KONYA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ – YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ

DİJİTAL GÖRÜNTÜ İŞLEME DERSİ

VİZE ÖDEVİ RAPORU

Hazırlayan

Zişan AYDOĞAN- 211229029

GitHub Hesabım

https://github.com/ZisanAydogan

Öğretim Üyesi

Dr. Öğr. Üyesi Burak Yılmaz

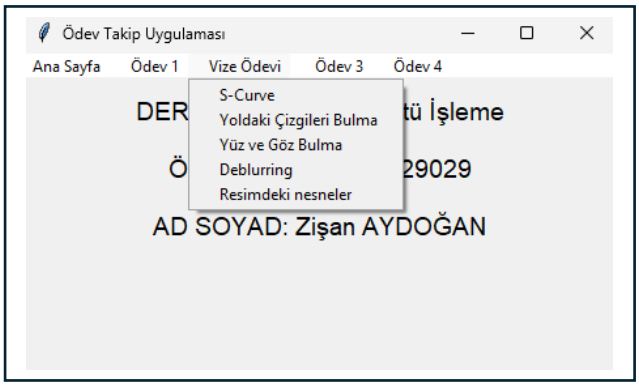
KONYA,2024

PROJE KONUSU:

Bu ödevde yaptıklarım:

1. Standart Sigmoid Fonksiyonu, Yatay Kaydırılmış Sigmoid Fonksiyonu, Eğimli Sigmoid Fonksiyonu ve kendi ürettiğim bir fonksiyonu kullanarak S- Curve metodu ile kontrast güçlendirme işlemi yaptım.
2. Hough Transform metodunu kullanarak, yoldaki çizgileri tespit eden bir uygulama ve bir de yüz resminde gözleri tespit eden bir uygulama yaptım.
3. Hareketli cisimlerin fotoğrafları çekildiğinde, hareketten dolayı oluşan blur ve bozulmaları gidermek için **deblurring** işlemi yaptım.
4. Bir tarladan drone ile çekilmiş hiperspektral görüntünün RGB'ye indirgenmiş halindeki "koyu yeşil" bölgeleri tespit edip bir excel tablosu oluşturacak kodu yazdım.

GUI GÖRÜNÜMÜ:



PROJE ADIMLARI:

1. S-Curve metodu ile kontrast güçlendirme işlemi:

Kontrast güçlendirme işleminde S-Curve metodu kullanılarak yapılan işlem, genellikle aşağıdaki adımları içerir:

- **Veri Toplama:** İlk adım, işlenecek olan verinin toplanmasıdır. Bu veriler, genellikle bir görüntünün piksel değerleri veya bir değişkenin zamana göre değişen değerleri olabilir.
- **Fonksiyon Seçimi:** S-Curve metodu kullanılarak kontrast güçlendirme işlemi için bir veya daha fazla fonksiyon seçilir. Bu fonksiyonlar, genellikle Sigmoid fonksiyonları gibi, belirli bir giriş aralığını belirli bir çıkış aralığına eşleyen matematiksel fonksiyonlardır.
- **Fonksiyon Parametrelerinin Ayarlanması:** Seçilen fonksiyonların parametreleri, genellikle deneme yanılma yoluyla veya belirli bir hedefe ulaşmak için optimize edilir. Bu adım, genellikle veriler üzerindeki bir dizi deneme-yanılma işlemi ile gerçekleştirilir.

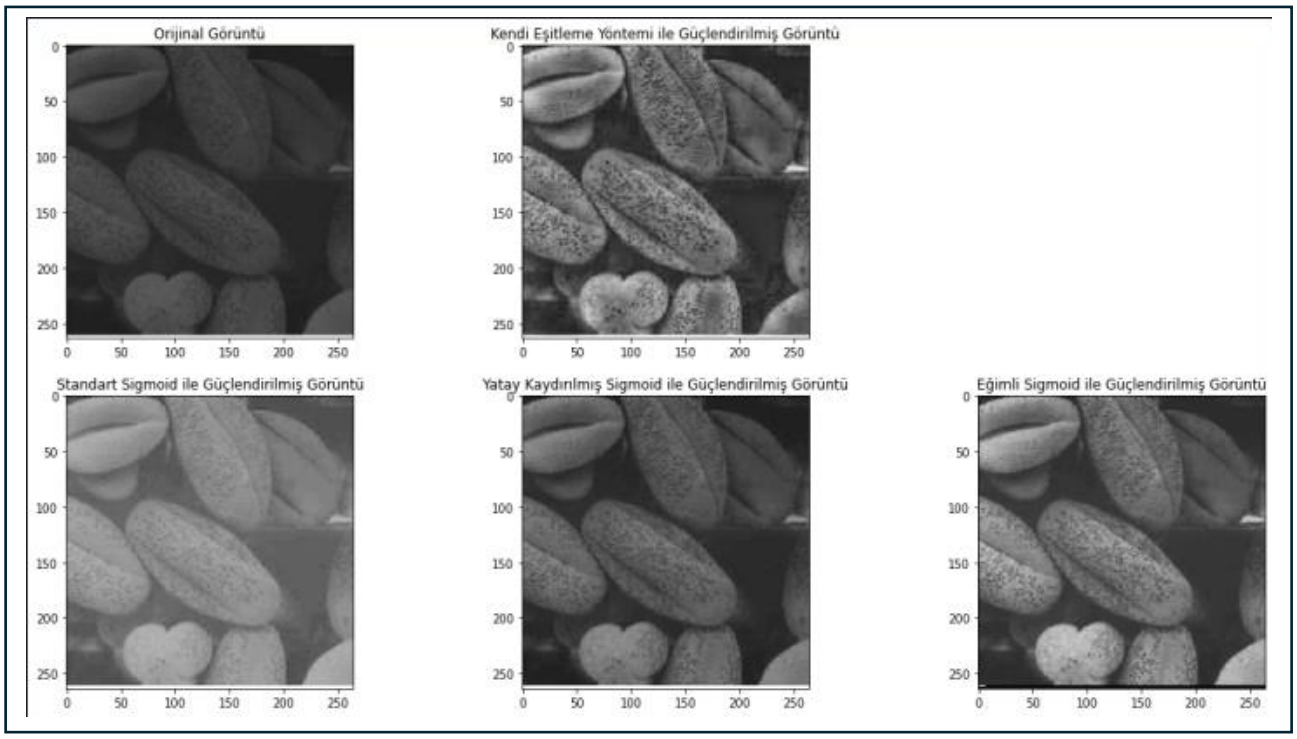
- **Uygulama ve Sonuçların Değerlendirilmesi:** Seçilen fonksiyonlar ve ayarlanmış parametreler, veri üzerinde uygulanır ve kontrast güçlendirme işlemi gerçekleştirilir. Sonuçlar, görsel olarak incelenir ve istenilen kontrast artışının sağlanıp sağlanmadığı değerlendirilir.

S-Curve metodu, özellikle doğrusal olmayan ilişkileri ve değişen davranışları modellemek ve analiz etmek için kullanışlıdır. Kontrast güçlendirme gibi uygulamalarda, bu metodoloji, verilerin doğrusal olmayan bir şekilde yeniden şekillendirilmesi veya dönüştürülmesi gerektiğinde sıkça kullanılır. Bu sayede, görüntülerin veya diğer veri türlerinin daha net veya daha belirgin hale getirilmesi sağlanabilir.

```python
# Sigmoid fonksiyonu
def sigmoid(x, a=1):
    return 1 / (1 + np.exp(-a*x))

# Yatay kaydırılmış sigmoid fonksiyonu
def shifted_sigmoid(x, a=1, b=0):
    return 1 / (1 + np.exp(-a*(x - b)))

# Eğimli sigmoid fonksiyonu
def tilted_sigmoid(x, a=1, b=0, c=1):
    return 1 / (1 + np.exp(-a*(x - b))) + c

# S-Curve kontrast arttırma fonksiyonu
def s_curve_contrast(image_path, func, **kwargs):
    # Görüntüyü yükle
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Normalize et
    normalized_image = image / 255.0

    # İşlevi görüntü üzerine uygula
    transformed_image = func(normalized_image, **kwargs)

    # İşlem sonrası görüntüyü [0, 255] aralığına geri getir
    transformed_image = (transformed_image * 255).astype(np.uint8)

    return transformed_image

# Kontrast arttırma fonksiyonu
def contrast_enhancement(image_path):
    # Görüntüyü yükle
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Adaptif histogram eşitleme işlemini uygula
    clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(16, 16))
    enhanced_image = clahe.apply(image)

    return enhanced_image
```

```python
def s_curve_uygula(self):
    image_path = 'C:/Users/Zisann/Desktop/bulanik.jpg'
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    enhanced_image = contrast_enhancement(image_path)

    # Standart Sigmoid Fonksiyonu ile işlem
    selected_function = sigmoid
    function_params = {'a': 5}
    sigmoid_transformed_image = s_curve_contrast(image_path, selected_function, **function_params)

    # Yatay Kaydırılmış Sigmoid Fonksiyonu ile işlem
    selected_function = shifted_sigmoid
    function_params_shifted = {'a': 10, 'b': 0.5}
    shifted_sigmoid_transformed_image = s_curve_contrast(image_path, selected_function, **function_params_shifted)

    # Eğimli Sigmoid Fonksiyonu ile işlem
    selected_function = tilted_sigmoid
    function_params = {'a': 18, 'b': 0.5, 'c': 0.1}
    tilted_sigmoid_transformed_image = s_curve_contrast(image_path, selected_function, **function_params)

    # Resimleri göster
    plt.figure(figsize=(20, 10))

    plt.subplot(2, 3, 1)
    plt.imshow(original_image, cmap='gray')
    plt.title('Orijinal Görüntü')

    plt.subplot(2, 3, 2)
    plt.imshow(enhanced_image, cmap='gray')
    plt.title('Kendi Eşitleme Yöntemi ile Güçlendirilmiş Görüntü')

    plt.subplot(2, 3, 4)
    plt.imshow(sigmoid_transformed_image, cmap='gray')
    plt.title('Standart Sigmoid ile Güçlendirilmiş Görüntü')

    plt.subplot(2, 3, 5)
    plt.imshow(shifted_sigmoid_transformed_image, cmap='gray')
    plt.title('Yatay Kaydırılmış Sigmoid ile Güçlendirilmiş Görüntü')

    plt.subplot(2, 3, 6)
    plt.imshow(tilted_sigmoid_transformed_image, cmap='gray')
    plt.title('Eğimli Sigmoid ile Güçlendirilmiş Görüntü')

    plt.show()
```

Yukarıdaki Görüntünün S-Curve İşlemi Sonucundaki Durumları:



2. Hough Transform metodu:

Hough Transform, özellikle çizgi, daire veya başka şekiller gibi belirli geometrik şekillerin tespit edilmesi için kullanılan bir görüntü işleme tekniğidir. Bu yöntem, bir görüntüdeki belirli şekillerin varlığını tespit etmek için kullanılır ve özellikle kenar tespiti gibi ön işleme adımlarının ardından sıklıkla kullanılır.

Hough Transform genellikle şu adımları içerir:

- **Kenar Tespiti:** İlk adım, genellikle Canny kenar dedektörü gibi bir kenar tespit algoritması kullanılarak, görüntünün kenarlarının belirlenmesidir. Bu adım, görüntünün her pikselindeki yoğunluk değişikliklerini tespit ederek kenarları belirler.
- **Parametre Uzayının Oluşturulması:** Hough Transform, belirli bir geometrik şeklinin görüntüdeki varlığını tespit etmek için bir parametre uzayı oluşturur. Örneğin, bir doğru

tespiti için iki parametre kullanılır: doğrunun eğimi (m) ve orijinalden geçen yüksekliği (b). Her olası doğru kombinasyonu, parametre uzayında bir noktaya karşılık gelir.

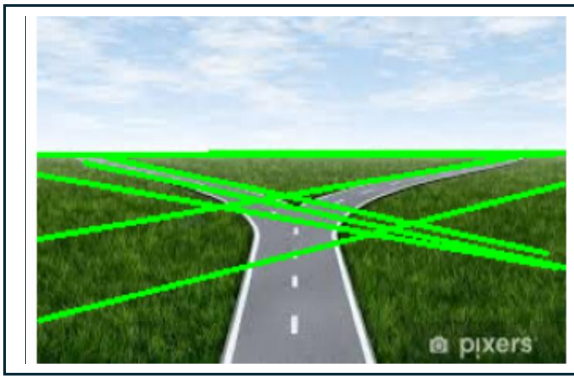
- **Oylama (Voting):** Her kenar pikseli, parametre uzayında muhtemel tüm şekillerin üzerinde oy kullanır. Örneğin, bir kenar pikseli, üzerinde bulunduğu doğrunun parametre uzayında bir oya sahip olduğu noktaya oy verir.
- **Oylama Sonuçlarının Değerlendirilmesi:** En fazla oya sahip noktalar genellikle gerçek şeklin parametrelerini temsil eder. Bu adımda, belirli bir eşik değeri kullanılarak en yüksek oya sahip noktalar belirlenir.
- **Sonuçların Çıkarılması**: En yüksek oya sahip noktaların gerçek dünyadaki geometrik şekillere karşılık geldiği düşünülür. Örneğin, en yüksek oya sahip noktalar, doğru tespiti için parametrelerin bulunduğu yerlerdir.

Hough Transform, özellikle kenar tespiti ve nesne algılama gibi alanlarda yaygın olarak kullanılır. Bu yöntem, karmaşık ve düzensiz görüntülerde bile belirli geometrik şekilleri tespit etmek için oldukça etkilidir.

Yoldaki çizgileri tespit etmek için Hough Transform kullanılırken, genellikle aşağıdaki adımlar izlenir:

1. **Kenar Tespiti:** İlk adım, yoldaki çizgilerin kenarlarını belirlemektir. Bu adım için genellikle bir kenar tespit algoritması kullanılır. Canny kenar dedektörü gibi algoritmalar sıkça tercih edilir.
2. **Hough Transform Uygulaması:** Kenar pikselleri kullanılarak Hough Transform uygulanır. Her bir kenar pikseli, parametre uzayında belirli bir doğru için bir oy kullanır. Bu adımda, doğruların parametrelerini belirlemek için bir oylama süreci gerçekleşir.
3. **Doğruların Belirlenmesi:** En yüksek oya sahip olan parametreler, yoldaki çizgilerin temsil ettiği doğruların parametrelerini temsil eder. Bu parametreler kullanılarak, yoldaki çizgilerin denklemleri hesaplanır ve görüntüye çizilir.

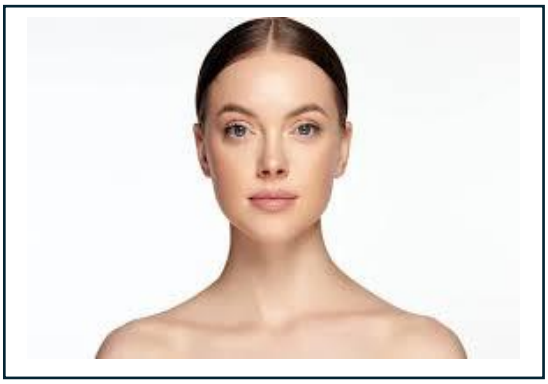
```python
def yol_cizgileri_bulma(self):
    # Görüntüyü yükle
    image = cv2.imread("road.jpg")
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Kenar tespiti
    edges = cv2.Canny(gray, 50, 150)

    # Hough Dönüşümü
    lines = cv2.HoughLinesP(edges, 1, np.pi/180, 100, minLineLength=100, maxLineGap=50)

    # Tüm çizgileri çizme
    for line in lines:
        x1, y1, x2, y2 = line[0]
        cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)

    # Sonucu görselleştirme
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Road Lines Detection")
    plt.axis('off')
    plt.show()
```

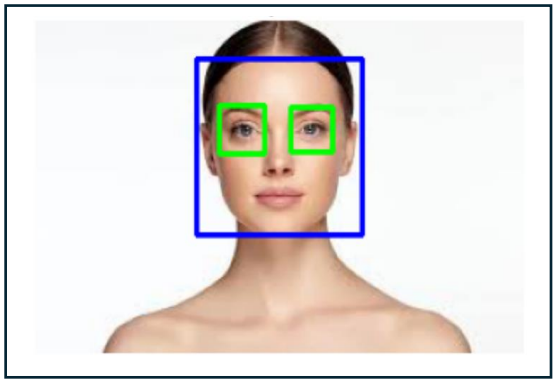


Yüz resminde gözleri tespit etmek için Hough Transform kullanmak daha karmaşıktır ve genellikle daha fazla adım içerir:

1. **Görüntü İşleme ve Ön İşleme:** İlk adım, yüzün tespit edilmesi ve görüntünün önceden işlenmesidir. Öncelikle yüz tespiti için bir nesne algılama veya yüz tanıma algoritması kullanılabilir. Ardından, yüz bölgesi belirlenir ve gözleri tespit etmek için bu bölgede Hough Transform uygulanır.
2. **Göz Bölgesinin Belirlenmesi:** Yüz bölgesi içindeki gözlerin tespiti için Hough Transform uygulanır. Gözler genellikle daire veya elips şeklinde olduğu için, Hough Transform'un dairesel varyantı kullanılır.
3. **Hough Transform Uygulaması:** Gözlerin kenarlarını belirlemek için bir kenar tespit algoritması kullanılır. Ardından, Hough Transform uygulanarak daire veya elips parametreleri belirlenir.
4. **Gözlerin Belirlenmesi:** En yüksek oya sahip olan daire veya elips parametreleri, gözlerin konumunu temsil eder. Bu parametreler kullanılarak, görüntüdeki gözlerin konumları belirlenir.

```python
def göz_bulma(self):
    # Görüntüyü yükle
    image = cv2.imread("face.jpg")
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Yüz tespiti için sınıflandırıcıyı yükle
    face_cascade = cv2.CascadeClassifier("C:\\Users\\Zisann\\anaconda3\\Library\\etc\\haarcascades\\haarcascade_frontalface_default.xml")

    # Göz tespiti için sınıflandırıcıyı yükle
    eye_cascade = cv2.CascadeClassifier("C:\\Users\\Zisann\\anaconda3\\Library\\etc\\haarcascades\\haarcascade_eye.xml")

    # Yüz tespiti
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Tüm yüzleri çizme
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = image[y:y+h, x:x+w]

        # Gözleri tespit etme
        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)

    # Görüntüyü Matplotlib ile gösterme
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Face with Eyes Detection")
    plt.axis('off')
    plt.show()
```





3. Deblurring Algoritması:

Hareket bulanıklığı (motion blur), bir kameranın veya bir nesnenin hareket ederken görüntüde bıraktığı bulanıklıktır. Bu tür bulanıklık, genellikle hızlı hareket eden nesneleri çekerken veya düşük ışık koşullarında fotoğraf çekerken meydana gelir. Örneğin, bir arabanın hızla geçişi sırasında veya elde tutulan bir kamerayla çekilen bir fotoğrafın titreme sonucu oluşan bulanıklık, hareket bulanıklığına örnek olarak verilebilir.

Motion deblurring, yani hareket bulanıklığının giderilmesi veya azaltılması, bu tür bulanıklığı azaltmak veya kaldırmak için kullanılan görüntü işleme tekniklerini ifade eder. Bu teknikler, genellikle bulanıklığın oluşma şeklini (örneğin, hareket yönü ve hızı) modelleyerek ve bu modele uygun bir geri dönüşüm işlemi uygulayarak çalışır.

```python
def deblurring(self):
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt

    def sharpen_image(image, kernel_size=(5, 5)):
        # Kenar belirginleştirme için filtre oluşturma
        kernel = np.array([[-1, -1, -1],
                           [-1, 9, -1],
                           [-1, -1, -1]])
        sharpened_image = cv2.filter2D(image, -1, kernel)
        return sharpened_image

    # Görüntüyü yükle
    image = cv2.imread("blurlu.jpg")

    # Görüntüyü keskinleştir
    sharpened_image = sharpen_image(image)

    # Görüntüleri görselleştirme
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(sharpened_image, cv2.COLOR_BGR2RGB))
    plt.title("Sharpened Image")
    plt.axis('off')

    plt.show()
```

4. Resimdeki nesneleri sayma ve özellik çıkarma:

```python
def nesne_bul(self):
    file_path = "image.jpg"
    # Resmi işle
    image = cv2.imread(file_path)
    # Renk filtresi uygula (koyu yeşil)
    lower_green = np.array([0, 100, 0])
    upper_green = np.array([100, 255, 100])
    mask = cv2.inRange(image, lower_green, upper_green)
    # Konturları bul
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    # Özellikleri saklamak için boş bir DataFrame oluştur
    data = []
    # Her kontur için işlem yap
    for i, contour in enumerate(contours):
        # Alanı hesapla
        area = cv2.contourArea(contour)
        # Konturun dış dikdörtgenini al
        x, y, w, h = cv2.boundingRect(contour)
        # Diagonal hesapla
        diagonal = np.sqrt(w*2 + h*2)
        # Momentleri hesapla
        moments = cv2.moments(contour)
        # Momentlardan enerji ve entropi hesapla
        energy = -1 * np.sum([p * np.log(p + 1e-6) for p in moments.values() if p > 0])
        entropy = -1 * np.sum([p / np.sum(list(moments.values())) * np.log(p / np.sum(list(moments.values()))) for p in moments.values() if p > 0])
        # Gri ton ortalama ve medianını hesapla
        mean_val = np.mean(image[y:y+h, x:x+w])
        median_val = np.median(image[y:y+h, x:x+w])
        # Ortalama, median ve moment değerlerini sakla
        data.append([i+1, (x+w/2, y+h/2), w, h, diagonal, energy, entropy, mean_val, median_val])
    # DataFrame oluştur
    df = pd.DataFrame(data, columns=["No", "Center", "Length", "Width", "Diagonal", "Energy", "Entropy", "Mean", "Median"])
    # Excel dosyasına yaz
    excel_path = "koyu_yesil_bolgeler.xlsx"
    df.to_excel(excel_path, index=False)
```

	A	B	C	D	E	F	G	H	I	J
	No	Center	Length	Width	Diagonal	Energy	Entropy	Mean	Median	
1	1	(733.0, 63	4	1	4,123106	0	0	105,9167	92,5	
2	2	(565.0, 63	2	1	2,236068	0	0	117,8333	96,5	
3	3	(1095.5, 6	5	3	5,830952	-2,1E+11	1,228335	102,4444	99	
4	4	(808.0, 63	6	3	6,708204	-6,2E+10	1,359276	93,5	96	
5	5	(711.0, 63	6	5	7,81025	-4,5E+11	1,386669	78,15556	77,5	
6	6	(1076.5, 6	7	6	9,219544	-1,5E+12	1,236521	76,31746	68	
7	7	(1112.0, 6	8	7	10,63015	-2E+12	1,21889	78,89286	74,5	
8	8	(969.0, 63	8	8	11,31371	-1,5E+12	1,288375	84,15104	81,5	
9	9	(773.5, 63	7	9	11,40175	-1,3E+12	1,369667	74,15344	78	
10	10	(1042.5, 6	7	7	9,899495	-1,6E+12	1,249769	92,53061	91	
11	11	(662.5, 63	9	8	12,04159	-9E+11	1,393329	86,40741	94	
12	12	(609.5, 62	7	5	8,602325	-4E+11	1,394731	92,87619	91	
13	13	(1133.5, 6	15	13	19,84943	-5E+12	1,204833	127,6923	123	
14	14	(628.0, 63	18	13	22,2036	-2,2E+12	1,395162	124,7849	113	
15	15	(515.0, 62	6	12	13,41641	-4,4E+11	1,372084	123,9815	101,5	
16	16	(884.5, 62	9	8	12,04159	-1,9E+12	1,321168	86,48148	89	
17	17	(750.5, 62	9	16	18,35756	-2,9E+12	1,374297	85,29398	84,5	
18	18	(735.5, 62	11	14	17,80449	-1,7E+12	1,378264	109,7078	103	
19	19	(566.5, 62	7	8	10,63015	-7E+11	1,390154	69,2619	53	
20	20	(536.5, 62	13	14	19,10497	-9,9E+11	1,382126	125,8626	106,5	
21	21	(1083.0, 6	14	17	22,02272	-8,7E+12	1,225929	97,61345	100	
22	22	(1055.5, 6	7	6	9,219544	-1,2E+12	1,234111	101,0635	102,5	
23	23	(1015.5, 6	5	7	8,602325	-9,9E+11	1,254765	92,78095	91	
24	24	(595.0, 62	8	16	17,88854	-1,4E+12	1,394004	86,3151	85,5	
25	25	(697.0, 62	12	12	16,97056	-2,1E+12	1,386572	81,71065	71	