Московский Авиационный Институт
(Государственный Технический Университет)

Факультет прикладной математики и физики.
Кафедра вычислительной математики и программирования.

Лабораторная работа №2
по курсу «Численные методы»

VI семестр

Студент Баскаков О.А.
Группа 08-306
Вариант 1

Москва, 2011.

Постановка задачи

1 Решение нелинейных уравнений

Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

$$2^x - x^2 - 0.5 = 0$$

Эквивалентный вид:

$$\log_2 x^2 + 0.5$$

Производная:

$$\ln 2 \cdot 2^x - 0.5\, x$$

2 Решение систем нелинейных уравнений

Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

$$\begin{cases} (x_1^2 + a^2)x_2 - a^3 = 0, \\ (x_1 - a/2)^2 + (x_2 - a/2)^2 - a^2 = 0. \end{cases} \qquad a = 2.$$

Теоретическая часть

1 Решение нелинейных уравнений

Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида

$$f(x) = 0 \tag{2.1}$$

заключается в нахождении значений *x*, удовлетворяющих (с заданной точностью) данному уравнению и состоит из следующих основных этапов:

1. *Отделение* (изоляция, локализация) корней уравнения.
2. *Уточнение* с помощью некоторого вычислительного алгоритма конкретного выделенного корня с заданной точностью.

Целью первого этапа является нахождение отрезков из области определения функции $f(x)$, внутри которых содержится только один корень решаемого уравнения. Иногда ограничиваются рассмотрением лишь какой-нибудь части области определения, вызывающей по тем или иным соображениям интерес. Для реализации данного этапа используются *графические* или *аналитические* способы.

При аналитическом способе отделения корней полезна следующая теорема:

Теорема: Непрерывная строго монотонная функция имеет и притом единственный нуль на отрезке [a,b] тогда и только тогда, когда на его концах она принимает значения разных знаков.

Достаточным признаком монотонности функции $f(x)$ на отрезке $[a,b]$ является сохранение знака производной функции.

Графический способ отделения корней целесообразно использовать в том случае, когда имеется возможность построения графика функции $y = f(x)$. Наличие графика исходной функции дает непосредственное представление о количестве и расположении нулей функции, что позволяет определить промежутки, внутри которых содержится только один корень. Если построение графика функции $y = f(x)$ вызывает затруднение, часто оказывается удобным преобразовать уравнение (2.1) к эквивалентному виду $f_1(x) = f_2(x)$ и построить графики функций $y = f_1(x)$ и $y = f_2(x)$. Абсциссы точек пересечения этих графиков будут соответствовать значениям корней решаемого уравнения.

Так или иначе, при завершении первого этапа, должны быть определены промежутки, на каждом из которых содержится только один корень уравнения.

Для уточнения корня с требуемой точностью обычно применяется какой-либо итерационный метод, заключающийся в построении числовой последовательности $x^{(k)}$ (k=0,1,2,…), сходящейся к искомому корню $x^{(*)}$ уравнения (2.1).

Метод Ньютона (метод касательных). При нахождении корня уравнения (2.1) методом Ньютона, итерационный процесс определяется формулой

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k = 0,1,2,... \tag{2.2}$$

Для начала вычислений требуется задание начального приближения $x^{(0)}$.

В качестве условия окончания итераций в практических вычислениях часто используется правило $\left|x^{(k+1)} - x^{(k)}\right| < \varepsilon \quad \Rightarrow \quad x^{(*)} \approx x^{(k+1)}$.

Метод простой итерации. При использовании метода простой итерации уравнение (2.1) заменяется эквивалентным уравнением с выделенным линейным членом

$$x = \varphi(x) \tag{2.5}$$

Решение ищется путем построения последовательности

$$x^{(k+1)} = \varphi(x^{(k)}) \qquad k = 0,1,2,... \tag{2.6}$$

начиная с некоторого заданного значения $x^{(0)}$. Если $\varphi(x)$ - непрерывная функция, а $x^{(k)} \quad (k = 0,1,2,...)$ - сходящаяся последовательность, то значение $x^{(*)} = \lim\limits_{k\to\infty} x^{(k)}$ является решением уравнения (2.5).

Условия сходимости метода и оценка его погрешности определяются теоремой [2]:

Теорема 2.3. Пусть функция $\varphi(x)$ определена и дифференцируема на отрезке [a,b]. Тогда если выполняются условия:

1) $\varphi(x) \in [a,b] \quad \forall x \in [a,b]$,

2) $\exists q : \left|\varphi'(x)\right| \le q < 1 \quad \forall x \in (a,b)$,

то уравнение (2.5) имеет и притом единственный на [a,b] корень $x^{(*)}$;

к этому корню сходится определяемая методом простой итерации последовательность $x^{(k)}$ $(k = 0,1,2,...)$, начинающаяся с любого $x^{(0)} \in [a,b]$.

При этом справедливы оценки погрешности ($\forall k \in N$):

$$\left|x^{(*)} - x^{(k+1)}\right| \le \frac{q}{1-q}\left|x^{(k+1)} - x^{(k)}\right|$$

$$\left|x^{(*)} - x^{(k+1)}\right| \le \frac{q^{k+1}}{1-q}\left|x^{(1)} - x^{(0)}\right|$$

2 Решение систем нелинейных уравнений

Систему нелинейных уравнений с *n* неизвестными можно записать в виде

$$\begin{cases} f_1(x_1, x_2, ..., x_n) = 0 \\ f_2(x_1, x_2, ..., x_n) = 0 \\ ..... \\ f_n(x_1, x_2, ..., x_n) = 0 \end{cases} \qquad (2.11)$$

или, более коротко, в векторной форме

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \qquad (2.12)$$

где $\mathbf{x}$ - вектор неизвестных величин, $\mathbf{f}$ - вектор-функция

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ ... \\ f_n(\mathbf{x}) \end{pmatrix}, \qquad \mathbf{0} = \begin{pmatrix} 0 \\ 0 \\ ... \\ 0 \end{pmatrix}.$$

В редких случаях для решения такой системы удается применить метод последовательного исключения неизвестных и свести решение исходной задачи к решению одного нелинейного уравнения с одним неизвестным. Значения других неизвестных величин находятся соответствующей подстановкой в конкретные выражения. Однако в подавляющем большинстве случаев для решения систем нелинейных уравнений используются итерационные методы.

В дальнейшем предполагается, что ищется изолированное решение нелинейной системы.

Как и в случае одного нелинейного уравнения, локализация решения может осуществляться на основе специфической информации по конкретной решаемой задаче (например, по физическим соображениям), и - с помощью методов математического анализа. При решении системы двух уравнений, достаточно часто удобным является графический способ, когда месторасположение корней определяется как точки пересечения кривых $f_1(x_1, x_2) = 0$, $f_2(x_1, x_2) = 0$ на плоскости (x_1, x_2).

Метод Ньютона. Если определено начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})^{\mathrm{T}}$, итерационный процесс нахождения решения системы (2.11) методом Ньютона можно представить в виде

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} + \Delta x_1^{(k)} \\ x_2^{(k+1)} = x_2^{(k)} + \Delta x_2^{(k)} \\ \quad \ldots\ldots\ldots\ldots\ldots \\ x_n^{(k+1)} = x_n^{(k)} + \Delta x_n^{(k)} \end{cases} \qquad k = 0,1,2,... \tag{2.13}$$

где значения приращений $\Delta x_1^{(k)}$, $\Delta x_2^{(k)}$,..., $\Delta x_n^{(k)}$ определяются из решения системы линейных алгебраических уравнений, все коэффициенты которой выражаются через известное предыдущее приближение $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})$

$$\begin{cases} f_1(\mathbf{x}^{(k)}) + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ f_2(\mathbf{x}^{(k)}) + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ \quad \ldots\ldots\ldots\ldots\ldots \\ f_n(\mathbf{x}^{(k)}) + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \end{cases} \tag{2.14}$$

В векторно-матричной форме расчетные формулы имеют вид

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{\Delta x}^{(k)} \qquad k = 0,1,2,... \tag{2.15}$$

где вектор приращений $\mathbf{\Delta x}^{(k)} = \begin{pmatrix} \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ ... \\ \Delta x_n^{(k)} \end{pmatrix}$ находится из решения уравнения

$$\mathbf{f}(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)})\mathbf{\Delta x}^{(k)} = \mathbf{0} \tag{2.16}$$

Здесь $\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \dots \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} \dots \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \dots\dots\dots\dots\dots\dots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} \dots \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$ - матрица Якоби первых производных вектор-функции $\mathbf{f}(\mathbf{x})$.

Выражая из (2.16) вектор приращений $\Delta\mathbf{x}^{(k)}$ и подставляя его в (2.15), итерационный процесс нахождения решения можно записать в виде

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}^{-1}(\mathbf{x}^{(k)})\mathbf{f}(\mathbf{x}^{(k)}) \qquad k = 0,1,2,... \qquad (2.17)$$

где $\mathbf{J}^{-1}(\mathbf{x})$ - матрица, обратная матрице Якоби. Формула (2.17) есть обобщение формулы (2.2) на случай систем нелинейных уравнений.

При реализации алгоритма метода Ньютона в большинстве случаев предпочтительным является не вычисление обратной матрицы $\mathbf{J}^{-1}(\mathbf{x}^{(k)})$, а нахождение из системы (2.14) значений приращений $\Delta x_1^{(k)}$, $\Delta x_2^{(k)}$,..., $\Delta x_n^{(k)}$ и вычисление нового приближения по (2.13). Для решения таких линейных систем можно привлекать самые разные методы, как прямые, так и итерационные (см. раздел 1.1), с учетом размерности *n* решаемой задачи и специфики матриц Якоби $\mathbf{J}(\mathbf{x})$ (например, симметрии, разреженности и т.п.).

Использование метода Ньютона предполагает дифференцируемость функций $f_1(\mathbf{x})$, $f_2(\mathbf{x})$,..., $f_n(\mathbf{x})$ и невырожденность матрицы Якоби ($\det \mathbf{J}(\mathbf{x}^{(k)}) \neq 0$). В случае, если начальное приближение выбрано в достаточно малой окрестности искомого корня, итерации сходятся к точному решению, причем сходимость квадратичная.

В практических вычислениях в качестве условия окончания итераций обычно используется критерий [2,5]

$$\left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\| \leq \varepsilon, \qquad (2.18)$$

где ε - заданная точность.

Метод простой итерации. При использовании метода простой итерации система уравнений (2.11) приводится к эквивалентной системе специального вида

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, ..., x_n) \\ x_2 = \varphi_2(x_1, x_2, ..., x_n) \\ \dots\dots\dots\dots\dots\dots \\ x_n = \varphi_n(x_1, x_2, ..., x_n) \end{cases} \qquad (2.21)$$

или, в векторной форме

$$\mathbf{x} = \boldsymbol{\varphi}(\mathbf{x}), \qquad \boldsymbol{\varphi}(\mathbf{x}) = \begin{pmatrix} \varphi_1(\mathbf{x}) \\ \varphi_2(\mathbf{x}) \\ \dots \\ \varphi_n(\mathbf{x}) \end{pmatrix} \qquad (2.22)$$

где функции $\varphi_1(\mathbf{x})$, $\varphi_2(\mathbf{x})$, ..., $\varphi_n(\mathbf{x})$ - определены и непрерывны в некоторой окрестности искомого изолированного решения $\mathbf{x}^{(*)} = (x_1^{(*)}, x_2^{(*)}, ..., x_n^{(*)})^{\mathrm{T}}$.

Если выбрано некоторое начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})^{\mathrm{T}}$, последующие приближения в методе простой итерации находятся по формулам

$$\begin{cases} x_1^{(k+1)} = \varphi_1(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \\ x_2^{(k+1)} = \varphi_2(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \\ \quad ..... \\ x_n^{(k+1)} = \varphi_n(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \end{cases} \qquad k = 0,1,2,... \tag{2.23}$$

или, в векторной форме

$$\mathbf{x}^{(k+1)} = \boldsymbol{\varphi}(\mathbf{x}^{(k)}),\ k = 0,1,2,... \tag{2.24}$$

Если последовательность векторов $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})^{\mathrm{T}}$ сходится, то она сходится к решению $\mathbf{x}^{(*)} = (x_1^{(*)}, x_2^{(*)}, ..., x_n^{(*)})^{\mathrm{T}}$.

Достаточное условие сходимости итерационного процесса (2.23) формулируется следующим образом [3]:

Теорема 2.4. Пусть вектор-функция $\boldsymbol{\varphi}(\mathbf{x})$ непрерывна, вместе со своей производной

$$\boldsymbol{\varphi}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial \varphi_1(\mathbf{x})}{\partial x_1} & \frac{\partial \varphi_1(\mathbf{x})}{\partial x_2} ... \frac{\partial \varphi_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial \varphi_2(\mathbf{x})}{\partial x_1} & \frac{\partial \varphi_2(\mathbf{x})}{\partial x_2} ... \frac{\partial \varphi_2(\mathbf{x})}{\partial x_n} \\ ..... \\ \frac{\partial \varphi_n(\mathbf{x})}{\partial x_1} & \frac{\partial \varphi_n(\mathbf{x})}{\partial x_2} ... \frac{\partial \varphi_n(\mathbf{x})}{\partial x_n} \end{bmatrix},$$

в ограниченной выпуклой замкнутой области G и

$$\max_{\mathbf{x} \in G} \left\| \boldsymbol{\varphi}'(\mathbf{x}) \right\| \le q < 1, \tag{2.25}$$

где q - постоянная. Если $\mathbf{x}^{(0)} \in G$ и все последовательные приближения

$$\mathbf{x}^{(k+1)} = \boldsymbol{\varphi}(\mathbf{x}^{(k)}),\ k = 0,1,2,...$$

также содержатся в G, то процесс итерации (2.23) сходится к единственному решению уравнения

$$\mathbf{x} = \boldsymbol{\varphi}(\mathbf{x})$$

в области G и справедливы оценки погрешности $(\forall k \in N)$:

$$\left\| \mathbf{x}^{(*)} - \mathbf{x}^{(k+1)} \right\| \le \frac{q^{k+1}}{1-q} \left\| \mathbf{x}^{(1)} - \mathbf{x}^{(0)} \right\|,$$

$$\left\| \mathbf{x}^{(*)} - \mathbf{x}^{(k+1)} \right\| \le \frac{q}{1-q} \left\| \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right\|$$

Исходный код на языке Python

1 Решение нелинейных уравнений

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log

def iterations(phi,segment,q,eps):
            xk = 0.0
            xkm = (segment[0]+segment[1])/2.0
            counter = 1
            while 42:
                  counter+= 1
                  if counter > 9000 : break
                  xk = phi(xkm)
                  if q/(1-q)*abs(xk-xkm) < eps:
                        #print ("finish with %d iters" % counter)
                        return xk, counter
                  xkm = xk

def Newton(func, func1, segment, eps):
            xk = 0.0
            xkm = (segment[0]+segment[1])/2.0
            counter = 1
            while 42:
                  counter+= 1
                  if counter > 9000 : break
                  xk = xkm - func(xkm)/func1(xkm)
                  if abs(xk-xkm) < eps:
                        #print ("finish with %d iters" % counter)
                        return xk, counter
                  xkm = xk

def func(x):
      return 2**x - x*x - 0.5

def phi(x):
      return log(x*x+1/2, 2)

def func1(x):
      return (2**x)*log(2) - 2.0*x

def main():
      eps = float( input("задайте eps: ") )
      s1 = iterations(phi,[4.0,4.5], 0.9, eps)
      print ("решение итерациями: %f за %d шагов" % s1)

      s2 = Newton(func,func1,[4.0,4.5], eps)
      print ("решение Ньютоном: %f за %d итераций" % s2)

      print("Зависимость сходимости от числа итераций:")
      print("eps\titer\tNewton")
      for eps in [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001]:
            s1, c1 = iterations(phi,[4.0,4.5], 0.9, eps)
            s2, c2 = Newton(func,func1,[4.0,4.5], eps)
            print(eps, "\t%d \t%d" %(c1,c2) )

      return

if (__name__ == "__main__"):
      main()
```

2 Решение систем нелинейных уравнений

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy
from matrix_2 import *

def f1(x):
      return (x[0][0]**2 + 4)*x[1][0] - 8

def f2(x):
      return (x[0][0]-1)**2 + (x[1][0]-1)**2 - 4

def phi1(x):
      return 8.0/(x[0][0]**2+4)

def phi2(x):
      return sqrt(4-(x[1][0]-1)**2)+1

def df1dx1(x):
      return 2*x[0][0]*x[1][0]

def df1dx2(x):
      return x[0][0]**2+4

def df2dx1(x):
      return 2*(x[0][0]-1)

def df2dx2(x):
      return 2*(x[1][0]-1)


def maximal(x):
      return max( x.transponate()[0] )

Af = [[df1dx1,df1dx2],[df2dx1,df2dx2]]
f = [f1,f2]
phis = [phi2,phi1]

segment = [[2.0,3.0],[0.5,1.0]]
m = 2
n = 2

def main():
      eps = 0.00001 #   eps = float(input("задайте eps: ") )
      dx = Matrix("",m,1)
      A = Matrix("",m,n)
      b = Matrix("",m,1)
      xk = Matrix("",m,1)
      xk.M = [ [(segment[i][0] + segment[i][1])/2.0] for i in range(m)]

      count = 0
      while 42:
            count+=1

            A.M = [ [ Af[i][j](xk)  for j in range(n)]
                                              for i in range(m)]

            b.M =[ [-f[i](xk)] for i in range(m)]

            A.b = b.transponate()[0]
```

```
            A.build_LU()
            v = A.solve(A.shift_b(A.b))

            dx.M = [ [v[i]] for i in range(dx.m)]
            xk = xk + dx

            if abs(maximal(dx))<eps:
                  print ("ответ Ньютон:")
                  print ("за %d шага" % count)
                  xk.pr()
                  break
      q = 0.5
      koef = q/(1-q)
      xk = Matrix("",m,1)
      xkm = Matrix("",m,1)
      xkm.M = [ [(segment[i][0] + segment[i][1])/2.0] for i in range(m)]

      count = 0
      while 42:
            count+=1
            xk.M = [ [phis[i](xkm)] for i in range(m)]
            if koef*abs((xk-xkm).norm()) < eps:
                  print ("ответ Итерации:")
                  print ("за %d шагов" % count)
                  xk.pr()
                  break
            xkm = deepcopy(xk)
      return 0

if (__name__ == "__main__"):
      main()
```

Протокол тестирования

1 Решение нелинейных уравнений

```
oleg@debian:~/lab2_release$ ./lab2_1.py
задайте eps: 0.00000001
решение итерациями: 4.142310 за 47 шагов
решение Ньютоном: 4.142310 за 5 итераций
Зависимость сходимости от числа итераций:
eps         iter  Newton
0.1         5     2
0.01        11    3
0.001       17    4
0.0001      23    4
1e-05       29    5
1e-06       35    5
1e-07       41    5
```

2 Решение систем нелинейных уравнений

```
oleg@debian:~/lab2$ ./lab2_2.py
ответ Ньютон:
за 4 шага
Matrix 2x1 :
      2.96463
      0.62554
-----
ответ Итерации:
за 9 шагов
Matrix 2x1 :
      2.96463
      0.62554
-----
```

Выводы

1 Решение нелинейных уравнений

Все методы справились со своей задачей. В ходе отработки решения тестовых задач наблюдалась закономерность, согласно которой для большей точности требуется большее количество итераций. Среди представленных методов наивысшую эффективность продемонстрировал метод Ньютона. Однако, метод ньютона не всегда пригоден, так как требует вычисления производных.

2 Решение систем нелинейных уравнений

Снова хорошо себя проявил метод Ньютона.