \tag{8}$$

При реализации алгоритма метода Ньютона в большинстве случаев предпочтительным является не вычисление обратной матрицы Якоби, а нахождение из системы (7) значений приращений $\Delta x^{(k)}$ и вычисление нового приближения по (6).

Использование метода Ньютона предполагает дифференцируемость функций $f_1(x), \ldots, f_n(x)$ и невырожденность матрицы Якоби. В случае, если начальное приближение выбрано в достаточно малой окрестности искомого корня, итерации сходятся к точному решению, причем сходимость квадратичная(если только якобиан в точке решения не равен или близок к нулю).

В практических вычислениях в качестве условия окончания итераций обычно используется критерий

$$||x^{(k+1)} - x^{(k)}|| \leq \epsilon, \tag{9}$$

где ϵ — заданная точность.

Листинг программы

```
1  #include <iostream>
2  #include <functional>
3  #include <vector>
4  #include <cmath>
5  #include<fstream>
6  #include<limits>
7  #include<unistd.h>
8
9  using std::function;
10 using std::vector;
11 using std::cout;
12 using std::endl;
13 using std::ofstream;
14 using std::numeric_limits;
15
16 class MF2;
17 class F2;
18 class V2;
19 F2 grad(function<double(V2)>);
20
21 class V2 {
22 public:
23   double x,y;
24   double norm_eu()const{return sqrt(x*x + y*y);};
25   double norm_max()const{return fabs(x)>fabs(y) ? fabs(x) : fabs(
       y);};
26   V2(double _x=0, double _y=0) { x = _x; y = _y;};
27   V2(const V2 &v){x = v.x; y = v.y;};
28   V2 operator+ (V2 p2)const {return V2(this->x + p2.x, this->y +
       p2.y);};
29   V2 operator- ()const {return V2(-x,-y);};
30   V2 operator- (V2 p2)const {return V2(*this) + (-p2);};
31   V2 operator* (double k)const{return V2(k*x, k*y);}
32 };
33 const V2 hx(10e-10,0);
34 const V2 hy(0,10e-10);
35
36 class MF2 {
37 public:
38   function<double(V2)> f11,f12,f21,f22;
39   MF2(){}
40   MF2 (function<double(V2)> f1,function<double(V2)> f2,
41        function<double(V2)> f3,function<double(V2)> f4){
42     this->f11 = f1;
43     this->f12 = f2;
44     this->f21 = f3;
```

```
45      this->f22 = f4;
46    }
47    function<double(V2)> Det(){
48      return [this](V2 arg){
49        return
50        (this->f11)(arg)*(this->f22)(arg) - (this->f12)(arg)*(this
              ->f21)(arg);
51      };
52    }
53    double Det(V2 v){
54      return this->Det()(v);
55    }
56  };
57
58  class F2 {
59  public:
60    function<double(V2)> f1,f2;
61    F2(){}
62    F2 (function<double(V2)> f1,function<double(V2)> f2){
63      this->f1 = f1;
64      this->f2 = f2;
65    }
66    MF2 jacobian() const {
67      MF2 res(
68           grad(f1).f1,
69           grad(f1).f2,
70           grad(f2).f1,
71           grad(f2).f2
72              );
73      return res;
74    }
75    V2 operator()(V2 x){V2 res(f1(x),f2(x));return res;};
76  };
77  F2 grad(const function<double(V2)> f){
78    F2 res(
79    [f](V2 x){
80      return (f(x+hx) - f(x))/hx.norm_eu();
81    },
82    [f](V2 x){
83      return (f(x+hy) - f(x))/hy.norm_eu();
84    });
85    return res;
86  };
87  double argmin(function<double(double)> f, double eps, double l,
        double r){
88    static double m1,m2;
89    while(fabs(r - l) > eps){
90      m1 = l + (r-l)/3;
91      m2 = r - (r-l)/3;
```

```
92      if(f(m1) > f(m2))
93        l = m2;
94      else
95        r = m1;
96    }
97    return (r+l)/2;
98  }
99
100 template<class VecFun, class VecScal>
101 V2 gradient_descent(const VecFun f, VecScal p0, double eps,
        VecScal app_result, double radius, size_t maxItCount){
102   function<double(V2)> phi = [f](V2 x){return f.f1(x)*f.f1(x) + f
          .f2(x)*f.f2(x);};
103   V2 p1 = p0;
104   int i = 0;
105
106   ofstream ofs;
107   ofs.open("xs.dat", ofstream::out);
108   ofs << p1.x << "\t" << p1.y << endl;
109   double k;
110   do {
111     p0 = p1;
112     k = argmin(
113         [phi,p0](double a){return phi(p0 - phi(p0) * a);},
114         eps,
115         10e-8,
116         1 / (grad(phi)(p0)).norm_eu()
117         );
118     ++i;
119     p1 = p0 - grad(phi)(p0)*k;
120     ofs << p1.x << "\t" << p1.y << endl;
121     if (i > maxItCount || (app_result - p1).norm_eu() > radius){
122       cout << "Algorithm does not converge" << endl;
123       ofs.close();
124       p1 = V2( std::numeric_limits<double>::infinity(), std::
              numeric_limits<double>::infinity());
125       ofs.open("xs.dat", ofstream::out);
126       break;
127     }
128   } while ((p0 - p1).norm_max() > eps);
129   ofs.close();
130   return p1;
131 }
132 template<class VecFun, class VecScal>
133 V2 find_root(VecFun f, VecScal p0, double eps, VecScal app_result
        , double radius, size_t maxItCount){
134   V2 p1 = p0;
135   V2 dp(0,0);
136   MF2 J = f.jacobian();
```

```
137   MF2 A1, A2;
138
139   A1.f11 = f.f1;
140   A1.f12 = J.f12;
141   A2.f11 = J.f11;
142   A2.f12 = f.f1;
143   A1.f21 = f.f2;
144   A1.f22 = J.f22;
145   A2.f21 = J.f21;
146   A2.f22 = f.f1;
147   int i = 0;
148   ofstream ofs;
149   ofs.open("xs.dat", ofstream::app);
150   do{
151     i++;
152     p0 = p1;
153     p1.x = p0.x - A1.Det(p0) / J.Det(p0);
154     p1.y = p0.y - A2.Det(p0) / J.Det(p0);
155     ofs << p1.x << "\t" << p1.y << endl;
156     if (i > maxItCount || (app_result - p1).norm_eu() > radius){
157       cout << "Algorithm does not converge" << endl;
158       ofs.close();
159       p1 = V2( std::numeric_limits<double>::infinity(), std::
              numeric_limits<double>::infinity());
160       ofs.open("xs.dat", ofstream::out);
161       break;
162     }
163   }while((p0-p1).norm_max() > eps);
164   ofs.close();
165   return p1;
166 }
167
168 int main(int argc, const char *argv[])
169 {
170   F2 f;
171   f.f1 = [](V2 p) {return p.x*p.x + p.y*p.y -1 ;};
172   f.f2 = [](V2 p) {return p.y - p.x - 0.5 ;};
173   V2 app_result(0.4,0.9);
174   ofstream of("p0stats.dat");
175   double x=-0.1,y=0.5;
176   for (;x<0.9;x+=0.1){
177     cout << "x = " << x << endl;
178     for (;y<1.3;y+=0.1) {
179       V2 p00(x,y);
180       V2 p0 = gradient_descent<F2,V2>(f, p00 , 0.01, app_result
              ,1.0f, 1000);
181       if(numeric_limits<double>::infinity() != p0.x ||
              numeric_limits<double>::infinity() != p0.y){
```

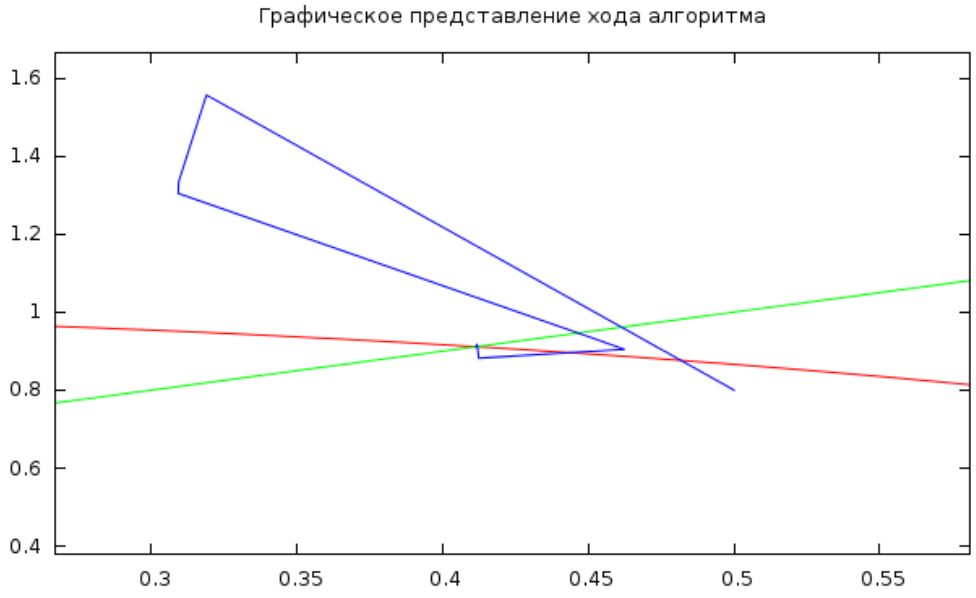
```
182           V2 result = find_root<F2,V2>(f,p0, 0.00001, app_result,
                  0.8,1000);
183           cout << result.x << ", " << result.y << endl;
184           of << endl << x << "\t" << y << "\t" << 0 << "\t" << 0 <<
                   "\t" << 0 << "\t" << endl;
185           of.close();
186           system("wc -l xs.dat | awk '{print $1}' >> p0stats.dat");
187           of.open("p0stats.dat",std::ostream::app);
188         }
189         else {
190           cout << "I know it does not converge here" << endl;
191           of << endl << x << "\t" << y << "\t" << 0 << "\t" << 0 <<
                   "\t" << 0 << "\t" << endl;
192           of << 0 << endl;
193         }
194       }
195       y = 0.3;
196     }
197     of.close();
198     double eps_0 = 10e-02;
199     double eps = 10e-02;
200     V2 p00(0.5,0.8);
201     ofstream ofe("epsstats.dat");
202     for(;eps > 10e-09;eps /= 10){
203       V2 p0 = gradient_descent<F2,V2>(f, p00 , eps_0, app_result
              ,0.8,10e+05);
204         if(numeric_limits<double>::infinity() != p0.x ||
              numeric_limits<double>::infinity() != p0.y){
205         V2 result = find_root<F2,V2>(f,p0, eps/100, app_result
              ,0.8,10e+06);
206       }
207       ofe << eps << endl;
208       ofe.close();
209       system("wc -l xs.dat | awk '{print $1}' >> epsstats.dat");
210       ofe.open("epsstats.dat",std::ostream::app);
211     }
212     ofe.close();
213     return 0;
214 }
```

Пример работы

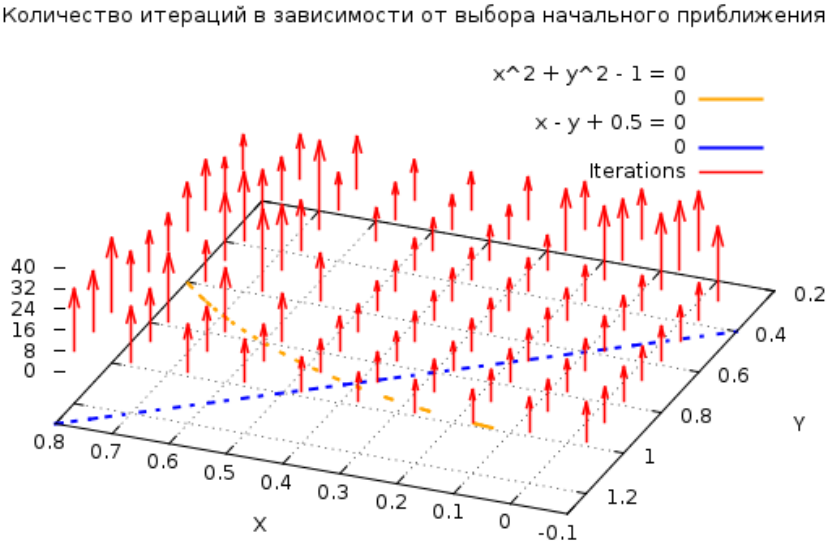
Для примера будет использоваться система из двух нелинейных уравнений

$$\begin{cases} x^2 + y^2 - 1 & = 0 \\ x - y + 0.5 & = 0 \end{cases} \tag{10}$$

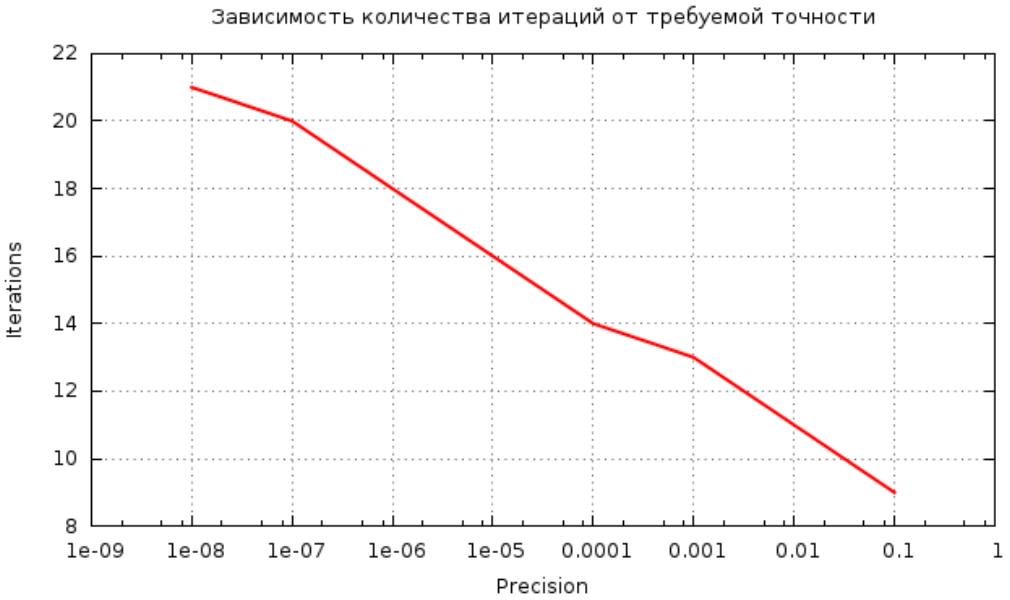
Решением которой с точностью до 10^{-7} является $(0.411438, 0.911438)$.



Зависимости количества итераций от начальных приближений и точности



На рисунке явно просматривается тенденция увеличения в разы количества итераций при удалении от точки решения.



Заключение

В результате выполнения работы были достигнуты следующие результаты:

- Написана программа, решающая системы нелинейных уравнений различной размерности, в пространствах с разными нормами с задаваемой точностью
- Исследована скорость сходимости связки методов при различных входных данных и точности

Список литературы

[1] Худак Ю. И. Бакушинский А. Б. *Основы функционального анализа.* 2009.

[2] Вержбицкий В. М. *Численные методы (Линейная алгебра и нелинейные уравнения). Учебное пособие для ВУЗов.* 2000.

[3] Вержбицкий В. М. *Численные методы (Математический анализ). Учебное пособие для ВУЗов.* 2000.

[4] Richard Hamming. *Numerical methods for scientists and engineers.* 1987.

[5] Robert Sedgewick. *Algorithms, 4th ed.* 2011.