بخش چهارم: برنامه کامپیوتری محاسبهٔ ظرفیت خازنی

۴-۱. انتخاب زبان کامپیوتری مناسب

برای انجام محاسبات و نوشتن کدهای لازم ابتدا باید زبان کامپیوتری مناسب انتخاب می‌شد. اگـر بین زبان‌هـای رایج و نسبتاً پرطرفدار انتخاب صورت گیرد، یکی از چهار زبان زیر را باید انتخاب کرد.

۱- fortran

۲- matlab

۳- c++/c

۴- python

معمولاً کارهای علمی در یکی از چهار زبان بالا صورت می‌گیرد. از بین این چهار زبان زبان‌هـای فرتـرن و سـی، برنامه را به صـورت کامـل ترجمـه (compile) می‌کننـد. دو زبـان پـایتون و متلب مفسـر (interpreter) هسـتند. در حالت کلی مفسرها سرعت پایین‌تری نسبت به مترجم‌ها دارند. اما هر دو زبـان پـایتون و متلب از توابـع آمـاده‌ای اسـتفاده می‌کنند که قبلاً ترجمه شده‌اند. اگر به درستی از این توابع استفاده شود، سرعت اجرای برنامه در آن‌ها نیز بالا می‌رود.

در نهایت با توجه به اینکه تفاوت چندانی میـان زبان‌هـای نـامبرده شـده از لحـاظ اجـرا نمی‌باشـد، انتخـاب زبـان مناسب بستگی به عوامل دیگری همچون تسلّط برنامه‌نویس، خوانا و قابل فهم بودن کد نوشته شده برای دیگران، امکـان توسعهٔ کد و در نهایت سلیقهٔ برنامه‌نویس دارد. برای نوشتن برنامهٔ محاسبهٔ ظـرفیت خـازنی از زبـان پـایتون و از کتابخانـهٔ numpy استفاده شد. در این فصل ابتـدا در مـورد کتابخانـهٔ numpy توضـیحاتی داده می‌شـود و سـپس قسـمت‌های مختلف برنامهٔ نوشته شده شرح داده می‌شود.

۴-۲. کتابخانهٔ numpy

نام‌پای(numpy)ـ یک کتابخانه برای زبان پایتون است. در نام‌پای می‌توانیم با آرایه‌های عددی مانند متغیرهای معمولی کار کنیم. توابع بسیاری برای کار با آرایه‌ها در نام‌پای وجود دارند. می‌توان گفت که نام‌پای کتابخانهٔ اصلی پایتون برای محاسبات عددی و برنامه‌های علمی است. از این کتابخانه در بسیاری از کتابخانه‌های دیگر در زمینه‌های پردازش سیگنال و هوش مصنوعی استفاده می‌شود. به عبارت دیگر پایهٔ محاسبات برداری و ماتریسی در زبان پایتون کتابخانهٔ numpy است.

به عنوان مثال اگر بخواهیم یک آرایه را در نام‌پای تعریف کنیم از دستور زیر استفاده می‌کنیم.

```
import numpy as np
x = np.array([1, 2, 3])
print (x)
...
[1,2,3]
```

و یا برای درست کردن یک آرایهٔ ۱۰ تایی می‌توانیم برنامهٔ زیر را بنویسیم.

```
import numpy as np
y = np.arange(10)
print (y)
...
[0 1 2 3 4 5 6 7 8 9]
```

برای اینکه مقادیر یک آرایه را دو برابر کنیم، دو راه داریم. در روش اول با ایجاد یک حلقه تک تک مقادیر آرایه را دو برابر می‌کنیم. همان کاری که برای مقادیر یک لیست (list) در زبان پایتون می‌توانیم انجام دهیم.

```
import numpy as np
y = np.arange(10)
print (y)
for i in range(10):
 y[i] *= 2
print (y)
...
[0 1 2 3 4 5 6 7 8 9]
[ 0  2  4  6  8 10 12 14 16 18]
```

روش دیگر استفاده از قابلیت نام‌پای در انجام عملیات ماتریسی بدون استفاده از حلقه است. خیلی ساده برنامه زیر را می‌نویسیم.

```
import numpy as np
y = np.arange(10)
print (y)
y *= 2
print (y)
...
[0 1 2 3 4 5 6 7 8 9]
[ 0  2  4  6  8 10 12 14 16 18]
```

روش دوم مهم‌تر از آنکه ظاهر ساده‌تری دارد، سرعت اجرای بسیار بالاتری نیز دارد. همانطور که گفتیم پایتون یک زبان مفسر(interpretter) است. یعنی برخلاف زبان‌های مترجم (compiler) خط به خط برنامه را ترجمه و اجرا می‌کند. حال اگر در برنامه از دستورات حلقه استفاده کنیم، دستوراتی که در حلقه تکرار می‌شوند، به ازاء هر بار تکرار یک بار هم ترجمه(compile) می‌شوند و این موضوع باعث اتلاف زمان بسیاری در اجرای برنامه می‌شود. البته در مفسرهای هوشمند جدید، قبل از اجرای برنامه، تا حدی ترجمه صورت می‌گیرد. به عبارت دیگر نمی‌توان به زبان‌های پایتون و جاوا و زبان‌های مشابه به راحتی مفسر صرف اتلاق کرد. این زبان‌ها تا آنجا که بتوانند در شروع کار اجرا، برنامه را ترجمه می‌کنند و قسمتی را که نمی‌توانند در زمان اجرا ترجمه می‌کنند. اما با این در نهایت باز هم سرعتشان از زبان‌هایی که مترجم هستند، کمتر است.

اما کاری که نام‌پای می‌کند استفاده از توابعی است که قبلاً ترجمه شده‌اند. وقتی در نام‌پای خط زیر را می‌نویسیم درواقع یک تابع را صدا می‌زنیم.

```
y *= 2
```

این تابع تک تک مقادیر موجود در آرایهٔ y را در دو ضرب می‌کند. البته این تابع به زبان سی نوشته شده است و در پایتون صرفاً فراخوانی می‌شود. بنابراین بسیار به سرعت اجرا می‌شود. البته می‌توانیم به جای آنکه دستور بالا را به صورت عملگری بنویسیم، مستقیماً تابع ضرب را فراخوانی کنیم.

```
y = np.multiply(y, 2)
```

تقریباً تمام توابع ریاضی که در کتابخانه‌های دیگر وجود دارند در کتابخانهٔ نام‌پای هم موجود هستند و این توابع می‌توانند بدون اینکه نیاز به استفاده از دستورات حلقه باشد، بر روی یک آرایه اعمال شوند. توابع مثلثاتی، توابع خاص،

توابع اعداد اتّفاقی و توابع مربوط به نظریهٔ اعداد همگی در نام‌پای موجود هستند. مثلاً برای اینکه سینوس مقادیر یک آرایه را حساب کنیم. به راحتی برنامهٔ زیر را می‌نویسیم.

```
import numpy as np
x = np.linspace(0, np.pi / 2, 5)
print (x)
y = np.sin (x)
print (y)
…
[0.         0.39269908 0.78539816 1.17809725 1.57079633]
[0.         0.38268343 0.70710678 0.92387953 1.        ]
```

در این برنامه «linspace» آرایه‌ای به تعداد عناصر معلوم با ابتدا و انتهای مشخص درست می‌کند. با کمک این تابع، ۵ مقدار مختلف برای x در فاصلهٔ صفر تا $\frac{\pi}{2}$ ایجاد می‌کنیم. تابع «np.pi» هم مقدار عدد π را حساب می‌کند. و دستور «np.sin» مقدار عددی سینوس تمامی مقادیر آرایهٔ x را محاسبه می‌کند.

کتابخانهٔ نام‌پای این توانایی را دارد که توابع پیچیده‌تری را بر روی آرایه‌ها انجام دهد. نام‌پای می‌تواند یک تابع را برای قسمتی از یک آرایه اعمال کند. مثلاً اگر بخواهیم که در یک آرایهٔ ده‌تایی ۵ عنصر اول را دو برابر کنیم، برنامهٔ زیر را می‌نویسیم.

```
import numpy as np
y = np.arange(10)
print (y)
y[:5] *= 2
print (y)
...
[0 1 2 3 4 5 6 7 8 9]
[0 2 4 6 8 5 6 7 8 9]
```

قابلیت دیگر نام‌پای امکان استفاده از توابع مشروط است. فرض کنید که می‌خواهیم در یک آرایهٔ ده‌تایی مقادیر اعداد زوج را بر دو تقسیم نماییم، این برنامه به این صورت در می‌آید.

```
import numpy as np
y = np.arange(10)
print (y)
y[y%2 == 0] //= 2
print (y)
...
[0 1 2 3 4 5 6 7 8 9]
[0 1 1 3 2 5 3 7 4 9]
```

در این برنامه عبارت «y%2== 0» به معنای مقادیری از آرایهٔ y است که باقی‌ماندهٔ آن‌ها بر دو برابر صفر است. و عملگر «//» برای تقسیم اعداد صحیح به کار می‌رود.

برای تسلط بیشتر به کتابخانهٔ نام‌پای می‌توان به کتاب‌ها و منابع مختلفی که در شبکهٔ اینترنت وجود دارند مراجعه نمود. در اینجا صرفاً برای آشنایی اجمالی با قابلیت‌ها و توانایی‌های این کتابخانه مطالبی به اختصار گفته شد. در هر صورت برای درک کامل توابع نوشته شده در برنامهٔ محاسبهٔ ظرفیت خازنی و استفاده از آن‌ها

۴ - ۳. نرم‌افزار محاسبه ضریب جفت شدگی

```
    #Coupling Coeffitien
    #Coupling Coeffitient
    from scipy.special import erf
    import numpy as np
    k0 = 9e9
    e0 = 1 / (4 * np.pi * k0)
    def Iss(x1, x2, y1, y2, z):
     x = np.abs(x1 - x2)
     y = np.abs(y1 - y2)
     z = np.abs(z)
     I1 = 1 / 12 * (- x**2 - y**2 + 2 * z**2) *
( x**2 + y**2 + z**2) ** (1/2)
    I2 =  1 / 4 * (y * (x**2 - z**2) ) *
np.arcsinh(y / np.hypot(x, z) )
    I3 =  1 / 4 * (x * (y**2 - z**2) ) *
np.arcsinh(x / np.hypot(y, z) )
    I4 = -1 / 2 * x * y * z * np.arctan(x * y /(z *
np.sqrt(x**2 + y**2 + z**2) ) )
    II = np.sqrt(np.pi) * ( I1 + I2 + I3 + I4 )
    return II
    def Iss_coplanar(x1, x2, y1, y2):
     x = np.abs(x1 - x2)
     y = np.abs(y1 - y2)
     I1 = 1 / 12 * (- x**2 - y**2) * ( x**2 + y**2)
** (1/2)
     x0 = (x == 0)
     y0 = (y == 0)
```

```
    xn = np.logical_not(x0)
    yn = np.logical_not(y0)
    I2 = np.zeros_like (x)
    I3 = np.zeros_like (x)
    I2[x0] = 0
    I2[xn] =  1 / 4 * ( y[xn] * x[xn]**2 ) *
np.arcsinh( y[xn] / x[xn] )
    I3[y0] = 0
    I3[yn] =  1 / 4 * ( x[yn] * y[yn]**2 ) *
np.arcsinh( x[yn] / y[yn] )
    II = np.sqrt(np.pi) * ( I1 + I2 + I3 )
    return II
   def ISum(Limits, z):
    s = 0
    for i in [0, 1]:
        for j in [0, 1]:
            for k in [0, 1]:
                for l in [0, 1]:
                    if (i + j + k + l) % 2 == 0:
                        A = 1
                    else:
                        A = -1
                    s += A * Iss(Limits[0][i],
Limits[1][j], Limits[2][k], Limits[3][l], z)
    return s

   def ISum_coplanar(Limits):
    s = 0
    for i in [0, 1]:
        for j in [0, 1]:
            for k in [0, 1]:
                for l in [0, 1]:
                    if (i + j + k + l) % 2 == 0:
                        A = 1
                    else:
                        A = -1
                    s += A * Iss_coplanar(Limits[0]
[i], Limits[1][j], Limits[2][k], Limits[3][l])
```

```
    return s

   def saeed(Lx ,Ly,x0,y0,z):
    x1L = [x0-Lx/2, x0+Lx/2]
    x2L = [-Lx/2, +Lx/2]
    y1L = [y0-Ly/2, y0+Ly/2]
    y2L = [-Ly/2, +Ly/2]
    Limits = [x1L, x2L, y1L, y2L]
    return ISum(Limits, z) * k0 / (Lx*Ly)**2 *2
/np.sqrt(np.pi)
   def saeed_coplanar(Lx ,Ly,x0,y0):
    x1L = [x0-Lx/2, x0+Lx/2]
    x2L = [-Lx/2, +Lx/2]
    y1L = [y0-Ly/2, y0+Ly/2]
    y2L = [-Ly/2, +Ly/2]
    Limits = [x1L, x2L, y1L, y2L]
    return ISum_coplanar(Limits) * k0 / (Lx*Ly)**2
*2 /np.sqrt(np.pi)
   def saeed_sc(x ,y):
    II = 1 / 3 * (x**3 + y**3) + 1 / 3 * (-x**2 -
y**2)*np.hypot(x, y) + \
         y * x**2 * np.arcsinh (y / x) + x * y**2 *
np.arcsinh (x / y)
    return II * k0 / (x*y)**2 *2
   def hitoshi_i(x ,y, d):
    r = np.sqrt(d*d + x*x +y * y)
    I1 = -d * np.arctan(x * y /(d * r))
    I2 = y * np.log(x + r)
    I3 = x * np.log(y + r)
    return I1 + I2 + I3
   def hitoshi_coplanar_i(x ,y):
    r = np.hypot(x, y)
    I = x * np.log(r + y) + y * np.log(r + x)
    return I
   def hitoshi(Lx ,Ly,x0,y0,z):
    I = hitoshi_i(x0+Lx/2, y0+Ly/2, z) -
hitoshi_i(x0+Lx/2, y0-Ly/2, z)
```

```
    J = hitoshi_i(x0-Lx/2, y0-Ly/2, z) -
hitoshi_i(x0-Lx/2, y0+Ly/2, z)
    return (I+J)/(Lx*Ly) * k0
   def hitoshi_coplanar(Lx ,Ly,x0,y0):
    I = hitoshi_coplanar_i(x0+Lx/2, y0+Ly/2) -
hitoshi_coplanar_i(x0+Lx/2, y0-Ly/2)
    J = hitoshi_coplanar_i(x0-Lx/2, y0-Ly/2) -
hitoshi_coplanar_i(x0-Lx/2, y0+Ly/2)
    return (I+J)/(Lx*Ly)*k0
   def orion(Lx, Ly):
    II =  1/Lx * np.arcsinh(Lx / Ly) + 1/Ly *
np.arcsinh(Ly / Lx) + \
    (Lx / Ly**2 + Ly / Lx**2 - (1 / Lx**2 + 1 /
Ly**2) * np.hypot(Lx,Ly) )/3
    return II * 2 * k0
   def zho(Lx ,Ly,x0,y0,z):
    return 1/np.sqrt(x0**2 + y0**2 + z**2)*k0
   def zho_coplanar(Lx ,Ly,x0,y0):
    return 1/np.sqrt(x0**2 + y0**2)*k0
   if __name__ == "__main__":
    d = 0
    Lx = 1
    Ly = 1
    x1L = [d-Lx/2, d+Lx/2]
    x2L = [-Lx/2, +Lx/2]
    y1L = [-Ly/2, +Ly/2]
    y2L = [-Ly/2, +Ly/2]
    Limits = [x1L, x2L, y1L, y2L]
    z = 1e-9
    Io = orion(Lx, Ly)
    print ('orion %e' %Io)
    Isc = saeed_sc(Lx, Ly)
    print ('saeed self copling %e' %Isc,
'saeed_sc/orion',Isc / Io)

    Icp = saeed_coplanar(Lx, Ly, d, 0)
    print ('saeed coplanar %e' %Icp,
'saeed/orion',Icp / Io)
```

```
    Is = saeed(Lx, Ly, d, 0, z)
    print ('saeed %e' %Is, 'saeed/orion',Is / Io)

    Ih = hitoshi(Lx, Ly, d, 0, z)
    print ('hitoshi',Ih, 'saeed/hitoshi',Is/Ih)
```

ضریب جفت‌شدگی برای قطعات عمود بر هم با این فرمول بدست می‌آید.