Записки за упражненията по Числени методи, СИ, ИС, II курс, зимен семестър, 2024/2025

10 декември 2024 г.

Съдържание

Глава 1

Въведение

1.1 Какво представляват числените методи?

Най-общо казано, числените методи са техники, чрез които математически задачи се представят във вид, в който могат да бъдат решени с помощта на аритметични операции. Въпреки че има много видове числени методи, те имат обща характеристика – изискват голям брой аритметични пресмятания. Ето защо тяхното прилагане става посредством имплементирането им в компютърни програми.

Обикновено числените методи включват **апроксимация** (т.е. приближение) на оригиналната математическа задача. Ето защо голяма част от тях можем да разглеждаме като техники за **приближеното решаване** на дадена математическа задача посредством аритметични операции.

В настоящия курс ще се занимаем с въпросите за приближаването на функции, приближеното пресмятане на производни и интеграли, приближеното намиране на корените на дадено уравнение.

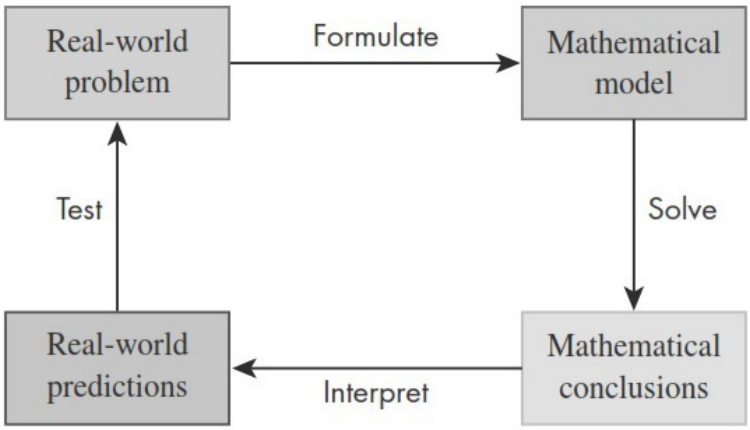
Преди да преминем към разглеждането на конкретни числени методи, нека разгледаме въпроса защо изобщо е необходимо тяхното използване. Математиката е езикът, на който се описват процесите от света около нас. За да изучим даден реален процес или да решим дадена практическа задача, ние трябва да определим кои са основните характеристики, които ги описват – това са някакви величини, които дават информация за съответния процес (например време, скорост, температура, сила, бързодействие на алгоритъм, компресия на данни и др.). Величините се измерват в дадени мерни единици, т.е. им се съпоставят някакви числени стойности. Изучавайки даден процес, ние искаме да изучим зависимостите между величините, които го описват, като за целта създаваме и изследваме математически модел на процеса.

Най-общо казано, **математически модел** е описание на някакъв реален процес или реална задача на езика на математиката. Често това става чрез функция или уравнение (или система от уравнения), много често – диференциално, свързващо величините, описващи процеса. Математическият модел обаче може да представлява и друг математически обект. Например, изследвайки една

компютърна мрежа, може да се наложи решаването на задача от теория на графите.

Целта на математическото моделиране е да се опише даденият процес и по-добре да се разберат механизмите, които го обуславят, както и, евентуално, да се направят компютърни симулации и/или предвиждания за бъдещото му поведение.

Често в литературата се дава следната схема, описваща методологията на математическото моделиране:



Да коментираме накратко етапите, описани в нея.

1. Имайки някаква реална задача, първото, което трябва да направим, е да **формулираме математически модел**, който да я описва. За целта трябва да определим основните величини, които характеризират процеса (от гледна точка на математиката – променливи и параметри), и да съставим математическата задача, която ги свързва (например диференциално уравнение, оптимизационна задача и др.). Важно е да се има предвид, че **всеки математически модел е една абстракция, идеализация на реалния процес**. В него трябва да има баланс – от една страна, моделът трябва достатъчно подробно да описва процеса, така че резултатите от него да бъдат полезни, но, от друга страна, трябва да е достатъчно прост, за да позволява математическо изследване. Всеки модел се базира на някакви допускания (абстракции), които позволяват опростяването на реалната ситуация. При създаването на математически модел използваме физически закони, обуславящи процеса, и математически техники, за да получим уравнения (или други обекти), свързващи променливите. В ситуации, когато не са известни физически закони, които да ни ръководят, може да е необходимо да се съберат данни от експерименти, на базата на които да се състави математическият модел.

2. Имайки предвид, че математическият модел на един процес представлява математическа задача, **вторият етап е да решим тази задача** и да получим математически заключения. **В настоящия курс ние ще разгледаме именно техники, които ще можем да използваме в този етап**. Важно е да се отбележи, че практическите задачи водят твърде

често до математически задачи, които не могат да бъдат решени със стандартните аналитични техники. Както знаем, дори просто изглеждащи алгебрични уравнения като полиномиалните уравнения от пета и по-висока степен в общия случай не могат да бъдат решени точно. Същото се отнася за повечето определени интеграли и др. Въпреки това обаче съществуват техники за тяхното **приближено решаване** и именно с такива ще се запознаем в курса по Числени методи.

3. След като сме решили (в някакъв смисъл) математическата задача, **следва да интерпретираме резултатите от гледна точка на реалния процес**.

4. Да обърнем внимание, че резултатите за реалния процес, които получихме, са следствие на математическия модел, а не на самия процес. От друга страна, казахме, че математическият модел е една абстракция на реалния процес, т.е. може и да не го описва достатъчно добре. Затова е необходимо да направим **проверка дали тези резултати съответстват на реалността**. Ако това е така, можем да считаме, че моделът ни е удачен. В противен случай се връщаме в началото и трябва да модифицираме модела така, че той да отразява действителността по-добре. С други думи, математическото моделиране е един **итеративен процес**.

Основни теми, които ще бъдат застъпени в курса, са:

1. Приближаване на функции – функциите са основен математически обект и затова ние ще посветим голяма част от курса именно на въпроса за тяхното приближаване.

2. Приближаване на производни – **производната на една функция описва скоростта на изменение на функцията в дадена точка**. Тя е основна характеристика на една функция. Затова ние ще се занимаем с въпроса за тяхното апроксимиране.

3. Приближаване на интеграли – интегрирането е основно действие в математиката. От друга страна, повечето интеграли не могат да бъдат решени точно. Ето защо методите за тяхното приближено пресмятане с достатъчно висока точност са от много голяма важност.

4. Приближено решаване на алгебрични уравнения.

В заключение да отбележим няколко причини за изучаването на числени методи:

- Числените методи са много мощни средства за решаването на реални задачи. С тяхна помощ е възможно решаването на големи системи уравнения, справянето с нелинейности и сложни геометрии, които са присъщи за задачите от практиката и към които често е невъзможно да се подходи аналитично.

- Често в практиката се налага използването на готови софтуерни продукти, чието действие се базира на дадени числени методи. Интелигентното използване на тези продукти изисква познаването на основната теория, обуславяща съответните числени методи.

- Невинаги готовите софтуерни продукти са достатъчни за решаването на дадена практическа задача. В тези случаи познаването на основната теория в областта на числените методи ни позволява проектирането и направата на собствени програми.

1.2 Грешка. Източници на грешка. Представяне на числата в компютъра.

Както отбелязахме, повечето числени методи включват някаква апроксимация. Ето защо разбирането на идеята за грешка е от много голяма важност за ефективното им използване. Нека първо дадем следните дефиниции:

Дефиниция 1. ***Абсолютна грешка*** *наричаме разликата между точната и приближената стойност при дадена апроксимация:*

$$\varepsilon_a := exact\ value - approximation.$$

Дефиниция 2. ***Относителна грешка*** *дефинираме по следния начин:*

$$\varepsilon_r := \frac{exact\ value - approximation}{exact\ value} = \frac{\varepsilon_a}{exact\ value}.$$

Основните източници на грешка при решаването на една практическа задача са следните:

- Математическият модел – както казахме, математическият модел сам по себе си е една апроксимация на реалността, с други думи самото му съставяне въвежда грешка по отношение на реалния процес.
- Грешка от числения метод – обикновено числените методи се базират на някаква апроксимация, т.е. въвеждат някаква грешка. Тъй като ние на практика не знаем точното решение на съответната математическа задача, обикновено е невъзможно да намерим каква е грешката при въпросната апроксимация. От друга страна, за да разберем дали даден числен метод е приложим, или не, ние трябва да знаем с каква точност той ще реши съответната задача. Затова се налага да се правят оценки на грешката, например да се намери някаква стойност, която тя със сигурност не надминава, или да се определи нейният порядък. Така, при изучаването на различните числени методи в настоящия курс, ние най-често ще се спираме на два основни момента:
  - описание на самия метод;
  - начини за оценка на грешката.
- Грешки от – те са свързани с начина, по който числата се представят в компютъра. Ще се спрем по-подробно на този вид грешка в настоящия параграф.
- Грешки от входните данни – математическите модели обикновено зависят от някакви параметри, стойностите на които се определят чрез провеждането на експерименти, правенето на измервания. Дори и най-съвършената

техника позволява измерване с определена точност, т.е. стойностите на измерените величини, с които работим, също носят определена грешка.

Задача 1. Постройте в една координатна система графиките на функциите

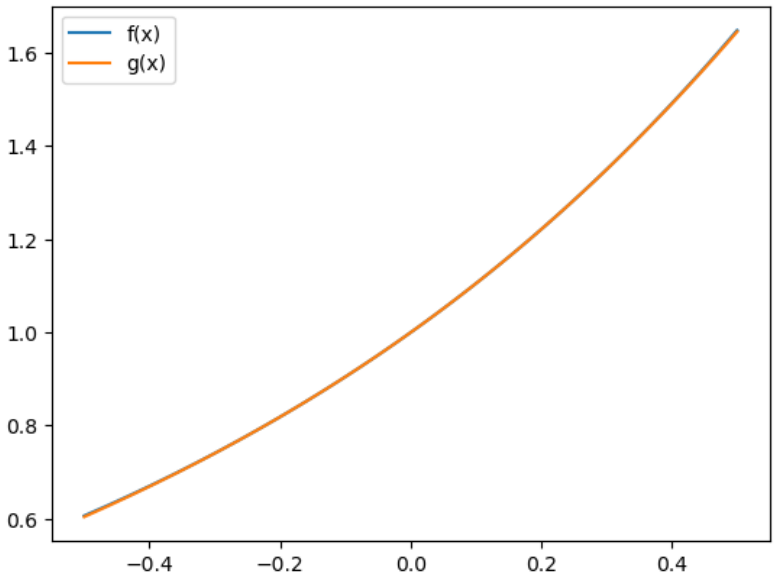
$$f(x) = e^x \text{ и } g(x) = 1 + x + 0.5x^2 + 0.1667x^3$$

в интервала [-0.5,0.5]. Постройте в същия интервал графиките на абсолютната и относителната грешка, които се получават при приближаването на $f(x)$ с $g(x)$, като функции на x.

Решение. Първо построяваме съответните графики, използвайки *Jupyter Notebook* и *Python*. За целта, най-напред ще вмъкнем няколко външни библиотеки, които ще ни дадат достъп до разнообразни математически операции и възможност за визуализиране на графики на функции:

[1]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
#np.linspace generates an array of 1000 equidistant  points in the␣
 ↪range -0.5 - 0.5
x_axis = np.linspace(-0.5, 0.5, 1000)

def f(x):
    return (math.e)**x
def g(x):
    return 1 + x + 0.5 * x**2 + 0.1667 * x**3

plt.plot(x_axis, f(x_axis), x_axis, g(x_axis))
plt.legend(['f(x)', 'g(x)'])
plt.show()
```



Визуално двете графики изглежда, че съвпадат. Това, което постигаме е, че експоненциалната функция, стойността на която не е лесно да се пресметне, сме приближили с алгебричен полином.

Абсолютната грешка, според Дефиниция 1, е

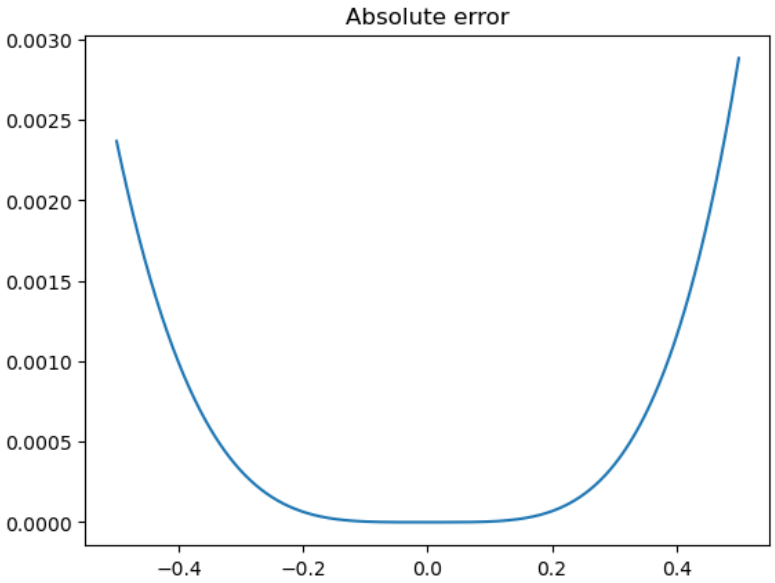
$$\varepsilon_a(x) = f(x) - g(x) = e^x - (1 + x + 0.5x^2 + 0.1667x^3),$$

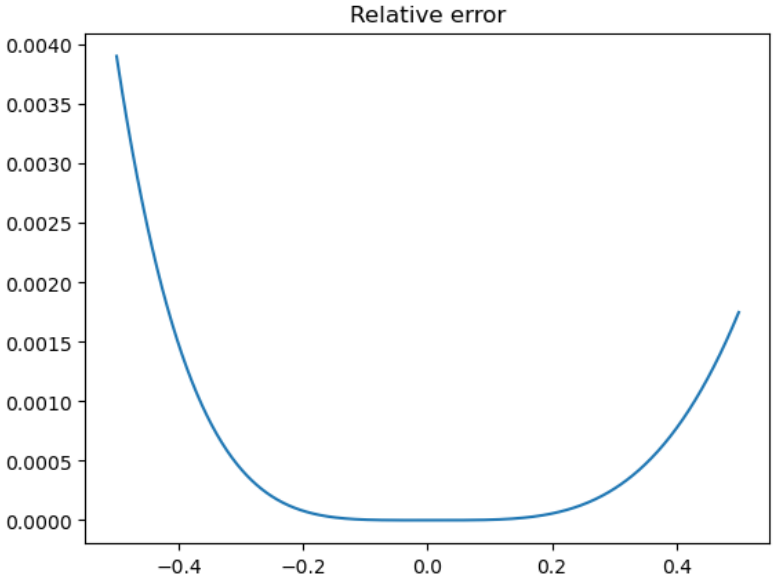
а за относителната грешка, според Дефиниция 2, имаме

$$\varepsilon_r(x) = \frac{f(x) - g(x)}{f(x)} = \frac{e^x - (1 + x + 0.5x^2 + 0.1667x^3)}{e^x}.$$

Да начертаем и техните графики:

[3]:
```
def abs_error(x):
    return abs(f(x) - g(x))

plt.plot(x_axis, abs_error(x_axis))
plt.title("Absolute error")
plt.show()

def relative_error(x):
    return abs(f(x) - g(x)) / f(x)

plt.plot(x_axis, relative_error(x_axis))
plt.title("Relative error")
plt.show()
```





От двете графики за грешката виждаме, че функцията $g(x)$ приближава сравнително добре функцията $f(x)$ в дадения интервал. Разбира се, дали точността на приближението е достатъчно добра, зависи от конкретния контекст, в който се разглежда задачата. Очевидно относителната грешка в разглеждания интервал не надминава 0.4%. □

Сега ще се спрем на грешката от закръгляване. За да илюстрираме какво представлява тя, да опитаме да съберем 2 десетични дроби с *Python*:

[4]:
```
0.1 + 0.2
```

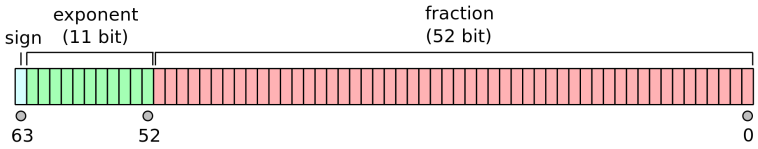
[4]:
```
0.30000000000000004
```

От примера забелязваме, че резултата от събирането не е коректен. Причината за това е начинът, по който числата се представят в компютъра. По-точно, ще се занимаем с т.нар. числа с плаваща точка (floating-point). В компютърната памет те се представят по следния начин:

$$m.b^e,$$

където m е дробна част, наречена **мантиса**, b е основата на бройната система, в която работим (в компютъра $b = 2$), а e е цяло число, наречено **експонента**. Например $156.78 = 0.15678 \times 10^3$ е представянето във вид на число с плаваща точка на числото 156.78 в десетична бройна система. Да обърнем внимание, че обикновено дробната част се нормализира, така че първият знак след десетичната точка да бъде различен от нула.

В компютърната памет, разбира се, числата се съхраняват в двоична бройна система. Да разгледаме, например, стандартния IEEE 754 тип double, който се използва за съхранение на числа с плаваща запетая с двойна точност в повечето съвсременни машини. При този формат едно число се съхранява в общо 64 бита, разпределени по следния начин:



- 1 бит за знак;
- 11 бита за експонента;
- 52 бита за мантиса (общо 53, един от които е запазен за цялата част).

Този формат позволява да записваме числа в огромен диапазон, използвайки сравнително малко памет – 11-битовата експонента дава възможност да съхранение на числа в интервала от 10^{-308} до 10^{308}. От друга страна обаче, мантисата може да съдържа краен брой цифри. Проблемът с това е, че съществуват безброй много десетични числа, които нямат крайно представяне в двоична бройна система. Такова е например числото 0.1, което използвахме в горния пример. Неговото двоично представяне е 0.00011001100110011..., т.е. в двоична бройна система то е безкрайна периодична дроб и е невъзможно тя да бъде записана в ограничената памет, с която мантисата разполага. Най-общо казано, в машина с t-битова дума могат да се представят най-много 2^t различни реални числа. Очевидно има безброй много числа, които не могат да бъдат представени точно. За тяхното представяне се използва най-близкото число, което може да се представи точно в 52-битовата мантиса. Тази операция се нарича „закръгляване“(roundoff) и е причината въпросните грешки от закръгляване.
Нещо повече, тъй като има максимално (по абсолютна стойност) число, което можем да запишем в паметта, то при опит да запишем число, което има по-голяма стойност, получаваме т.нар грешка “overflow”. Освен това, по аналогична причина, не можем да представяме много малки по абсолютна стойност числа (т.е. близки до нулата). Опитът за записването на такова число води до грешка “underflow”. Нека отбележим, че някои компютри заместват “underflow” с нула.

За да илюстрираме ефектите от грешките от закръгляване, нека разгледаме един хипотетичен компютър, който използва десетична бройна система и представя числата с плаваща точка чрез 1-цифрена експонента със знак и 3-цифрена мантиса.

Най-малкото положително число, което можем да представим в този компютър, е 0.100×10^{-9}, а следващото по големина число е 0.101×10^{-9}. Всяко друго число между тези две трябва да бъде апроксимирано. Това ни дава минимална грешка от закръгляване 0.5×10^{-12}.

Най-голямото число, което можем да представим, е 0.999×10^9, докато следващото по-малко число е 0.998×10^9, което дава максимална грешка 0.5×10^6.

Вижда се, че грешката съществено зависи от големината на числата, които апроксимираме. Затова е по-смислено да говорим за относителната вместо за абсолютната грешка. Може да се покаже, че тя е под 5×10^{-3}, т.е. под 0.5%. Да разгледаме следния пример, който ще ни покаже защо относителната грешка е по-добрия показател за точността на приближението. Ясно е, че абсолютна грешка, равна на 1, при число от порядъка на 10^8 е, по принцип, много

по-пренебрежима, отколкото грешка от 0.001 при число от порядъка на 10^{-2}. Относителните грешки в този случай са съответно 10^{-8} и 0.1.

Може да се покаже, че за относителната грешка е в сила

$$|\varepsilon_r| < 0.5 \times 10^{-p},$$

където p е броят значещи цифри в мантисата. За числа с двойна точност (double) $p \approx 16$, а с единична (float) – $p \approx 7$. Отчитайки този извод, както и факта, че машините работят в двоична бройна система, то може да се покаже, че при 64-битовия тип double могат да се извършват пресмятания с приблизително 16 значещи цифри след десетичната запетая ($53 \log_{10}(2) \approx 15.955$), а относителната грешка от закръгляване е от порядъка на 10^{-16}.
Като резултат от грешките от закръгляване, дори фундаменталните асоциативни и дистрибутивни закони на алгебрата може и да не са в сила при числени пресмятания. Да разгледаме следните примери:

- Асоциативност на събирането

  $$a + (b + c) = (a + b) + c.$$

  Нека $a = 0.456 \times 10^{-2}$, $b = 0.123 \times 10^{0}$, $c = -0.128 \times 10^{0}$. Тогава

  $$(a + b) + c = 0.128 \times 10^{0} - 0.128 \times 10^{0} = 0,$$
  $$a + (b + c) = 0.456 \times 10^{-2} - 0.500 \times 10^{-2} = -0.440 \times 10^{-3}.$$

  Очевидно първият резултат не е верен и причината за това е **събирането на голямо с малко число**. Можем да разгледаме и още по-показателен пример за този проблем – ако съберем 0.100×10^{0} с 0.100×10^{-3}, резултатът е 0.100×10^{0}, т.е. все едно не сме извършили събирането!

- Асоциативност на умножението

  $$a \times (b \times c) = (a \times b) \times c.$$

  При стойности $a = 10^{-6}$, $b = 10^{-6}$, $c = 10^{8}$ лявата страна на асоциативния закон дава верен резултат. При използване на дясната страна обаче, при изчисленията ще се получи "underflow". Виждаме, че дори при работата с числа, които могат да бъдат представени точно, не сме застраховани от наличието на тази грешка. Следователно **действието на един алгоритъм може да зависи съществено от реда, в който се извършват операциите в него.**

Горните примери ни показват, че всяка аритметична операция, която извършваме, би могла да въведе грешка. Както казахме, числените методи се базират на голям брой аритметични операции, така че това е нещо, което не можем да пренебрегнем при тяхното използване.

За да илюстрираме ефекта на грешките от закръгляване, нека разгледаме следния пример.

Задача 2. Даден е алгебричният полином

$$p_1(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 = p_2(x)$$

Да се построи неговата графика, като за пресмятане на стойностите му в точката x се използва

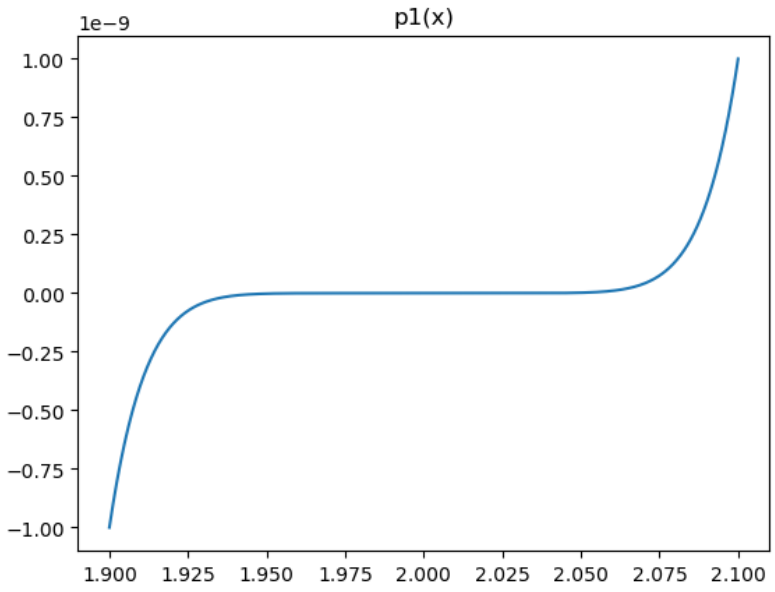
а)$p_1(x) = (x-2)^9$

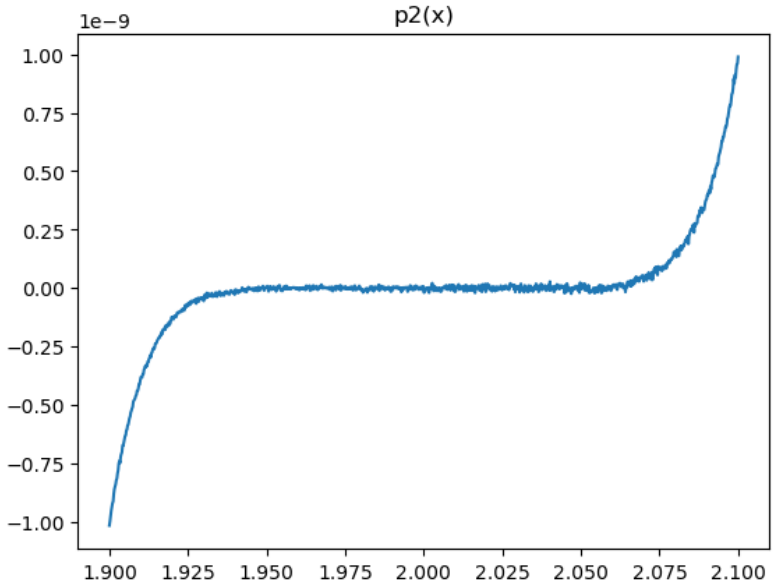
б)$p_2(x) = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512.$

Решение. Решението прилагаме по-долу.

[5]:
```
def p1(x):
    return (x - 2) ** 9
def p2(x):
    return x**9 - 18 * x**8 + 144 * x**7 - 672 * x**6 + 2016 *␣
 ↪x**5 - 4032 * x**4 + 5376 * x**3 - 4608 * x**2 + 2304 * x - 512
```

[6]:
```
x_axis = np.linspace(1.9,2.1,1000)
plt.plot(x_axis, p1(x_axis))
plt.title('p1(x)')
plt.show()

plt.plot(x_axis, p2(x_axis))
plt.title('p2(x)')
plt.show()
```





Във втория случай (когато използваме дефиницията $p_2(x)$), вместо графиката да изглежда също толкова „гладка“като графиката на $p_1(x)$, в резултатът се появява някакъв „шум“. Причината за разликата между двете графики се крие в това, че за пресмятане на стойностите на $p_2(x)$ е необходимо да се извършат много по-голям брой аритметични операции в сравнение с пресмятането на $p_1(x)$. От този експеримент можем да заключим, че **грешките от закръгляване се натрупват с увеличаване броя на аритметичните операции!** Това може да доведе до съществено отклонение от отчакваните резултати.

□

Вземайки предвид казаното дотук, **числените методи, които използваме трябва да са такива, че грешките от закръгляване да не водят до драстично изменение на резултата. Такива методи се наричат устойчиви.**

Глава 2

Интерполация

2.1 Идея на интерполацията

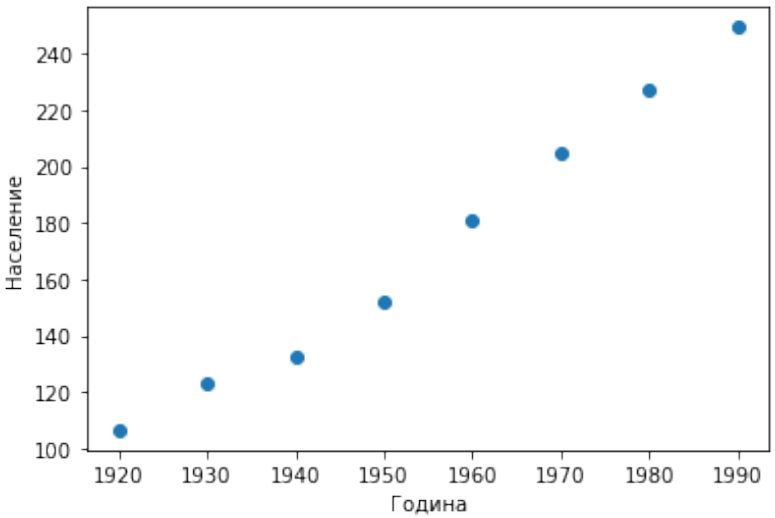
Изучавайки света около нас, ние искаме да намерим зависимости между различни величини. За да се изследва дадено явление или даден процес, е необходимо да се направят експерименти, които да дадат информация за него. След това данните от тези експерименти се използват, за да се създаде математически модел, който ги описва. Нека разгледаме следната задача.

Задача 3. В таблицата са дадени данни за населението на САЩ в млн. в периода 1920-1990.

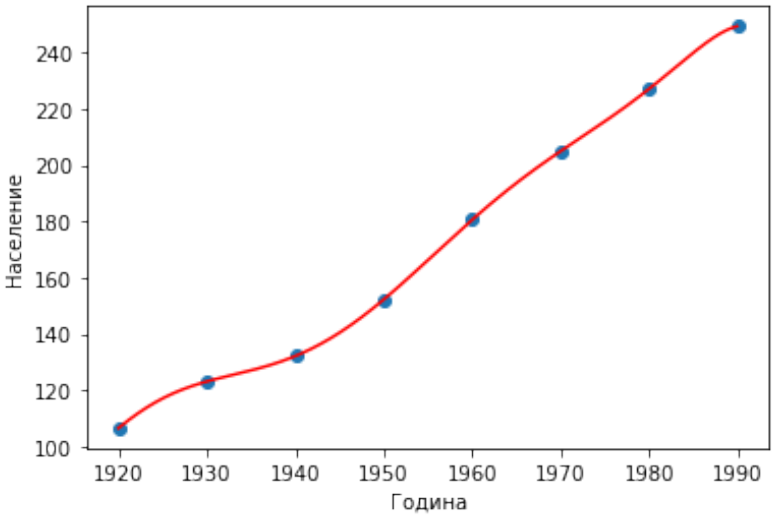
Година	1920	1930	1940	1950	1960	1970	1980	1990
Население	106.46	123.08	132.12	152.27	180.67	205.05	227.23	249.46

Да се намери функция, описваща изменението на населението през този период.

Коментар по задачата. В случая търсим зависимост между две величини, а резултатите от измерванията можем да интерпретираме геометрично като точки в равнината.



Тогава един възможен начин да опишем това явление е да намерим функция, чиято графика минава през дадените точки.



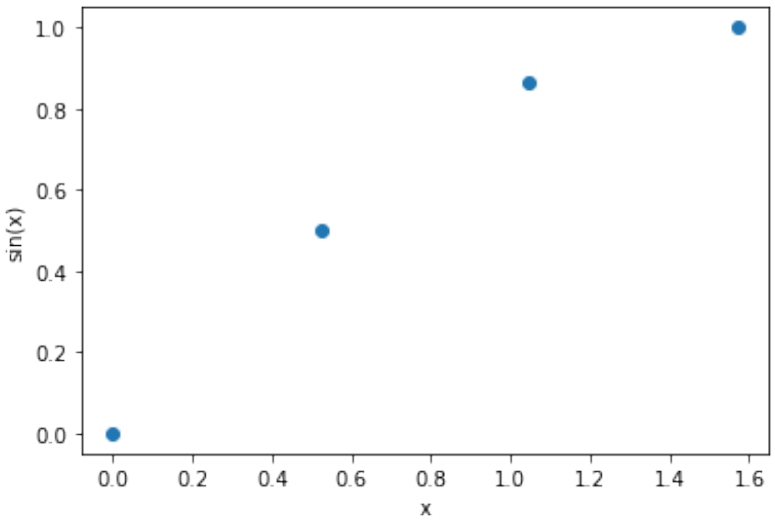
□

Намирането на функция, чиято графика минава през дадени точки, се нарича **интерполация**.

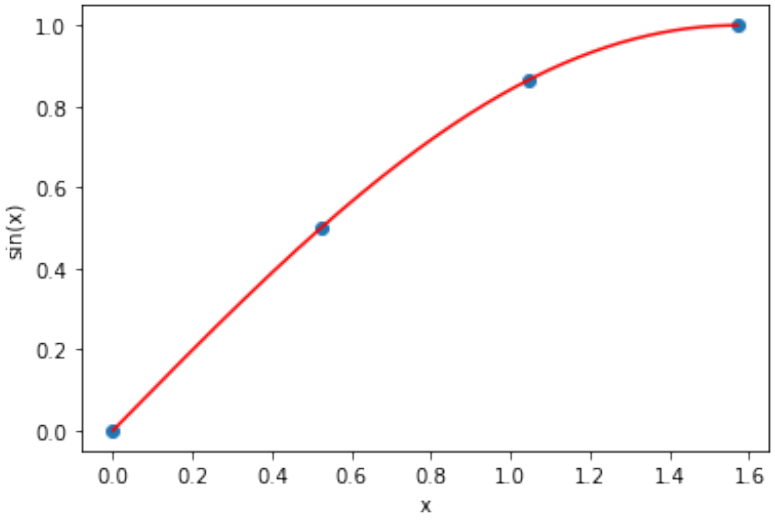
Нека сега разгледаме още една ситуация, в която ще използваме интерполация. На практика често се налага да се намират стойностите в дадена точка на функции като $\sin x$, $\cos x$, e^x, $\ln x$ и др. Както сами можем да се убедим, в общия случай това не изглежда тривиално. Един възможен подход ще илюстрираме със следващия пример.

Задача 4. Да се намери приближение на стойността на функцията $f(x) = \sin x$ за $x = \pi/5$.

Коментар по задачата. Оказва се, че можем да сведем тази задача до аналогична на предходната. Стойността на функцията $f(x) = \sin x$ ни е известна например за $x_0 = 0$, $x_1 = \frac{\pi}{6}$, $x_2 = \frac{\pi}{3}$, $x_3 = \frac{\pi}{2}$ (съответните стойности на $f(x)$ в тези точки са $0, 1/2, \sqrt{3}/2, 1$). Геометрично тази информация е представена на следващата фигура:



Тогава, ако намерим някаква функция $g(x)$, чиято графика да минава през тези точки (и чиято стойност в дадена точка може да бъде лесно пресметната!), ние ще имаме приближение на стойността на $f(x)$.



Така ние ще можем да намерим приблизително $\sin \pi/5$, като пресметнем $g(\pi/5)$. Тук възниква много важният въпрос колко точно ще бъде нашето приближение, т.е. колко ще се отличава неговата стойност от стойността на оригиналната функция. В настоящата глава ще коментираме и него. □

Изобщо казано, интерполацията ни позволява да намерим приближение на дадена функция, използвайки стойностите ѝ в дадени точки.[1] И така, в настоящата глава ние ще търсим отговора на следните въпроси:

- Как да намерим функция, чиято графика минава през дадени точки?
- Как да оценим точността на приближението, което сме получили?

[1] Както ще видим по-нататък, можем да наложим условия и върху стойностите на нейните производни, но засега ще разглеждаме ситуацията, когато сме наложили условия само върху стойностите на функцията.

2.2 Интерполационна задача на Лагранж

Алгебричните полиноми са функции, чиято стойност в дадена точка може да бъде пресметната лесно (един бърз алгоритъм за целта е например схемата на Хорнер). Ето защо те се явяват добър избор за решаване на задачата, която си поставихме. И така, ние ще търсим алгебричен полином, който минава през дадени точки. Нека сега формулираме точно поставената вече задача.

Постановка на интерполационната задача на Лагранж.

Нека x_0, x_1, ..., x_n са дадени различни точки от реалната права (възли) и $y_0, y_1, \ldots, y_n$ са дадени реални числа (стойности). Искаме да построим полином $P(x) \in \pi_n$ (π_n – класът от всички алгебрични полиноми от степен, ненадминаваща n) такъв, че

$$\begin{vmatrix} P(x_0) = y_0 \\ P(x_1) = y_1 \\ \quad \vdots \\ P(x_n) = y_n \end{vmatrix} . \tag{2.1}$$

Винаги, когато формулираме една математическа задача, много съществен е въпросът за съществуване и единственост на решението. От гледна точка на програмното реализиране на алгоритми за нейното решение например, е важно да знаем дали задачата винаги е решима и, ако не е, да можем да обработваме съответните изключения. В случая на интерполационната задача на Лагранж е в сила следното твърдение.

Твърдение 1. *Съществува, при това единствен полином $P(x) \in \pi_n$, удовлетворяващ интерполационната задача на Лагранж за произволни възли и стойности.*

Да отбележим още веднъж, че геометричната интерпретация на тази задача е следната – дадени са $n+1$ точки в равнината и търсим алгебричен полином от степен, ненадминаваща n, чиято графика минава през тези точки.

2.3 Интерполационна формула на Лагранж

Твърдение 2 (Интерполационна формула на Лагранж)**.** *Полиномът, удовлетворяващ условията (2.1), се представя по формулата*

$$P(x) = \sum_{k=0}^{n} l_k(x) y_k,$$

където $l_0(x), l_1(x), \ldots, l_n(x)$ са базисните полиноми на Лагранж. Те изпълняват условията

$$l_k(x_i) = \begin{cases} 0, & \text{ако } k \neq i \\ 1, & \text{ако } k = i \end{cases}$$

и се задават с формулата

$$l_k(x) = \prod_{i=0,\ i\neq k}^{n} \frac{x - x_i}{x_k - x_i}.$$

Ще обясним смисъла на формулата чрез пример.

Задача 5. Като се използва интерполационната формула на Лагранж, да се намери полином $P(x) \in \pi_3$, удовлетворяващ условията

$$P(1) = 2;\ P(2) = 9;\ P(4) = 41;\ P(6) = 97$$

Решение. Нека означим

$$x_0 = 1,\ x_1 = 2,\ x_2 = 4,\ x_3 = 6;$$

$$y_0 = 2,\ y_1 = 9,\ y_2 = 41,\ y_3 = 97.$$

Първо ще построим базисните полиноми на Лагранж. За полинома $l_0(x)$ искаме да се нулира във всички възли освен в x_0. В x_0 стойността му трябва да бъде 1. Тогава имаме

$$l_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}.$$

Действително, във възлите x_1, x_2, x_3 съответно първият, вторият и третият множител в числителя става 0 и цялата дроб е 0. Във възела x_0 получаваме

$$l_0(x_0) = \frac{(x_0-x_1)(x_0-x_2)(x_0-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = 1,$$

т.е. така дефинираният полином $l_0(x)$ изпълнява поставените му условия. Като заместим x_0, x_1, x_2, x_3 с техните равни, получаваме окончателно за $l_0(x)$

$$l_0(x) = \frac{(x-2)(x-4)(x-6)}{(1-2)(1-4)(1-6)} = -\frac{(x-2)(x-4)(x-6)}{15}.$$

Аналогично имаме

$$l_1(x) = \frac{(x-1)(x-4)(x-6)}{(2-1)(2-4)(2-6)} = \frac{(x-1)(x-4)(x-6)}{8};$$
$$l_2(x) = \frac{(x-1)(x-2)(x-6)}{(4-1)(4-2)(4-6)} = -\frac{(x-1)(x-2)(x-6)}{12};$$
$$l_3(x) = \frac{(x-1)(x-2)(x-4)}{(6-1)(6-2)(6-4)} = \frac{(x-1)(x-2)(x-4)}{40}.$$

Тогава интерполационният полином, удовлетворяващ задачата, може да бъде представен във вида

$$P(x) = l_0(x).y_0 + l_1(x).y_1 + l_2(x).y_2 + l_3(x).y_3.$$

За да се убедим в това, нека проверим какво се случва например в точката $x = x_1$. Имаме $l_0(x_1) = l_2(x_1) = l_3(x_1) = 0$ и $l_1(x_1) = 1$. Тогава

$$P(x_1) = 0.y_0 + 1.y_1 + 0.y_2 + 0.y_3 = y_1.$$

Аналогично се вижда, че този полином удовлетворява интерполационните условия и в другите възли.

И така, получихме

$$P(x) = l_0(x).2 + l_1(x).9 + l_2(x).41 + l_3(x).97.$$

След заместване и опростяване, получаваме окончателно

$$P(x) = 3x^2 - 2x + 1.$$

□

Задача 6. Да се построи интерполационният полином на Лагранж от първа степен за функцията $f(x) = 1/(1+x)$ с възли 0 и 1. Като се използва така намерения полином, да се пресметне приближено стойността $f(0.75)$. Да се пресметне абсолютната грешка от това приближение.

Решение. **Важно означение**: Когато построяваме алгебричен полином от степен n, който итнерполира стойностите на някоя функция $f(x)$, чийто явен вид ни е известен, обикновено го означаваме чрез $L_n(f;x)$ и четем „интерполационния полином от ред n за функцията f с възли $x_0, \ldots, x_n$“. Съгласно формулата на Лагранж, той ще има вида

$$L_n(f;x) = f(x_0)l_0(x) + f(x_1)l_1(x) + \cdots + f(x_n)l_n(x).$$

Като използваме така въведеното означение, търсим $L_1(f;x)$, където $f(x) = 1/(1+x)$ с възли $x_0 = 0$ и $x_1 = 1$. И така, по формулата на Лагранж този полином ще има вида

$$L_1(f;x) = f(x_0)l_0(x) + f(x_1)l_1(x) = f(0)l_0(x) + f(1)l_1(x) = l_0(x) + \frac{1}{2}l_1(x).$$

Да намерим базисните полиноми на Лагранж:

$$l_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} = 1 - x;$$
$$l_1(x) = \frac{(x - x_0)}{(x_1 - x_0)} = x.$$

Заместваме базисните полиноми в $L_1(f;x)$ и получаваме

$$L_1(f;x) = 1 - x + \frac{1}{2}x = 1 - \frac{1}{2}x.$$

Тогава за абсолютната грешка от приближението имаме

$$|f(0.75) - L(f;0.75)| = |4/7 - 5/8| = 3/56.$$

□

Да обърнем внимание, че когато пресмятаме абсолютна грешка, най-често се интересуваме от нейната големина, а не от знака и. Затова и обикновено пресмятаме абсолютната грешка по асболютна стойност.

Да забележим също така, че за да можем да пресмятаме асболютна грешка, е необходимо да знаем точната стойност, която се опитваме да приближим. Обикновено причината, поради която правим приближения, е именно за да избегнем пресмятането на тази точна стойност. Следващото твърдение дава отговор на въпроса как да оценим каква е точността на приближението, което правим, заменяйки дадена функция с нейния интерполационен полином без да използваме точните и стойности.

Твърдение 3. *(**Теорема за оценка на грешката при интерполация.**) Нека $[a,b]$ е даден краен интервал и $x_0, \ldots, x_n$ са различни точки в него. Нека функцията $f(x)$ има непрекъсната $(n+1)$-ва производна в този интервал. Тогава за всяко $x \in [a,b]$ съществува точка $\xi = \xi(x) \in [a,b]$ такава, че*

$$f(x) - L_n(f;x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x),$$

където $\omega(x) = (x-x_0)(x-x_1)\ldots(x-x_n)$.

Преди да продължим, да обърнем внимание на няколко неща в горната теорема. Последната ни казва, че ако искаме да пресметнем грешката в някоя точка $x \in [a,b]$, то тя ще бъде точно равна на дясната страна в (3). Тъй като обаче за всяко x числото ξ е различно и освен това то на практика не може да се определи, последната теорема често се използва за намиране на някаква оценка на грешката **отгоре**. Това означава, че изолзвайки тази теорема, не можем да пресметнем точната стойност на грешката, но можем да намерим някаква стойност, която тя няма да надвишава. Както вече споменахме, когато говорим за грешка, ние често няма да се интересуваме от нейния знак, т.е. дали надценяваме или подценяваме истинския резултат, а ще се интересуваме от това колко далеч се намираме от реалната стойност, т.е. ще разглеждаме грешката, взета по модул:

$$|f(x) - L_n(f;x)| = \left|\frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\ldots(x-x_n)\right|.$$

От горното лесно се вижда, че ако можем да направим оценка отгоре на $(n+1)$-вата производна за $\xi \in [a,b]$, то можем да дадем оценка отгоре на грешката по модул за всяко $x \in [a,b]$. С други думи, ако $|f^{(n+1)}(\xi)| \leq M_{n+1}$, където $M_{n+1} \in \mathbb{R}$, то

$$|f(x) - L_n(f;x)| \leq \frac{M_{n+1}}{(n+1)!}|(x-x_0)(x-x_1)\ldots(x-x_n)|.$$

Нещо повече, ако поискаме да дадем оценка отгоре на грешката по модул за целия интервал $[a,b]$, е в сила

$$|f(x) - L_n(f;x)| \leq \frac{M_{n+1}}{(n+1)!}\max_{x\in[a,b]}|(x-x_0)(x-x_1)\ldots(x-x_n)|.$$

Да опитаме да приложим последното твърдение, за да намерим оценка на грешката, допусната при приближението, направено в Задача 6:

Задача 7. Като се използва теоремата за оценка на грешката при интерпоалция, да се направи оценка на грешката, която се допуска при приближение на $f(x) = 1/(1+x)$ с интерполационния полином на Лагранж от първа степен с възли 0 и 1

- при $x = 0.75$;
- в целия интервал $[0, 1]$.

Решение. От теоремата за оценка на грешката имаме

$$R(x) := f(x) - L_n(f; x) = \frac{f^{(2)}(\xi)}{2!}(x-0)(x-1),$$

за някое $\xi \in [0, 1]$. Да пресметнем втората производна

$$f''(x) = \frac{2}{(x+1)^3},$$

взета в точката ξ, и да я заместим по-горе. Имаме

$$R(x) = \frac{2x(x-1)}{2!(\xi+1)^3} = \frac{x(x-1)}{(\xi+1)^3}.$$

1. Нека сега видим как изглежда оценката на грешката по модул за произволно x в интервала. Имаме

$$|R(x)| = \left|\frac{x(x-1)}{(\xi+1)^3}\right|.$$

За $\xi \in [0, 1]$ е в сила неравенството

$$\frac{1}{(\xi+1)^3} \leq \frac{1}{(1+0)^3} = 1,$$

което влече

$$|R(x)| = \left|\frac{x(x-1)}{(\xi+1)^3}\right| \leq |x(x-1)|, \forall x \in [0, 1]. \tag{2.2}$$

Неравенство (2.2) следва от факта, че $\dfrac{1}{(\xi+1)^3}$ има максимална стойност за $\xi \in [0, 1]$ в точката $\xi = 0$. В частност, при $x = 0.75$, получаваме

$$|R(0.75)| \leq \left|\frac{3}{4}\left(\frac{3}{4} - 1\right)\right| = \frac{3}{16}.$$

Ако сравним получения резултат с абсолютната грешка, получена в Задача 6, ще забележим, че действително допуснатата грешка е **значително по-малка**.

2. Нека сега направим оценка отгоре за целия интервал $[0,1]$, т.е. да максимизираме по x оценката на грешката. С други думи, искаме да определим максималната стойност на $|x(x-1)|$ в интервала $[0,1]$. Последната функция очевидно има максимална стойност в точката $x = 1/2$, в която се намира върхът на параболата (максимумът може да се намери и като се вземат екстремалните точки на функцията в интервала). Следователно за всяко $x \in [0,1]$ имаме

$$|R(x)| \leq \left|\frac{1}{2}\left(\frac{1}{2}-1\right)\right| = \frac{1}{4}.$$

□

Нека сега разгледаме още един пример, показващ как се оценява грешката при приближаване на дадена функция с нейния интерполационен полином.

Задача 8. Стойността на $\ln 15.2$ е намерена приблизително по следния начин: взети са точните стойности на $\ln 15$ и $\ln 16$ и е използвана линейна интерполация (построен е интерполационният полином от първа степен за възлите $x_0 = 15$ и $x_1 = 16$). Нека с f и p са означени съответно точната и приближената стойност на $\ln 15.2$. Докажете, че

$$0 < f - p < 4.10^{-4}$$

Решение. Нека $f(t) = \ln t$. Тогава грешката е

$$f - p = \frac{f''(\xi)}{2}(15.2-15)(15.2-16), \quad \xi \in (15,16)$$

Тъй като $f''(t) = -\dfrac{1}{t^2}$, получаваме

$$f - p = -\frac{0.2.(-0.8)}{2\xi^2} = \frac{4}{50\xi^2}.$$

От една страна, $f-p$ очевидно е положително. От друга, най-голямата стойност на грешката се достига, когато знаменателят е най-малък, т.е. при $\xi = 15$. Тогава имаме

$$0 < f - p < \frac{4}{50.15^2} = \frac{4}{2.3^2.5^4} < \frac{4}{10^4}.$$

□

Задача 9. Да се намери приближено стойността на $\sin\dfrac{\pi}{5}$, като за целта се построи интерпоалционният полином на Лагранж за функцията $\sin(x)$ с възли $x_0 = 0$, $x_1 = \frac{\pi}{6}$, $x_2 = \frac{\pi}{3}$, $x_3 = \frac{\pi}{2}$. Да се начертаят графиките на двете функции, заедно с точките на интерполация в една координатна система. Да се направи оценка на грешката от така направеното приближение и да се сравни с абсолютната грешка.

„Измина още един ден, в който синус и косинус не ми послужиха за нищо...“ *За някои фундаментални операции в областта на компютърната графика често се налага пресмятането на стойности на тригонометричните функции* $\sin(x)$ *и* $cos(x)$*. Такива операции са например ротация на 3D обект (напр. в компютърните игри) и създаването на светлинни модели (т. нар shading) при генериране на фотореалистични изображения. При тези приложения често се налага пресмятане на огромен брой стойности на тези фунцкии за относително кратко време. Това налага употребата на различни техники за ефективно приближено пресмятане на стойностите им, които често включват интерполация.*

Решение. Очевидно оценяването на функцията $f(x) = \sin x$ в дадена точка не е никак проста работа. Ето защо, вместо да работим с тази функция, ние ще намерим нейно приближение и ще работим с него. Точки, в които стойността на $\sin x$ ни е известна, са например $x_0 = 0$, $x_1 = \frac{\pi}{6}$, $x_2 = \frac{\pi}{3}$, $x_3 = \frac{\pi}{2}$ (съответните стойности на $f(x)$ в тези точки са $0, 1/2, \sqrt{3}/2, 1$). Ще намерим полинома $L_3(f; x)$, който интерполира функцията $f(x)$ в тези точки. (За извършване на всички изчисления, дадени по-долу, можем да използваме *Python*.) Да намерим най-напред базисните полиноми на Лагранж:

$$l_0(x) = \frac{(x-\frac{\pi}{6})(x-\frac{\pi}{3})(x-\frac{\pi}{2})}{(0-\frac{\pi}{6})(0-\frac{\pi}{3})(0-\frac{\pi}{2})};$$
$$l_1(x) = \frac{(x-0)(x-\frac{\pi}{3})(x-\frac{\pi}{2})}{(\frac{\pi}{6}-0)(\frac{\pi}{6}-\frac{\pi}{3})(\frac{\pi}{6}-\frac{\pi}{2})};$$
$$l_2(x) = \frac{(x-0)(x-\frac{\pi}{6})(x-\frac{\pi}{2})}{(\frac{\pi}{3}-0)(\frac{\pi}{3}-\frac{\pi}{6})(\frac{\pi}{3}-\frac{\pi}{2})};$$
$$l_3(x) = \frac{(x-0)(x-\frac{\pi}{6})(x-\frac{\pi}{3})}{(\frac{\pi}{2}-0)(\frac{\pi}{2}-\frac{\pi}{6})(\frac{\pi}{2}-\frac{\pi}{3})}.$$

Тогава получаваме

$$L_3(f; x) = l_0(x).0 + l_1(x).\frac{1}{2} + l_2(x).\frac{\sqrt{3}}{2} + l_3(x).1.$$

След кратки преобразования получаваме

$$L_3(x) \approx 1.02043x - 0.0654708x^2 - 0.113872x^3.$$

Сега вече можем да намерим стойността на полинома $L_3(x)$ при $x = \frac{\pi}{5}$, тъй като тя ще бъде „близо” до истинската стойност на $\sin\frac{\pi}{5}$. Получаваме $L_3\left(\frac{\pi}{5}\right) \approx 0.587061$.

Остана да дадем оценка за това колко „близо” всъщност е стойността, която ние сме намерили, до точната стойност. С други думи, искаме да дадем оценка за грешката

$$R\left(\frac{\pi}{5}\right) = \left|f\left(\frac{\pi}{5}\right) - L_3\left(f; \frac{\pi}{5}\right)\right|.$$

От Твърдение 3 непосредствено следва, че

$$R\left(\frac{\pi}{5}\right) = \frac{\left|f^{(4)}(\xi)\right|}{4!}\left|\omega\left(\frac{\pi}{5}\right)\right|,$$

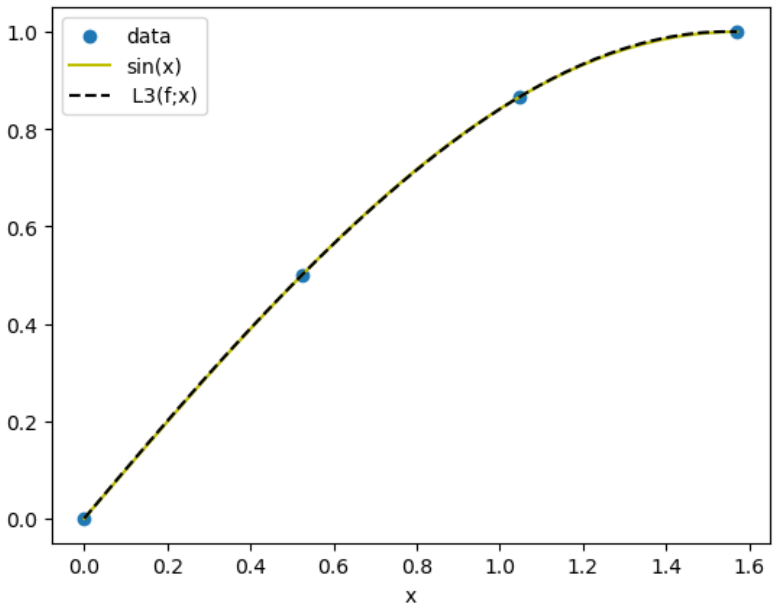
където ξ е число от интервала $\left[0, \frac{\pi}{2}\right]$. Имаме $f^{(4)}(\xi) = \sin\xi$.В интервала $\left[0, \frac{\pi}{2}\right]$ функцията $\sin x$ приема стойности между 0 и 1, т.е. $|f^{(4)}(\xi)| \leq 1$. Тогава

$$R\left(\frac{\pi}{5}\right) \leq \frac{1}{24}\left|\omega\left(\frac{\pi}{5}\right)\right| \approx 0.00108232.$$

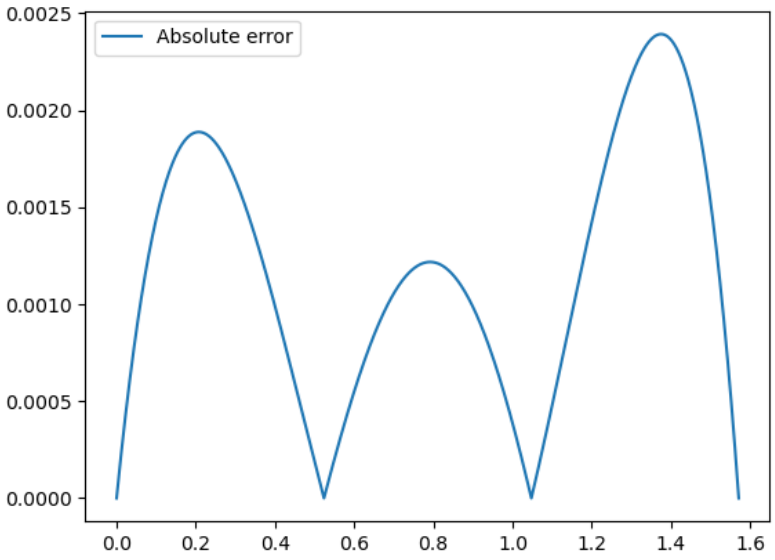
Окончателно получихме, че грешката по абсолютна стойност не надминава 0.0011. Ако сравним стойността, която ние получихме (0.587061) със стойността, която *Python NumPy* връща за $\sin\frac{\pi}{5}$, ще се убедим, че това действително е така. Обърнете внимание, че това е оценка отгоре за грешката, т.е. тя може и да е значително по-малка.

Да обърнем внимание и че можехме да намерим по-добро приближение на $\sin\frac{\pi}{5}$, ако бяхме подбрали интерполационните възли по по-подходящ начин или бяхме взели повече възли. □

Нека коментираме още няколко неща, свързани с предходната задача. Да илюстрираме първо нейното решение графично – заместваме функцията $\sin x$ (която на фигурата е с черната пунктирана линия) с интерполационния полином от степен 3, $L_3(f;x)$ (жълтата непрекъсната линия). Както виждаме, двете графики почти съвпадат в интервала на интерполация $[0, \pi/2]$, което обосновава приближаването на $\sin\frac{\pi}{5}$ с $L_3(f;\frac{\pi}{5})$.

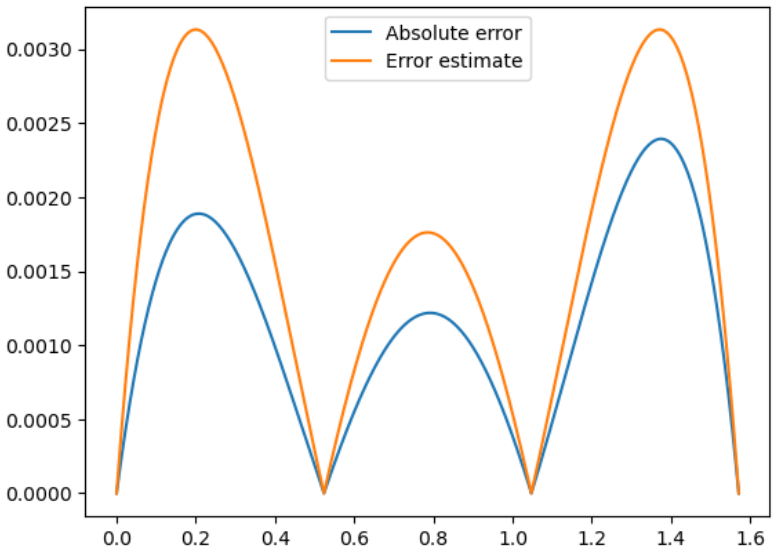


Привеждаме и графиката на абсолютната грешка (по модул) при приближаването на $\sin x$ с $L_3(f;x)$ в разглеждания интервал:

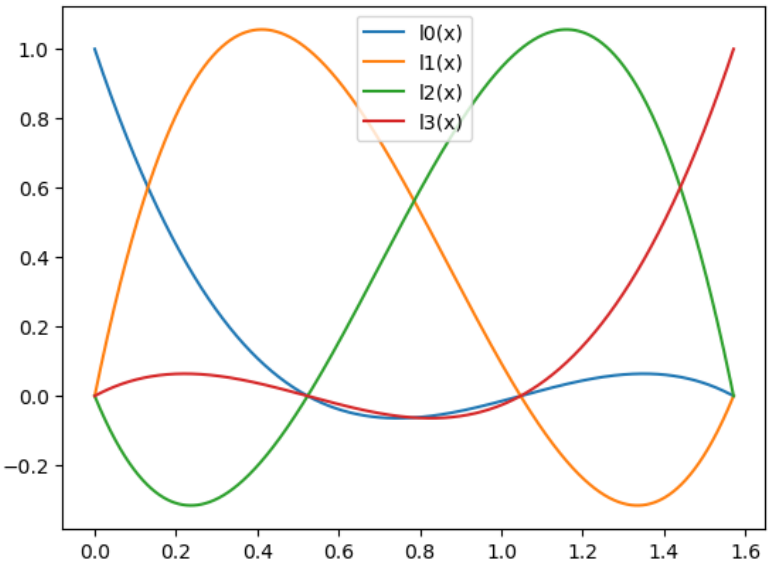


Допуснатата грешка, както можем да очакваме, във всички точки е не по-голяма от изведената оценка

$$R(x) \leq \frac{1}{24} |\omega(x)| :$$



Нека сега разгледаме графиките на базисните полиноми на Лагранж, които намерихме в предходната задача:



Графиката добре илюстрира условието, което наложихме на базисните полиноми на Лагранж при тяхното дефиниране – всеки от тях има стойност 1 във възела, за който "отговаря" и 0 във всички останали възли.

Дефиниция 3. *Нека са дадени точките* $x_0 < \cdots < x_n$. *Казваме, че функциите* $\varphi_0(x), \ldots, \varphi_n(x)$ *образуват интерполационен базис, ако*

$$\varphi_i(x_j) = \begin{cases} 0 & \text{ако } i \neq j, \\ 1 & \text{ако } i = j. \end{cases}$$

Ясно е, че ако $\varphi_0(x), \ldots, \varphi_n(x)$ образуват интерполационен базис и във формулата на Лагранж заместим базисните полиноми $l_i(x)$ с $\varphi_i(x)$, то получената функция $\overline{\varphi}(x)$ ще изпълнява условията $\overline{\varphi}(x_i) = y_i$.

2.3.1 Едно практическо приложение на интерполацията в библиотеката *matplotlib*

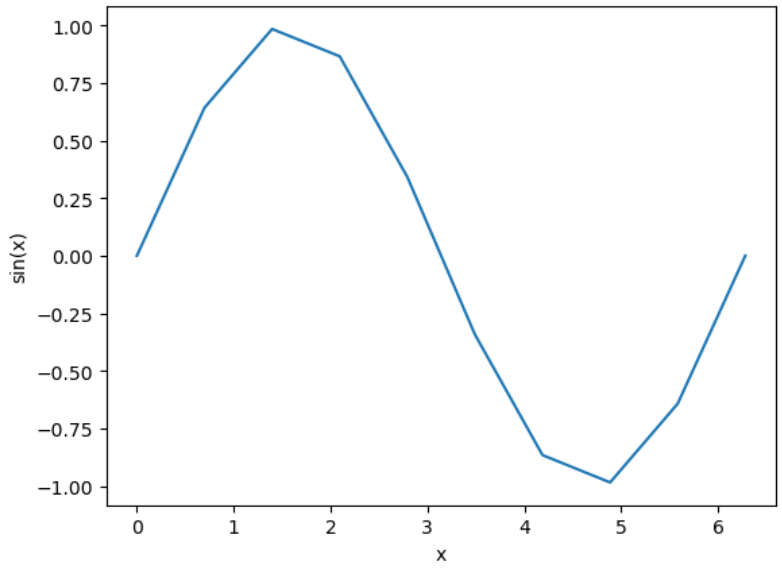
За изчертаване графика на непрекъсната функция с модула *matplotlib.pyplot*, е необходимо да укажем масив от точки от реалната права, в които да се пресметнат съответни стойности на функцията, за да може графиката и да се визуализира на екрана. Нека да се опитаме да генерираме масив от 10 числа в интервала $[0, 2\pi]$, с чиято помощ да начертаем графиката на функцията $\sin(x)$:

```
[*]: # generate 10 equidistant points from 0 to 2pi
     x_axis_10 = np.linspace(0, 2*math.pi, 10)
     print(x_axis)
```

```
[0.         0.6981317  1.3962634  2.0943951  2.7925268  3.4906585
4.1887902  4.88692191 5.58505361 6.28318531]
```

Графиката, която ще получим като резултат, няма да прилича съвсем на графиката на функцията $\sin(x)$, която познаваме. Вместо това, на екрана ще видим една начупена линия:

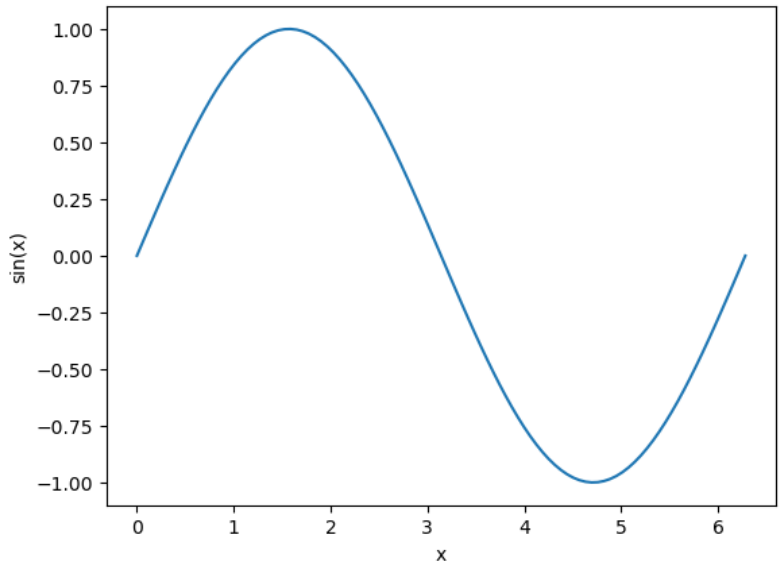
[*]:
```
# plot the graph of sin(x) using these 10 points
plt.plot(x_axis_10, np.sin(x_axis_10))
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```



Причината за това е, че за изчертаване на съответната графика *pyplot* използва **по части линейна интерполация**, т.е. построява множество интерполационни полиноми от първа степен за фунцкията $\sin(x)$, графиката на всеки от които е права линия, свързваща две съседни точки от списъка $\{x_i, \sin(x_i)\}, i = 0, \ldots, 9$.

Ако увеличим броя на точките в масива, то разтоянията между тях ще станат по-малки и правите, с чиято помощ се изчертава графиката на функцията, ще бъдат визуално неразличими за нас, затова и графиката изглежда "гладка".

[*]:
```
# use 100 equidistant points in 0-2pi to plot sin(x)
x_axis100 = np.linspace(0, 2*math.pi, 100)
plt.plot(x_axis100, np.sin(x_axis100))
plt.xlabel('x')
plt.ylabel('sin(x)')plt.show()
```



2.4 Разделени разлики. Интерполационна формула на Нютон.

В настоящия параграф ще коментираме още един начин за построяване на интерполационния полином на Лагранж – формулата на Нютон с разделени разлики.

Дефиниция 4. *Нека $x_0, x_1, \ldots, x_n$ са дадени различни точки.* ***Разделената разлика*** *на функцията $f(x)$ в точките $x_0, \ldots, x_n$ се бележи с $f[x_0, \ldots, x_n]$ и се дефинира индуктивно по следния начин:*

$$f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}, \quad n = 1, 2, \ldots,$$

като приемаме, че $f[x_i] := f(x_i)$ за всяка точка x_i.

Твърдение 4 (Интерполационна формула на Нютон)**.** *Интерполационният полином на Лагранж може да се представи чрез формулата*

$$L_n(f; x) = f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k](x - x_0) \ldots (x - x_{k-1}).$$

Идея на доказателството. Формулата на Нютон се основава на връзката между полинома $L_n(f, x)$, интерполиращ $f(x)$ в точките $x_0, \ldots, x_n$, и полинома $L_{n-1}(f, x)$, интерполиращ $f(x)$ в точките $x_0, \ldots, x_{n-1}$. Да разгледаме разликата $L_n(f, x) - L_{n-1}(f, x) \in \pi_n$. Ясно е, че стойността на тази разлика е 0 за $x = x_0, \ldots, x_{n-1}$, тъй като в тези точки и двата полинома съвпадат с $f(x)$. Тогава

$$L_n(f, x) - L_{n-1}(f, x) = C(x - x_0) \ldots (x - x_{n-1}),$$

т.е.

$$L_n(f, x) = C(x - x_0) \ldots (x - x_{n-1}) + L_{n-1}(f, x).$$

Може да се покаже, че $C = f[x_0, \ldots, x_n]$. Разсъждавайки индуктивно, можем да изразим L_{n-1} чрез L_{n-2} и т.н., с което получаваме формулата на Нютон. □

Задача 10. Като използвате интерполационната формула на Нютон, намерете полинома $P(x)$, който интерполира функцията $f(x) = \sqrt{x}+3x^2-x+2$ в точките 0,1,4.

Решение. Нека първо систематизираме интерполационните условия в таблица. За целта трябва да пресметнем стойността на $f(x)$ в точките 0, 1, 4. Получаваме

x	0	1	4
$f(x)$	2	5	48

Използвайки интерполационната формула на Нютон, получаваме представянето

$$L_2(f;x) = f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1).$$

Тогава трябва да пресметнем разделените разлики, участващи във формулата. Удобно е тези пресмятания да се систематизират в следната таблица:

Възли	Ред 0	Ред 1	Ред 2
$x_0 = 0$	$f[x_0] = 2$	$f[x_0,x_1] = 3$	$f[x_0,x_1,x_2] = \frac{34}{12} = \frac{17}{6}$
$x_1 = 1$	$f[x_1] = 5$	$f[x_1,x_2] = \frac{43}{3}$	
$x_2 = 4$	$f[x_2] = 48$		

Коефициентите, необходими ни за формулата на Нютон, се намират в първия ред на така получената таблица. Получаваме

$$L_2(f;x) = 2+3(x-0)+\frac{17}{6}(x-0)(x-1) = 2+3x+\frac{17}{6}x^2-\frac{17}{6}x = 2+\frac{1}{6}x+\frac{17}{6}x^2.$$

□

Задача 11. Точките

x	-2	1	4	-1	3	-4
y	-1	2	59	4	24	-53

лежат на полином. Определете степента на този полином.

Решение. От единствеността на интерполационния полином и прилагайки формулата на Нютон, получаваме $P(x) = f[x_0] + \sum_{k=1}^{5} f[x_0,\ldots,x_k](x-x_0)\ldots(x-x_{k-1})$ Както в предишната задача правим таблица с необходимите ни разделени разлики (обърнете внимание, че не е необходимо възлите да бъдат в нарастващ ред):

Възли	Ред 0	Ред 1	Ред 2	Ред 3	4	5
$x_0 = -2$	$f[x_0] = -1$	$f[x_0,x_1] = 1$	$f[x_0,x_1,x_2] = 3$	$f[x_0,x_1,x_2,x_3] = 1$	0	0
$x_1 = 1$	$f[x_1] = 2$	$f[x_1,x_2] = 19$	$f[x_1,x_2,x_3] = 4$	$f[x_1,x_2,x_3,x_4] = 1$	0	
$x_2 = 4$	$f[x_2] = 59$	$f[x_2,x_3] = 11$	$f[x_2,x_3,x_4] = 6$	$f[x_2,x_3,x_4,x_5] = 1$		
$x_3 = -1$	$f[x_3] = 4$	$f[x_3,x_4] = 5$	$f[x_3,x_4,x_5] = -2$			
$x_4 = 3$	$f[x_4] = 24$	$f[x_4,x_5] = 11$				
$x_5 = -4$	$f[x_5] = -53$					

Коефициентите във формулата на Нютон лежат на първия ред и тогава последният ненулев член е $f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$. Следователно полиномът е от трета степен. □

Нека сега коментираме накратко въпроса **защо са необходими различни формули за решаване на една и съща задача**.

Имплементирайки даден алгоритъм, основен е въпросът за неговото бързодействие. Именно тук се появява разликата между различните формули. В зависимост от това каква конкретна практическа задача искаме да решим, ще искаме различни неща от съответния алгоритъм.

- Често на практика, когато искаме да апроксимираме стойността на дадена функция в точката x, като са известни стойностите в точките $x_0, x_1, \ldots$, се построява редица от полиноми – първо, полином $p_1(x)$ от първа степен през двете най-близки точки, след това – полином $p_2(x)$ от втора степен през вече използваните две точки и следващата по отдалеченост и т.н. Това се прави, докато две съседни приближения дадат един и същ (с точност до някаква допустима грешка) резултат. След като сме получили един и същи резултат по два различни начина, можем да считаме, че този резултат е достоверен. За построяването на такава редица от полиноми е явно преимуществото на формулата на Нютон, тъй като, за да добавим нов възел, е достатъчно да добавим един ред в таблицата с разделени разлики и един член в сумата, описваща полинома. С други думи, можем да използваме вече направените изчисления. По формулата на Лагранж би ни се наложило да направим всички изчисления отначало, тъй като базисните полиноми зависят от възлите.

- Понякога искаме да построим полиноми с различни стойности, но еднакви възли. Например в биологията, когато се правят експерименти за развитието на една популация от микроорганизми, се следва експериментален протокол. Измервания се правят във фиксирани моменти от време. Тогава, ако искаме да сравним развитието на две популации, ще трябва да построим полиноми с едни и същи възли (фиксираните моменти от време), но различни стойности (числеността на различните популации в дадените моменти от време). Когато искаме да построим различни интерполационни полиноми, които имат обаче еднакви възли, по-удачно ще бъде да използваме формулата на Лагранж. След като възлите не се променят, базисните полиноми остават същите и трябва само да сметнем линейната им комбинация с новите стойности. Формулата на Нютон в този случай не е удобно да се използва, тъй като, сменяйки стойностите, трябва да пресметнем всички разделени разлики отначало (да отбележим още веднъж, че базата на рекурсията зависи от стойностите на приближаваната функция).

- В случаите, когато възлите са на равни разстояния един от друг, могат да се използват формулите на Нютон с крайни разлики. По формулата

на Лагранж полиномът се представя във вида

$$\sum_{i=0}^{n} y_i \prod_{k \neq i} \frac{x - x_k}{x_i - x_k}.$$

Имаме $n+1$ събираеми, като за всяко трябва да се извършат от порядъка на n операции, т.е. сложността е $O(n^2)$. Формулата на Нютон за интерполиране напред (вж. учебника) представя полинома във вида

$$\sum_{i=0}^{n} \binom{t}{i} \Delta^i f_0,$$

т.е. се извършват $O(n)$ операции. Тази формула обаче не е приложима в общия случай, а само при равноотдалечени възли

С други думи, различните формули ни предоставят различни инструменти, които ни позволяват да изберем подходящото средство за всеки конкретен случай.

2.5 Някои практически въпроси, свързани с интерполирането с алгебрични полиноми

Числените методи, както казахме, включват голям брой аритметични пресмятания. Затова тяхното прилагане става посредством имплементирането им в компютърни програми. Нека сега дефинираме функция в Mathematica, която намира автоматично полинома на Лагранж по формулата на Нютон.

Ясно е, че за да имплементираме формулата на Нютон, трябва да можем да пресмятаме разделени разлики. Вземайки предвид това, ще дефинираме следните функции:

- $divided_difference(nodes, values)$ – изчислява разделената разлика на функция със стойности $values$ в точките $nodes$ ($nodes$ и $values$ се задават като списъци или масиви от реални числа);
- $newton_poly(nodes, values, x)$ – пресмята стойността на интерполационния полином на Лагранж, построен за възли $nodes$ и стойности $values$ в точката x.

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[4]:
```
def divided_difference(nodes, values):
    if(nodes.size == 1):
        return values[0]
    else:
        return (divided_difference(nodes[1 : ], values[1 : ]) -␣
↪divided_difference(nodes[ : -1],values[ : -1])) / (nodes[-1] -␣
↪nodes[0])
```

[6]:
```
def newton_poly(nodes, values, x):
    poly = 0
    product = 1
    for i in range(nodes.size):
        poly += divided_difference(nodes[ : i + 1],values[ : i +␣
 ↪1]) * product
        product *= (x - nodes[i])
    return poly
```

Сега, използвайки дефинираните функции, ще разгледаме някои особености на интерполирането с алгебрични полиноми, които трябва да се имат предвид, когато приближаваме дадена функция или моделираме дадено явление. Ще започнем с една дефиниция.

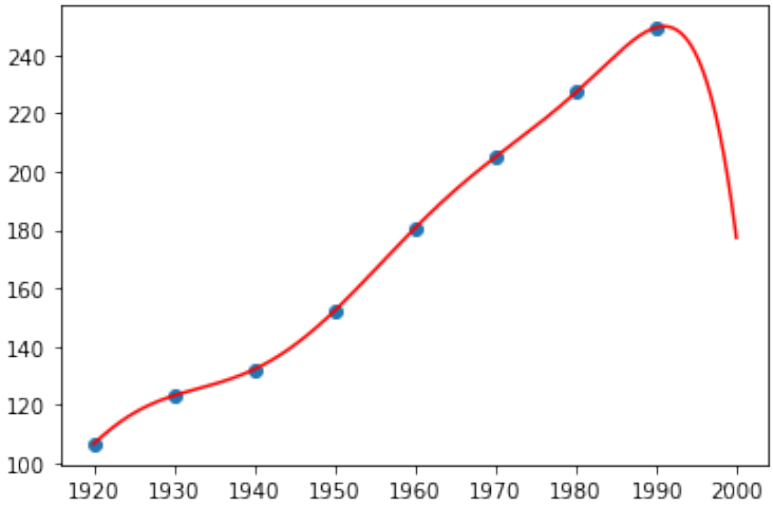
Дефиниция 5. *Когато използваме интерполационния полином за приближаване на стойност в интервала, определен от интерполационните възли, говорим за* ***интерполация****. В противен случай говорим за* ***екстраполация****.*

Задача 12. В таблицата са дадени данни за населението на САЩ в периода 1920-1990. Да се построи полином от седма степен, интерполиращ таблицата. Да се даде приближение на населението през 1952, 1974, 2000 година и да се сравни с действителните стойности – съответно 157 млн., 214 млн., 281.42 млн.

Година	1920	1930	1940	1950	1960	1970	1980	1990
Население	106.46	123.08	132.12	152.27	180.67	205.05	227.23	249.46

Решение. За решаване на задачата ще използваме двете функции, които предложихме в началото на настоящия параграф.

[8]:
```
years = np.arange(1920, 2000, 10)
population = np.array([106.46, 123.08, 132.12, 152.27, 180.67, 205.
 ↪05, 227.23, 249.46])
def poly1(x):
    return newton_poly(years, population, x)
x = np.arange(1920, 2000, 0.1)
plt.scatter(years, population)
plt.plot(x, poly1(x), color='r')
plt.show()
```



[9]:
```
print(poly1(1952))
print(poly1(1974))
print(poly1(2000))
```

```
157.72802626559996
213.51053127680007
175.08000000000226
```

Получихме следните приближения:

- През 1952 година – 157.728 млн. (в действителност 157 млн.)
- През 1974 година – 213.511 млн. (в действителност 214 млн.)
- През 2000 година – 175.08 млн. (в действителност 281,42 млн.)

В първите два случая приближението е добро и относителната грешка е малка. В третия обаче полученият с интерполационния полином резултат няма нищо общо с действителността. Отново виждаме, че при екстраполация не можем да разчитаме на добри резултати. Обърнете внимание как в границите на интерполация (между 1920 и 1990) поведението му добре моделира нарастването на населението, а извън тях това не е така.

□

Със следващата задача ще покажем, че, противно на интуицията, често при налагане на повече интерполационни условия (съответно използвайки полиноми от по-висока степен) получаваме по-лошо приближение. Това е така, защото полиномите от по-висока степен имат „по-лошо" поведение – появяват се осцилации.

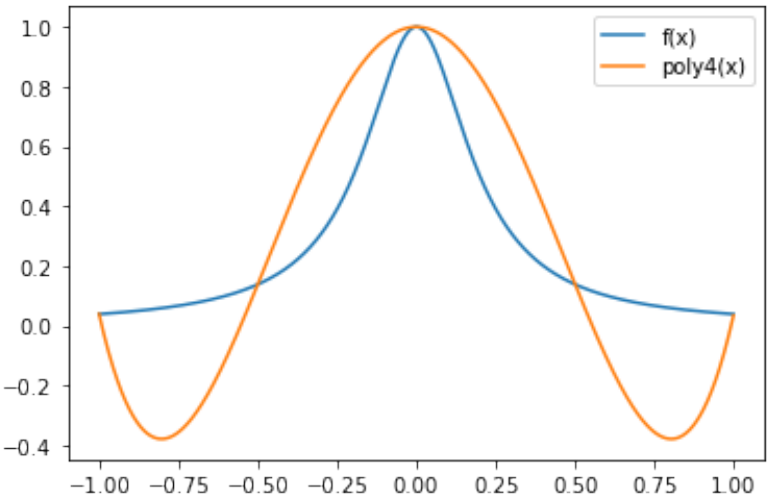
Задача 13. Да се приближи функцията[2] $f(x) = \dfrac{1}{1+25x^2}$ в интервала, като се намерят интерполационните полиноми от степени 10 и 4 при равноотдалечени възли в интервала.

[2]Това е т.нар. функция на Рунге, носеща името на Карл Рунге – немски математик, забелязал особеността при интерполиране с полиноми от висока степен, която коментираме. Тази особеност е известна в литературата като „феномен на Рунге".

Да се построят графиките на абсолютните грешки в двата случая.

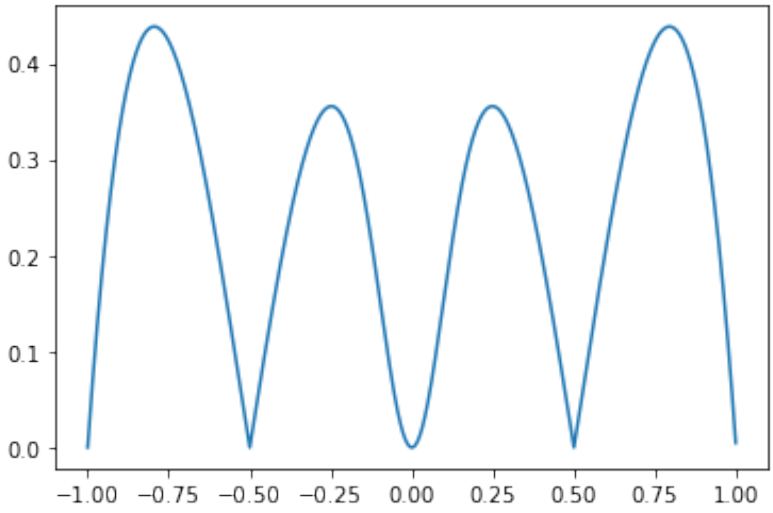
Решение. Първо да намерим полинома от четвърта степен.

[11]:
```
#linspace returns fixed number of enevly spaced points in an␣
 ↪interval
#arange works like range in Wolfram
nodes4 = np.linspace(-1, 1, 5)
def f(x):
    return 1 / (1 + 25 * x**2)
values4 = f(nodes4)
def poly4(x):
    return newton_poly(nodes4, values4, x)
x=np.arange(-1, 1 ,0.001)
plt.plot(x ,f(x))
plt.plot(x, poly4(x))
plt.legend(['f(x)','poly4(x)'])
plt.show()
```



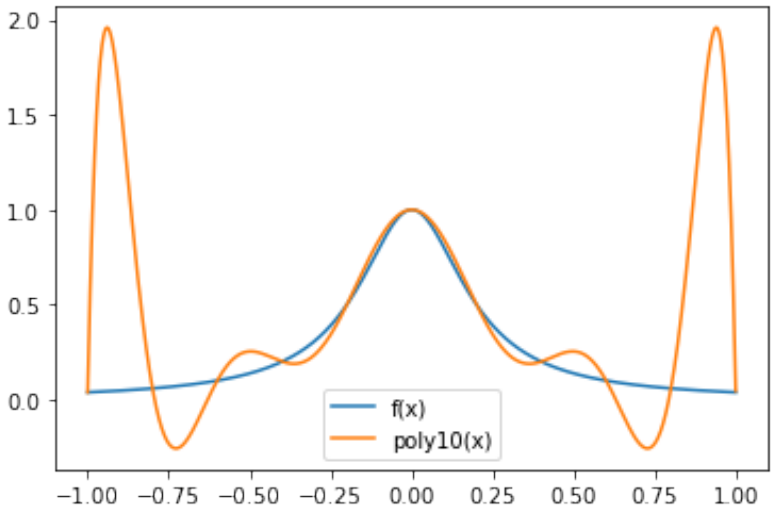
Да построим и графика на абсолютната грешка:

[12]:
```
plt.plot(x,abs(f(x) - poly4(x)))
plt.show()
```



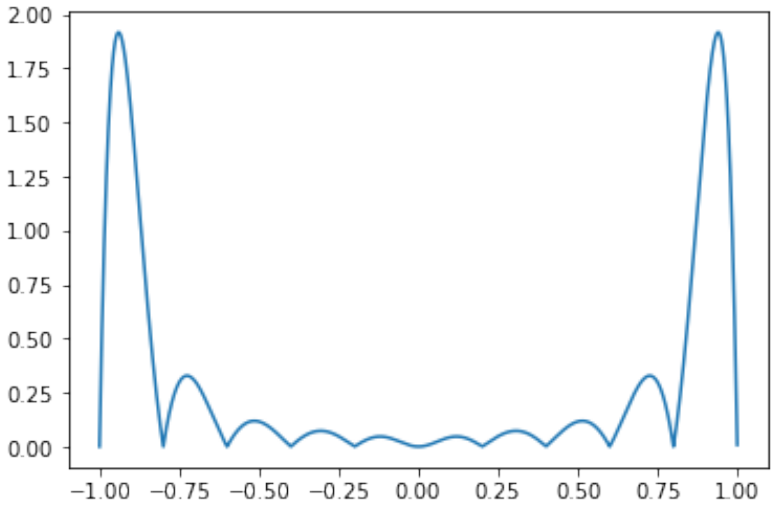
Поведението на интерполационния полином очевидно съществено се разминава с това на функцията на Рунге. Да опитаме сега да повишим степента на интерполационния полином с намерението да получим по-добро приближение:

[13]:
```
nodes10 = np.linspace(-1, 1, 11)
values10 = f(nodes10)
def poly10(x):
    return newton_poly(nodes10, values10, x)
x=np.arange(-1, 1, 0.0001)
plt.plot(x, f(x))
plt.plot(x, poly10(x))
plt.legend(['f(x)','poly10(x)'])
plt.show()
```



Прилагаме и графика на абсолютната грешка:

[14]:
```
plt.plot(x,abs(f(x) - poly10(x)))
plt.show()
```



Противно на очакванията ни, цялостното приближение с полинома от висока степен не е по-добро. В краищата на интервала наблюдаваме силни осцилации, които водят до огромни отклонения от стойностите на оригиналната функция. При интерполиране с полиноми от висока степен това е очаквано поведение и затова, когато е възможно, е добре да се избягва употребата им.

Вижда се, че и в двата случая грешката в средата на интервала е по-малка отколкото в краищата. Това особено добре си личи за полинома от десета степен. Затова е добре, когато интерполираме, възлите да са подбрани така, че точката, в която искаме да намерим приближената стойност, да е близо до средата на интервала. □

Задача 14. Известно е, че най-добрите възли за интерполация в интервала $[-1, 1]$ са т.нар. Чебишови възли, които се задават по формулата

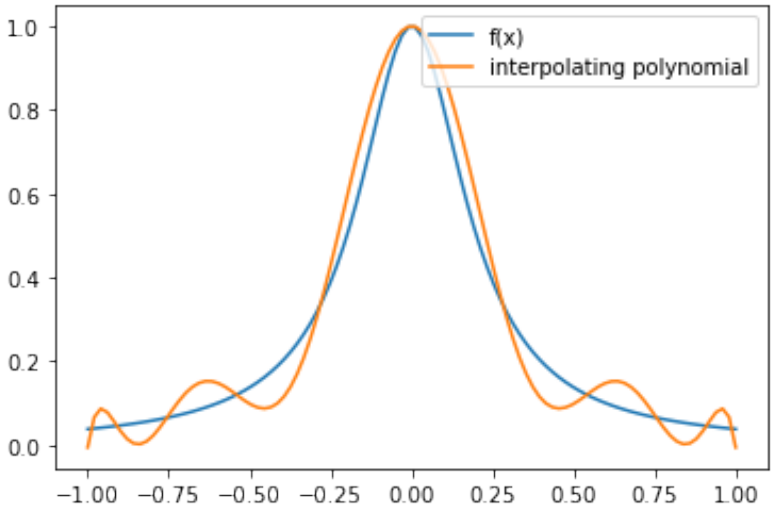
$$k = \cos(\frac{2k-1}{2n}\pi), k = 1, \ldots, n$$

.

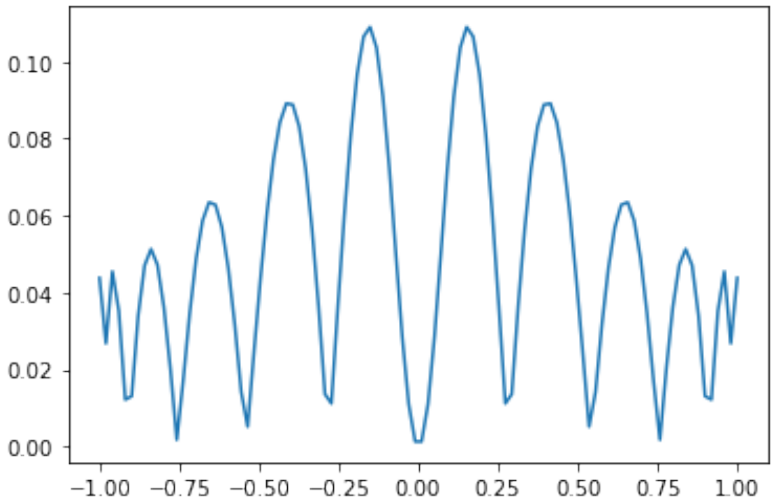
Да се построи интерполационен полином от степен 10 за функцията на Рунге в интервала $[-1, 1]$, като за целта се използват съответните Чебишови възли. Да се построи графика на абсолютната грешка от приближението.

[16]:
```
def f(x):
    return 1 / (1 + 25 * x**2)
n = 11
chebyshev_nodes = np.empty(n)
for k in range(n):
    chebyshev_nodes[k] = np.cos((2 * (k + 1) - 1) / (2 * n) * np.
 ↪pi)

chebyshev_values = f(chebyshev_nodes)

#def chebyshev_interp(x):
```

```
#    newton_poly(chebyshev_nodes,chebyshev_values,x)

x = np.linspace(-1, 1, 100)
plt.plot(x, f(x))
plt.plot(x, newton_poly(chebyshev_nodes, chebyshev_values, x))
plt.legend(['f(x)','interpolating polynomial'])
plt.show()
```



[17]:
```
plt.plot(x, abs(f(x) - newton_poly(chebyshev_nodes,␣
 ↪chebyshev_values, x)))
plt.show()
```



Задача 15. Проведени са експерименти, за да се определи бързодействието на един алгоритъм за сортиране в зависимост от броя елементи. Резултатите са представени в следната таблица:

бр.е/ти (х1000)	10	20	50	100	150	200	250
време, сек	0.1639275	0.53282	3.00007	11.20784	26.7486723	47.3297	76.80605

Да се определи колко най-много елемента могат да се сортират за не повече от

30 сек.

Решение. Нека с x означим броя елементи. Ще използваме **обратна интерполация**. Т.е. ще построим интерполационния полином $f(x)$, съответсващ на таблицата, и ще намерим стойността на x, за която $f(x) = 30$. За целта ще ни е необходимо да знаем интерполационния полином в явен вид. За това, а и за да можем да решим уравнението $f(x) = 30$ ще ни бъдат необходими инструменти за символни пресмятания в *Python*. По-конкретно, ще използваме библиотеката *SymPy*.

[20]:
```
# SymPy is a Python library for symbolic computations
from sympy import Symbol, expand, Eq, solve
x = Symbol('x')
left_part = expand(newton_poly(elements, time, x))
equation = Eq(left_part, 30)
sol = solve(equation, x)
print(sol)
```

```
[-59.2372258204670, 158.463886561511, 19.0353604249825 - 118.
 ↪902227037914*I,
19.0353604249825 + 118.902227037914*I, 258.811310015336 - 91.
 ↪415888526688*I,
258.811310015336 + 91.415888526688*I]
```

Само един от получените корени на уравнението е реално положително число – 158.46. От данните в таблицата е ясно, че търсената стойност за x ще е между 150 и 200 хил., така че можем да твърдим, че за под 30 сек. ще можем да сортираме приблизително 158 хил. елемента.

□

2.6 Интерполационна задача на Ермит. Разделени разлики с кратни възли.

Освен да налагаме условия върху стойността на функцията, можем да налагаме условия и върху стойностите на нейните производни в дадени точки. Това ни позволява по-добре да „контролираме" нейното поведение. Ние сега ще разгледаме интерполационната задача на Ермит.

Постановка на задачата. Нека

$$x_0, \ldots, x_n$$

са дадени различни точки от реалната права (възли). Нека

$$\nu_0, \ldots, \nu_n$$

са цели положителни числа (кратности) и

$$\{y_{k\lambda}, k = 0, \ldots, n, \lambda = 0, \ldots, \nu_k - 1\}$$

е таблица от произволни реални стойности. Нека

$$N := \nu_0 + \cdots + \nu_n - 1.$$

Интерполационната задача на Ермит е да се построи полином P от степен N, който удовлетворява условията

$$P^{(\lambda)}(x_k) = y_{k\lambda}, k = 0, \ldots, n, \lambda = 0, \ldots, \nu_k - 1.$$

С други думи, за всеки възел x_k налагаме условия за стойността на функцията и първите ѝ $\nu_k - 1$ производни (или общо толкова условия колкото е кратността на всеки възел) в съответния възел. По този начин имаме общо $\nu_0 + \cdots + \nu_n$ условия, които могат да определят еднозначно коефициентите на полином от степен $N := \nu_0 + \cdots + \nu_n - 1$.

Твърдение 5. *Задачата на Ермит има, при това единствено решение при всеки избор на интерполационни възли, кратности и стойности.*

Оказва се, че за да намерим интерполационния полином на Ермит, можем да използваме отново формулата на Нютон. Единствената разлика е, че трябва да обобщим понятието за разделени разлики така, че да можем да пресмятаме разделени разлики с кратни възли.

Твърдение 6. *Нека $f \in C^{(k)}[a, b]$. Тогава за произволни точки $x_0 \leq \ldots \leq x_n$ е в сила рекурентната връзка*

$$f[x_0, \ldots, x_n] = \begin{cases} \dfrac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}, & \text{ако } x_0 < x_n \\ \dfrac{f^{(n)}(x_0)}{n!}, & \text{ако } x_0 = x_n. \end{cases}$$

Задача 16. Да се интерполира функцията $f(x)$, като се използват следните интерполационни условия:

$$f(0) = -1, f'(0) = -2, f(1) = 0, f'(1) = 10, f''(1) = 40.$$

Решение. На базата на дадените 5 интерполационни условия можем да построим полином $P(x) \in \pi_4$, който интерполира функцията $f(x)$ във възлите 0 и 1 със съответни кратности 2 и 3. Ще го построим, използвайки формулата на Нютон. За целта трябва първо да пресметнем необходимите ни разделени разлики. Там, където пресмятаме разделена разлика, в която всички възли са равни, ще отбелязваме с * ($f[.,.,.]^*$), за да обръщаме внимание на този факт.

Възли	Ред 0	Ред 1	Ред 2	Ред 3	Ред 4
$x_0 = 0$	$f[x_0] = -1$	$f[x_0, x_1]^* = -2$	$f[x_0, x_1, x_2] = 3$	$f[x_0, x_1, x_2, x_3] = 6$	5
$x_1 = 0$	$f[x_1] = -1$	$f[x_1, x_2] = 1$	$f[x_1, x_2, x_3] = 9$	$f[x_1, x_2, x_3, x_4] = 11$	
$x_2 = 1$	$f[x_2] = 0$	$f[x_2, x_3]^* = 10$	$f[x_2, x_3, x_4]^* = 20$		
$x_3 = 1$	$f[x_3] = 0$	$f[x_3, x_4]^* = 10$			
$x_4 = 1$	$f[x_4] = 0$				

Обърнете внимание как е попълнена първата колона. $f[x_1]$ е равно на -1 а не на -2, тъй като $f[x_1] = f[0] = f(0)$. По същата причина $f[x_3] = f[x_4] = 0$.

Сега коефициентите лежат в първия ред. Получаваме

$$P(x) = -1 + (-2)(x-0) + 3(x-0)(x-0) + 6(x-0)(x-0)(x-1) \\ + 5(x-0)(x-0)(x-1)(x-1) = 5x^4 - 4x^3 + 2x^2 - 2x - 1.$$

□

Задача 17. Да се построи полиномът, интерполиращ таблицата

0	0	0	1
1	0	2	-1

Решение. Казано с други думи, трябва да построим полином $P(x) \in \pi_3$ такъв, че

$$P(0) = 1, P'(0) = 0, P''(0) = 2, P(1) = -1.$$

Имаме

Възли	Ред 0	Ред 1	Ред 2	Ред 3
$x_0 = 0$	$f[x_0] = 1$	$f[x_0, x_1]^* = \frac{0}{1!} = 0$	$f[x_0, x_1, x_2]^* = \frac{2}{2!} = 1$	$f[x_0, x_1, x_2, x_3] = -3$
$x_1 = 0$	$f[x_1] = 1$	$f[x_1, x_2]^* = \frac{0}{1!} = 0$	$f[x_1, x_2, x_3] = -2$	
$x_2 = 0$	$f[x_2] = 1$	$f[x_2, x_3] = -2$		
$x_3 = 1$	$f[x_3] = -1$			

Получаваме

$$P(x) = 1 + 0.(x-0) + 1(x-0)(x-0) + (-3)(x-0)(x-0)(x-0) = 1 + x^2 - 3x^3.$$

□

Да разгледаме още един възможен начин, по който може да бъде формулирана задачата на Ермит.

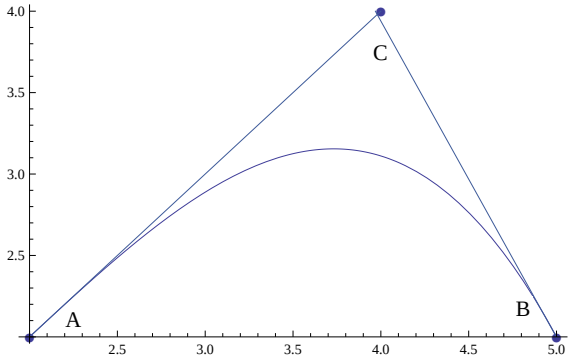
Задача 18. Да се построи интерполационният полином за функцията $f(x) = \cos x$ с възли 0 и $\pi/6$ със съответни кратности 2 и 3.

Упътване. В този случай най-напред трябва да построим интерполационните условия за задачата на Ермит, като пресметнем стойностите на функцията $f(x)$, която ни е дадена в явен вид и нейните производни в дадените интерполационни възли и след това да процедираме тъй както и в предходните две задачи. □

В софтуера за графичен дизайн една от базовите функционалности, които трябва да бъдат реализирани, е изобразяването на криви. Да разгледаме един простичък начин как може да стане това. За да илюстрираме това, за което става въпрос, нека използваме програмата Paint, която е достъпна за всеки. За да построим крива, първо трябва да провлачим мишката между две точки, нека ги наречем A и B (построявайки отсечка), и след това да кликнем с нея някъде (например в точка C), за да определим „кривината" ѝ. Както можете

да се уверите, оказва се, че през точка C минават допирателните към кривата в точките A и B. Да видим как може да бъде реализирано това.

От гледна точка на математиката, информацията, която имаме, са координатите на 3 точки в равнината. Нека ги означим с $A = (x_0, y_0)$, $B = (x_1, y_1)$ и $C = (x_2, y_2)$. Задачата ни е да построим крива, която минава през точките A и B, а допирателните към нея в точките A и B минават през точката C.



За да опишем кривата, нека я разгледаме като графика на някакъв кубичен полином

$$p(x) = ax^3 + bx^2 + cx + d.$$

Тогава можем да пресметнем производната на $p(x)$ в точките x_0 и x_1 (геометрично, това е тангенсът на ъгъла, който допирателната сключва с абсцисната ос). Съответните стойности са $p'(x_0) = \dfrac{y_2 - y_0}{x_2 - x_0}$ и $p'(x_1) = \dfrac{y_1 - y_2}{x_1 - x_2}$. Сега вече имаме стойностите на функцията и на първите производни в x_0 и x_1. Намирането на полинома при тези данни е интерполационната задача на Ермит. Решавайки я, имаме функция, която описва кривата и която можем да изобразим на екрана. Разбира се, на практика кривите се описват с много разнообразни функции и условията, които се налагат върху тях, могат да бъдат различни.

2.7 Интерполиране с обобщени полиноми. Интерполиране с тригонометрични полиноми.

Дотук разглеждахме задачата за намиране на алгебричен полином, чиято графика минава през дадени точки. По-общо въпросът за апроксимиране на дадена функция може да се разглежда така. Дадена е функцията f и искаме да намерим функция g от определен вид, която я приближава (например като я интерполира в дадени възли). Досега ние търсихме функцията g във вид на алгебричен полином, ненадминаващ дадена степен, т.е. я търсихме в множеството π_n.

Множеството от алгебрични полиноми π_n е линейно пространство, тъй като, ако $p, q \in \pi_n$, то

$$p + q \in \pi_n, \quad \lambda p \in \pi_n$$

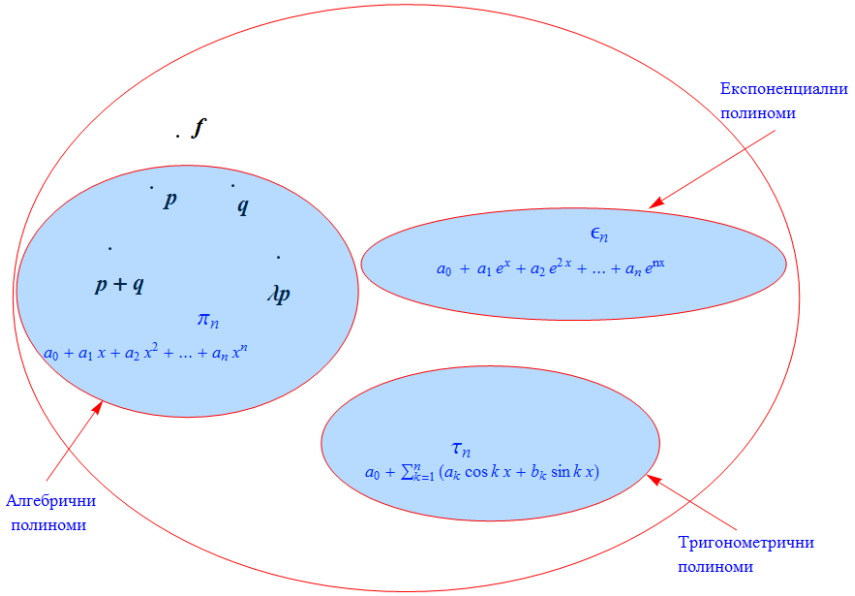
за кое да е число λ. Казано с други думи, можем да събираме два полинома или да умножаваме полином с число и това, което получаваме, е също полином от степен, ненадминаваща n. Най-простият базис в линейното пространство π_n

е $\{1, x, x^2, ..., x^n\}$ и следователно π_n **е крайномерно пространство**, тъй като се поражда от краен брой базисни функции.

Разбира се, вместо да търсим "най-доброто" (в някакъв смисъл) приближение на f във вид на алгебричен полином, т.е. в пространството π_n, можем да търсим друга по вид функция, т.е. от друго линейно пространство, която да е "близо" до f (вж. фигурата по-долу).

Да разгледаме, например, следните

Например, ако имаме функция f, която расте много бързо, бихме могли да я приближаваме с експоненциален полином, т.е. с функция от линейното пространство, породено от базиса $\{1, e^x, \ldots, e^{nx}\}$. Ако имаме периодична функция f или периодично явление, можем да използваме пространството от тригонометрични полиноми (определящо се от базиса $\{1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots, \sin nx, \cos nx\}$). Изобщо, имайки дадена функция, можем да търсим приближението ѝ във вид, който е най-удобен, т.е. отговаря на свойствата на оригиналната функция. Нека отбележим все пак, че **на практика най-често се използват алгебрични полиноми**, тъй като имат редица хубави свойства – лесно се намира стойността им в дадена точка, лесно се диференцират, интегрират и т.н.



Дефиниция 6. *Нека* $\{\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x)\}$ *образуват базис на дадено крайномерно линейно пространство. Тогава линейната комбинация*

$$\varphi(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + \ldots + a_n\varphi_n(x)$$

наричаме ***обобщен полином*** *по системата* $\{\varphi_k(x)\}_{k=0}^n$.

Забележка. Обобщените полиноми са обобщение на алгебричните полиноми. В случая, когато базисът ни е $\{1, x, x^2, \ldots, x^n\}$, обобщените полиноми са всъщност алгебрични полиноми.

Забележка. Всяко крайномерно линейно пространство има краен базис $\{\varphi_0(x), \ldots, \varphi_n(x)\}$. Следователно функциите в него представляват обобщени

полиноми по дадения базис. **Обобщените полиноми представляват най-общия вид на функции от кое да е крайномерно линейно пространство.**

Предимството на това да работим в крайномерни пространства е, че знаем вида на функциите в тях. Ако търсим обобщем полином от вида

$$\varphi(x) = a_0\varphi_0(x) + a_1\varphi_1(x) + \ldots + a_n\varphi_n(x),$$

единствените неизвестни са коефициентите в него. Нека са дадени точките $(x_0, y_0), \ldots, (x_n, y_n)$. За да намерим обобщения полином $\varphi(x)$, който минава през тях, трябва да решим линейната система $\varphi(x_0) = y_0, \ldots, \varphi(x_n) = y_n$, т.е.

$$\begin{aligned}
a_0\varphi_0(x_0) + \cdots + a_n\varphi_n(x_0) &= y_0,\\
a_0\varphi_0(x_1) + \cdots + a_n\varphi_n(x_1) &= y_1,\\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&\\
a_0\varphi_0(x_n) + \cdots + a_n\varphi_n(x_n) &= y_n.
\end{aligned}$$

Задача 19. В таблицата са дадени данни за развитието на бактериална популация.

t,h	1	2	3	4	5
клетки (x1000)	1	12	110	1037	12218

Да се намери подходяща функция, която интерполира дадените данни.

Решение. Дадените данни предполагат да търсим функцията в линейно пространство от бързо растящи функции, например експоненциални. Имаме 5 дадени точки, следователно ще изберем 5 базисни функции. Нека това са $\{1, e^x, e^{2x}, e^{3x}, e^{4x}\}$. Общият вид на функциите в линейното пространство, породено от този базис, е

$$\varphi(x) = a_0 + a_1e^x + a_2e^{2x} + a_3e^{3x} + a_4e^{4x}.$$

Ще определим коефициентите така, че

$$\varphi(1) = 1,\ \varphi(2) = 12,\ \varphi(3) = 110,\ \varphi(4) = 1037,\ \varphi(5) = 12218,$$

като решим съответната линейна система. Ще направим това, като използваме метода *linalg.solve* от библиотеката *NumPy*. Този метод приема като вход матрицата с коефициентите на линейната система и вектора на десните и страни. Да разгледаме най-напред общия случай на построяване на експоненциалне полином. Ако означим $\{x_i\}_{i=0}^n$ интерполационните възли, а с $\{y_i\}_{i=0}^n$ съответните им стойности, то системата от уравнения, която трябва да решим, записана във векторно-матрична форма, ще има следния вид:

$$A.\mathbf{a} = \mathbf{b},$$

където

$$A = \begin{pmatrix} 1 & e^{x_0} & e^{2x_0} & \cdots & e^{nx_0} \\ 1 & e^{x_1} & e^{2x_1} & \cdots & e^{nx_1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{x_4} & e^{2x_4} & \cdots & e^{nx_n} \end{pmatrix}, \tag{2.3}$$

и

$$\mathbf{b} = (y_0, y_1, \ldots, y_n)^T.$$

Тъй като в конкретната задача имаме 5 инрерполационни условия, то матрицата на системата ще бъде матрица от горния вид с размерност 5×5, а вектора на десните страни ще има вида

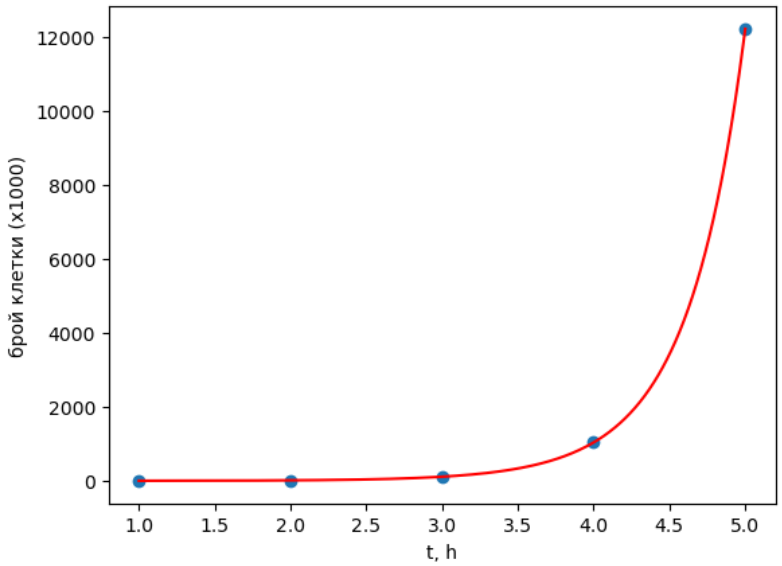
$$\mathbf{b} = \begin{pmatrix} 1 & 12 & 110 & 1037 & 12218 \end{pmatrix}^T. \tag{2.4}$$

Генерираме матрицата в *Python* и решаваме системата:

[5]:
```
t = np.array([1, 2, 3, 4, 5])
cells = np.array([1, 12, 110, 1037, 12218])
A = np.zeros([5,5])
for i in range (5):
    for j in range (5):
        A[i,j] = np.exp(j * t[i])
sol = np.linalg.solve(A, cells)
```

Сега, замествайки получените коефициенти на съответните места в общия вид на интерполационния полином, получаваме

[6]:
```
def exp_poly_1(x):
    poly = 0;
    for i in range(cells.size):
        poly += sol[i] * np.exp(i * x)
    return poly

x_axis = np.linspace(1, 5, 1000)
plt.plot(x_axis, exp_poly_1(x_axis), color = 'r')
plt.scatter(t, cells)
plt.xlabel('t, h')
plt.ylabel('брой клетки (x1000)')
plt.show()
```



□

Да разгледаме още един практически пример, аналогичен на предходния.

Задача 20. В таблицата са дадени данни за зависимостта между нивото на алкохол в кръвта (BAC - Blood Alcohol Level) и относителния риск за попадане в ПТП (т.е. колко пъти се увеличава рискът спрямо водач, който не е употребявал алкохол)

BAC	0	0.03	0.07	0.15	0.21	0.27
relative risk of crashing	1	1.06	2.09	22.1	99.78	328.602

Да се построи интерполационен полином по подходящ базис, който описва данните от таблицата. Да се начертае графиката му, заедно с данните от таблицата.

Тръгвайки да решаваме дадена интерполационна задача, бихме искали да знаем дали тя има решение и дали то е единствено. Отговорът на този въпрос е свързан с т.нар. Чебишови системи.

Дефиниция 7. *Казваме, че функциите* $\{\varphi_k(x)\}_{k=0}^n$ *образуват* ***Чебишова система*** *(Т-система) в интервала* $[a,b]$*, ако всеки ненулев обобщен полином по тази система има не повече от* n *различни нули в дадения интервал.*

Разглеждането на функции, които образуват системи на Чебишов е важно, тъй като е в сила следното твърдение:

Твърдение 7. *Нека функциите* $\varphi_0(x), \ldots, \varphi_n(x)$ *образуват система на Чебишов в интервала* $[a,b]$*. Тогава при дадени произволни възли* $x_0 < \cdots < x_n$ *в* $[a,b]$ *и стойности* $y_0, \ldots, y_n$ *интерполационната задача*

$$a_0\varphi_0(x_k) + \cdots + a_n\varphi_n(x_k) = y_k, \;\; k = 0, \ldots, n$$

има единствено решение.

С други думи, ако дадени функции образуват система на Чебишов в даден интервал, можем да използваме обобщените полиноми по тази система функции,

за решаване на интерполационна задача, в случай, че възлите на интерполация са произволни точки в дадения интервал.

Да разгледаме примери за функции, образуващи Чебишова система.

Задача 21. Да се докаже, че $\{1, x, x^2, \ldots, x^n\}$ образуват Чебишова система във всеки интервал $[a, b]$.

Решение. Нека вземем един произволен обобщен полином по тази система, т.е. една тяхна произволна линейна комбинация:

$$\varphi(x) = a_0.1 + a_1.x + a_2.x^2 + \ldots + a_n.x^n.$$

Искаме да докажем, че $\varphi(x)$ има не повече от n различни нули в интервала $[a, b]$. Но това е алгебричен полином от степен най-много n и от Основната теорема на алгебрата непосредствено следва, че той има не повече от n нули в кой да е интервал. С това задачата е доказана. □

Забележка. По този начин още веднъж доказахме, че ако $P(x) \in \pi_n$, то интерполационната задача $P(x_i) = y_i$, $i = 0, 1, \ldots, n$ (т.е. интерполационната задача на Лагранж) има единствено решение.

Задача 22. Да се докаже, че $\{x^{2k+1}\}_{k=0}^n$ образуват Чебишова система във всеки интервал $[\alpha, \beta]$, $0 < \alpha < \beta$.

Решение. Вземаме обобщен полином по тази система:

$$\varphi(x) = a_0 x + a_1 x^3 + a_2 x^5 + \ldots + a_n x^{2n+1}.$$

Искаме да покажем, че той има не повече от n различни нули в интервала $[\alpha, \beta]$ и затова разглеждаме уравнението

$$\varphi(x) = a_0 x + a_1 x^3 + a_2 x^5 + \ldots + a_n x^{2n+1} = 0$$

Тъй като 0 не е в разглеждания интервал, можем спокойно да разделим двете страни на x. Полагаме $x^2 = y$ и получаваме

$$\varphi(x) = a_0 + a_1 y + a_2 y^2 + \ldots + a_n y^n = 0.$$

Това уравнение има най-много n корена y^*. На всеки такъв корен съответстват два корена за $x - \pm\sqrt{y^*}$. Със сигурност обаче най-много 1 от тях е положителен, т.е. най-много 1 от тях е в интервала $[\alpha, \beta]$. Следователно $\varphi(x)$ има най-много n нули в $[\alpha, \beta]$ и задачата е доказана. □

Със следващата задача ще илюстрираме факта, че това дали дадени функции образуват система на Чебишов зависи съществено от интервала, в който ги разглеждаме.

Задача 23. Да се докаже, че $\{1, \cos x\}$ образуват Чебишова система в интервала $[0, \pi)$, но не образуват в $[0, 2\pi)$.

Решение. Ще докажем, че

$$\varphi(x) = a.1 + b.\cos x$$

има най-много 1 нула в интервала $[0, \pi)$. Първо да отбележим, че ако $b = 0$, то, за да има уравнението $\varphi(x) = 0$ корени, е необходимо $a = 0$, но тогава $\varphi(x)$ е нулевият полином. Следователно е достатъчно да разглеждаме $b \neq 0$ и нулите на $\varphi(x)$ се получават при $\cos x = -\frac{a}{b}$. Ако $-1 \leq -\frac{a}{b} \leq 1$ уравнението има 1 решение в интерала $[0, \pi)$, иначе – няма решения. С други думи $\varphi(x)$ наистина има най-много 1 нула в разглеждания интервал и $\{1, \cos x\}$ образуват Чебишова система в него.

Да разгледаме сега същите функции върху интервала $[0, 2\pi)$. Достатъчно е да покажем, че съществува поне един обобщен полином по тази система, който да има повече от 1 нула в $[0, 2\pi)$. Нека вземем

$$\psi(x) = 0.1 + 1.\cos x = \cos x.$$

Той очевидно има 2 корена в $[0, 2\pi)$ и това са $\frac{\pi}{2}$ и $\frac{3\pi}{2}$, т.е. $\{1, \cos x\}$ не образуват система на Чебишов в интервала $[0, 2\pi)$. □

Много важен частен случай на обобщени полиноми са т.нар. тригонометрични полиноми.

Дефиниция 8. *Обобщените полиноми по системата*

$$\{1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots, \sin nx, \cos nx\},$$

които имат вида

$$\tau_n(x) = a_0 + \sum_{k=1}^{n} (a_k \cos kx + b_k \sin kx),$$

наричаме тригонометрични полиноми от ред n.

Може да се докаже, че следното твърдение

Твърдение 8. *Нека $\alpha \leq x_0 < x_1 < \ldots < x_{2n} < \alpha + 2\pi$. Тогава за всяка функция f, определена в точките $\{x_i\}_0^{2n}$ съществува единствен тригонометричен полином τ_n от ред n такъв, че*

$$\tau_n(x_i) = f(x_i),\ i = 0, \ldots, 2n.$$

Тъй като тригонометричните полиноми са очевидно 2π-периодични функции, те се явяват удобен апарат за моделиране на периодични явления. Нека разгледаме следния пример:

Задача 24. В таблицата са дадени данни от сигнал на акселерометър в 5 момента от време:

t, ms	1	1.5	3	4	6
ускорение, m/s^2	0	1	1.2	4	2

Да се намери обощен полином по подходящ базис, който интерполира тези данни, ако е известно, че сигналът се описва от периодична функция с период а) $T = 2\pi$; б) $T = 8$. Да се начертае графиката на полинома в интервала $[0, 4\pi]$ заедно с точките в една координатна система във всеки от случаите.

Решение. □

Първо, да отбележим, че за да има смисъл да приближаваме една функция с тригонометричен полином, тази функция трябва да бъде 2π-периодична. Да разгледаме най-напред подусовие а), където това е изпълнено. Дадените 5 интерполационни условия могат да определят еднозначно 5 коефициента и следователно можем да построим върху тях тригонометричен полином от ред 2. Търсим го във вида

$$\tau_2(t) = a_0 + a_1 \cos t + b_1 \sin t + a_2 \cos 2t + b_2 \sin 2t.$$

Системата уравнения, удовлетворяваща интерпоалционните условия, следва отново да бъде записана във векторно-матрична форма. За определяне на тригонометричен полином от ред n, с възли $\{t_i\}_{i=0}^n$ и съответни стойности $\{y_i\}_{i=0}^n$, матрицата на системата има вида

$$A = \begin{pmatrix} 1 & \cos t_0 & \sin t_1 & \cos 2t_0 & \sin 2t_0 \cdots & \cos nt_0 & \sin nt_0 \\ 1 & \cos t_1 & \sin t_1 & \cos 2t_1 & \sin 2t_1 \cdots & \cos nt_1 & \sin nt_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \\ 1 & \cos t_n & \sin t_n & \cos 2t_n & \sin 2t_n \cdots & \cos nt_n & \sin nt_n. \end{pmatrix}, \tag{2.5}$$

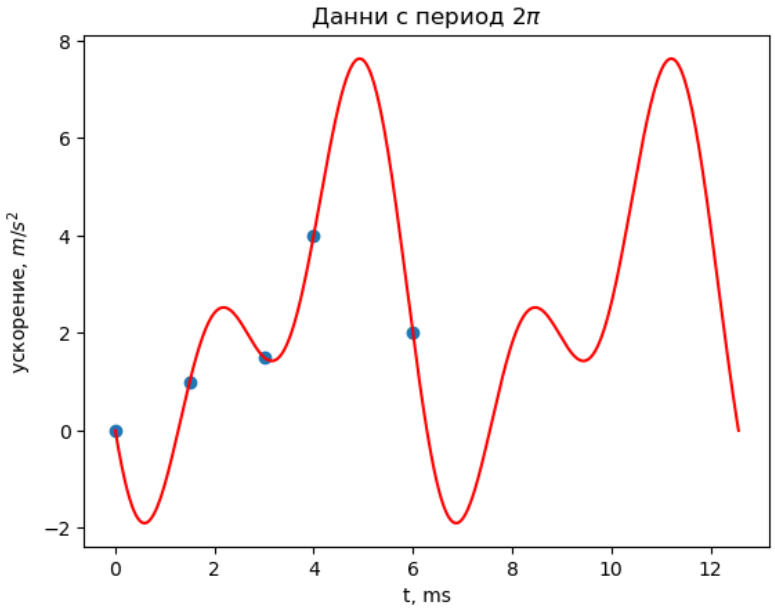
В часност, за разглежданата задача, матрицата на системата ще бъде отново матрица 5×5 от горепосочения вид. Да я гнерираме с *Pyhton* и да решим съответната система.

[16]:
```
t = np.array([0, 1.5, 3, 4, 6])
acceleration = np.array([0, 1, 1.5, 4, 2])
A = np.ones([5, 5])
for i in range (5):
    for j in range (1, 5):
        if(j % 2 != 0):
            A[i, j] = np.cos((math.floor(j / 2) + 1) * t[i])
        else:
            A[i, j] = np.sin(j / 2 * t[i])


sol = np.linalg.solve(A, acceleration)
```

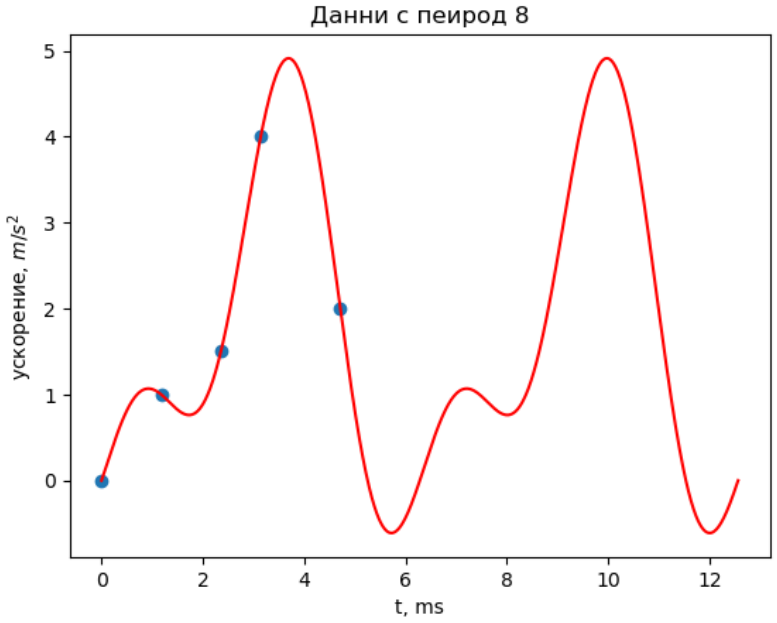
```
       [ 2.51827865 -0.71350272 -3.04121566 -1.80477593 -1.
↪56550947]
```

[17]:
```
def trig_poly_1(x):
    return sol[0] + sol[1] * np.cos(x) + sol[2] * np.sin(x) +␣
↪sol[3] * np.cos(2 * x) + sol[4] * np.sin(2 * x)
```

```
x_axis = np.linspace(0, 4 * np.pi, 1000)
plt.plot(x_axis, trig_poly_1(x_axis), color = 'r')
plt.scatter(t, acceleration)
plt.title("Данни с период $2 \pi$")
plt.xlabel('t, ms')
plt.ylabel('ускорение, $m/s^2$')
plt.show()
```



В подусловие б) е указано, че периодът на разглежданите данни е 8. Това означава, че за да може да ги интепролираме с тригонометричен полином, е необходимо да направим линейна смяна на променливата, която описва възлите. Тази смяна има за цел да свие интервала $t \in [0, 8]$ в интервала $t* \in [0, 2\pi]$ и има вида $t^* = \frac{2\pi}{8}t$. Вече сме готови да построим интепролационния полином от ред 2 в термините на новата променлива t^* по абсолютно аналогичен на вече описания в подусловие а) начин. Така получаваме



Задача 25. В таблицата са дадени данни за средните месечни количества слънчева радиация на територията на България, като лиспват данни за м. август

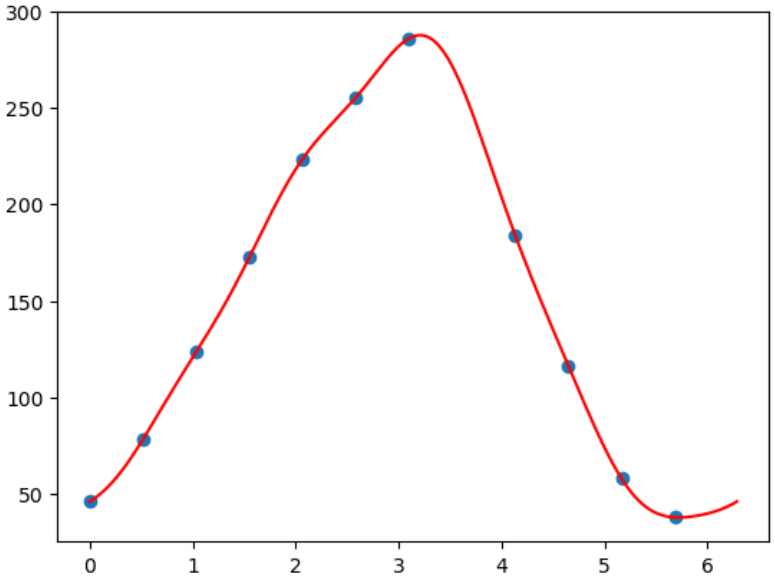
t, месец	Я	Ф	М	А	М	Ю	Ю	А	С	О	Н	Д
слънчева радиация, W/m^2	45.9	78.2	123.5	172.6	223.5	255.3	286.0		183.9	116.2	57.8	37.7

Данните са осреднени на 30 дни, т.е. можем да считаме, че разстоянието между измерванията в два съседни месеца е 30. Като се има предвид това, както и факта, че разглежданият процес е с период 365 дни, да се намери обобщен полином по подходящ базис, който описва данните в таблицата. Като се използва така намерения полином, да се пресметне приближено количеството слънчева радиация, съответстващо на м. август. Да се сравни с действителната стойност - $257.9 W/m^2$.

Решение. Естеството на разглеждания процес, както и самите данни, подсказват, че става дума за периодично явление, което е естествено да се моделира с тригонометричен полином. Тъй като разстоянието между две съседни измервания е 30, то например, започвайки от 0, на всеки от месеците можем да асоциираме равноотдалечени на разстояние 30 числа. Преди да започнем с определянето на неизвестните коефициенти в полинома, е необходимо още да направим смяна на променливата $t^* = \dfrac{2\pi}{365} t$:

[21]:
```
days_with_aug = np.arange(0, 12 * 30, 30)
days = np.delete(days_with_aug, 7)
radiation = np.array([45.9, 78.2, 123.5, 172.6, 223.5, 255.3, 286.
 ↪0, 183.9, 116.2, 57.8, 37.7])
days_changed = (2 * np.pi) / 365 * days
```

Вече сме готови да намерим търсения тригонометричен полином. По зададените 11 точки в ранвнината можем да определим тригонометричен полином от ред 5, т.е. матрицата на системата уравненения, която трябва да решим, ще е матрица от вида (2.5) с размерност 11×11. Извършваме съответните пресмятания и получаваме графиката на полинома



За стойността, съответстваща на м. август, получаваме

[23]:
```
trig_poly_2((2 * np.pi) / 365 * 210, sol)
```

[23]:
```
260.79356396914744
```

Може да се покаже, че функциите

$$\tau_k(x) = \prod_{i=0, i\neq k}^{n} \frac{\sin\frac{x-x_i}{2}}{\sin\frac{x_k-x_i}{2}}$$

образуват интерполационен базис за пространството от тригонометрични полиноми, т.е. те са такива тригонометрични полиноми, че

$$\tau_k(x_k) = 1, \quad \tau_k(x_i) = 0, i \neq k.$$

Тоагава можем да намерим един тригонометричен полином като използваме формулата на Лагранж с базисни функции функциите $\tau_k(x), k = 0, \ldots, n$.

Глава 3

Приближения в линейни нормирани пространства.

3.1 Метод на най-малките квадрати

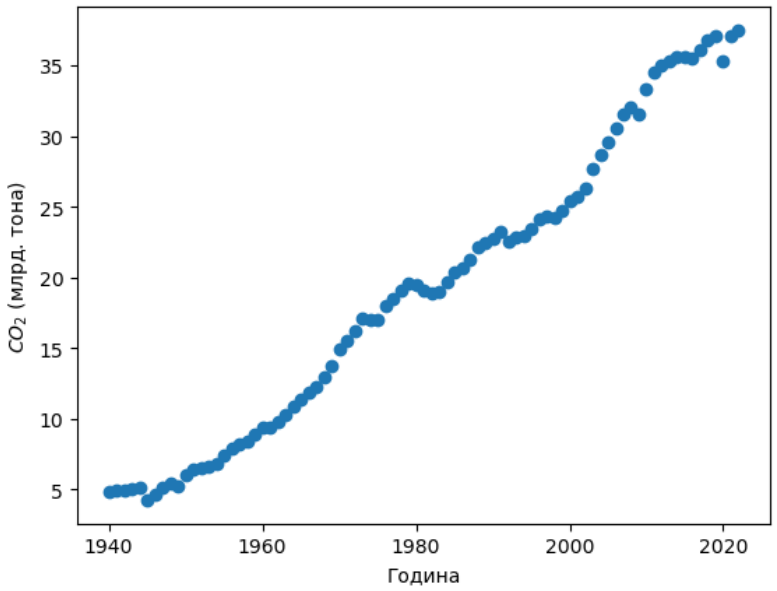
Дотук разгледахме различни начини за решаване на следната задача:

Дадени са точките $(x_0, y_0),\ldots,(x_n, y_n)$. Търсим функция, чиято графика минава точно през тези точки. Както отбелязахме, една възможна интерпретация на тази задача, от гледна точка на практиката, е че са направени краен брой измервания (експерименти) при изследването на дадено явление и търсим функция, която да моделира явлението, като отговаря на тези емпирични резултати. И с най-съвършената техника обаче има някаква допустима грешка при правенето на тези измервания. В много случаи тази грешка не може да бъде пренебрегната. Тогава какъв би бил смисълът да намерим функция, която да минава през тези точки, след като дори самите те не са „на мястото си"? Друг проблем, който видяхме при интерполацията е, че често полиномите от висока степен имат „лошо" поведение, т.е. може да имаме проблеми при моделирането на явление, за което искаме да приближим голямо количество данни.

Оказва се, че можем да постъпим и по друг начин – да търсим функция, която следва поведението на данните (без задължително да съвпада с тях в която и да е точка) и която е „близо" до тези данни.

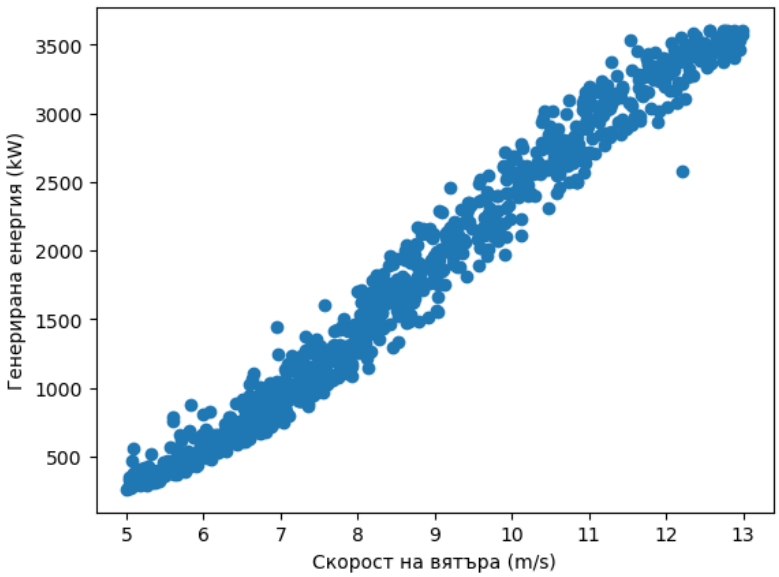
Преди да коментираме самия метод, нека разгледаме няколко реални процеса и да коментираме каква функция е подходяща, за да ги опише (на база на данните, които имаме).

- На графиката са дадени данни за нивата на въглеродния диоксид в млрд. тонове в периода 1940 – 2020г.



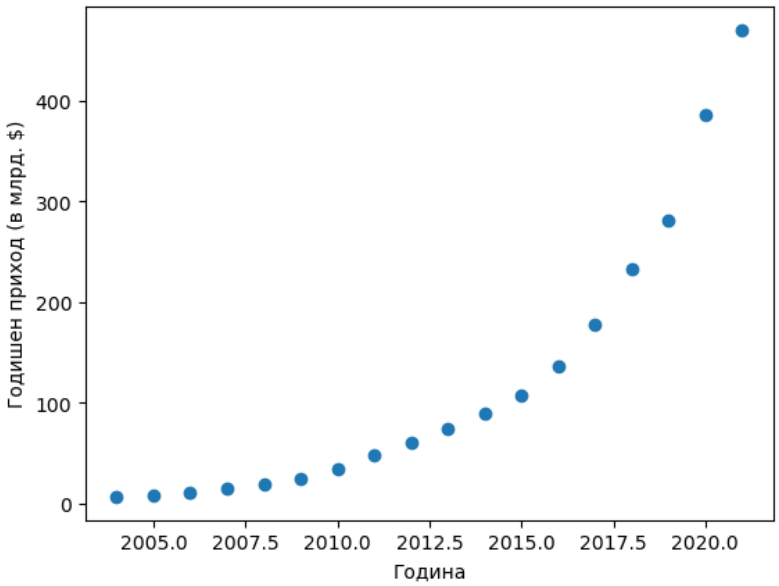
Точките очевидно не лежат на права линия, но общата тенденция на данните, би могла да се опише с такава, т.е. можем да търсим приближение на данните във вида $f(x) = ax + b$.

- Известно е, че връзката между скоростта на вятъра и генерираното количество електроенергия от една вятърна турбина може да се опише чрез кубична функция. На графиката са дадени данни за произведеното количество енергия от конкретна вятърна турбина като функция на скоростта на вятъра.



В този случай ще търсим функцията, моделираща процеса, във вида $f(x) = ax^3 + bx^2 + cx + d$.

- Да разгледаме и един процес, който може да се моделира с функция, която не е алгебричен полином. На графиката са визуализирани данни за годишния приход на *Amazon* в млрд. щатски долари за периода 2004 - 2021г.



Тук, на база на експерименталните данни, следва да търсим експоненциална зависимост. Ще търсим функцията във вида $f(x) = ae^{bx}$.

Ще оставим тези 3 процеса за малко и ще се върнем към тях, след като първо изясним метода, по който ще търсим съответните функции.И така, от горните примери е ясно, че след като сме избрали вида на функцията, с която ще приближаваме, трябва да определим параметрите в нея (например коефициентите на квадратния тричлен във втория пример) така, че функцията да се окаже възможно „най-близо" до данните, които приближаваме. Първото, което трябва да направим, е вече да формализираме понятието „близо".

Нека точките, които искаме да приближим, имат координати

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n),$$

а $f(x)$ е функция, с която ги приближаваме. Да означим с e_i грешката, т.е. разликата между действителната и приближената стойност в точката x_i

$$e_i := f(x_i) - y_i.$$

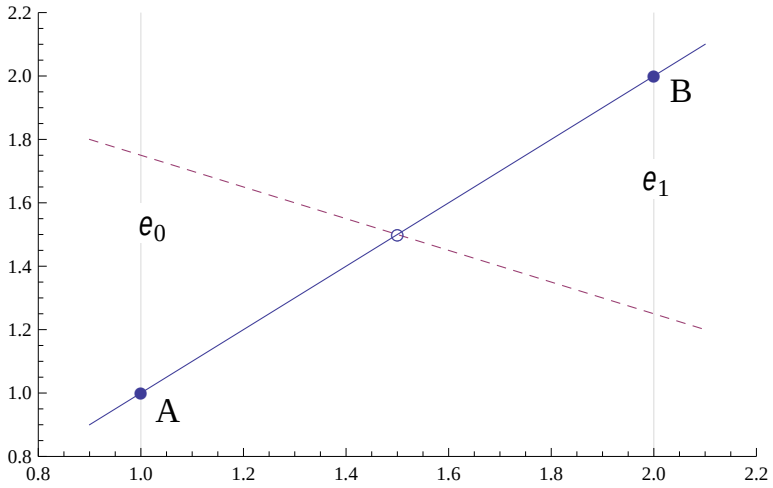
Да разгледаме някои възможни (но, по една или друга причина, неудачни) начини за да дефинираме понятието „най-близо":

- Първата очевидна идея е да търсим функцията $f(x)$ така, че сумата от всички грешки да е възможно най-малка, т.е. да е възможно най-малко

$$\sum_{i=0}^{n} e_i.$$

Да разгледаме обаче следния пример (вж. фигурата). Имаме две точки в равнината $A = (x_0, y_0)$, $B = (x_1, y_1)$. Търсим линейна функция, която да е „най-близо" до тях. Очевидно, това би следвало да е правата, която минава през тези две точки. Но, ако вземем произволна друга права, която минава през средата на отсечката AB, за нея също ще е изпълнено $e_0 + e_1 = 0$, тъй като в точките A и B грешките ще имат една и съща абсолютна

стойност и ще са различни по знак. С други думи, всяка такава права ще удовлетворява условието $\sum_{i=0}^{n} e_i$ да е минимално.



- Възможен начин да избегнем проблема е да не позволим грешките да бъдат с различни знаци, т.е. да ги вземаме по модул и да поставим условието функцията $f(x)$ да е избрана така, че

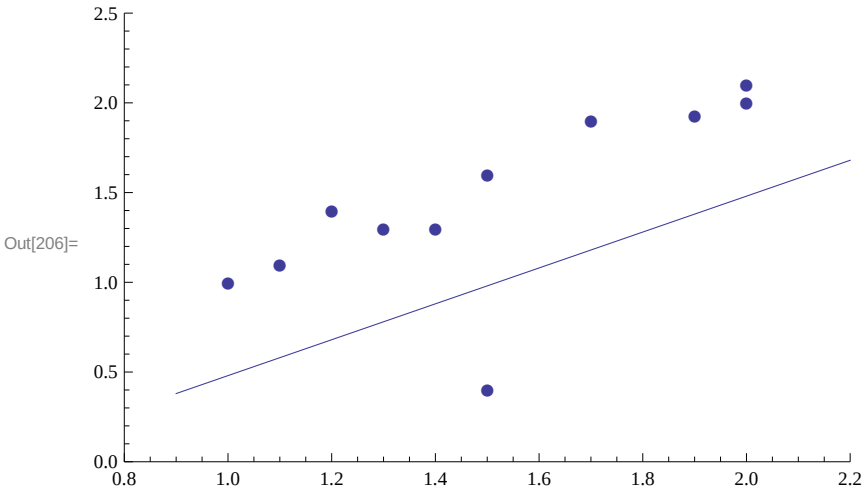
$$\sum_{i=0}^{n} |e_i|$$

да е минимално. Оказва се обаче, че и при това условие (макар и напълно логично и удовлетворително) в общия случай не можем да намерим единствена функция, която да го изпълнява.

- Можем да подберем функцията $f(x)$ така, че да е най-малка максималната грешка, т.е. да е най-малко

$$\max_{i=0}^{n} e_i.$$

Да вземем обаче следния пример:



Всички точки, с изключение на една, лежат приблизително на права, но тази „странична" точка, която вероятно е резултат от шум или грешка в

измерванията, има в някакъв смисъл същото влияние върху избора на правата, както всички останали, взети заедно.

И така, след като показахме някои неудачни начини, сега вече да се концентрираме върху същността на **метода на най-малките квадрати**. Търсим функцията $f(x)$ така, че

$$\sum_{i=0}^{n} e_i^2$$

да е възможно най-малко. Освен, че решава проблема с наличието на различни знаци при сумирането на грешките, оказва се, че този подход води и до единствено решение.

Задачата за минимизиране на тази сума може да се интерпретира и по друг начин. Нека означим с y и F съответно векторите с елементи точните и приближените стойности

$$y = (y_0, y_1, \ldots, y_n)$$
$$F = (f(x_0), f(x_1), \ldots, f(x_n)).$$

Тогава $e = F - y$ ще бъде векторът с грешките

$$e = (e_0, e_1, \ldots, e_n).$$

За него имаме

$$|e| = \sqrt{e_0^2 + e_1^2 + \ldots + e_n^2},$$

т.е. задачата за минимизиране на сумата

$$\sum_{i=0}^{n} e_i^2$$

можем да разглеждаме и като минимизиране на дължината на вектора на грешката.

Следващата стъпка е да покажем как точно ще намерим функцията $f(x)$ така, че тази сума да е възможно най-малка. Ще го покажем с пример

Задача 26. Да се намери линейна функция, която приближава по метода на най-малките квадрати таблицата

x_i	0	1	2	3	4
y_i	0	1	1	2	2

Решение. От условието следва, че ще търсим функцията във вида $f(x) = ax+b$. Трябва да определим параметрите a и b така, че

$$\sum_{i=0}^{4} e_i^2 = \sum_{i=0}^{4} (f(x_i) - y_i)^2$$

да е възможно най-малко. Имаме

$$f(0) = b;\ f(1) = a + b;\ f(2) = 2a + b;\ f(3) = 3a + b;\ f(4) = 4a + b.$$

Тогава

$$\sum_{i=0}^{4} e_i^2 = b^2 + (a+b-1)^2 + (2a+b-1)^2 + (3a+b-2)^2 + (4a+b-2)^2.$$

Разглеждаме този израз като функция на двете променливи a и b (нека я означим с $\Phi(a,b)$). Необходимо условие тази функция да има минимум в дадена точка е първите частни производни по отношение на всеки параметър да са нули. Получаваме системата

$$\begin{vmatrix} \dfrac{\partial \Phi}{\partial a} = 0 \\ \dfrac{\partial \Phi}{\partial b} = 0 \end{vmatrix}$$

По този начин получаваме система с толкова уравнения, колкото са и параметрите в задачата. Решаваме я и еднозначно определяме a и b.

$$\begin{vmatrix} 2(a+b-1) + 2(2a+b-1).2 + 2(3a+b-2).3 + 2(4a+b-2).4 = 0 \\ 2b + 2(a+b-1) + 2(2a+b-1) + 2(3a+b-2) + 2(4a+b-2) = 0 \end{vmatrix}$$

Окончателно имаме $a = \dfrac{1}{2}, b = \dfrac{1}{5}$. Така получихме търсената функция във вида

$$f(x) = \frac{1}{2}x + \frac{1}{5}.$$

□

Нека сега да видим как бихме могли да имплементираме решението на Задача 26 с помощта на *Python*.

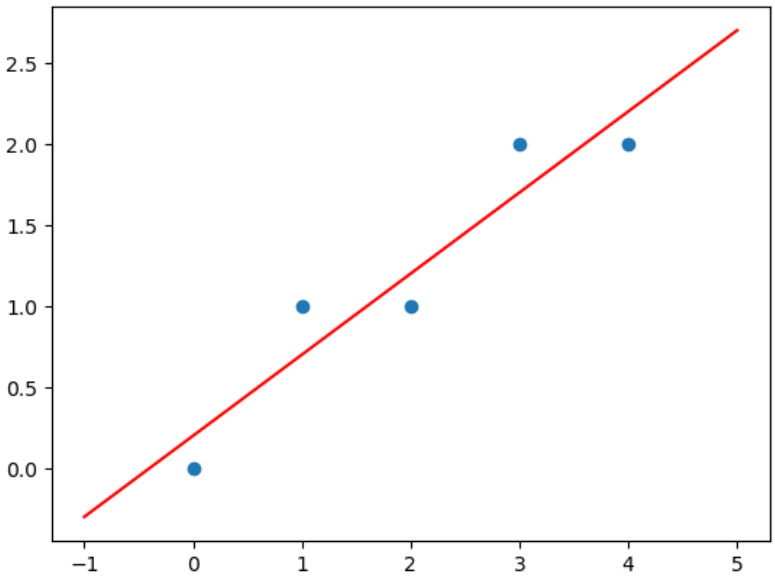
Най-напред ще подходим към решаването на задачата символно (аналитично), т.е. ще построим системата от урванения 26 в *Python* и ще я решим с помощта на инструменти за символни пресмятания в езика. За целта ще използваме библиотеката *sympy*:

[4]:
```
from sympy import symbols, diff, Eq, solve

a, b = symbols('a, b')

def phi(a, b):
    return b ** 2 + (a + b - 1) ** 2 + (2 * a + b - 1) ** 2 + (3 *␣
 ↪a + b - 2) ** 2 + (4 * a + b - 2) ** 2

equations = [Eq(diff(phi(a, b), a), 0),
             Eq(diff(phi(a, b), b), 0)]

solve(equations)
```

[4]:
```
{a: 1/2, b: 1/5}
```

Така получихме стойностите на неизвестните коефициенти. Остава само да изчертаем графиката на получената функция заедно с данните в една координатна система:

[5]:
```
def f(x):
    return 1/2 * x + 1/5

x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 1, 1, 2, 2])
x_axis = np.linspace(-1, 5, 100)
plt.scatter(x, y)
plt.plot(x_axis, f(x_axis), 'red')
plt.show()
```



И така, след като изяснихме общия подход за имплементация на решението на задача за най-малки квадрати в *Python*, сме готови да се върнем към трите примера, разгледани в началото на настоящата секция. Те са свързани с моделирането на реални данни, чиито източници ще бъдат посочени във всяка от задачите. Поради по-големия обем на данните, те се съхраняват в *.csv* файлове, които можете да откриете на страницата на курса в *Moodle*.

Задача 27. Във файла *CO_2_data.csv* се съдържат данни за нивата на въглеродния диоксид в атмосферата (в млрд. тонове) за периода 1940-2022г. Да се зареди файла и да се визуализират данните. По метода на най-малките квадрати да се построи линейна функция, която приближава данните.

Източник: https://www.statista.com/statistics/276629/global-co2-emissions/

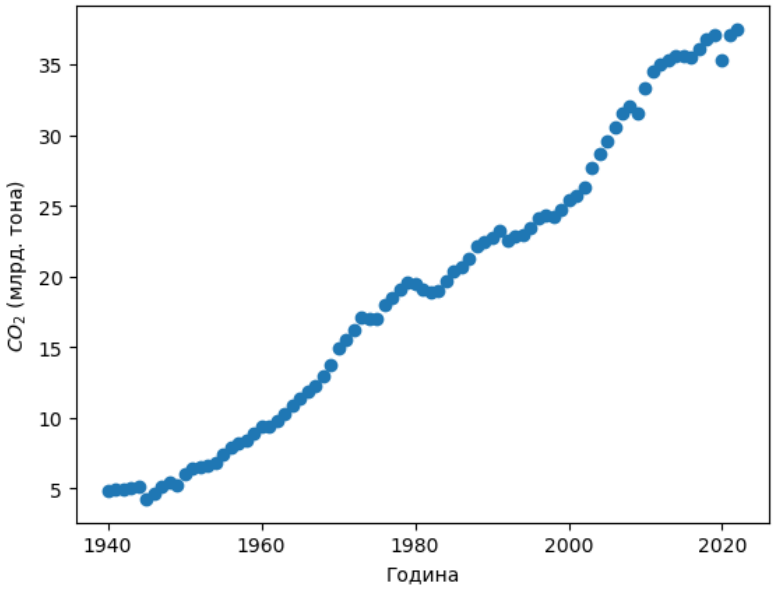
Решение. Най-напред да заредим и да илюстрираме данните в един *Jupyter Notebook*. За целта ще използваме библиотеката *Pandas*, предназначена за обработка на данни.

[10]:
```
import pandas as pd
df = pd.read_csv('CO_2_data.csv')
df.head()
```

[10]:
```
year   co2
0  1940  4.85
1  1941  4.97
2  1942  4.95
3  1943  5.04
4  1944  5.11
```

Данните се зареждат във файла във вид на *dataframe* - таблица, чиито редове и стълбове можем да достъпваме, за да извлечем необходимите ни записи, напр. по следния начин:

[11]:
```
years = df.loc[:, "year"]
co2 = df.loc[:, "co2"]
plt.scatter(years, co2)
plt.xlabel('Година')
plt.ylabel('$CO_2$ (млрд. тона)')
plt.show()
```



И така, след като сме заредили данните, остава да определим неизвестните параметри в търсената линейна функция по начин, аналогичен на решението на предходната задача:

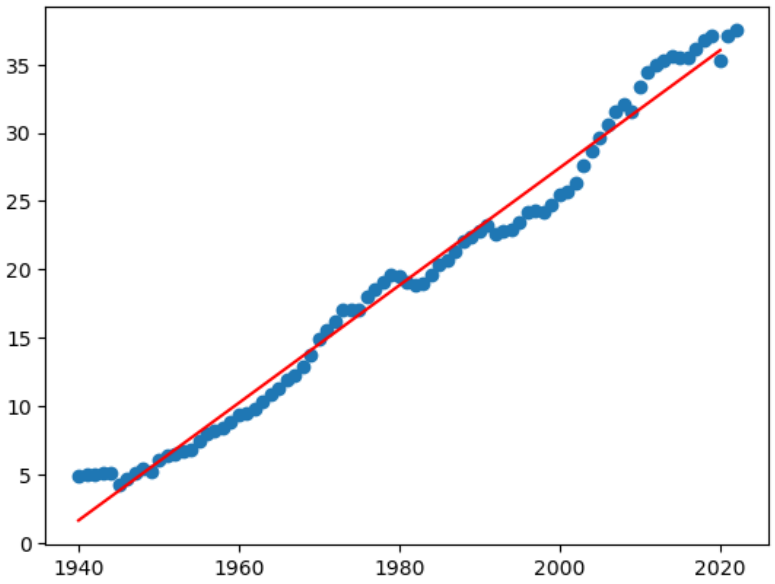
[10]:
```
a, b = symbols('a, b')

def f(a, b, x):
    return a * x + b
```

```
def phi(a, b):
    sum_of_squares = 0
    for i in range(years.size):
        sum_of_squares += (f(a, b, years[i]) - co2[i]) ** 2
    return sum_of_squares

equations = [Eq(diff(phi(a, b), a), 0),
             Eq(diff(phi(a, b), b), 0)]

sol = solve(equations)
print(sol)
```

```
{a: 0.430363124973763, b: -833.280314428446}
```

Прилагаме и графика на получената линйена функция, заедно с данните:

[11]:
```
x_axis = np.linspace(1940, 2020, 1000)
plt.scatter(years, co2)
plt.plot(x_axis, f(sol[a], sol[b], x_axis), 'red')
plt.show()
```



□

Задача 28. Известно е, че връзката между скоростта на вятъра и генерираното количество електроенергия от една вятърна турбина може да се опише чрез кубична функция. Във файла *wind_ turbine_ data_ sample.csv* се съдържат измервания за количеството произведена електроенергия (kW) от конкретен вятърен генератор в зависимост от скоростта на вятъра (m/s). Да се зареди файла и да се визуализират данните. По метода на най-малките квадрати да се построи полином от трета степен, който приближава данните.

Източник: https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset/

Решение. Отново първата ни стъпка е зареждането на данните

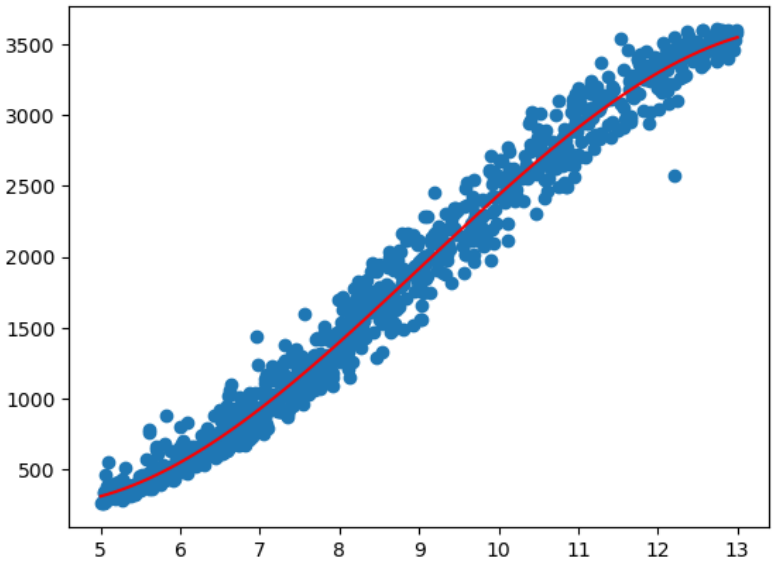
[16]:
```
df = pd.read_csv('wind_turbine_data_sample.csv')
wind_speed = df.loc[:, "wind speed"]
power_output = df.loc[:, "power output"]
```

Единствената разлика, в сравнение с решението на предходната задача, е в общия вид на функцията, чрез която ще моделираме даните - вместо линейна функция, този път тя ще бъде алгебричен полином от трета степен.

[15]:
```
a, b, c, d = symbols('a, b, c, d')

def f(a, b, c, d, x):
    return a * x ** 3 + b * x ** 2 + c * x + d

def phi(a, b, c, d):
    sum_of_squares = 0
    for i in range(wind_speed.size):
        sum_of_squares += (f(a, b, c, d, wind_speed[i]) -␣
 ↪power_output[i]) ** 2
    return sum_of_squares

equations = [Eq(diff(phi(a, b, c, d), a), 0),
             Eq(diff(phi(a, b, c, d), b), 0),
             Eq(diff(phi(a, b, c, d), c), 0),
             Eq(diff(phi(a, b, c, d), d), 0)]

sol = solve(equations)
print(sol)
```

```
{a: -7.59606064982793, b: 205.917112180286, c: -1334.75406713179, d:
 ↪2787.95787856771}
```

[16]:
```
x_axis = np.linspace(5, 13, 100)
plt.scatter(wind_speed, power_output)
plt.plot(x_axis, f(sol[a], sol[b], sol[c], sol[d], x_axis), 'red')
plt.show()
```



□

Забележка: Тъй като системата, която е необходимо да се реши в този случай е значително по-сложна, в сравнение с предходната, то е възможно символното и решаване да отнеме доста повече време. Ще се върнем на този въпрос малко по-нататък.

Преди да се концентрираме върху третия модел ще коментираме една техника, която е полезна в определени ситуации.

Нека търсим приближението във вида $f(x) = \alpha e^{\beta x}$. Очевидно в този случай системата уравнения, която ще получим за параметрите a и b ще бъде нелинейна. В някои ситуации е уместно да решаваме системата в този вид, но често това може да създаде проблеми. Затова можем да подходим и по друг начин. Имаме

$$y_i \approx f(x_i) = \alpha e^{\beta x_i}$$

Тогава, логаритмувайки двете страни, получаваме

$$\ln y_i \approx \ln \alpha + \beta x_i$$

и сега отново определяме коефициентите така, че да минимизират

$$\sum_{i=0}^{n} {e_i}^2,$$

но за e_i вземаме $\ln y_i - (\ln \alpha + \beta x_i)$ (т.е. минимизираме $\sum_{i=0}^{n} (\ln y_i - \ln f(x_i))^2$. Полагайки $\ln \alpha = c$, получената система ще е линейна спрямо c и β. След като намерим c, връщайки се в полагането, имаме $\alpha = e^c$.

След тези предварителни сведения вече сме готови да моделираме и третия пример от началото на параграфа.

Задача 29. Във файла *amazon_sales_net_revenue.csv* се съдържат данни за годишния приход на Amazon в млрд. щатски долари за периода 2004 - 2021г. Да се зареди файла и да се визуализират данните. По метода на най-малките квадрати, да се построи функция от вида $f(x) = ae^{bx}$, която моделира данните.

Източник: https://www.statista.com/statistics/266282/annual-net-revenue-of-amazoncom/

Решение. Ще търсим приближението във вида

$$f(x) = ae^{bx}$$

Както казахме, ще минимизираме сумата от квадратите не на $f(x_i) - y_i$, а на $\ln f(x_i) - \ln y_i$, т.е. на разликата от логаритмите на приближената и точната стойност. С други думи, искаме да определим коефициентите a и b така, че

$$\sum (\ln y_i - (c + bx_i))^2,$$

където $c = \ln a$, да е възможно най-малко.

Зареждаме данните и построяваме линейна функция по познатия вече начин, използвайки логаритмуваните стойности на годишния приход :

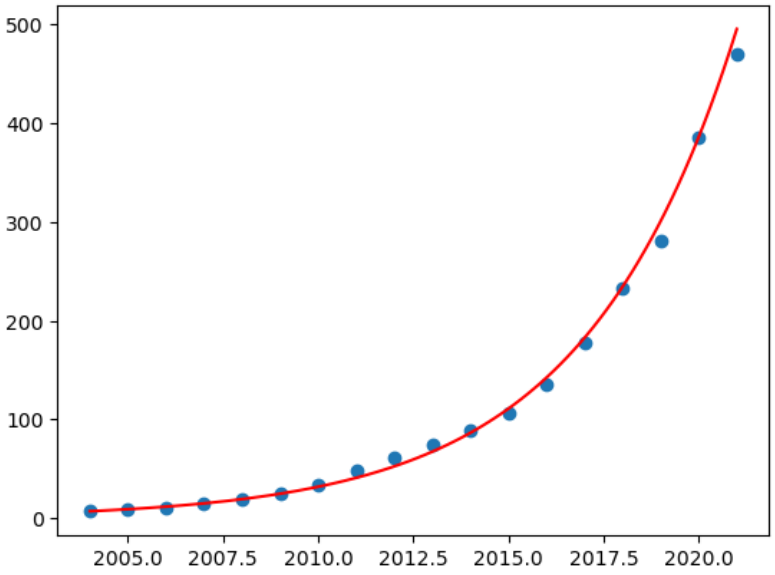
[21]:
```
df = pd.read_csv('amazon_sales_net_revenue.csv')
years = df.loc[:, "year"]
revenue = df.loc[:, "revenue"]
```

[20]:
```
c, b = symbols('c, b')

def f(c, b, x):
    return c * x + b

def phi(c, b):
    sum_of_squares = 0
    for i in range(years.size):
        sum_of_squares += (f(c, b, years[i]) - np.log(revenue[i]))␣
 ↪** 2
    return sum_of_squares

equations = [Eq(diff(phi(c, b), b), 0),
             Eq(diff(phi(c, b), c), 0)]

sol = solve(equations)
print(sol)
```

```
{b: -497.459709696735, c: 0.249215934116305}
```

След получаване на параметрите в линейната връзка, остава само да възстановим приближението в първоначалния му вид.

[21]:
```
def exp_func(x):
    return np.exp(float(sol[c]) * x + float(sol[b]))


x_axis = np.linspace(2004, 2021, 100)
plt.scatter(years, revenue)
plt.plot(x_axis, exp_func(x_axis), 'red')
plt.show()
```



□

Може да разгледаме още една подобна ситуация (не е правено на упражнения, виж допълнителната задача към темата за метода на най-малките квадрати): Търсим приближението във вида

$$f(x) = \frac{\alpha x}{\beta + x}.$$

С други думи, имаме

$$y_i \approx \frac{\alpha x_i}{\beta + x_i}$$

Вземаме реципрочните стойности на двете страни и получаваме

$$\frac{1}{y_i} \approx \frac{\beta}{\alpha}\frac{1}{x_i} + \frac{1}{\alpha}.$$

Дефинираме

$$e_i := \frac{1}{y_i} - \left(\frac{\beta}{\alpha}\frac{1}{x_i} + \frac{1}{\alpha}\right),$$

полагаме $\frac{\beta}{\alpha} = A$ и $\frac{1}{\alpha} = B$ и получената система ще бъде линейна спрямо A и B.