1. Понятие ООП. Инкапсуляция, полиморфизм, наследование.

Структурное программирование означает то, что есть некие блоки, на которые делится программа.
ООП – технология, возникшая в кризис ПО.
Любая моделированная система состоит из объектов. Каждый объект характеризуется своим внутренним состоянием.

Инкапсуляция – свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса, а также объединить и защитить данные.
Наследование – позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Абстрактный класс – класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. От такого класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного.
Множественное наследие: у класса может быть более одного предка. Класс наследует методы всех предков.
Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию. (один интерфейс, множество реализаций)

Абстракция – придание объекту характеристик, которые отличают его от всех других объектов.

2. Классы и объекты. Общий формат объявления классов. Доступ к членам классов.

Класс объектов – абстрактное описание, которое объединяет набор объектов. В структурном программировании ключевым объектов является действие.

Классы – описание некоторой структуры программы, обладающей набором внутренних переменных (свойств) и функций, имеющих доступ к свойствам – методов.

Класс объединяет в себе как данные, так и методы обработки этих данных. Класс – это тип данных, созданный пользователем. Под класс память не выделяется. Класс – это всего лишь описание типа данных.

class имя_класса
{закрытые поля и методы класса
Public: открытые поля и методы класса
} список объектов

К закрытым членам класса можно обращаться только внутри класса.

Создание объекта класса: SimpleClass MyObj
Полю объекта присваивается значение: MyObj.number = 5

3. Открытые и закрытые члены класса. Объявление. Отличия.

В C++ по умолчанию члены класса являются закрытыми. Закрытыми члены класса делают для ограничения несанкционированного внешнего доступа к этим членам.
К закрытым членам класса можно обращаться только внутри класса.

class SimpleClass{
int m,n; - закрытые члены класса
public: - ключевое слово
void show();}; - открытые члены класса

Можно использовать ключевое слово private.
class SimpleClass{
private: - ключевое слово
int m,n; - закрытые
public: - ключевое слово
void show();}; - открытые

4. Статистические члены класса. Объявление статических членов класса.

Статистический член является общим для всех объектов класса. В отличии от обычного поля класса статистическое поле не исчезает при удалении объекта.

Когда член класса объявляется как статический, то тем самым компилятору дается указание, что должна существовать только одна копия этого члена, сколько бы объектов этого класса ни создавалось.
static int m;
static void msum(int k);

Пример программы:
```
#include <iostream.h>
class counter {
static int count;
public:
void setcount(int i) {count = i;};
void showcount () {cout << count << " "; }
};
int counter::count; // определение count
int main() {
counter a, b;
a.showcount (); // выводит 0
b.showcount (); // выводит 0
a.setcount (10); // установка статического count в 10
a.showcount (); // выводит 10
b.showcount (); // также выводит 10
return 0;}
```

5. Дружественные функции. Объявление и использование.

Внешние функции, которые имеют доступ к закрытым членам класса, называются

дружественными функциями.
Чтобы задекларировать функцию как дружественную для класса, необходимо указать прототип этой функции в описании класса, предварив его ключевым словом friend.
friend void show(MyClass obj);
Дружественная функция имеет доступ к закрытым членам класса.

```
class B
class A{
double x;
public:
A(double z) {x = z;}
friend double summa (A a, B b);
}a(3.5);
class B{
double y;
public:
B(double z) {y = z;}
friend double summa (A a, B b);
}b(2.3);
double summa (A a, B b){
return a.x+b.y;}
int main(){
cout << “Total is” << summa(a,b) << endl;
return 0;}
```

Дружественными по отношению к классу могут быть отдельные методы другого класса или целый класс. В последнем случае все методы дружественного класса имеют доступ к закрытым полям и методам класса.

```
class B;
class A{
double x;
public:
A(double z) {x = z;}
double summa (B b);
} a(3.5);
class B{
double y;
public:
B(double z) {y = z;}
friend double A::summa(B b);
} b(2.3);
int main(){
cout << “Total is” << a.summa(b) << endl;
return 0;}
double A::summa(B b){
return x + b.y;}
```

Дружественные классы используются не очень часто – существуют более эффективные способы реализации доступа методов одного класса к членам другого, главным среди которых является механизм наследования.

6. Создание конструктора в классе и отличие от создания метода в классе.

Конструктор – метод, который автоматически вызывается при создании объекта.
Деструктор – метод, вызываемый автоматически при выгрузке объекта из памяти.

Создание и перегрузка конструктора:
1. Имя конструктора совпадает с именем класса.
2. Конструктор не возвращает результат и для него тип результата не указывается вообще.

Во всем остальном конструктор напоминает обычный метод, за исключением лишь той разницы, что метод необходимо вызывать в явном виде, а конструктор вызывается автоматически и только при создании объекта. Конструктор может иметь аргументы, а может и не иметь. Конструктор можно перегружать.

```
class MyClass{
public:
int m.n;
myClass(){
m = 0;
n = 0;}
void show(){
cout << “m=” << m << endl;
cout << “n=” << n << endl;}};
int main(){
MyClass obj;
obj.show();
return 0;}
```

Правила перегрузки конструктора такие же как и правила перегрузки методов и функций. Наличие перегруженного конструктора существенно повышает гибкость программного кода.
Конструктор для создания нового объекта на основе уже существующего объекта:
Аргументом конструктора является объект того же класса, в котором описан конструктор.

```
MyClass(MyClass obj){
m = obj.m;
n = obj.n;}
```

Конструкторы незаменимы с практической точки зрения, их применение позволяет легко и быстро решать сложные задачи.

7. Наследование классов и типы наследования.

Одним из фундаментальных механизмов, обеспечивающих уникальную гибкость и продуктивность программ, написанных на C++, является наследование. Наследование позволяет одним объектам получать свойства других объектов.

В C++ классы могут наследовать друг друга. Это значит, что класс-наследник(производный класс) получает свойства своего родительского(базового) класса. Кроме этого, производный класс может добавлять к свойствам, полученным от базового класса, свои собственные. Производный класса также может быть базовым по отношению к другому классу.

При наследовании классов производный класс фактически создается не на пустом месте, а на основе базового класса, получая в наследство от своего родителя поля и методы. Какие поля и методы наследуются, а какие нет, определяются их доступностью. Они могут быть открытыми (public), закрытыми (private) или защищенными (protected). Защищенные члены класса фактически являются закрытыми, но от private они отличаются способом наследования.

Public-член базового класса наследуется производным классом при любом механизме наследования, но в производном классе унаследованный член может иметь разный уровень доступа. При public-наследовании он останется public-членом в производном классе.

При private-наследовании public-член базового класса становится private-членом производного класса.

Механизм наследования vs. тип члена	public-наследование	private-наследование	protected-наследование
public-член	public	private	protected
private-член	не наследуется	не наследуется	не наследуется
protected-член	protected	private	protected

Создание на основе базового класса производного класса:

class производный_класс: тип_наследования базовый класс{
//программный код производного класса
}список_объектов;

Если производный класс B создается на основе базового класса A через открытое наследование (public-наследование):

class A{
//Программный код класса A
}список_объектов;
//…
class B: public A{
//программный код класса B
}список_объектов;

```
#include <iostream>
using namespace std;
//Базовый класс:
class A{
  private:
  int x;
  public:
  int y;
  };
//Производный класс:
class B: public A{
  public:
  int z;
  void show(){
    cout<<"y = "<<y<<endl;
    cout<<"z = "<<z<<endl;}
  };
int main(){
  B obj;
  obj.y=1;
  obj.z=2;
  obj.show();
  return 0;}
```

В результате выполнения этой программы получим

```
y = 1
z = 2
```

8. Работа с файлами в C++. Описание файловой переменной. Способ доступа к файлам и их содержимому.

Для работы с файлами необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> — файловый ввод и <ofstream> — файловый вывод.
Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

```
//запись
ofstream fout;
fout.open("cppstudio.txt");
fout << "Работа с файлами в C++"; // запись строки в файл
fout.close(); // закрываем файл
//чтение
char buff[50];
ifstream fin("cppstudio.txt");
if (!fin.is_open()) // если файл не открыт
cout << "Файл не может быть открыт!\n"; // сообщить об этом
else
{
fin >> buff;
cout << buff << endl;
}
ofstream fout("cppstudio.txt", ios_base::app)
```

ios_base::in открыть файл для чтения
ios_base::out открыть файл для записи
ios_base::ate при открытии переместить указатель в конец файла
ios_base::app открыть файл для записи в конец файла
ios_base::trunc удалить содержимое файла, если он существует
ios_base::binary открытие файла в двоичном режиме

9. Механизм ввода\вывода в C++. Потоки в C++. Связь потока с файлами. Чтение и запись текстовых файлов.

В C++ используется механизм потокового ввода/вывода. Поток - механизм преобразования значений различного типа в последовательность символов (вывод) и наоборот (ввод), в значение переменной.
cout - стандартный поток вывода (экран)
cerr - стандартный поток вывода ошибок (экран)
cin - стандартный поток ввода (клавиатура)
В поток ошибок и вывода можно писать, используя оператор <<
Из потока ввода можно читать, используя оператор >>
Так же, существуют различные манипуляторы. Используются они так:

```
cout<<setw(4)<<setfill('#')<<left<<12;
```

showbase
noshowbase инициирует отображение основания системы счисления
showpos
noshowpos инициирует явное отображение знака (+) для положительных значений

uppercase
nouppercase инициирует преобразование выводимых символов в верхний регистр
showpoint
noshowpoint инициирует отображение десятичной точки при выводе вещественных чисел
skipws
noskipws инициирует пропуск пробельных символов при вводе
left инициирует левое выравнивание, заполнение справа
right инициирует правое выравнивание, заполнение слева
internal инициирует внутреннее заполнение (между значением и знаком или основанием системы счисления)
scientific инициирует вывод вещественных значений в научном формате (d.ddddddddE+dd)
fixed инициирует вывод вещественных значений в фиксированном формате (d.dddddd)
setbase (int base) изменяет систему счисления (10, 8, 16)
dec инициирует вывод целочисленных значений в десятичной системе счисления
oct инициирует вывод целочисленных значений в восьмеричной системе счисления
hex инициирует вывод целочисленных значений в шестнадцатеричной системе счисления
setfill (char n) задает заполняющий символ
setprecision (int n) изменяет точность выводимых значений
setw (int n) задает ширину поля вывода
Но самые важные -
endl; // заканчивается строка, т.е. вставляется символ '\n'
ends; // заканчивается строка записью ' '
flush; // очистка буфера потока

Так как синтаксис работы с потоками ввода/вывода идентичен файловому, тот же код можно использовать и при работе с файлами

```
#include <fstream>
using namespace std;
void main()
{
ifstream inputfile("z:\data.txt"); // создается поток ввода из файла
inputfile>>x; // все то же самое что и для cin
ofstream outputfile("z:\res.txt"); // создается поток вывода в файл
outputfile<<x; // все то же самое что и для cout
}
```

10. Абстрактные классы.

Иногда возникает необходимость определить класс, который не предполагает создания конкретных объектов. Например, класс фигуры. В реальности есть конкретные фигуры: квадрат, прямоугольник и тд.

Абстрактные классы - это классы, которые содержат или наследуют без переопределения хотя бы одну чистую виртуальную функцию.
Абстрактный класс определяет интерфейс для переопределения производными классами.

```
class Figure
{
public:
```

```
virtual double getSquare() = 0;
virtual double getPerimeter() = 0;
virtual void showFigureType() = 0;
};
class Rectangle : public Figure
{
private:
double width;
double height;
public:
Rectangle(double w, double h) : width(w), height(h)
{
}
double getSquare() override
{
return width * height;
}
double getPerimeter() override
{
return width * 2 + height * 2;
}
void showFigureType()
{
std::cout << "Rectangle" << std::endl;
}
};
```

11. Переопределение методов и виртуальные функции.

Полиморфизм времени исполнения обеспечивается за счет использования производных классов и виртуальных функций. Виртуальная функция — это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или в нескольких производных классах. Виртуальные функции являются особыми функциями, потому что при вызове объекта производ-ного класса с помощью указателя или ссылки на него C++ определяет во время исполнения про-граммы, какую функцию вызвать, основываясь на типе объекта. Для разных объектов вызываются разные версии одной и той же виртуальной функции. Класс, содержащий одну или более вир-туальных функций, называется полиморфным классом

Переопределение означает, что вы создали иерархию классов, у которой в базовом классе есть виртуальная функция и вы можете переопределить её в производном классе

```
class Base {
public:
virtual void prepare() { cout << "Base";}
};
class Derived : public Base {
public:
  virtual void prepare() { cout << "Derived";}
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
```

```
    Base *b=new Derived();
    b->prepare();
    return a.exec();
}
```
Переопределение относится именно к виртуальным функциям.

12. Создание и перегрузка конструктора.

Конструктор - это метод класса, выполняемый при создании объекта данного класса. Он не имеет типа возвращаемого значения и должен называться так же, как и сам класс. Конструктор должен иметь спецификатор доступа public.

```
class Test{
public:
    Test(){}
};
```

Конструктор нельзя вызвать.
Можно переопределить конструктор параметрами, так же как и функцию

13. Динамическое выделение памяти под объекты и высвобождение её.

Для динамического выделения памяти под объекты класса используют оператор new. Сначала объявляется указатель на объект соответствующего класса, после чего в формате указатель = new класс соответствующей командой под объект выделяется место в памяти и адрес передается указателю. Если при создании объекта конструктору необходимо передать аргументы, они указываются в круглых скобках после имени класса. Для удаления объекта из памяти используется команда delete указатель, где указатель является указателем на объект.

Конструктор – метод, вызываемый при создании объекта, деструктор вызывается при удалении объекта из памяти. Динамические массивы объектов создаются аналогично динамическим массивам базовых типов.

```
class MyClass {
public:
double x;
void show(){
cout << ”x=” << x << endl;}
MyClass (double z){
x = z;
cout << “Object with x = “ << x << “has been created!\n”;}
MyClass(){
x = 0;
cout << “Object with x = “ << x << “has been created!\n”;}
~MyClass(){
cout << “Object with x = “ << x << “has been deleted!\n”;}};
int main(){
MyClass *p;
p = new MyClass;
p -> show();
delete p;
```

```
p = new MyClass(1);
p -> show();
delete p;
return 0;}
```

Для элементарных типов данных:

```
int *p_darr = new int[num];
```

Для объектов:

```
Test **objPtr = new Test*[COUNT_OBJ];
for (int i = 0; i < COUNT_OBJ; ++i) objPtr[i] = new Test(i + 5);
```

Для освобождения используется оператор delete

```
delete p_darr;
```

Матрица:

```
int **a = new int* [n]; // Создаем массив указателей
for (int i = 0; i < n; i++)
{
   a[i] = new int [n]; // Создаем элементы
}
```

14. Абсолютная и относительная погрешность

Абсолютная и относительная погрешности
Δ – абсолютная погрешность, δ – относительная погрешность

Алгебраическое выражение	Абсолютная погрешность	Относительная погрешность
Суммы $a+b$	$\Delta(a+b)=\Delta a+\Delta b$	$\delta(a+b)\leqslant\delta(a)$, где $\delta(a)\geqslant\delta(b)$
Разности $a-b$	$\Delta(a-b)=\Delta a+\Delta b$	–
Произведения $a\cdot b$	$\Delta(a\cdot b)=a\cdot\Delta b+b\cdot\Delta a$	$\delta(ab)=\delta(a)+\delta(b)$
Частного $\frac{a}{b}$	$\Delta\left(\frac{a}{b}\right)=\frac{b\cdot\Delta a+a\cdot\Delta b}{b^2}$	$\delta\left(\frac{a}{b}\right)=\delta(a)+\delta(b)$
Степени a^n	$\Delta(a^n)=n\cdot a^{n-1}\cdot\Delta a$	$\delta(a^n)=n\cdot\delta(a)$
Корня $\sqrt[n]{a}$	$\Delta(\sqrt[n]{a})=\frac{\Delta a}{n\cdot\sqrt[n]{a^{n-1}}}$	$\delta(\sqrt[n]{a})=\frac{\delta(a)}{n}$

Абсолютная погрешность — это разность между измеренным и действительным значениями измеряемой величины:

$$\Delta A = A_{изм} - A,$$

где $A_{изм}$, A - измеряемое и действительное значения; ΔA - абсолютная погрешность.

Абсолютную погрешность выражают в единицах измеряемой величины. Абсолютную погрешность, взятую с обратным знаком, называют поправкой.

Относительная погрешность р равна отношению абсолютной погрешности ΔA к действительному значению измеряемой величины и выражается в процентах:

$$\beta = \frac{\Delta A}{A}100$$

15. Численные методы решения алгебраических и трансцендентных уравнений. Постановка задачи. Методы отделения корней.

РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ

Постановка задачи и этапы решения.

При решении алгебраических и трансцендентных уравнений, встречающихся на практике, очень редко удается найти точное решение. Поэтому приходится применять различные приближенные способы определения корней. В общей постановке задачи обычно требуют непрерывность функции f(x), корни которой ищутся с заданной точностью.

Пример отделения корней.

Приведем лишь один ПРИМЕР: определить количество и приближенное расположение корней уравнения sinX - 0.2X=0.

Для решения перепишем уравнение в виде sinX=0.2X. Поскольку значения функции y=sinX лежат между -1 и 1, то корни уравнения могут быть только на отрезке [-5,5]. Ясно, что один из корней - это X=0 . Если же на отрезке [-5,5] нарисовать графики функций $y_1(X)$= sinX и $y_2(X)$=0.2X, то сразу будет видно, что точки их пересечения (а это и есть корни нашего уравнения) расположены на отрезках [-3,-2] и [2,3].

Ответ: исходное уравнение имеет 3 корня: $X_1=0$, $X_2 \in [-3,-2]$ и $X_3 \in [2,3]$.

В дальнейшем мы будем считать, что уравнение f(X)=0 задано на отрезке [a,b], на котором расположен ровно один его корень, и исследовать решение второй части задачи - уточнение корней. По-видимому, эта задача является самой простой из всех вычислительных задач, встречающихся на практике. Существуют несколько хороших методов решения данной задачи.

Метод половинного деления

хорошо знаком по доказательству теоремы о промежуточном значении в курсе математического анализа. Его суть заключается в построении последовательности вложенных отрезков, содержащих корень. При этом на каждом шаге очередной отрезок делится пополам и в качестве следующего отрезка берется та половина, на которой значения функции в концах имеют разные знаки. Процесс продолжают до тех пор, пока длина очередного отрезка не станет меньше, чем величина 2ε. Тогда его середина и будет приближенным значением корня с точностью ε.

Алгоритм данного метода можно записать так:

1.Ввести данные (a, b, ε).

2.Если нужная точность достигнута (| b - a | < 2ε) то иди к п.6

3.Возьми середину очередного отрезка (C = (a + b)/ 2).

4.Если значения функции в точках a и C одного знака (f(a)*f(C)>0), то в качестве следующего отрезка возьми правую половину (a=C), иначе левую (b=C).

5.Иди к п.2.

6.Напечатать ответ ((a + b) / 2)

Метод хорд и касательных

(или **метод Ньютона)** применяется только в том случае, когда . f'(X) и f''(X) не изменяют знака на отрезке [a,b], т.е.функция f(X) на отрезке [a,b] монотонна и не имеет точек перегиба.

Суть метода та же самая - построение последовательности вложенных отрезков, содержащих корень, однако отрезки строятся по-другому. На каждом шаге через концы дуги графика функции f(X) на очередном отрезке проводят хорду и из одного конца проводят касательную. Точки пересечения этих прямых с осью OX и образуют следующий отрезок. Процесс построения прекращают при выполнении того же условия (| b - a | < 2ε).

Для того, чтобы отрезки получались вложенными, нужно проводить ту касательную из конца, которая пересекает ось OX на отрезке [a,b]. Перебрав четыре возможных случая, легко увидеть, что касательную следует проводить из того конца, где знак функции совпадает со знаком второй производной. Также несложно заметить, что касательная проводится либо все время из правого, либо все время из левого конца. Будем считать для определенности , что этот конец - b .

Формулы, употребляемые в методе Ньютона, хорошо известны из аналитической геометрии:

Уравнение хорды, проходящей через точки (a,f(a)) и (b,f(b)): y = f(a)+(x-a)*(f(b)-f(a))/(b-a),

откуда точка пересечения с осью OX: X= a - f(a) *(b-a)/(f(b)-f(a)).
Уравнение касательной, проходящей через точку (b,f(b)): -y=f(b)+f'(b)(x-b),
откуда точка пересечения с осью OX: X= b - f(b)/f'(b).

При составлении алгоритма снова естественно использовать для концов отрезка только две переменные a и b и писать: a= a - f(a) *(b-a)/(f(b)-f(a)) и

(1.1)

b= b - f(b)/f'(b) (1.2)

Однако, в этом случае важен порядок формул (1.1) и (1.2).

Метод итераций

применяется к уравнению вида X= u(x) на отрезке [a,b], где:
а) модуль производной функции u(x) невелик: | u'(x) | <= q < 1 (x∈[a,b])
б) значения u(x) лежат на [a,b] ,т.е. a <= u(x) <= b при x∈[a,b].

Если заранее известно, что на отрезке [a,b] расположен ровно один корень уравнения X=u(x), то достаточно проверить выполнение условия а).
Упражнения: определить, применим ли метод итераций для уравнений:
1.7 X=ln(3X+2) на отрезке [0,5]. А на отрезке [1,5]?
1.8 $X=e^{x-9}$ на отрезке [10,12]. А на отрезке [0,1]?

Сведение исходного уравнения к виду, пригодному для применения метода итераций.

Сведение уравнения f(x)=0 к нужному виду обычно осуществляют одним из двух способов:

1) Выражают один из X, входящих в уравнение, например уравнение e^x - x-=0 приводят к виду:

$X=e^x$ или $X = \ln(x)$

2) Подбирают множитель и производят преобразования: f(x)=0 => k*f(x)=0 => x=x + k*f(x), т.е. u(x)=x+ k*f(x). Например, если 0 < m < f'(x) <= M при X∈[a,b], то можно в качестве k взять величину - 1/M, и тогда 0 <= u'(x) = 1 +к* f'(x)= 1- 1/M * f' (x) <= 1- m/M

Суть и обоснование метода итераций.

Суть метода итераций заключается в построении рекуррентной последовательности чисел, сходящейся к решению, по формуле $x_{к+1} = u(x_к)$, к=0,1,2,..., где x_0∈[a,b] - произвольная точка.

Справедливость метода обосновывает следующая ТЕОРЕМА:

Пусть на [a,b] задана функция u(x), удовлетворяющая условиям а) и б), а x_0 - произвольная точка отрезка [a,b], причем уравнение x=u(x) имеет корень. Тогда последовательность {Xк}, построенная по формуле $x_{к+1} = u(x_к)$ сходится к решению не медленнее, чем геометрическая прогрессия со знаменателем q.

Доказательство: Сравним расстояния от точек $x_{к+1}$ и $x_к$ до решения (обозначим его C), используя теорему Лагранжа:

$| x_{к+1}-C| = |u(x_к)- u(C)| = | (x_к-c) u'(y)|<= |(x_к-c)|* \max | u'(x) | = q |(x_к-c)|$,

что и требовалось доказать.

Замечание 1. Требование существования корня приведено в теореме лишь для простоты доказательства.

Замечание 2. Теорема является одним из частных случаев применения принципа сжимающих отображений, который часто применяется в самых разных вопросах многих точных наук.

Условие окончания вычислений в методе итераций.

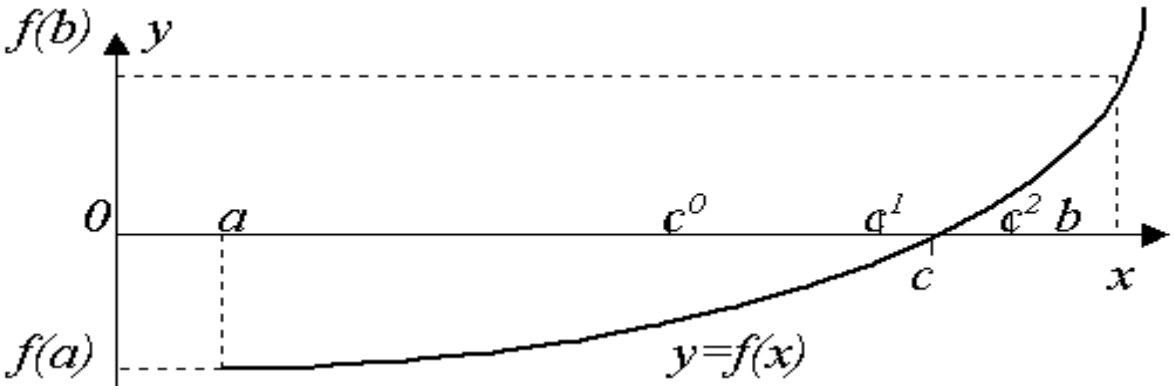
Замечание 3. Процесс построения последовательности следует обрывать, когда станет верным неравенство $|x_{к+1}-x_к|< \varepsilon*(1-q)/q$. В этом случае $x_{к+1}$ и дает приближение к решению с требуемой точностью.

16. Итерационные Методы уточнения корней

1.1. Отделение корней

В общем случае отделение корней уравнения *f(x)=0* базируется на известной теореме, утверждающей, что если непрерывная функция *f(x)* на концах отрезка *[a,b]* имеет значения разных знаков, т.е. *f(a)ґ f(b)J 0*, то в указан-ном промежутке содержится хотя бы один корень. Например, для уравнения *f(x)= x^3-6x+2=0* видим, что при *x®Ґ f(x)>0*, при *x®-Ґ f(x)<0*, что уже свидетельствует о наличии хотя бы одного корня.
В общем случае выбирают некоторый диапазон, где могут обнаружиться корни, и осуществляют "прогулку" по этому диапазону с выбранным шагом *h* для обнаружения перемены знаков *f(x)*, т.е. *f(x)ґ f(x+h)<0*.
При последующем уточнении корня на обнаруженном интервале не надейтесь никогда найти *точное* значение и добиться обращения функции в нуль при использовании калькулятора или компьютера, где сами числа представлены ограниченным числом знаков. Здесь критерием может служить приемлемая *абсолютная* или *относительная погрешность* корня. Если корень близок к нулю, то лишь относительная погрешность даст необходимое число значащих цифр. Если же он весьма велик по абсолютной величине, то критерий абсолютной погрешности часто дает совершенно излишние верные цифры. Для функций, быстро изменяющихся в окрестности корня, может быть привлечен и критерий: *абсолютная величина значения функции* не превышает заданной допустимой погрешности.

1.2. Уточнение корней методом половинного деления (дихотомии)



Самым простейшим из методов уточнения корней является метод половинного деления, или метод дихотомии, предназначенный для нахождения корней уравнений, представленных в виде *f(x)=0*.
Пусть непрерывная функция *f(x)* на концах отрезка [a,b] имеет значения разных знаков, т.е. *f(a)ґ f(b) J 0*(рис. 1), тогда на отрезке имеется хотя бы один корень.
Возьмем середину отрезка *c=(a+b)/2*. Если *f(a)ґ f(c) J 0*, то корень явно принадлежит отрезку от *a* до *(a+b)/2* и в противном случае от *(a+b)/2* до *b*.

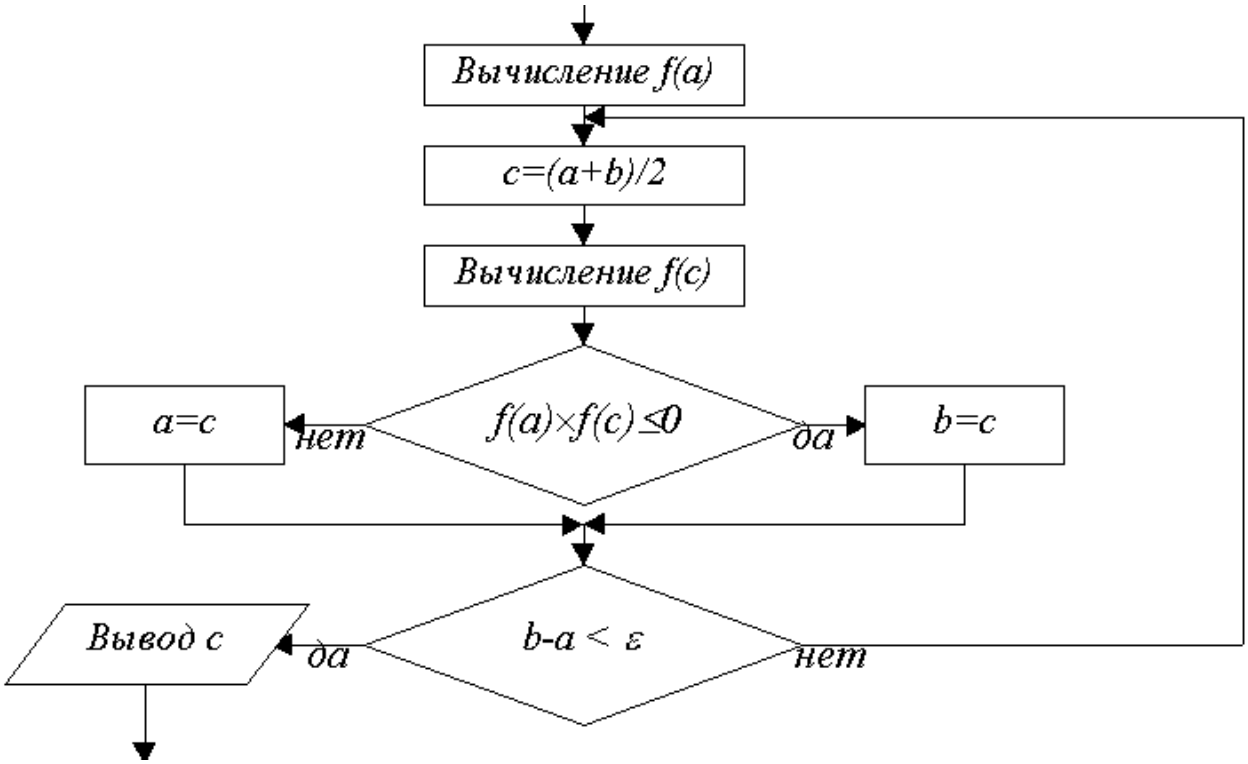


Рис. 2. Блок-схема метода половинного деления

Поэтому берем подходящий из этих отрезков, вычисляем значение функции в его середине и т.д. до тех пор, пока длина очередного отрезка не окажется меньше заданной предельной абсолютной погрешности $(b-a)<e$.

Так как каждое очередное вычисление середины отрезка c и значения функции $f(c)$ сужает интервал поиска вдвое, то при исходном отрезке [a,b] и предельной погрешности e количество вычислений n определяется условием $(b-a)/2^n<e$, или $n \sim \log_2((b-a)/e)$. Например, при исходном единичном интервале и точности порядка 6 знаков ($e \sim 10^{-6}$) после десятичной точки достаточно провести 20 вычислений (итераций) значений функции.

С точки зрения машинной реализации (рис. 2) этот метод наиболее прост и используется во многих стандартных программных средствах, хотя существуют и другие более эффективные по затратам времени методы.

1.3. Уточнение корней методом хорд

В отличие от метода дихотомии, обращающего внимание лишь на знаки значений функции, но не на сами значения, метод хорд использует пропорциональное деление интервала (рис. 3).

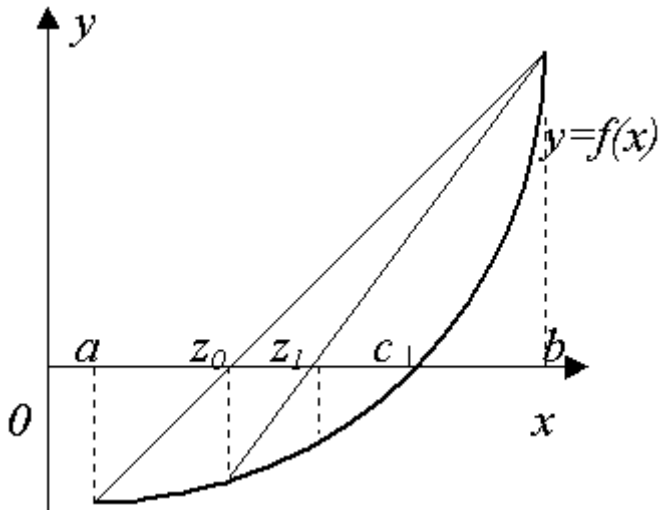


Рис. 3. Метод хорд

Здесь вычисляются значения функции на концах отрезка, и строится "хорда", соединяющая точки *(a,f(a))* и *(b,f(b))*. Точка пересечения ее с осью абсцисс

$$Z = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

принимается за очередное приближение к корню. Анализируя знак *f(z)* в сопоставлении со знаком *f(x)* на концах отрезка, сужаем интервал до [*a,z*] или [*z,b*] и продолжаем процесс построения хорд до тех пор, пока разница между очередными приближениями не окажется достаточно малой (в пределах допустимой погрешности) $|Z_n - Z_{n-1}| < e$.
Можно доказать, что истинная погрешность найденного приближения:

$$\left|X^* - Z_n\right| \le \frac{M-m}{m}\left|Z_n - Z_{n-1}\right|,$$

где X^* - корень уравнения, Z_n и Z_{n+1} - очередные приближения, *m* и *M* - наименьшее и наибольшее значения *f(x)* на интервале [*a,b*].

1.4. Уточнение корней методом касательных (Ньютона)

Обширную группу методов уточнения корня представляют *итерационные методы* - методы последовательных приближений. Здесь в отличие от метода дихотомии задается не начальный интервал местонахождения корня, а его начальное приближение.
Наиболее популярным из итерационных методов является *метод Ньютона (метод касательных)*.

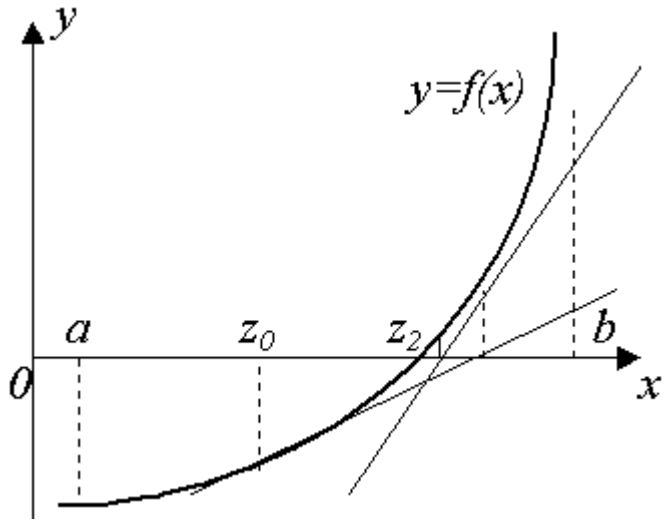


Рис. 4. Метод касательных

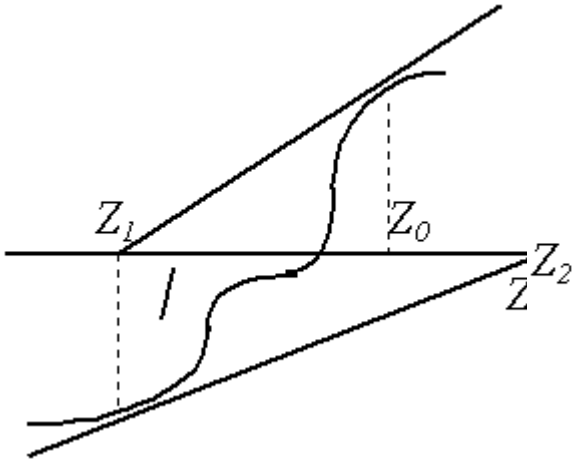
Пусть известно некоторое приближенное значение Z_n корня X^*. Применяя формулу Тейлора и ограничиваясь в ней двумя членами, имеем

$$f(X^*) \approx f(Z_n) + (X^* - Z_n) f'(Z_n) = 0,$$

откуда

$$X^* \approx Z_{n+1} = Z_n - \frac{f(Z_n)}{f'(Z_n)}.$$

Геометрически этот метод предлагает построить касательную к кривой *y=f(x)* в выбранной точке $x=Z_n$, найти точку пересечения её с осью абсцисс и принять эту точку за очередное приближение к корню (рис. 4).



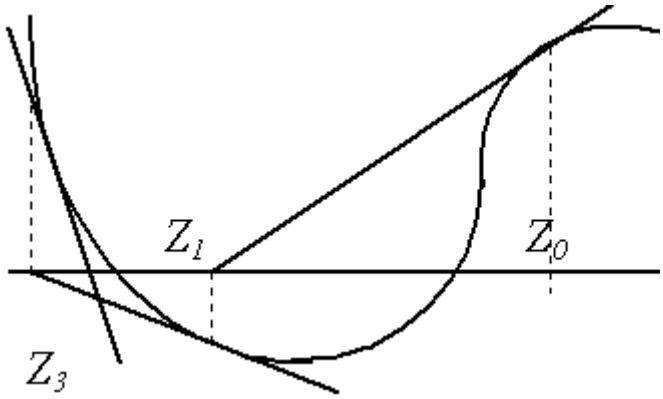


Рис. 5. Расходящийся процесс *Рис. 6. Приближение к другому корню*

Очевидно, что этот метод обеспечивает сходящийся процесс приближений лишь при выполнении некоторых условий (например при непрерывности и знакопостоянстве первой и второй производной функции в окрестности корня) и при их нарушении либо дает расходящийся процесс (рис. 5), либо приводит к другому корню (рис. 6).
Очевидно, что для функций, производная от которых в окрестности корня близка к нулю, использовать метод Ньютона едва ли разумно.
Если производная функции мало изменяется в окрестности корня, то можно использовать видоизменение метода

$$Z_{n+1} = Z_n - \frac{f(Z_n)}{f'(Z_0)}, \quad n=0,1,2,\ldots$$

Существуют и другие модификации метода Ньютона.

1.5. Уточнение корней методом простой итерации

Другим представителем итерационных методов является *метод простой итерации*.
Здесь уравнение $f(x)=0$ заменяется равносильным уравнением $x=j(x)$ и строится последовательность значений

$$X_{n+1} = \varphi(X_n), \quad n=0,1,2,\ldots$$

Если функция $j(x)$ определена и дифференцируема на некотором интервале, причем $|j'(x)|< 1$, то эта последовательность сходится к корню уравнения $x=j(x)$ на этом интервале.
Геометрическая интерпретация процесса представлена на рис. 7. Здесь первые два рисунка (а, б) демонстрируют одностороннее и двустороннее приближение к корню, третий же (в) выступает иллюстрацией расходящегося процесса ($|j'(x)| > 1$).

а *б* *в*

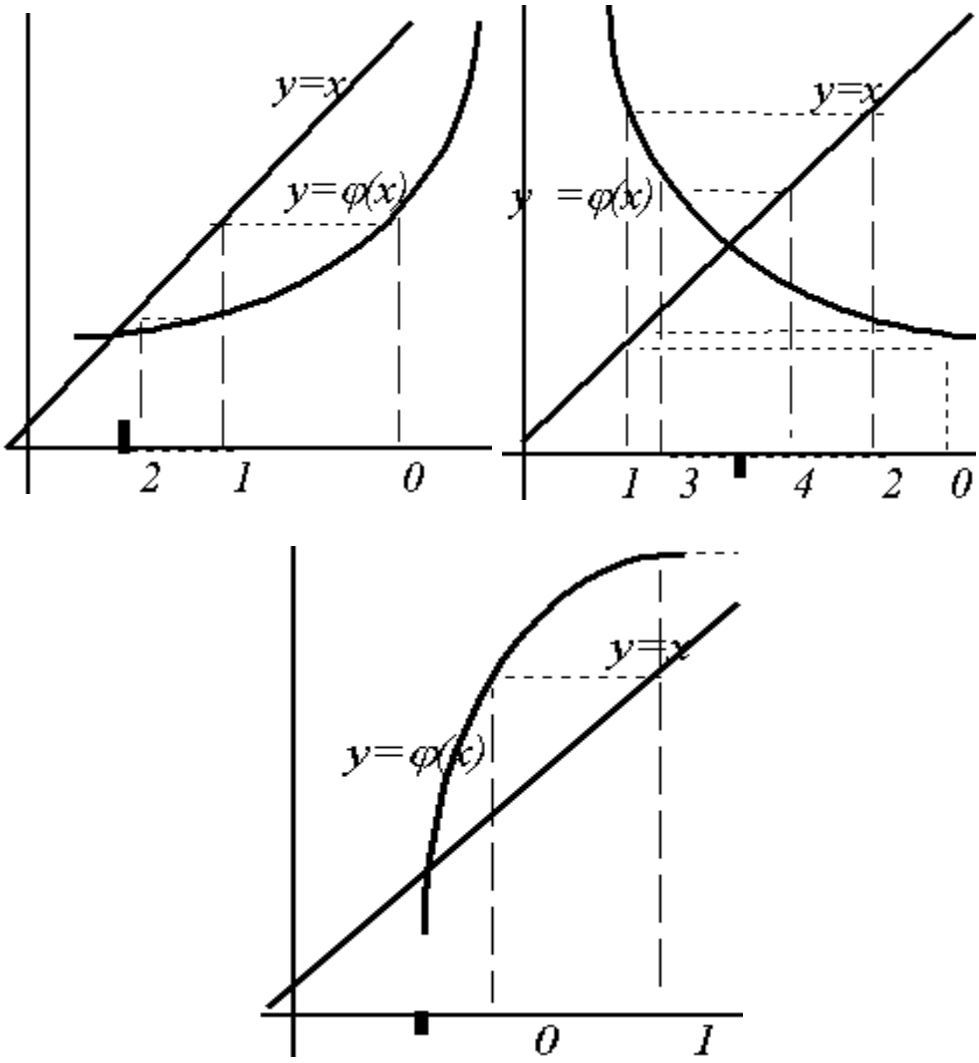


Рис. 7. Геометрическая интерпретация метода простой итерации

Если *f '(x)>0*, то подбор равносильного уравнения можно свести к замене *x=x-l Ч f(x)*, т.е. к выбору *j (x)= x-l Ч f(x)*, где *l >0* подбирается так, чтобы в окрестности корня *0 < j '(x)=1- l Ч f '(x) J 1*. Отсюда может быть построен итерационный процесс

$$X_{n+1} = X_n - \frac{f(X_n)}{M}, \quad n = 0, 1, 2, \ldots$$

где *M i max |f '(x)|* (в случае *f '(x)< 0* возьмите функцию *f(x)* с противоположным знаком). Возьмем для примера уравнение $x^3 + x - 1000 = 0$. Очевидно, что корень данного уравнения несколько меньше *10*. Если переписать это уравнение в виде $x = 1000 - x^3$ и начать итерационный процесс при $x_0=10$, то из первых же приближений очевидна его расходимость. Если же учесть $f'(x)=3x^2+1>0$ и принять за приближенное значение максимума *f '(x) M=300*, то можно построить сходящийся итерационный процесс на основе представления

$$X = X - \frac{X^3 + X - 1000}{300}$$

Можно и искусственно подобрать подходящую форму уравнения, например:

$$X = \sqrt[3]{1000 - X} \text{ или } X = \frac{1000}{X^2} - \frac{1}{X}$$

Заметим, что существуют и другие методы (наискорейшего спуска, Эйткена-Стеффенсена, Вегстейна, Рыбакова и т.д.) уточнения корней, обладающие высокой скоростью сходимости.

17. Постановка задачи аппроксимации функции. Существование и единственность интерполяционного многочлена (полинома)

Приближение функции более простой функцией называется аппроксимацией (от латинского approximo – приближаюсь). Аппроксимирующую функцию строят таким образом, чтобы отклонения (в некотором смысле) от в заданной области было наименьшим. Понятие “малого отклонения” зависит от того, каким способом оценивается близость двух функций , поэтому оно будет уточняться в дальнейшем при рассмотрении конкретных методов аппроксимации

Если для табличной функции y=f(x) имеющей значения x0, f(x0) требуется построить аппроксимирующую функцию g(x), совпадают в узлах xi с заданной, то такой способ называется интерполяцией.

Задача аппроксимации: Функция y= f(x) задана таблицей. Необходимо найти функцию заданного вида: y=F(x) которая в точках x1, x2, …, xn принимает значения, как можно более близкие к табличным y1, y2, …, yn.

Существование и единственность интерполяционного многочлена (полинома)

Пусть на отрезке [a, b] **задана сетка** $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ **, в узлах которой заданы значения**

$y_i = f(x_i)$, $i = 0,1,\ldots,n$, $x_i \neq x_j$ **при** $i \neq j$**. Тогда существует единственный полином** $P_n(x)$ **степени** n **такой, что**

$P_n(x_i) = y_i$.

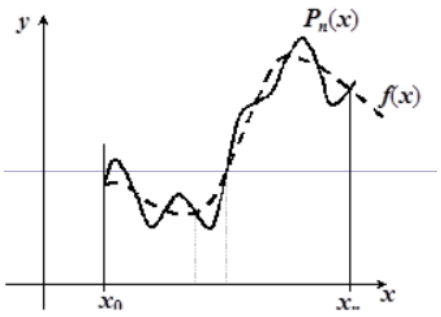


Рис. 5.1. Интерполяционный полином

Таким образом, по любой интерполяционной таблице, у которой все xi различны между собой, всегда можно построить единственный интерполяционный полином, степень которого на единицу меньше , чем размерность интерполяционной таблицы.

Доказательство теоремы я выписывал отдельно ниже после 18 вопроса, там очень странно описание, поэтому используйте на свой страх и риск.

18. Постановка задачи численного дифференцирования

В точках x_i, $i = \overline{0,n}$-известны значения функции $y_i = f(x_i)$. Задача численного дифференцирования – найти значение производной f ’ или f ” или … $f^{(n)}$ в любой наперед заданной точке x. Поступаем также как при интерполяции.

Применяется при невозможности нахождения обычным дифференциированием производной.

Основные методы решения:

Метод наименьших квадратов

Полином Лангранжа

Полином Ньютона

Полином Стирлинга

Примеры объяснения методов на всякий случай

Полином Лангранжа

$$L_n(x) = \sum_{i=0}^{n} p_i(x) y_i \quad , \qquad (2)$$

В самом деле, во-первых, очевидно, степень построенного полинома $L_n(x)$ не выше n и, во-вторых, в силу условия (3) имеем:

$$L_n(x) = \sum_{i=0}^{n} p_i(x_j) y_i = p_j(x_j) y_j = y_j \qquad (j = 0,1,2,,\ldots,n) .$$

Причем

$$p_i(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} .$$

Подставив значение $p_i(x)$ в формулу (2), получим:

$$L_n(x) = \sum_{i=0}^{n} y_i \frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} . \qquad (4)$$

Это и есть *интерполяционная формула Лагранжа.*

Формуле (4) Лагранжа можно придать более сжатый вид. Для этого, введя обозначение $\prod_{n+1}(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$, получим:

$$L_n(x) = \prod_{n+1}(x) \sum_{i=0}^{n} \frac{y_i}{\prod'_{n+1}(x_i)(x-x_i)} .$$

При $n = 1$ мы имеем две точки, и формула Лагранжа представляет в этом случае уравнение прямой $y = L_1(x)$, проходящей через две заданные точки:

$$y = \frac{x-b}{a-b} y_0 + \frac{x-a}{b-a} y_1 ,$$

где a, b - абсциссы этих точек.

При $n = 2$ получим уравнение параболы $y = L_2(x)$, проходящей через три точки:

$$y = \frac{(x-b)(x-c)}{(a-b)(a-c)} y_0 + \frac{(x-a)(x-c)}{(b-a)(b-c)} y_1 + \frac{(x-a)(x-b)}{(c-a)(c-b)} y_2 ,$$

где a, b, c - абсциссы данных точек.

Полином Стирлинга

Взяв среднее арифметическое первой и второй интерполяционных формул Гаусса (2) и (3) (предыдущей рассылки), получим *формулу Стирлинга*

$$P(x) = y_0 + q\frac{\Delta y_{-1} + \Delta y_0}{2} + \frac{q^2}{2}\Delta^2 y_{-1} + \frac{q(q^2-1^2)}{3!}\cdot\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2} +$$

$$+\frac{q^2(q^2-1^2)}{4!}\Delta^4 y_{-2} + \frac{q(q^2-1^2)(q^2-2^2)}{5!}\cdot\frac{\Delta^5 y_{-3} + \Delta^5 y_{-2}}{2} +$$

$$+\frac{q(q^2-1^2)(q^2-2^2)}{6!}\Delta^6 y_{-3} + \ldots + \frac{q(q^2-1^2)(q^2-2^2)(q^2-3^2)\ldots[q^2-(n-1)^2]}{(2n-1)!}\times$$

$$\times\frac{\Delta^{2n-1} y_{-n} + \Delta^{2n-1} y_{-(n-1)}}{2} + \frac{q(q^2-1^2)(q^2-2^2)\ldots[q^2-(n-1)^2]}{(2n)!}\Delta^{2n} y_{-n}, \qquad (1)$$

где $q = \frac{x - x_0}{h}$.

Легко видеть, что $P(x_i) = y_i$ при $i = 0, \pm 1, \ldots, \pm n$.

Пример. Используем интерполяционную формулу Стирлинга для решения примера 2. (см. выше в прошлой рассылке). Подставляя соответствующие коэффициенты из таблицы разностей (таблица 2) в формулу (1), получим:

$$P_n(x) = 1.8847 + \frac{0.1016 + 0.0912}{2}q - 0.0104\frac{q^2}{2} - \frac{0.0008 + 0.0004}{2}\cdot\frac{q(q^2-1^2)}{6}$$

или

$$P_n(x) = 1.8847 + 0.0964q - 0.0104\frac{q^2}{2} - 0.0001q(q^2-1^2),$$

где

$$q = \frac{x - 0.35}{0.05} = 20(x - 0.35).$$

Это и есть искомый интерполяционный полином Стирлинга.

Полином Ньютона

$$L_n(x) = y_0 + q\frac{\Delta y_0}{1!} + q(q-1)\frac{\Delta^2 y_0}{2!} + \ldots + q(q-1)\ldots(q-n+1)\frac{\Delta^n y_0}{n!}$$

Задача: Построить интерполяционный многочлен в форме Ньютона для функции, заданной таблицей.

X	0	1	2	3
y	-2	-5	0	-4

Решение:
Составим таблицу разделенных разностей.

$$f_{10}=\frac{y_1-y_0}{x_1-x_0}=-3$$

$$f_{20}=\frac{f_{11}-f_{10}}{x_2-x_0}=4$$

$$f_{11}=\frac{y_2-y_1}{x_2-x_1}=5 \qquad f_{30}=\frac{f_{21}-f_{20}}{x_3-x_0}=-\frac{17}{6}$$

$$f_{21}=\frac{f_{12}-f_{11}}{x_3-x_1}=-4,5$$

$$f_{12}=\frac{y_3-y_2}{x_3-x_2}=-4$$

Запишем формулу для интерполяционного многочлена Ньютона и подставим туда полученные значения:

$$P(x)=y_0+f_{10}(x-x_0)+f_0(x-x_0)(x-x_1)+f_{30}(x-x_0)(x-x_1)(x-x_2)=$$
$$=-2-3(x-0)+4(x-0)(x-1)-\frac{17}{6}(x-0)(x-1)(x-2)=$$
$$=-\frac{17}{6}x^3+12,5x^2-\frac{38}{3}x-2$$

Ответ: Интерполяционный многочлен Ньютона имеет вид: $P(x)=-\frac{17}{6}x^3+12,5x^2-\frac{38}{3}x-2$

Доказательство теоремы разобьем на две части. В первой части докажем существование интерполяционного многочлена, а во второй часть докажем единственность интерполяционного многочлена.

1. **Существование**

Пусть $A_{n+1} = (x - x_0)(x - x_1)\dots(x - x_n)$. Тогда по известной теореме алгебры правильная несократимая рациональная дробь

$$\frac{L_n(x)}{A_{n+1}(x)}$$

может быть представлена в виде суммы простейших дробей

$$\frac{L_n(x)}{A_{n+1}(x)} = \sum_{m=0}^{n} \frac{R_m}{(x - x_m)}. \qquad (2.3)$$

Из (2.3) можно получить, что

$$R_m = \frac{L_n(x_m)}{\widetilde{A}_{n+1}(x_m)} \qquad (2.4)$$

где $\widetilde{A}_{n+1}(x_m) = (x_m - x_0)(x_m - x_1)(x_m - x_{m-1})(x_m - x_{m+1})\dots$, Таким образом, в силу (2.1), (2.2) и (2.4)

$$L_n(x) = \sum_{i=0}^{n} P_{ni}(x) f_i, \qquad (2.5)$$

$P_{ni}(x) = \prod_{j=0}^{n} \frac{(x - x_j)}{(x_i - x_j)}$, $j \neq i$. Многочлен (2.5) называют многочленом Лагранжа.

2. **Единственность**

Доказательство проведем по методу от противного. Для этого предположим, что существует еще один интерполяционный многочлен $\widetilde{L}_n(x)$. Построим многочлен $L_m(x) = \widetilde{L}_n(x) - L_n(x)$, где $m \le n$. По основной теореме алгебры уравнение $L_m(x) = 0$ имеет m корней. Но так как

$$\begin{cases} L_n(x_i) = f_i \\ \widetilde{L}_n(x_i) = f_i \end{cases},$$

то уравнение $L_m(x) = 0$ имеет $(n+1)$ корней $x_0, x_1 \dots . x_n$, что противоречит основной теореме алгебры. Тем самым теорема доказана полностью.

19. Постановка задачи численного интегрирования

Численное интегрирование (историческое название: (численная) квадратура) — вычисление значения определённого интеграла (как правило, приближённое). Под численным интегрированием понимают набор численных методов для нахождения значения определённого интеграла.

- Численное интегрирование применяется, когда:
- Сама подынтегральная функция не задана аналитически. Например, она представлена в виде таблицы (массива) значений в узлах некоторой расчётной сетки.

Аналитическое представление подынтегральной функции известно, но её первообразная не выражается через аналитические функции. Например, $f(x) = \exp(-x^2)$.

В этих двух случаях невозможно вычисление интеграла по формуле Ньютона — Лейбница. Также возможна ситуация, когда вид первообразной настолько сложен, что быстрее вычислить значение интеграла численным методом.

Основная идея большинства методов численного интегрирования состоит в замене подынтегральной функции на более простую, интеграл от которой легко вычисляется аналитически. При этом для оценки значения интеграла получаются формулы вида:

$$I \approx \sum_{i=1}^{n} w_i \, f(x_i),$$

где n — число точек, в которых вычисляется значение подынтегральной функции. Точки Xi называются узлами метода, числа Wi — весами узлов. При замене подынтегральной функции на полином нулевой, первой и второй степени получаются соответственно методы прямоугольников, трапеций и парабол (Симпсона). Часто формулы для оценки значения интеграла называют квадратурными формулами.

Частным случаем является метод построения интегральных квадратурных формул для равномерных сеток, известный как **формулы Котеса**. Метод назван в честь Роджера Котса. Основной идеей метода является замена подынтегральной функции каким-либо интерполяционным многочленом. После взятия интеграла можно написать:

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n} H_i \, f(x_i) + r_n(f),$$

где числа H_i называются *коэффициентами Котеса* и вычисляются как интегралы от соответствующих многочленов, стоящих в исходном интерполяционном многочлене для подынтегральной функции при значении функции в узле $x_i = a + ih$ ($h = (b-a)/n$ — шаг сетки; n — число узлов сетки, а индекс узлов $i = 0 \ldots n$). Слагаемое $r_n(f)$ — погрешность метода, которая может быть найдена разными способами. Для нечетных $n \geqslant 1$ погрешность может быть найдена интегрированием погрешности интерполяционного полинома подынтегральной функции.

Частными случаями формул Котеса являются: формулы прямоугольников ($n = 0$), формулы трапеций ($n = 1$), формула Симпсона ($n = 2$), формула Ньютона ($n = 3$) и т. д.

20. Основные задачи линейной алгебры. Прямые методы решения систем линейных алгебраических уравнений: метод, использующий обратную матрицу, формулы Крамера, метод Гаусса и его устойчивость.

Выделяют четыре основные задачи линейной алгебры: решение СЛАУ, вычисление определителя матрицы, нахождение обратной матрицы, определение собственных значений и собственных векторов матрицы.

Решение систем линейных алгебраических уравнений с помощью обратной матрицы.

Решение систем линейных алгебраических уравнений (СЛАУ) с помощью обратной матрицы требует предварительного ознакомления с таким понятием как матричная форма записи СЛАУ. Метод обратной матрицы предназначен для решения тех систем линейных алгебраических уравнений, у которых определитель матрицы системы отличен от нуля. Естественно, при этом подразумевается, что матрица системы квадратна (понятие определителя существует только для квадратных матриц). Суть метода обратной матрицы можно выразить в трёх пунктах:

1. Записать три матрицы: матрицу системы A, матрицу неизвестных X, матрицу свободных членов B.
2. Найти обратную матрица A^{-1}.

$$A^{-1} = \frac{1}{\Delta A} \cdot A^{*T}$$

3. Используя равенство $X=A^{-1} \cdot B$ получить решение заданной СЛАУ.

Пример №1

Решить СЛАУ $\begin{cases} -5x_1 + 7x_2 = 29; \\ 9x_1 + 8x_2 = -11. \end{cases}$ с помощью обратной матрицы.

Решение

Запишем матрицу системы A, матрицу свободных членов B и матрицу неизвестных X.

$$A = \begin{pmatrix} -5 & 7 \\ 9 & 8 \end{pmatrix}; \; B = \begin{pmatrix} 29 \\ -11 \end{pmatrix}; \; X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

Найдём обратную матрицу к матрице системы, т.е. вычислим A^{-1}. В примере № 2 на странице, посвящённой нахождению обратных матриц, обратная матрица была уже найдена. Воспользуемся готовым результатом и запишем A^{-1}:

$$A^{-1} = -\frac{1}{103} \cdot \begin{pmatrix} 8 & -7 \\ -9 & -5 \end{pmatrix}.$$

Метод Крамера:

Алгоритм решения систем линейных алгебраических уравнений методом Крамера.

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

1. Вычисляем определитель основной матрицы системы убеждаемся, что он отличен от нуля.

2. Находим определители:

$$\Delta_{x_1} = \begin{vmatrix} b_1 & a_{12} & \cdots & a_{1n} \\ b_2 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ b_n & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

$$\Delta_{x_2} = \begin{vmatrix} a_{11} & b_1 & \cdots & a_{1n} \\ a_{21} & b_2 & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & b_n & \cdots & a_{nn} \end{vmatrix}$$

$\vdots$

$$\Delta_{x_n} = \begin{vmatrix} a_{11} & a_{12} & \cdots & b_1 \\ a_{21} & a_{22} & \cdots & b_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & b_n \end{vmatrix}$$

которые являются определителями матриц, полученных из матрицы A заменой k-ого столбца (k = 1, 2, …, n) на столбец свободных членов.

3. Вычисляем искомые неизвестные переменные x1, x2, …, xn по формулам:

$$x_1 = \frac{\Delta_{x_1}}{\Delta}, \quad x_2 = \frac{\Delta_{x_2}}{\Delta}, \quad \ldots, \quad x_n = \frac{\Delta_{x_n}}{\Delta}$$

4. Выполняем проверку результатов, подставляя x1, x2, …, xn в исходную СЛАУ. Все уравнения системы должны обратиться в тождества. Можно также вычислить произведение матриц A · X, если в результате получилась матрица, равная B, то решение системы найдено верно. В противном случае в ходе решения была допущена ошибка.

Метод Гауса:

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа.

1) На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. А именно, среди элементов первого столбца матрицы выбирают ненулевой, перемещают его на крайнее верхнее положение перестановкой строк и вычитают получившуюся после перестановки первую строку из остальных строк, домножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.
2) На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная,

поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

Пример:

Покажем, как методом Гаусса можно решить следующую систему:

$$\begin{cases} 2x+y-z &= 8 \\ -3x-y+2z &= -11 \\ -2x+y+2z &= -3 \end{cases}$$

Обнулим коэффициенты при x во второй и третьей строчках. Для этого прибавим к ним первую строчку, умноженную на $\frac{3}{2}$ и 1, соответственно:

$$\begin{cases} 2x+y-z &= 8 \\ \frac{1}{2}y+\frac{1}{2}z &= 1 \\ 2y+z &= 5 \end{cases}$$

Теперь обнулим коэффициент при y в третьей строке, вычтя из неё вторую строку, умноженную на 4:

$$\begin{cases} 2x+y-z &= 8 \\ \frac{1}{2}y+\frac{1}{2}z &= 1 \\ -z &= 1 \end{cases}$$

В результате мы привели исходную систему к треугольному виду, тем самым закончим первый этап алгоритма.

На втором этапе разрешим полученные уравнения в обратном порядке. Имеем:

$z=-1$ из третьего;

$y=3$ из второго, подставив полученное z

$x=2$ из первого, подставив полученные z и y.

Таким образом исходная система решена.

Устойчивость:

Если бы вычисления проводились без ошибок округления, то решение было бы точным. Но к сожалению, все расчеты с вещественными числами выполняются только приближенно. В зависимости от типа вещественного числа (Real,Single,Extendedи т.д.) точными являются 10-20 знаков.

21. Системы линейных алгебраических уравнений (СЛАУ). Решение СЛАУ численными методами.

Системой линейных алгебраических уравнений (СЛАУ) называется система вида:

$$\begin{cases} a_{11}\cdot x_1 + a_{12}\cdot x_2 + \ldots + a_{1n}\cdot x_n = b_1 \\ a_{21}\cdot x_1 + a_{22}\cdot x_2 + \ldots + a_{2n}\cdot x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}\cdot x_1 + a_{m2}\cdot x_2 + \ldots + a_{mn}\cdot x_n = b_m \end{cases}$$

СЛАУ называется совместной, если она имеет, хотя бы одно решение. В противном случае система называется несовместной.

Система называется определённой, если она совместна и имеет единственное решение. В противном случае (т.е. если система совместна и имеет более одного решения) система называется неопределённой.

Система называется однородной, если все правые части уравнений, входящих в нее, равны нулю одновременно.

Система называется квадратной, если количество уравнений равно количеству неизвестных.
Методы решения делятся на «Точные» и «Итерационные». Методы отличаются друг от друга эффективностью, требованиям к объемам машинной памяти (при реализации на ЭВМ), закономерностями накопления ошибок в ходе расчетов.
Точными являются: Метод Крамера, Матричный метод, Метод Гаусса, Метод прогонки.
Итерационными являются: Метод простых итераций, Метод Зейделя, Метод Ньютона.

22. Вычисление определенных интегралов методом Монте-Карло.

Метод Монте-Карло еще называют Методом статистических испытаний.

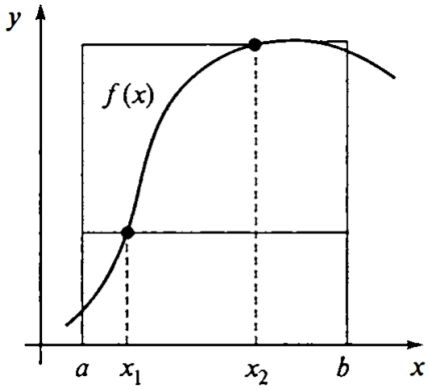


Рис. 2.36. Метод Монте-Карло

На отрезке интегрирования [a; b] выберем N случайных точек x_1, x_2, …, x_N, являющихся значениями случайной величины x с равномерным распределением на данном отрезке. Для каждой точки вычислим площадь прямоугольника, одна сторона которого равна b - a, а вторая – значению функции в данной точке $f(x_i)$: $S_i = (b - a)f(x_i)$.
Вследствие случайного выбора узла x_i значение площадей S_i также будет носить случайный характер. В качестве приближенного значения интеграла можно принять результат усреднения площадей S_i

$$\int_a^b f(x)dx \approx \frac{S_1 + S_2 + \ldots + S_N}{N} = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

Погрешность вычисления интеграла будет уменьшаться с ростом числа испытаний N по закону " "'N - '12 •
Полученная формула формально совпадает с формулой правых прямоугольников, но различие состоит в том, что в формуле узлы интегрирования расположены регулярно, а в данном методе расположение узлов носит случайный характер.

P.S Или можно тупо скринами (именно 22 вопрос), я не знаю в какой форме удобнее:

2.9.3. Метод Монте-Карло

Метод статистических испытаний, называемый также *методом Монте-Карло*, применяют для решения разнообразных задач вычислительной математики, в том числе для вычисления интегралов. Рассмотрим вначале один из простых вариантов метода Монте-Карло, который можно интерпретировать как статистический метод прямоугольников (рис. 2.36).

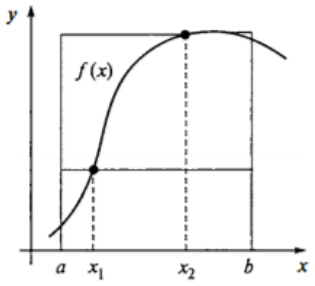


Рис. 2.36. Метод Монте-Карло

На отрезке интегрирования $[a; b]$ выберем N случайных точек $x_1, x_2, \ldots, x_N$, являющихся значениями случайной величины x с равномерным распределением на данном отрезке. Для каждой точки вычислим площадь прямоугольника, одна сторона которого равна $b - a$, а вторая — значению функции в данной точке $f(x_i)$: $S_i = (b - a) f(x_i)$.

Вследствие случайного выбора узла x_i значение площадей S_i также будет носить случайный характер. В качестве приближенного значения интеграла можно принять результат усреднения площадей S_i:

$$\int_a^b f(x)dx \approx \frac{S_1 + S_2 + \ldots + S_N}{N} = \frac{b - a}{N} \sum_{i=1}^{N} f(x_i). \qquad (2.18)$$

Погрешность вычисления интеграла будет уменьшаться с ростом числа испытаний N по закону $\varepsilon \approx N^{-1/2}$.

Полученная формула формально совпадает с формулой правых прямоугольников, но различие состоит в том, что в формуле (2.18) узлы интегрирования расположены регулярно, а в данном случае расположение узлов носит случайный характер.

Формула (2.18) непосредственно обобщается на кратные интегралы

$$\iint_G \ldots \int f(x, y, \ldots, z)dv = \frac{v_G}{N} \sum_{i=1}^{N} f(x_i, y_i, \ldots, z_i),$$

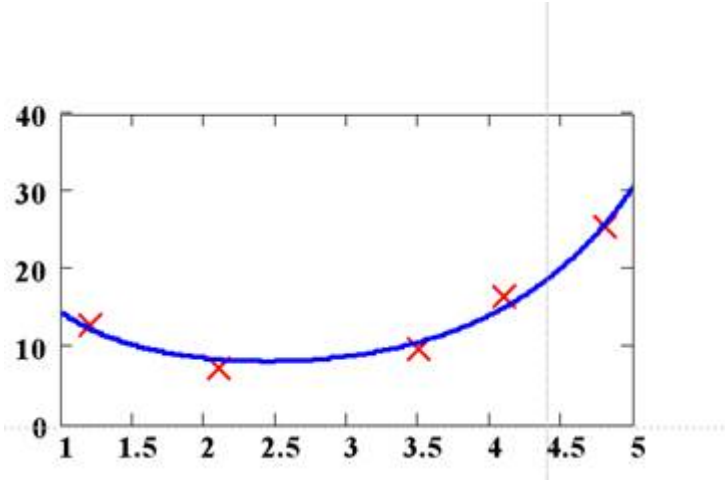
где v_G — объем области интегрирования. Например, для двукратного интеграла с прямоугольной областью интегрирования имеем

$$\int_a^b \int_c^d f(x, y)dxdy = \frac{(b - a)(d - c)}{N} \sum_{i=1}^{N} f(x_i, y_i).$$

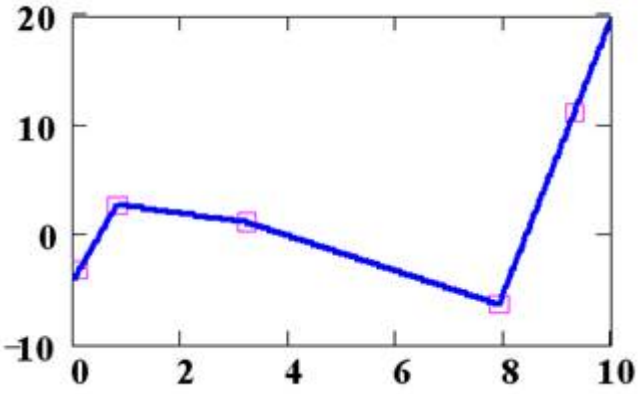
24. Математическая обработка экспериментальных данных: интерполирование и аппроксимация функций. Общая постановка задачи. Экстраполяция.

Аппроксимация – это замена исходной функции f(x) функцией φ(x) так, чтобы отклонение f(x) от φ(x) в заданной области было наименьшим. Функция φ(x) называется аппроксимирующей.

Если исходная функция f(x) задана таблично (дискретным набором точек), то аппроксимация называется дискретной. Если исходная функция f(x) задана аналитически (на отрезке), то аппроксимация называется непрерывной или интегральной.



Интерполяция – это замена исходной функции f(x) функцией φ(x) так, чтобы φ(x) точно проходила через точки исходной функции f(x). Интерполяция еще называется точечной аппроксимацией. Точки исходной функции f(x) называются узлами интерполяции.



Экстраполяцией называется аппроксимация вне заданной области определения исходной функции, т.е.

$x < x0$ и $x > xn$.

Найдя интерполяционную функцию, мы можем вычислить ее значения между узлами интерполяции, а также определить значение функции за пределами заданного интервала (провести экстраполяцию).

Основной мерой отклонения функции y(x) от функции f(x) при аппроксимации является величина, равная сумме квадратов разностей между значениями аппроксимирующей и исходной функций

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 \to \min$$

Основными задачами математической обработки экспериментальных данных являются: определение характеристик случайных величин и событий, сравнение между собой их вычисленных значений, построение законов распределения случайных величин, установление зависимости между полученными случайными величинами, анализ случайных процессов. Здесь же представляется целесообразным рассмотреть лишь особенности и возможности применения их при решении инженерно-психологических задач.

Основными характеристиками случайных величин являются их математическое ожидание и дисперсия, а случайных событий — вероятность их наступления. Математическое ожидание характеризует среднее значение наблюдаемой случайной величины (например, времени реакции, погрешности измерений, числа ошибок, допущенных человеком при выполнении работы и т. п.), а дисперсия является мерой рассеивания ее значений относительно среднего значения. Выборочные (опытные) значения математического ожидания и дисперсии вычисляются соответственно по формулам.

25. Классификация дифференциальных уравнений с частными производными: параболические, эллиптические и гиперболические уравнения. Численные методы решения задачи Коши.

Для решения многих практических задач необходимо рассматривать так называемые линейные или вполне линейные уравнения в частных производных:

$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$ (1)

Дифференциальное уравнение с частными производными II порядка с двумя независимыми переменными. A,B,C,D,E.F.G - функции от независимых переменных x и y, имеющие непрерывные частные производные.

Уравнение (1) всегда может быть приведено к одному из трёх стандартных канонических форм:

-- если $B^2 - 4AC < 0$, то уравнение (1) - **эллиптическое**;

-- если $B^2 - 4AC = 0$, то уравнение (1) - **параболическое**;

-- если $B^2 - 4AC > 0$, то уравнение (1) – **гиперболическое**.

Гиперболические уравнения

Это случай, когда $a_{12}^2 - a_{11}a_{22} > 0$. Общие интегралы характеристического уравнения $\varphi(x,y) = C_1;\ \psi(x,y) = C_2$. Выполняется замена $\xi = \varphi(x,y);\ \eta = \psi(x,y)$.

Параболические уравнения

Это случай, когда $a_{12}^2 - a_{11}a_{22} = 0$. Общий интеграл характеристического уравнения $\varphi(x,y) = C$. Выполняется замена $\xi = \varphi(x,y);\ \eta = \psi(x,y)$, где $\psi(x,y)$ - произвольная дважды дифференцируемая функция, для которой выполняется условие $\frac{D(\xi,\eta)}{D(x,y)} \neq 0$.

Эллиптические уравнения

Это случай, когда $a_{12}^2 - a_{11}a_{22} < 0$. Общий интеграл характеристического уравнения $\varphi(x,y) = C$. Выполняется замена $\xi = Re\varphi(x,y);\ \eta = Im\varphi(x,y)$.

Численные методы решения задачи Коши.

$y' = f(x,y),\ y|_{x=x_0} = y_0$

на равномерной сетке $(x_0 = a, x_1, x_2, \dots, x_m = b)$ отрезка $[a, b]$ с шагом $h = (b - a)/m$ являются методами Рунге-Кутта, если, начиная с данных (x_0, y_0), решение ведется по следующим рекуррентным формулам:

$$x_i = x_{i-1} + h,\ y_i = y_{i-1} + \Delta y_{i-1},\ i = 1,2,\dots,m$$

$$\Delta y_{i-1} = \sum_{j=1}^{p} d_j k_j^{[i-1]},\ k_j^{[i-1]} = hf\left(x_{i-1} + c_j h,\ y_{i-1} + c_j k_{j-1}^{[i-1]}\right). \quad (1)$$

Метод называют методом Рунге-Кутта порядка p, если он имеет p-й порядок точности по шагу h на сетке. Порядок точности p достигается с помощью формул (6) при определенных значениях коэффициентов c_j и d_j, $j = 1, 2, \dots, p$; коэффициент c_1 всегда полагают равным нулю. Эти коэффициенты вычисляются по следующей схеме:

1) точное решение $\varphi(x_0 + h)$ и его приближение $y_1 = y_0 + \Delta y_0(h)$ представляют в виде разложения по формуле Тейлора с центром x_0 вплоть до слагаемого порядка h^{p+1};

2) из равенств подобных членов при одинаковых степенях h в двух разложениях получают уравнения, решая которые находят коэффициенты c_j и d_j.

Метод Эйлера можно назвать методом Рунге-Кутта первого порядка. Действительно, для $p = 1,\ c_1 = 0,\ d_1 = 1$ формулы (1) преобразуются в соотношение:

$$x_i = x_{i-1} + h,\ y_i = y_{i-1} + \Delta y_{i-1},\ i = 1,\dots,m,$$

$$\Delta y_{i-1} = d_1 k_1^{[i-1]} = k_1^{[i-1]},\ k_1^{[i-1]} = hf(x_{i-1}, y_{i-1}) \text{ или}$$

$$x_i = x_{i-1} + h,\ y_i = y_{i-1} + hf(x_{i-1}, y_{i-1}).$$

Метод Рунге-Кутта второго порядка называют методом Эйлера-Коши, если $p = 1, c_1 = 0, c_2 = 1, d_1 = d_2 = 1/2$. Алгоритм метода Эйлера-Коши получается из формул (1):

$$x_i = x_{i-1} + h,\ y_i = y_{i-1} + \Delta y_{i-1},\ \Delta y_{i-1} = \frac{1}{2}\left(k_1^{[i-1]} + k_2^{[i-1]}\right),\ i = 1,\dots,m$$

$$k_1^{[i-1]} = hf(x_{i-1}, y_{i-1}),\ k_2^{[i-1]} = hf(x_{i-1} + h, y_{i-1} + k_1^{[i-1]}).$$

Для практической оценки погрешности решения можно применять правило Рунге, полагая в формуле (5) $p = 2$:

$$d = \left|\phi(x_i) - y_i(h/2)\right| \approx \frac{\left|y_i(h/2) - y_i(h)\right|}{3}.$$

Метод Рунге-Кутта четвертого порядка называют классическим методом Рунге-Кутта, если $p = 1, c_1 = 0, c_2 = c_3 = 1/2, c_4 = 1, d_1 = d_4 = 1/6, d_2 = d_3 = 1/3$. Из рекуррентных формул (6) получим алгоритм решения задачи Коши классическим методом Рунге-Кутта:

$$x_i = x_{i-1} + h,\ y_i = y_{i-1} + \Delta y_{i-1},\ i = 1,\dots,m,$$

$$\Delta y_{i-1} = \frac{1}{6}\left(k_1^{[i-1]} + 2k_2^{[i-1]} + 2k_3^{[i-1]} + k_4^{[i-1]}\right),$$

$$k_1^{[i-1]} = hf(x_{i-1}, y_{i-1}),\ k_2^{[i-1]} = hf\left(x_{i-1} + \frac{1}{2}h, y_{i-1} + \frac{1}{2}k_1^{[i-1]}\right),$$

$$k_3^{[i-1]} = hf\left(x_{i-1} + \frac{1}{2}h, y_{i-1} + \frac{1}{2}k_2^{[i-1]}\right),$$

$$k_4^{[i-1]} = hf\left(x_{i-1} + h, y_{i-1} + k_3^{[i-1]}\right).$$

Графиком приближенного решения является ломаная, последовательно соединяющая точки $P_i(x_i, y_i),\ i = 1, 2, \dots, m$. С увеличением порядка численного метода звенья ломанной приближаются к ломаной, образованной хордами интегральной кривой $y = \varphi(x)$, последовательно соединяющими точки $(x_i, \varphi(x_i))$ на интегральной кривой.

Правило Рунге для практической оценки погрешности приближенного решения для численного метода четвертого порядка имеет вид:

$$d = \left|\phi(x_i) - y_i(h/2)\right| \approx \frac{\left|y_i(h/2) - y_i(h)\right|}{15}.$$

26. Граничные условия 1-го, 2-го, 3-го рода решения краевой задачи.

Условия, относящиеся к фиксированным значениям координат, называются краевыми или граничными.

- **Граничные условия I рода** определяют значения функции на границе области её изменения;

$$T(t, x = 0) = \varphi_1(t), \quad T(t, x = l) = \varphi_2(t).$$

- **Граничные условия II рода** определяют значения градиента функции на границе области её изменения;

$$\frac{\partial T}{\partial x}(t, x = 0) = \varphi_1(t), \quad \frac{\partial T}{\partial x}(t, x = l) = \varphi_2(t).$$

- **Граничные условия III рода** определяют зависимость функции и её градиента на границе области изменения функции, т.е. в граничных точках пространства записывается дифференциальное уравнение.

$$\frac{\partial T}{\partial x}(t, x=0) = \alpha\left\{T(t, x=0) - T_{cp}\right\}, \qquad \frac{\partial T}{\partial x}(t, x=l) = \alpha\left\{T(t, x=l) - T_{cp}\right\}.$$