Modul Praktikum Pengantar Analisis Numerik

DRAFT

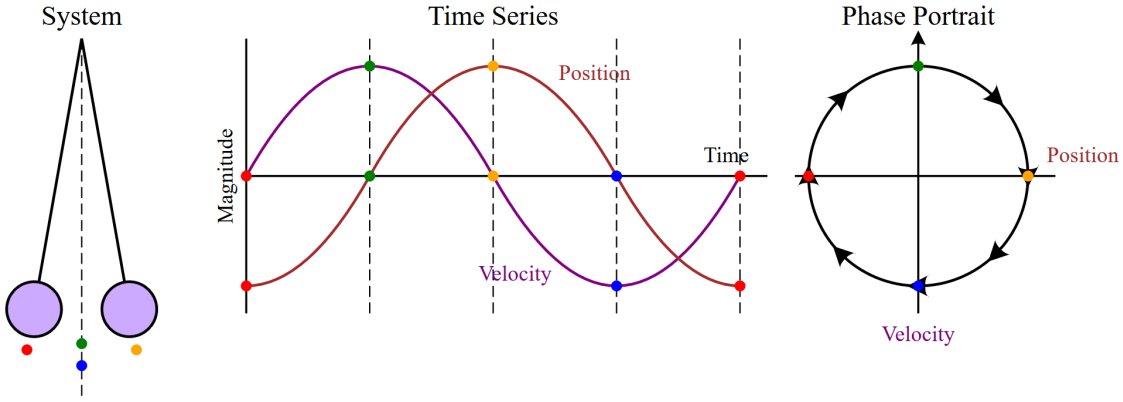
July 20, 2020

Contents

Chapter 1

Pencarian Akar Persamaan

Pada kehidupan nyata seringkali kita bertemu dengan masalah-masalah nonlinear yang membutuhkan penyelesaian. Meski hampir selalu dapat kita *linear*-kan [1] permasalahan tersebut, menemukan solusi dari permasalahan sebenarnya akan lebih memuaskan. Apalagi kalau ternyata permasalahan kita cukup rumit untuk dijadikan bentuk linear.



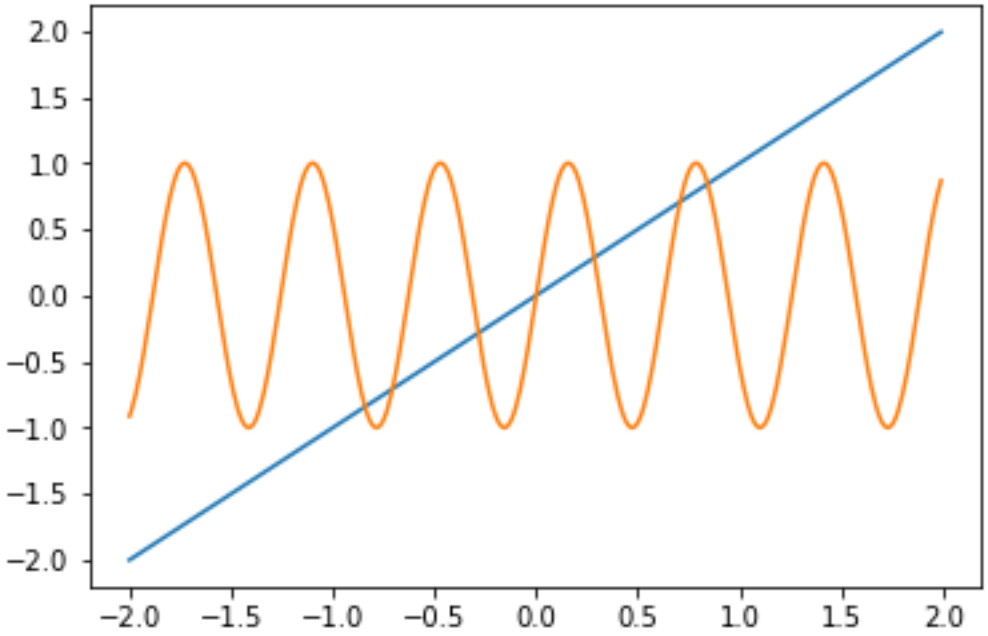
Masalah yang kemudian mungkin timbul adalah terdapat masalah yang apabila dibuat model matematikanya, kita tidak dapat mencari dengan cepat secara eksplisit solusi dari model

[1] Persamaan nonlinear bisa didekati dengan fungsi linear jika memenuhi syarat-syarat tertentu di daerah yang cukup dekat dengan suatu titik.

tersebut. Sebut saja misalnya kita diminta mencari solusi dari persamaan berikut yang cukup sering dilihat sehari-hari: fungsi linear bertemu fungsi trigonometri

$$x = \sin(10x).$$

Dengan sekali lihat, kita akan langsung tahu bahwa $x = 0$ adalah solusi. Tapi, apakah hanya itu satu-satunya solusi? Atau akan ada banyak solusi yang tidak dekat dengan 0? Ada berapa banyak solusi real untuk persamaan tersebut? Metode numerik digunakan untuk menyelesaikan permasalahan matematis yang tidak bisa diselesaikan secara analitik, atau mungkin dapat diselesaikan namun membutuhkan perhitungan yang tidak sederhana.



Meski pertanyaan-pertanyaan tersebut tidak akan dibahas pada bab ini secara menyeluruh, namun kita akan belajar mencari solusi-solusi persamaan-persamaan nonlinear dengan metode numerik.

Secara umum, pencarian akar dari fungsi ada beberapa metode. Beberapa yang cukup terkenal adalah:

1. Metode braket Metode ini adalah metode pencarian yang dilakukan dengan cara memperkecil interval pencarian yang diduga terdapat akar di dalamnya. Metode-metode ini menggunakan teorema nilai antara sebagai penjamin bahwa ada akar di dalam interval pencarian. Beberapa metode yang termasuk dalam kategori ini adalah 1.1 Metode Biseksi 1.2 Metode *Regula Falsi* 1.3 Metode Ridders
2. Metode Interpolasi Interpolasi adalah salah satu cara untuk menentukan akar persamaan. Dengan cara mendekati fungsi dengan polinomial berderajat rendah, diharapkan bahwa pendekatan ini cukup baik untuk menebak akar persamaan. Beberapa metode yang termasuk dalam kategori ini adalah 2.1 Metode Secant 2.2 Metode Muller
3. Metode Iteratif Iterasi disini lebih dimaksudkan kepada metode yang menggunakan suatu fungsi atau bentuk tertentu yang lebih spesifik. Tentu saja metode-metode yang lain membutuhkan iterasi dalam prakteknya. Beberapa metode yang bisa dikategorikan dalam metode iteratif adalah 3.1 Metode Newton-Rhapson dan temantemannya 3.2 Metode Secant (yang juga termasuk dalam metode interpolasi) 3.3 Metode Broyden 3.4 Metode Steffensen 3.5 Metode Interpolasi Invers Kuadrat
4. Kombinasi metode Salah satu metode yang cukup terkenal[^3] mengkombinasikan metode-metode yang lain adalah metode Brent atau metode Brent–Dekker. Mengkombinasikan metode biseksi, metode secant, dan metode interpolasi invers kuadrat.

Berikutnya akan dibahas beberapa metode pencarian akar persamaan.

1.1 Metode Biseksi

Metode biseksi adalah salah satu metode untuk mencari akar dari suatu fungsi. Idenya adalah mencari akar pada suatu area, suatu interval. Kemudian membagi dua interval tersebut (*bisect*). Dari kedua interval tersebut, ditentukan interval yang mana yang mempunyai akar sehingga kita punya interval yang lebih pendek, area yang lebih kecil untuk mencari akar persamaan yang diinginkan. Metode biseksi sering juga disebut metode pencarian biner

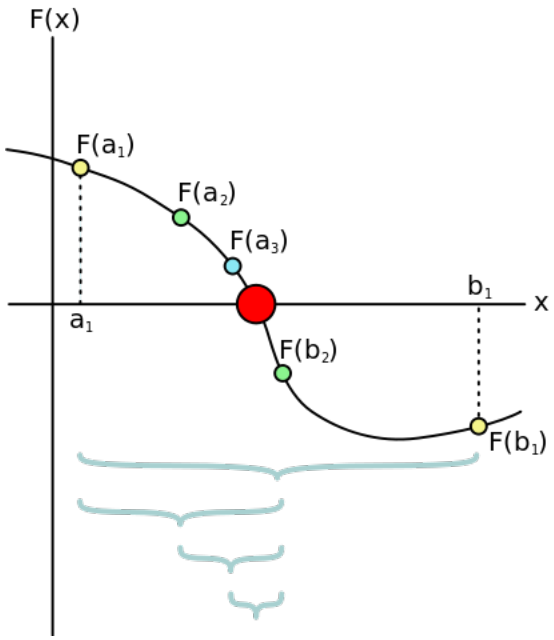
atau metode dikotomi [^2]. Berikut ilustrasi untuk metode biseksi untuk pencarian diskrit vs pencarian linear.

SOME ANIMATED PICTURES FROM MATHWAREHOUSE.COM

Pada pencarian diskrit hal yang harus diperhatikan adalah data harus urut (dari kecil ke besar ataupun sebaliknya). Hal ini lebih mudah dilakukan untuk fungsi yang kontinu. Untuk memastikan adanya akar pada satu interval, kita memerlukan beberapa teorema.

(Teorema Nilai Antara) Diberikan fungsi f yang kontinu di interval tertutup $I = [a, b]$. Jika u adalah nilai di antara $f(a)$ dan $f(b)$, maka terdapat c pada I sehingga $f(x) = u$.

Dalam penggunaannya, kita cari interval sehingga salah satu ujungnya hasil fungsinya positif dan ujung lainnya hasil fungsinya negatif, $u = 0$ adalah nilai yang dicari. Eksistensi nilai u dijamin oleh teorema nilai antara sebab a dan b berbeda tanda. Kemudian yang dilakukan adalah mencari titik tengah antara a dan b, sebut saja c, lalu melihat tanda dari $f(c)$ apakah sama dengan tanda (*sign*) $f(a)$ atau tanda $f(b)$. Jika sama dengan tanda $f(a)$, maka $[c, b]$ adalah interval baru yang dicari, kalau tidak, maka $[a, c]$ adalah interval baru yang dipilih. Proses ini diulangi sampai kita mendapatkan toleransi yang diinginkan.



1.1.1 Algoritma

Metode biseksi bisa ditulis dengan *pseudocode* sebagai berikut.

```
INPUT: Fungsi f, ujung interval LOW, HIGH, toleransi TOL, iterasi maksimum MAX
SYARAT AWAL: a < b,  f(a) * f(b) < 0
OUTPUT: nilai yang selisihnya dengan akar f(x)=0 kurang dari TOL


N ← 1
While N   NMAX
  MID ← (LOW+HIGH)/2
  If f(MID) = 0 or (HIGH-LOW)/2 < TOL then # MID adalah solusi
    Output(MID)
    Stop
  EndIf
  N ← N + 1 # increment step counter
  If (f(MID)*(f(LOW)>0 then LOW ← MID else HIGH ← MID # interval baru
EndWhile
Output("Metode biseksi gagal.") # Maksimum iterasi tercapai
```

1.1.2 Error dan Konvergensi

Pada metode biseksi, eksistensi akar dijamin ada oleh teorema nilai antara, hanya bila fungsi yang dicari akarnya kontinu serta kedua ujung interval nilai fungsinya berbeda tanda. Jadi dengan memilih interval awal yang tepat, kita selalu bisa mendekati akar yang diinginkan. Namun demikian, laju kekonvergenan fungsi ini hanyalah linear sebab pada setiap iterasi, kemungkinan eror hanya berkurang setengahnya. Misalkan f adalah fungsi yang dicari akarnya, c adalah akar dari f, dan c_n adalah tebakan ke n dari metode biseksi (yaitu MID), maka berlaku $ |c_n -c| < |a-b|/2^n$. Nilai absolut eror selalu berkurang setengah untuk

setiap iterasi. Akibatnya metode biseksi ini konvergen secara linear dengan faktor 1/2. Persamaan tersebut juga bisa digunakan untuk menentukan kira-kira berapa banyak iterasi yang dibutuhkan untuk konvergen ke solusi.

Misalkan ε adalah eror yang diperbolehkan dan n adalah banyak iterasi. Berlaku

$$|c_n - c| < |a - b|/2^n \iff 2^n < \frac{|a-b|}{|c_n - c|} \tag{1.1}$$

$$\iff n < \log_2\left(\frac{\varepsilon_0}{\varepsilon}\right) = \frac{\log \varepsilon_0 - \log \varepsilon}{\log 2} \tag{1.2}$$

1.1.3 Latihan

1. Buatlah fungsi biseksi dengan input interval ([LOW,HIGH]), fungsi (F), dan toleransi (TOL).
2. Tentukan salah satu akar dari persamaan $x^3 - x - 2 = 0$ serta rekam semua tebakan (LOW,MID,HIGH) dalam satu tabel. Perlihatkan pula bahwa nilai iterasi yang didapat tidak melebihi estimasi banyak iterasi pada persamaan (*)
3.

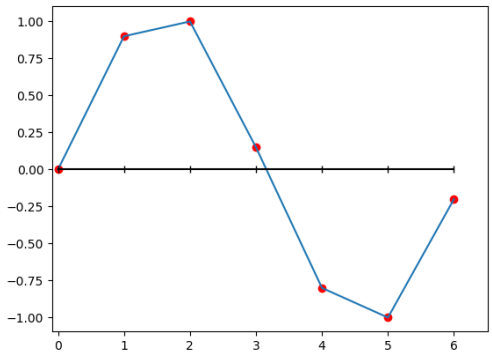
Chapter 2

Interpolasi

Dengan beberapa sampel data, kita bisa mengestimasi nilai titik-titik diantaranya. Salah satu caranya adalah dengan interpolasi. Interpolasi sendiri adalah menemukan fungsi yang tepat melewati titik-titik data yang diketahui. Ada beberapa interpolasi yang cukup terkenal yaitu: *piecewise-constant* , linear, polinomial, dan *cubic-spline*. Interpolasi dua dimensi dan tiga dimensi tidak akan dibahas di tulisan ini. Interpolasi yang pertama, *piecewise-constant* ini idenya adalah nilai yang akan diinterpolasi paling dekat ke titik mana, lalu nilai interpolasinya akan mengikuti nilai titik terdekat tersebut. Untuk interpolasi yang lain, akan dibahas pada sub bab berikut.

2.1 Interpolasi garis

Interpolasi linear adalah interpolasi yang menggunakan sarana garis lurus dengan melewati dua titik data. Pada interpolasi garis, semua titik diantara dua titik data didekati dengan linear atau dengan garis.



Meskipun ini adalah teknik interpolasi yang cukup mudah, namun teknik ini memiliki keunggulan yaitu kecepatan. Karena tidak mempertimbangkan titik data yang lain selain yang berada paling dekat dengan titik yang akan dicari nilai interpolasinya, hitungan menjadi sangat cepat. persamaan yang digunakan juga 'cuma' persamaan garis. Mencari nilai intervalnya akan memerlukan *cost* lebih besar daripada melakukan hitungan tebakannya.

Misalkan kita punya himpunan titik data $\{(x_i, y_i) \mid i = 1, 2, 3, \ldots, n\}$ dengan $x_i < x_j$ jika $i < j$. Misalkan titik yang ingin diinterpolasi nilainya adalah x_0 dan titik tersebut berada pada interval $[x_a, x_b]$, maka persamaan untuk mendapatkan nilai interpolasi y_0 adalah

$$y_0 = y_a + (x_0 - x_a)\frac{y_b - y_a}{x_b - x_a}.$$

Jika dicermati, sebenarnya ini hanyalah persamaan garis yang melalui (x_a, y_a) dan (x_b, y_b).

2.1.1 Algoritma

Algoritma untuk ini sebenarnya cukup mudah:

1. `Tentukan intervalnya.`
2. `Hitung nilainya berdasarkan persamaan garis`

Namun pada prakteknya, kita memerlukan pencarian interval yang memuat nilai yang akan

diinterpolasi. Di skrip Python berikut, metode yang digunakan untuk mencari interval adalah metode biseksi yang dimodifikasi.

[9]:
```
def carisegmen(xData,x0):
    '''mencari interval pada xData dimana x berada. Nilai yang␣
 ↪dihasilkan adalah batas kiri interval'''
    kiri = 0
    kanan = len(xData)- 1
    while True:
        if (kanan-kiri) <= 1:
            return kiri
        i =int((kiri + kanan)/2)
        if x0 < xData[i]: kanan = i
        else: kiri = i


def interpolasigaris(xData, yData, x0):
    '''menginterpolasi x berdasarkan xData'''
    x, y = xData, yData
    k = carisegmen(x, x0)
    y0 = y[k] + (x0-x[k])*(y[k+1] - y[k])/(x[k+1] - x[k])
    return(y0)
```

2.2 Interpolasi dengan Polinomial: Lagrange

Selain garis, hal natural yang digunakan untuk interpolasi adalah polinomial. Tentu saja kita selalu bisa menemukan polinomial yang melewati $n+1$ titik. Untuk menjamin keunikan dari polinomial ini, diambil polinomial berderajat n. Jika derajat lebih rendah dari n, belum tentu ada polinomial yang memenuhi; sebaliknya bila diambil derajat yang lebih tinggi dari n, kita

selalu bisa menemukan setidaknya dua polinomial yang melewati semua n titik tersebut*. Ada beberapa metode untuk mendapatkan polinomial unik tersebut. Berikut adalah cara yang digunakan Lagrange untuk mendapatkannya. Kita akan mulai dengan sebuah lemma yang cukup menarik.

Lemma. Misalkan $n \geq 1$. Terdapat suatu polinomial $\ell_k \in \mathcal{P}_n$ untuk $k = 0, 1, 2, ..., n$ sedemikian sehingga

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

untuk semua $i, k = 0, 1, 2, ..., n$.

Membuktikan eksistensi dari sesuatu seringkali tidak mudah. Namun, karena untuk kasus ini, eksistensi lebih mudah karena kita bisa melakukan konstruksi polinomial ℓ_k yang diinginkan.

Untuk kasus $n = 1$ kita bisa pilih ℓ_k dengan

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1}; \qquad \ell_1(x) = \frac{x - x_0}{x_1 - x_0}.$$

Melanjutkan pola yang sama, bisa dipilih ℓ_k dengan cara serupa

$$\ell_k(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_n)}$$

Dengan ide tersebut, kita bisa mulai berpikir untuk bisa mengubah 1 menjadi nilai yang kita inginkan di titik tertentu, namun bernilai 0 di titik lainnya. Diperoleh bahwa:

$$y_k \ell_k(x_i) = \begin{cases} y_i, & i = k \\ 0, & i \neq k \end{cases}$$

Selanjutnya, untuk dua bilangan bulat non negatif $p \neq q$, tinjau fungsi $f(x) = y_p\ell_p(x) + y_q\ell_q(x)$. Fungsi ini memiliki sifat $f(x_p) = y_p$ dan $f(x_q) = y_q$. Baik $y_p\ell_p$ maupun $y_q\ell_q$ tidak saling mempengaruhi di titik x_p dan x_q. Dengan ini, kita bisa memperluas untuk sebanyak mungkin titik data yang kita butuhkan, yaitu penjumlahan semua $y_k\ell_k$ untuk semua titik data.

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

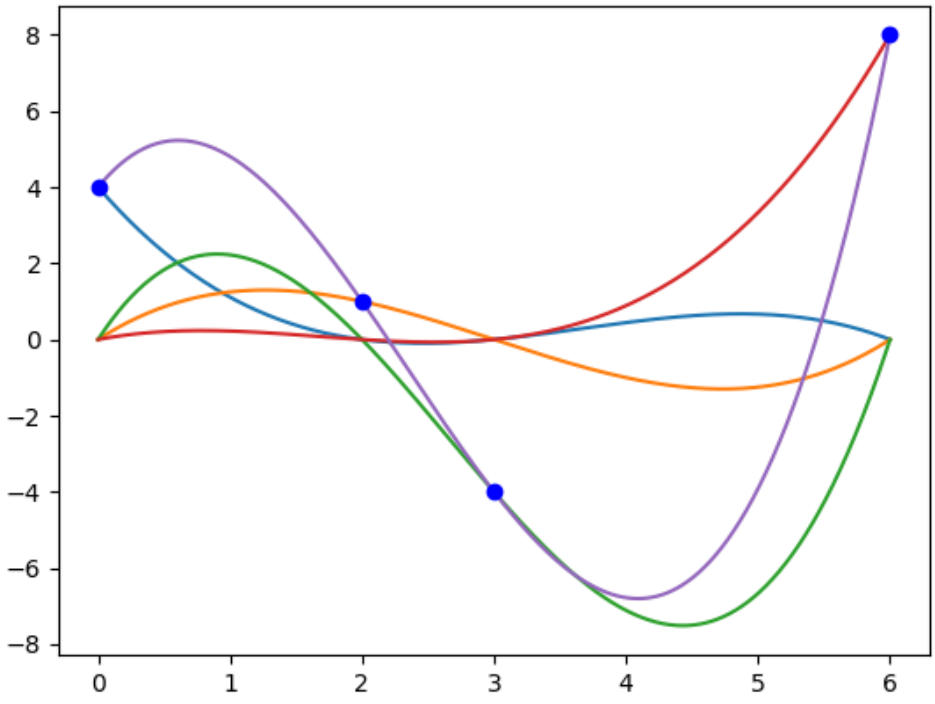
Contoh. Misalkan kita punya titik-titik data sebagai berikut $(0, 4), (2, 1), (3, -4), (6, 8)$. Menghitung masing-masing ℓ_k, diperoleh

$$\begin{aligned}
\ell_1 &= \left(\frac{x-2}{-2}\right)\left(\frac{x-3}{-3}\right)\left(\frac{x-6}{-6}\right) = -\frac{1}{36}x^3 + \frac{11}{36}x^2 - x + 1 \\
\ell_2 &= \left(\frac{x-0}{2-0}\right)\left(\frac{x-3}{2-3}\right)\left(\frac{x-6}{2-6}\right) = \frac{1}{8}x^3 - \frac{9}{8}x^2 + \frac{9}{4}x \\
\ell_3 &= \left(\frac{x-0}{3-0}\right)\left(\frac{x-2}{3-2}\right)\left(\frac{x-6}{3-6}\right) = -\frac{1}{9}x^3 + \frac{8}{9}x^2 - \frac{4}{3}x \\
\ell_4 &= \left(\frac{x-0}{6-0}\right)\left(\frac{x-2}{6-2}\right)\left(\frac{x-3}{6-3}\right) = \frac{1}{72}x^3 - \frac{5}{72}x^2 + \frac{1}{12}x.
\end{aligned}$$

Satu hal yang menarik dari contoh ini adalah penjumlahan $\sum_{k=1}^{n} \ell_k = 1$.

Sedangkan polinomial Lagrange yang didapatkan adalah

$$\sum_{k=1}^{n} y_k\ell_k = y_1l_1 + y_2l_2 + y_3l_3 + y_4l_4 = \frac{41}{72}x^3 - \frac{289}{72}x^2 + \frac{17}{4}x + 4$$



2.2.1 Algoritma

Berikut koding untuk mencari $\ell_k(x_0)$ dan $P(x_0)$ jika diberi sampel data.

[10]:
```
def Lk1(x, k, x0):
    ''' data disimpan di x, nilai yang dihitung x0 '''
    lj = 1
    for i in range(len(x)):
        if i == k:
            continue
        else:
            lj *= (x0 - x[i])/(x[k] - x[i])
    return(lj)
```

```
def lagrange1(x, y, x0):
    '''Interpolasi Lagrange dengan data input berupa matriks'''
    P = 0
    Lk = [Lk1(x, k, x0) for k in range(len(x))]
    for k in range(len(x)):
        P += y[k]*Lk[k]
    return(P)
```

Untuk interpolasi dengan menghasilkan koefisien polinomial, kita bisa menggunakan metode lagrange pada library Scipy dengan bantuan polinomoals dari library Numpy. Jika tidak, tentu Anda bisa mengembangkan kode Anda sendiri.

[18]:
```
import numpy as np
from scipy.interpolate import lagrange
from numpy.polynomial.polynomial import Polynomial

x = np.array([0,1,2])
y = np.array([1,3,9])
polinom = lagrange(x,y)
Polynomial(polinom).coef
```

[18]:
```
array([2., 0., 1.])
```

Pada contoh ini, outputnya `array([2., 0., 1.])`, yang bila diterjemahkan adalah $x^2 + 1$.

2.3 Interpolasi dengan Polinomial: Newton

Meskipun Lagrange memiliki algortima yang cukup mudah, ternyata cara ini tidak terlalu efektif karena melibatkan suku yang cukup banyak dan tidak ada suku yang berulang. Oleh

sebab itu Newton mengembangkan metode yang lebih efisien. Mengapa ini lebih efisien? sebab untuk menghitung polinom derajat n, cara ini melibatkan polinomial derajat $n-1$. Ini berbeda dengan cara interpolasi Lagrange yang langsung menghitung pada derajat ke n.

Polinomial interpolan $P_n \in \mathcal{P}_n$ ditulis dalam bentuk sebagai berikut.

$$\begin{aligned} P_n(x) =& a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_n) \\ =& a_0 + (x - x_0)\left(a_1 + (x - x_1)\left(a_2 + (x - x_2)\left(\cdots + a_n\left(x - x_n\right)\right)\right)\right) \end{aligned}$$

Dua bentuk diatas memiliki kelebihan masing-masing. Yang pertama untuk menghitung koefisien polinom lebih mudah, sedangkan bentuk kedua lebih cepat dalam menghitung nilai hasil interpolasi. Bentuk kedua bila ditulis ulang dalam bentuk rekursif, menjadi

$$P_k(x) = a_{n-k} + (x - x_{n-k})P_{k-1}$$

Cara ini akan efektif ketika kita tidak tahu ada berapa banyak data yang didapat. Di dunia nyata, ketika tiba-tiba mendapat data baru, kita bisa menambahkan data tersebut dalam interpolasi tanpa perlu mengulang interpolasi dari awal. Ini adalah salah satu kelebihan metode polinomial Newton dibanding metode Lagrange.

Berikutnya, kita perlu untuk mendapatkan koefisien dari polinom P_n. Untuk itu, kita akan mencocokkan polinom ini dengan data sehingga $P_n(x_i) = y_i$ untuk semua $i = 0, 1, 2, 3, ..., n$. Hal ini bisa ditulis dalam sistem persamaan sebagai berikut.

$$
\begin{aligned}
y_0 &= a_0 \\
y_1 &= a_0 + a_1(x_1 - x_0) \\
y_2 &= a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\
&\cdots \\
y_n &= a_0 + a_1(x_n - x_0) + a_2(x_n - x_0)(x_n - x_1) + \cdots + a_n(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})
\end{aligned}
$$

atau kalau ditulis dalam bentuk matriks menjadi

$$
\begin{pmatrix}
1 & 0 & \cdots & & 0 \\
1 & x_1 - x_0 & 0 & \cdots & 0 \\
1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & \vdots \\
\vdots & \vdots & & \ddots & \\
1 & x_n - x_0 & \cdots & \cdots & \prod_{j=0}^{k-1}(x_n - x_j)
\end{pmatrix}
\begin{pmatrix} a_0 \\ \\ \vdots \\ \\ a_n \end{pmatrix}
=
\begin{pmatrix} y_0 \\ \\ \vdots \\ \\ y_n \end{pmatrix}
$$

Menyelesaikan sistem persamaan ini cukup mudah sebab melibatkan matriks segitiga bawah. Dengan substitusi, kita bisa menyelesaikan persamaan ini secara iteratif. Langkah pertama kita dapatkan $a_0 = y_0$. Kedua, bisa didapat

$$a_1 = \frac{y_1 - a_0}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0}.$$

Berikutnya pada a_2 kita bisa mulai melihat pola.

$$a_2 = \frac{y_2 - a_0 - (x_2 - x_0)a_1}{(x_2 - x_0)(x_2 - x_1)} = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_1 - x_0}$$

Perhatikan diagram berikut.

$$
\begin{array}{llll}
x_0 & y_0 & & \\
 & & \frac{y_1-y_0}{x_1-x_0} & \\
x_1 & y_1 & & \frac{\frac{y_1-y_0}{x_1-x_0}-\frac{y_2-y_1}{x_2-x_1}}{x_2-x_0} \\
 & & \frac{y_2-y_1}{x_2-x_1} & \\
x_2 & y_2 & & \\
\end{array}
$$

Notasikan beda-pembagi $\nabla^k y_j$ sebagai berikut: untuk $k = 1$ notasikan $\nabla y_i = \frac{y_i - y_0}{x_i - x_0}$, $i = 1, 2, ..., n$ dan untuk $k > 1$ notasikan

$$\nabla^k y_i = \frac{\nabla^{k-1} y_i - \nabla^{k-1} y_{k-1}}{x_i - x_{k-1}}, \qquad i = k, k+1, ..., n.$$

Maka diagram di atas bisa ditulis ulang menjadi tabel berikut dengan solusi dari sistem persamaan yang dicari berada pada diagonal utama tabel.

x_0	y_0				
x_1	y_1	∇y_1			
x_2	y_2	∇y_2	$\nabla^2 y_2$		
$\vdots$				$\ddots$	
x_n	y_n	∇y_n	$\nabla^2 y_n$	$\cdots$	$\nabla^n y_n$

2.3.1 Algoritma

Untuk menyusun algortima untuk menyusun interpolasi ini, kita perlu dua step, mengambil koefisien polinom, lalu mengeksekusi nilainya. Berdasarkan persamaan 2.3 kita bisa menghitung hasil interpolasi secara rekursif. Unyuk mengambil koefisien polinom, kita bisa melakukannya dengan menghitung tabel 2.3.

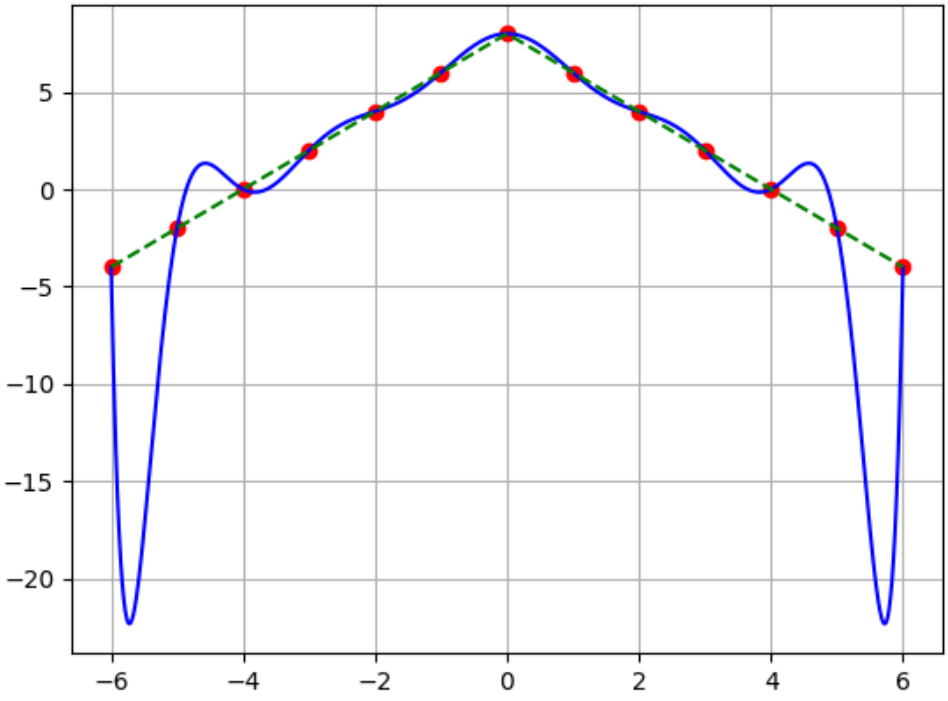
[5]:
```
def interpolasiNewtonPolinom(a,x):
    '''Menghitung P(x) dengan a adalah koefisien polinom
    x adalah data dan x0 adalah nilai yang akan diinterpolasi'''

    y = a[-1]
    for k in range(1,len(a)+1):
        y = a[-k] + (x0-x[-k])*y
    return(y)

def koefisienNewtonPolinom(x,y):
    '''x dan y adalah data yang akan diinterpolasi menjadi P(x)=y
    output fungsi ini adalah vektor berisi koefisien P'''
    m = len(x)
    a = y
    for k in range(1,len(x)):
        a[k:m] = (a[k:m] - a[k-1])/(x[k:m] - x[k-1])
    return(a)
```

Bandingkan hasil ini dengan koefisien yang diberikan oleh `numpy.lagrange`.

2.4 Interpolasi dengan Polinomial: Keterbatasan



Chapter 3

Diferensiasi Numerik

Diberikan fungsi f, tentukan $\dfrac{d^n f(x)}{dx^n}$.

Di dunia nyata, seringkali menghitung turunan diperlukan dalam rentang waktu yang singkat, meskipun harus mengorbankan akurasi. Ingat bahwa turunan pertama mewakili tren dan turunan kedua mewakili kecepatan tren berubah. Di kondisi yang lain, kita juga sering menemukan bahwa kita membutuhkan turunan, namun poin data yang dipunyai tidak banyak. Hanya titik per titik, sementara untuk melakukan *interpolasi* maupun *curve fitting* memakan *resource* yang lebih banyak.

Ini mengakibatkan dibutuhkannya suatu konsep untuk menghitung fungsi dengan cepat.

Ingat bahwa dari definisi turunan, kita punya

$$f'(x) = \lim_{x \to 0} \frac{f(x+h) - f(x)}{h}$$

Jika kita mengambil nilai h tertentu, kita akan dapatkan $f'(x)$ mendekati nilai pada ruas kanan (tanpa limit).

Sehingga kita bisa tulis ulang menjadi

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Beda hingga atau *finite difference* memiliki tiga metode.

- Beda maju
- Beda mundur
- Beda tengah

3.1 Beda Hingga

Dari persamaan sebelumnya $f'(x) \approx \frac{f(x+h)-f(x)}{h}$, kita bisa memilih nilai $h > 0$. Ini menjadi metode beda maju.

$$f'(x) = \frac{f(x+h) - f(x)}{h}.$$

Dari persamaan sebelumnya $f'(x) \approx \frac{f(x+h)-f(x)}{h}$, kita bisa memilih nilai $h < 0$. Ini menjadi metode beda maju. Jika kita pilih $k = -h$

$$f'(x) = \frac{f(x-k) - f(x)}{-k} = \frac{f(x) - f(x-k)}{k}$$

atau cukup ditulis

$$f'(x) = \frac{f(x) - f(x-k)}{k}.$$

Kalau kita ganti simbol k dengan h didapat metode beda mundur.

$$f'(x) = \frac{f(x) - f(x-h)}{h}.$$

Untuk beda tengah agak berbeda. Kalau beda maju adalah melakukan aproksimasi maju

satu step (satu h) dan beda mundur melakukan satu step ke belakang, kalau beda tengah ini melakukan aproksimasi dengan maju setengah step dan mundur setengah step. Ini didapat dengan menggabungkan beda maju dan beda mundur.

$$f'(x) = \frac{1}{2}\left(\frac{f(x+h) - f(x)}{h} + \frac{f(x) - f(x-h)}{h}\right) = \frac{f(x+h) - f(x-h)}{2h}$$

atau bila kita set $2h \to h$, maka $h \to \frac{1}{2}h$. Aproksimasi menjadi

$$f'(x) = \frac{f(x+\frac{1}{2}h) - f(x-\frac{1}{2}h)}{h}$$

Chapter 4

Appendix: Pengenalan Python

Bab ini disadur dari .

4.1 Getting Started

Surely we all heard about Python and all of the hype of data science and machine learning. But, sorry for dissapoint you, we are here wouldn't talk about it. At least for the this first episode of workshop series, we wouldn't get more than a glance of the hype. We are here to bring you to get into the world of open source, the era of you-can-googled-it-all.

For you to get here, I hope that you already have installed conda, or at least python (with pip), in your machine. If you aren't ready for that, please check the assistant in the back of the room, let them instal it for you. Through out the workshop, we will using Python 3.x not the old 2.x. Not to say that 2.x is bad, but rather to embrace the new things and let go the past.

First thing first, let us continue the tradition and let say hi to the world:

[1]:
```
print("Hello, World!")
```

```
Hello, World!
```

4.1.1 Editor

Open your laptop, open your Python. Choose any editor you would like to use. There are plenty numbers of editor out there. Here are my fav:

1. Notepad, Notepad++, if you are using Windows OS
2. gedit (gnome), kate, kwrite (kde), deepin editor (deepin), etc,
3. Python IDLE, if you are using windows and install python without install conda
4. Spyder (python library)
5. Jupyter and Jupyterhub (python librarty)
6. Sublime Text ()
7. Atom ()
8. any text editor

Just do not use a document editor such as Microsoft Word. It is doable, but rather inconvinience. You can directly using terminal if you need a quick glance of a syntax (could be cmd, powershell, or terminal). Just type "python" in your terminal, and you can simpy executed any python syntax in there after you saw `>>>` in the terminal rows.

A typical python file is with extension `.py`. You can execute any `.py` files with python simply by typing `python this.py` at your terminal (of course in appropriate folder). For those who didn't really familiar with terminal, just open your editor and run `this.py` at the your choice of text editor.

Last but not least, this workshop will try to engrave your heart and mind with one holy sentence:

PICTURE OF SIMPSON WRITE ON THE BLACKBOARD

4.1.2 Some Notes!

As previously said, we can print text with the syntax `print` such as:

```
print("Hello, World")!
```

This slightly different with Python 2.x where we could just type `print c` where `c` is the variable we want to print. In Python 3.x we using parentheses to print what's inside.

Indentation

In the world of Python, the indentation is very important. Unlike, in other programming languages where the indentation in code is for readability only, in Python the indentation was marked as continuation of the previous lines to indicate a block of code. Here is an example:

[2]:
```
if 5 > 2:
    print("Cakeep")
```

```
Cakeep
```

[3]:
```
if 5 > 2:
print("Salah ini")
```

```
  File "<ipython-input-3-61ebf1355001>", line 2
    print("Salah ini")
        ^
IndentationError: expected an indented block
```

You will find an error on the line 3 since the statement `if` is not complete. `PRO TIPS`: Do

not use a mixture of tabs and spaces for the indentation as it does not work across different platforms properly.

Commenting

You can add a comment simply by type # before a comment and the rest of the lines would be commented out.

[7]:
```
print("Jalan jalan ke pasar minggu,") # yang ini yang komen
# print("Cakeeep")
print("udah gitu doang.")
# print("weeew!")
```

```
Jalan jalan ke pasar minggu,
udah gitu doang.
```

Documentation

Python also has extended documentation capability, called docstrings. Docstrings can be one line, or multiline. Python uses triple quotes at the beginning and end of the docstring:

[9]:
```
"""Continuation of the tradition"""
print("Hello, World!")
```

```
Hello, World!
```

4.1.3 Before You Ask

But before you asking a technical question online, there are some things you should consider to do the following:

1. Try to find an answer by READING THE MANUAL!

2. Try to find an answer by **searching** the archives of the forum or mailing list you plan to post to.
3. Try to find an answer by **searching** the Web.
4. Try to find an answer by READING a FAQ.
5. Try to find an answer by inspection or experimentation.
6. Try to find an answer by asking a skilled friend.
7. If you're a programmer, try to find an answer by reading the source code.

When you ask your question, display the fact that you have done these things first; this will help establish that you're not being a lazy sponge and wasting people's time. Better yet, display what you have learned from doing these things. We like answering questions for people who have demonstrated they can learn from the answers.

4.1.4 References

Some references:

0. Python Documentation
1. Swaroop, C.H. A Byte of Python
2. Shaw, Zed. Learn Python the Hard Way.
3. W3Schools, old but gold.
4. Stackoverflow will help you to do some troubleshooting.
5. Duckduckgo is your true best friend. Google is not.

End of Notes

That's it! Now you ready to start ~~coding~~ Googling and learn Python!

4.2 The Basics

4.2.1 Literal Constants

The literal constants is: they are literally constants. The are as they are looked like. Numbers and strings are literal constants. 2 will always 2. You cannot change the meaning of 2 to be 10. The string `I am awesome` will always as it is, whether you (the one read it) awesome or not at all.

4.2.2 Strings and Numbers

Strings is just a data type. We indicate strings with one quotes (`'`), double quotes (`"`), or triple quotes (`'''` or `"""`). You always can escape from reality by using `\` symbols. Kinda looked like this:

[7]:
```
'They\'re never know what are we talked about.'
```

[7]:
```
"They're never know what are we talked about."
```

4.2.3 Numbers

As in your other preference programming class, number are treated as number. No less, no more. There are three distinct numeric types: integers, floating point numbers, and complex numbers. Once upon a time, there are two type of integer, plain integer and long integer. But that all changed ~~when the Fire Nation attacked~~ since this point onward, when the two types are merged. So you haven't worried about whether it's integer or long integers.

As you can now print something, do print some of these. No copy-paste allowed:

[2]:
```
print("Hello, World!")
print("I read this one.")
print("Duh, bentar lagi pengen copy-paste.")
```

```
print("Ini pasti udah nggak ada yang ngetik sama kek gini")
print("Itu yang males di ujung ngetik apaan woi?")
print("Jangan senyum-senyum sendiri, tar keterusan gila.")
print("Emang garing ini, tapi lu harus ngetik ini sampe beres gaboleh␣
 ↪copas, ga boleh ngeluh, ga boleh senyum, ga boleh ketawa, ga boleh skip␣
 ↪ini.")
```

```
Hello, World!
I read this one.
Duh, bentar lagi pengen copy-paste.
Ini pasti udah nggak ada yang ngetik sama kek gini
Itu yang males di ujung ngetik apaan woi?
Jangan senyum-senyum sendiri, tar keterusan gila.
Emang garing ini, tapi lu harus ngetik ini sampe beres gaboleh copas, ga␣
 ↪boleh
ngeluh, ga boleh senyum, ga boleh ketawa, ga boleh skip ini.
```

Now let's move to something else. Find two of your favorite numbers. Do some arithmetic operation on them, then print them out. On Python we have several arithmetic operation on Python by default. That is addition `+`, substraction `-`, multiplication `x`, division `/`, division with remainder(modulo) `%`, division with floor `//`, etc.

[6]:
```
print(1+2)
print(3*(4+5j))
print((1+1j)/(1-1j))
```

```
3
(12+15j)
1j
```

4.2.4 Variables

Somehow, we need some way of storing any information and manipulate them as well. This is where variables come into the picture. In mathematics, we also need some varible to labeled as an unknown. Variables are exactly what they mean - their value can vary i.e. you can store anything using a variable. In our case, variables are just parts of your computer's memory where you store some information. Unlike constants, you need some method of accessing these variables and hence you give them names.

4.2.5 Identifier Naming

Variables are examples of identifiers. Identifiers are names given to identify something. There are some rules you have to follow for naming identifiers:

- the first character should be a letter of alphabet or an underscore (`_`)
- the rest of the identifier should be a letter, a number, or an underscore
- identifier are case sensitive

For example, `_ini_`, `anu`, `iTu`, and `iT4` are valid identifier, but `this one`, `7hisOne` or `this-one` are not.

4.2.6 Object

We will skip this one for another workshop :p

4.2.7 Logical and Physical Lines

A physical line is what you see when you write the program. A logical line is what Python sees as a single statement. Python implicitly assumes that each physical line corresponds to a logical line. Use more than one physical line for a single logical line only if the logical line is really long. If you want to write a code but unfortunately your code is too long to be on one line, you just use backslash `/` to make a new physical lines (NOT a logical line).

4.2.8 Exercise

Do some exercise. Please do some research on your own before asking.

PICTURE OF SIMPSON WRITE ON THE BLACKBOARD

PROBLEM Print these sentences on your terminal. Try it with escape and without escape.

1. `My instructor are awesome.`
2. `You're welcome, apprentice.`

PROBLEM Can I make a variable like this: `13 = 'Bad Numbers'`? Why?

PROBLEM Assign two variables with some strings, i.e. `x=Department` and `y=Mathematics`. Then print `x+y`. Do the same thing with numbers and string of numbers. Could you spot the difference?. Let `z="."`. Try print `y+x+10*z`.

PROBLEM Just type this one on your favorite terminal or editor.

[13]:
```
name = "John"
age = 21
cars = "Nissan GTR"
years = 2018


print("My name is %s" % name)
print("Now, I am %d years old" % age)
print("I ride a %s %d" % (cars, years))
```

```
My name is John
Now, I am 21 years old
I ride a Nissan GTR 2018
```

4.3 Control Flow

One might want to execute program with some order in mind, with some flow that different than just do some series of statements as we did in previous chapter. Like many other languages, Python have the ability to do so. Namely, `if`, `while`, `for`, `break`, and `continue`. We may not cover all of these details right now, but we will try to cover the rough of the nature of these built in function.

4.3.1 If

Here are some codes from *A Byte of Python* with modification. The function `input()` is just a built-in function which prints it to the screen and waits for keyboard input from the user. Python 2.x might have it different way to do this.

[4]:
```
#!/usr/bin/python
# Filename: if.py
number = 23
guess = int(input('Enter an integer : '))
if guess == number:
    print('Congratulations, you guessed it.') # New block starts here
    print("(but you do not win any prizes!)") # New block ends here
elif guess < number:
    print('No, it is a little higher than that.') # Another block
    # You can do whatever you want in a block ...
else:
```

```
    print('No, it is a little lower than that.')
    # you must have guess > number to reach here
    print('Done')
    # This last statement is always executed, after the if statement is␣
↪executed
```

```
Enter an integer : 32
No, it is a little lower than that.
Done
```

The main blocks of this `if` are

```
if statement:
    some command
elif other-statement:
    some other command
else:
    another command
```

Basically the statement after `if` are the primary condition so that the following command could run. The `elif` are the second priority after the `if` or after another `elif` above this one. `else` is the lowest priority. The very minimal requirement of this function are one statment and command. For example:

[8]:
```
if 10 == 9+1:
    print(1.2 - 1.0)
```

```
0.19999999999999996
```

4.3.2 While

Another codes from the same book previously mentioned, *A Byte of Python*, with some modification for compability. `True` and `False` are also literal constants. You cannot doin anything with them.

[14]:
```
#!/usr/bin/python
# Filename: while.py
number = 23
running = True

while running:
    guess = int(input('Enter an integer : '))
    if guess == number:
        print('Congratulations, you guessed it.')
        running = False # this causes the while loop to stop
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
# Do anything else you want to do here
print('Done')
```

```
Enter an integer : 5
No, it is a little higher than that.
Enter an integer : 7
No, it is a little higher than that.
```

```
Enter an integer : 19
No, it is a little higher than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

While `if` is only work when the condition fulfilled, the while is almost the opposite: `while` automatically work and just stop when the condition is breach. Unlike some other programs, in the Python, `while` could be followed by `else`.

4.3.3 For

The `for` function on Python is one of the most sophisticated. You can do `for` followed by `else`. Not only that, you can almost literally looping anything with `for`. For example:

[17]:
```
for i in range(1,5):
    print(i)
else:
    print("printed with range")


fishes = ("teri", "kakap", "tongkol")
for fish in fishes:
    print("ini ikan"+fish)
    print("yang ini dapet sepeda")


for love in "i love you":
```

```
    print(love)
else:
    print("no matter what")
```

```
1
2
3
4
printed with range
ini ikanteri
yang ini dapet sepeda
ini ikankakap
yang ini dapet sepeda
ini ikantongkol
yang ini dapet sepeda
i

l
o
v
e

y
o
u
no matter what
```

Have same property as `if` and `while`, the block after the line with `for iterator in`

`iteration:` are the command that would run iteratively. And here, you see again that this `if` has `else` following it behind.

4.3.4 Break and Continue

Break and Continue are quite the opposite but not really an opposite at the same time. Here's an example.

[ ]:
```
while True:
    s = input('isi dong:')
    if s == 'quit' :
        break
    print('panjang string adalah', len(s))

    if len(s) < 3:
        continue
    print('Input is of sufficient length')
print('Done.')
```

```
isi dong:quite
panjang string adalah 5
Input is of sufficient length
isi dong:12
panjang string adalah 2
```

While `break` really break the flow, the `continue` is just like `pass` when you playing a card, the flow still, but you didn't get your turn.

4.3.5 Exercise

PROBLEM Make a list of the month of the year. Skipped in your month of your birthday. Print it out.

4.4 Function and Modules

4.4.1 Function

Function are block of statemenmt that given a name. You can call it anywhere in your program. Function started with `def`.

[ ]:
```
def Hello():
    print('Hello, World!') # this block is the block that given a name.


# You can end a function just like that.


Hello()
```

4.4.2 Function Parameter

Function may have parameter as an input and return as an output. It may looked like this

[ ]:
```
def small(x,y):
    if x == y:
        print('cek ulang gih')
    if x < y:
        return(x)
    else:
        return(y)
```

```
small(4,4)
```

In this case, `x` and `y` are the parameters, the return gave an output a value. For example, if you call the function `z = small(1,2)`, then after execute it, you will have the value `z` as 1.

If you have some functions with many parameters and you want to specify only some of them, then you can give values for such parameters by naming them - this is called keyword arguments - we use the name (keyword) instead of the position (which we have been using all along) to specify the arguments to the function.

[ ]:
```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

4.4.3 DocStrings

As mentione before, Python has a nifty feature called documentation strings which is usually referred to by its shorter name docstrings. DocStrings are an important tool that you should make use of since it helps to document the Functions program better and makes it more easy to understand. Amazingly, we can even get back the docstring from, say a function, when the program is actually running!

[ ]:
```
#!/usr/bin/python
# Filename: func_doc.py
def max(x, y):
        '''Prints the maximum of two numbers.
The two values must be integers.'''
```

```
        x = int(x) # convert to integers, if possible
        y = int(y)
        if x > y:
            print(x, 'is maximum')
        else:
            print(y, 'is maximum')

max(3, 5)
print(max.__doc__)
```

4.4.4 Modules

You see that function can be called anywhere anytime. What if your function is in another files? The answers is modules. To reuse module in other programs, the filename must be in `.py` extension. Module can be imported by another program to reuse what's inside. If you want to import and use all of the feature inside the module, you can just directly `import themodule`. If you just want to use some of it, then you can do `from somemodule import somefunction`. The module itself might be run independently apart for imported into another function.

Here is some example:

[ ]:
```
#!/usr/bin/python
# Filename: using_name.py

if __name__ == '__main__':
    print('This program is being run by itself')
else:
```

```
    print('I am being imported from another module')
```

Save that program from editor. And do some run with these two method. Feel the different when you called it with

```
$python using_name.py$python
>>>import using_name
```

Anyway, that wasn't really a necesarry to using `__main__` when your program is not too long. You probably won't need this in the next three days.

[ ]:
```
#!/usr/bin/python
# Filename: mymodule.py
def sayhi():
    print('Hi, this is mymodule speaking.')
    version = '0.1'
# End of mymodule.py
```

You can call the modules with `import` or `from ... import`

[ ]:
```
import mymodule
mymodule.sayhi()
print('Version', mymodule.version)
```

or if you only want to get the version, you might consider do the following

[ ]:
```
#!/usr/bin/python
# Filename: mymodule_demo2.py
from mymodule import version
print('Version', version)
```

4.4.5 Exercise

PROBLEM Try make a module that contain a mathematical function of two variables that give return the value of the volume and the surface of a cone, given its radii and height.

4.5 Data Structures

Data structures are structures which can hold some data together. In other words, they are used to store a collection of related data. There are three built in data structures in Python - list, tuple and dictionary.

4.5.1 List

List is an ordered data or collection of item. They are enclosed with brackets. You mau add or remove or search items from, to, or in the list.

[37]:
```
import sys
fishes = ["teri", "kakap", "tongkol"]
items = (" sword"," halberd"," axe"," knife"," blade")

for fish in fishes:
    print("ini ikan "+fish)

for item in items:
    print(item)

print(fishes[0])
print(items[-1])
del(fishes[-1])
```

```
for fish in fishes:
    print("ini ikan "+fish)
```

```
ini ikan teri
ini ikan kakap
ini ikan tongkol
 sword
 halberd
 axe
 knife
 blade
teri
 blade
ini ikan teri
ini ikan kakap
```

4.5.2 Tuple

Tuple is just like list, but one cannot modify typles. So you must be careful when defining a tuple. You can define tuple with parenthessis (in contrast of list that used brackets).

[14]:
```
name = "John"
age = 21
cars = "Nissan GTR"
years = 2018

print("My name is %s" % name)
```

```
print("Now, I am %d years old" % age)
print("I ride a %s %d" % (cars, years))
```

```
My name is John
Now, I am 21 years old
I ride a Nissan GTR 2018
```

4.5.3 Sequences

The sequences such as strings, list, and tuples are special. You may have indexing operation to fetch a data from it and you also could take "some slice" of it.

[35]:
```
pii = "3.14159265358979323846264 3"
fishes = ("teri", "kakap", "tongkol")

# Get a data
print(pii[0])
print(pii[-2])
print(pii[-3])
print(fishes[2])

# Slice
print(pii[10:15])
print(fishes[1:2])
```

```
3
4
6
tongkol
```

```
35897
```

```
('kakap',)
```

4.5.4 Dictionary

A dictionary is like an data base where you can find some data based only a "key". As in database, a key must be unique. While the previous type using brackets and parenthesis, the dictionary using a curly brackets. The following example taken from *A Byte of Python*.

[42]:
```
#!/usr/bin/python
# Filename: using_dict.py
# 'AB' is short for 'A'ddress'B'ook
AB = {          'Swaroop'   : 'swaroopch@byteofpython.info',
'Larry'     : 'larry@wall.org',
'Matsumoto' : 'matz@ruby-lang.org',
'Spammer'   : 'spammer@hotmail.com'
}
print("Swaroop's address is %s" % AB['Swaroop'])
# Adding a key/value pair
AB['Guido'] = 'guido@python.org'
# Deleting a key/value pair
del AB['Spammer']

print('\nThere are %d contacts in the address-book\n' % len(AB))

for name, address in AB.items():
    print('Contact %s at %s' % (name, address))
```

```
if 'Guido' in AB: # OR AB.has_key('Guido')
    print("\nGuido's address is %s" % AB['Guido'])
```

```
Swaroop's address is swaroopch@byteofpython.info


There are 4 contacts in the address-book


Contact Swaroop at swaroopch@byteofpython.info
Contact Larry at larry@wall.org
Contact Matsumoto at matz@ruby-lang.org
Contact Guido at guido@python.org


Guido's address is guido@python.org
```

With it yo can see that if we can identify the name, we can get the other content of the fields. Now look at the following example (*Learn Python the Hardway*). We can also adding another stuff into our dictionary by `dictionary['key']='value'`.

Also read this following example for your train of thought before taking exercise in the next chapter.

[1]:
```
cities = {'CA':'San Francisco','MI':'Detroit','FL':'Jacksonville'}
cities['NY'] = 'New York'
cities['OR'] = 'Portland'



def find_city(themap, state):
    if state in themap:
        return themap[state]
```

```
    else:
        return "Not found."


cities['_find'] = find_city


while True:
    print("State? (ENTER to quit)")
    state = input("> ")


    if not state:
        break


    # this line is the most important ever! study!
    city_found = cities['_find'](cities, state)


    print(city_found)
```

```
State? (ENTER to quit)
> CA
San Francisco
State? (ENTER to quit)
> MI
Detroit
State? (ENTER to quit)
> FL
Jacksonville
State? (ENTER to quit)
```

```
> NY
New York
State? (ENTER to quit)
> OR
Portland
State? (ENTER to quit)
>
```

Scipy and Numpy

Scipy builds on Numpy, and for all basic array handling needs you can use Numpy functions:

[ ]:
```
import numpy as np
```

If you need to using only some function, you may use the `from ... import` command.

[ ]:
```
from scipy import some_module
some_module.some_function()
```

This whole stuff with numpy is required a lot of practice since it's a big library itself. In this session we only cover some of it.

4.5.5 Array

You can make an array right of the blue with `np.array`.

[ ]:
```
import numpy as np


a = np.array([[1,2,3],[2,3,4]])
print(a)
print(a[0,1])
```

As usual, you could access the coordinates with the brackets. Remember that Python start

it's counting at 0. For better or worse, Numpy have a different data type than pure Python. We can access the dimension of the matrix with `a.shape` if `a` is the name of our matriks. It could also reshape with `a.reshape(m,n)` where `(m,n)` is the new dimension (NumPy can have more than 2 dimensional being). There's also `arange` to create an array of evenly spaced numbers. You also could transpose the matrix simply by `transpose`.

[ ]:
```
import numpy as np


a = np.array([[1,2,3],[2,3,4]])
b = np.arange(1,7)
c = np.arange(6)
b = b+c
b = b.reshape(2,3)


print(a+b)
```

Numpy has an ability to *broadcast* matrix, that is to treat arrays of different shapes during arithmetic operations. If the arrays are in the same shape, we did the binary operation coordinat-by-coordinate. But operations on arrays of non-similar shapes is still possible in NumPy.

[ ]:
```
a = np.array([1,2,3])
b = np.array([[10,20,30],[20,30,40],[30,40,50]])
a+b
```

4.5.6 Iteration on Array

You can also iterating in array using `nditer`. We can compare it if we reshape it first, then iterating as iteration on list, or we iterating directly with nditer.

[ ]:
```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print('Original array is:')
print(a)
print('\n')

print('Transpose of the original array is:')
b = a.T
print(b)
print('\n')

print('Sorted in C-style order:')
c = b.copy(order = 'C')
print(c)
for x in np.nditer(c):
   print(x),

print('\n')

print('Sorted in F-style order:')
c = b.copy(order = 'F')
print(c)
for x in np.nditer(c):
   print(x),
```

4.5.7 Arithmetic Operations

NumPy is not failed our expectation with linear algebra. Beside binary operation, NumPy also support the "usual" arithmetic operation on matrix, namely: `np.add`,`np.subtract`,`np.multiply`.

[1]:
```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)
b = np.arange(9, dtype = np.float_).reshape(3,3)
print(a)
print(np.add(a,b))
print(np.subtract(a,b))
print(np.multiply(a,b))
```

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
[[ 0.  2.  4.]
 [ 6.  8. 10.]
 [12. 14. 16.]]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[ 0.  1.  4.]
 [ 9. 16. 25.]
 [36. 49. 64.]]
```