Metody Numeryczne projekt 1 Piotr Sieński 184297

March 25, 2022

1 Projekt 1 Metody Numeryczne Piotr Sieński 184297 - Implementacja wskaźnika giełdowego MACD

1.1 Omówienie ogólnej struktury programu i implementacji wskaźnika $MACD$

Wskaźniki $MACD$ i $SIGNAL$ obliczane są przy uzyciu średniej kroczącej EMA dla N okresów danej wzorem :

$$EMA_N = \frac{p_0 + (1-\alpha)p_1 + (1-\alpha)^2 p_2 + ... + (1-\alpha)^N p_N}{1 + (1-\alpha) + (1-\alpha)^2 + ... + (1-\alpha)^N}$$

gdzie p_i jest próbką sprzed i dni a $\alpha = \frac{2}{N+1}$

Funkcja EMA przyjmuje jako parametr N elementowy wektor danych

[ ]:
```
def EMA(v):
    n = len(v)
    a = 2 / (n + 1)
    numerator = 0
    denominator = 0
    for i in range(n):
        numerator += v[i] * ((1 - a) ** i)
        denominator += (1 - a) ** i
    return numerator / denominator
```

Wskaźnik $MACD$ jest obliczany jako $EMA_{12} - EMA_{26}$ z danych, a wstaźnik $SIGNAL$ jako EMA_9 z $MACD$

Funkcja $MACD$ przyjmuje jako parametr wektor danych, którego ostatnim elementem jest dzień dla którego liczony jest wskaźnik, podobnie jak funkcja $SIGNAL$

[ ]:
```
def MACD(v):
    ema_12 = EMA(v[-12:])
    ema_26 = EMA(v[-26:])
    return ema_12 - ema_26


def SIGNAL(macd):
    return EMA(macd[-9:])
```

Generowanie wykresu funkcji odbywa się w funkcji main, jako parametr podany jest wektor danych wczytany na początku skryptu za pomocą biblioteki pandas

[ ]:
```
if __name__ == "__main__":
    column = -2
    data_dir = 'wig20.csv'
    data = pd.read_csv(data_dir)
    data_vector = data.iloc[:, column].to_numpy()
    main(data_vector[:1000], sim_type="basic")
```

Parametrami funkcji main są wektor danych i opcjonalne parametry określające metodę symulacji.

W funkcji main iteracyjnie wypełniane są tablice zawierające historię wartości wskaźników $MACD$ i $SIGNAL$. Zmienne ***last_action*** i ***prev_dif*** zawierać będą parametry do różnych metod symulacji (dla podstawowej zapisujemy różnicę między $MACD$ i $SIGNAL$ w każdej w każdej iteracji głównej pętli, a dla zaawansowanej poprzenio wykonaną akcję tzn, kupno/ sprzedaż) i potrzebne będą w późniejszej ocenie przydatności wskaźników w analizie technicznej. Inicjalizowane są również zmienne określające ilość posiadanych akcji i pieniędzy oraz początkowa wartość posiadanego kapitału

[ ]:
```
def main(data, sim_type="advanced", dist=1.5, tau=0.8, amount=1):
    macd_arr = np.empty(0)
    signal_arr = np.empty(0)

    if sim_type == "basic":
        prev_dif = 0
    else:
        last_action = "none"

    if amount > 1:
        amount = 1

    actions = 1000
    money = 0
    start_value = money + actions * data_vector[0]
```

Najpierw w pętli iterującej od 26 do 35 (9 - razy - tyle ile potrzeba do wyznaczenia $SIGNAL$) elementu wektora danych obliczane są wartości $MACD$ a następnie obliczana jest pierwsza wartość $SIGNAL$

[ ]:
```
     for i in range(26, 35):
        macd_arr = np.append(macd_arr, MACD(data[:i]))

    signal_arr = np.append(signal_arr, SIGNAL(macd_arr))
```

Przed główną pętlą programu przygotowywany jest podział wykresu na dwie części : na jednej z nich znajdować będą się wskaźniki, a na drugiej wyrysowujemy dane przy pomocy wykresy słupkowego.

```
[ ]:    fig, ax = plt.subplots(2, sharex=True)
        plt.xlabel("dni")
        ax[1].set_ylabel("cena akcji")
        ax[0].set_ylabel("wartość wskaźników")
        axes_1 = fig.gca()
        # narysowanie wykresu danych wejściowych
        ax[1].bar(np.arange(0, len(data)), data - min(data), width=1,␣
    ↪color='#80ede2', bottom=min(data))
        # zapisujemy wartości do późniejszego użycia
        y_min_1, y_max_1 = axes_1.get_ylim()
```

W głównej pętli rysowanie wykresów przebiega poprzez rysowanie odcinków pomiędzy kolejnymi wartościami $MACD$ i $SIGNAL$

```
[ ]:     for i in range(35, len(data)):
                macd_arr = np.append(macd_arr,  MACD(data[:i]))
                signal_arr = np.append(signal_arr, SIGNAL(macd_arr))

                ax[0].plot([i - 1, i], [signal_arr[-2], signal_arr[-1]], color='b',␣
    ↪label='SIGNAL')
                ax[0].plot([i - 1, i], [macd_arr[-2], macd_arr[-1]], color='r',␣
    ↪label='MACD')
```

W każdej iteracji pętli dodatkowo podejmowana jest decyzja o zakupie / sprzedaży akcji. Użyte funkcje omówione będą w dalszej części sprawozdania.

```
[ ]:  if sim_type == "basic":
                prev_dif, actions, money = basicSim(actions, amount, ax, i,␣
    ↪macd_arr, money, prev_dif, signal_arr, y_max_1,
                                                    y_min_1)
            else:
                last_action, actions, money = advancedSim(actions, amount, ax,␣
    ↪dist, i, last_action, macd_arr, money,
                                                          signal_arr, tau, y_max_1,␣
    ↪y_min_1)
```

Po wykonaniu się głownej pętli obliczana jest końcowa wartość posiadanego kapitału

```
[ ]: end_value = money + actions * data_vector[len(data_vector) - 1]
        print(f"wartość startowa: {start_value}, wartosć końcowa : {end_value},␣
    ↪różnica: {end_value - start_value} "
              f"( {100 * end_value /start_value}  % wartości startowej)")
```

W końcu wyświetlane są wykresy

```
[ ]: # usunięcie powtarzających się etykiet i wyświetlenie legend dla obu wykresów
        handles, labels = ax[1].get_legend_handles_labels()
```

```
    by_label = dict(zip(labels, handles))
    ax[1].legend(by_label.values(), by_label.keys(), loc='lower left')
    handles, labels = ax[0].get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    ax[0].legend(by_label.values(), by_label.keys(), loc='lower left')
# wyświetlenie wykresów
    fig.tight_layout()
    plt.show()
```

1.2 Omówienie implementacji funkcji symulujących

Do symulacji wykorzystywane są funkcje buy i sell

[ ]:
```
def buy(actions, money, rate, price):
    amount = int(rate * money / price)
    return actions + amount, money - amount * price


def sell(actions, money, rate, price):
    amount = int(actions * rate)
    return actions - amount, money + price * amount
```

1.2.1 Prosta funkcja decyzji

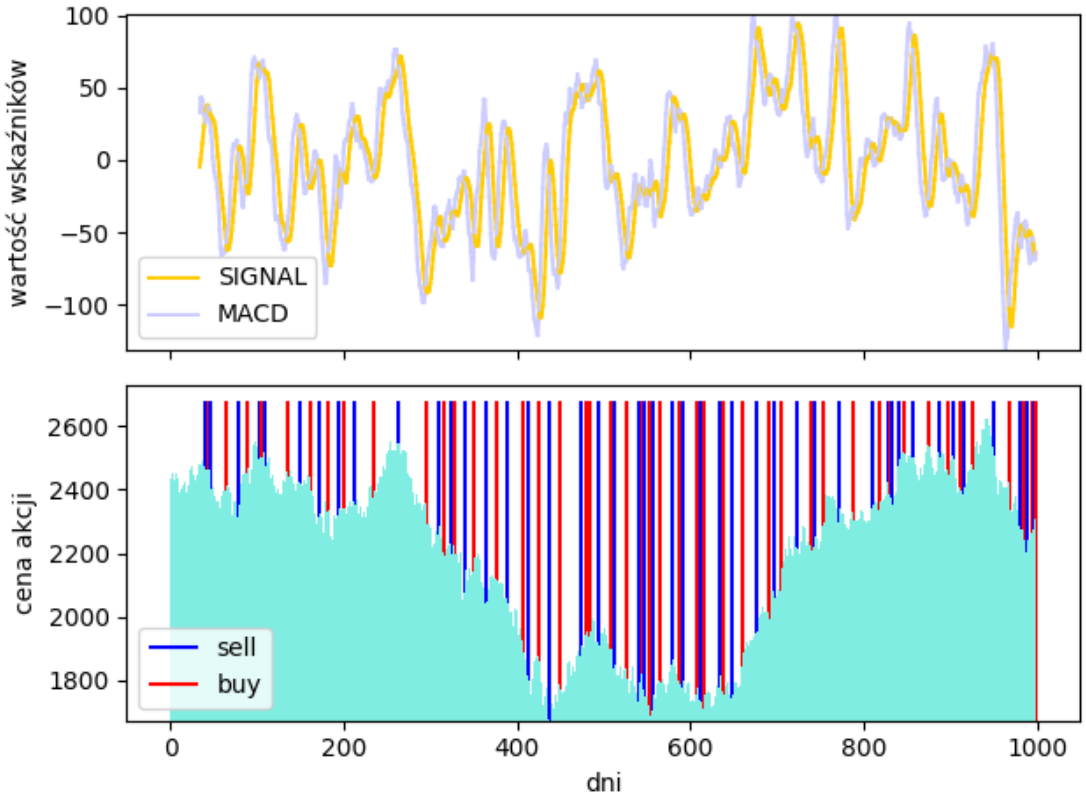
Pierwsza, (prosta) funkcaj przewidująca moment kupna / sprzedaży oparta jest na następującej zasadzie: momenty przecięcia się wykresów $MACD$ i $SIGNAL$ oznaczają moment kupna / sprzedaży - jeśli $MACD$ przecina $SIGNAL$ od dołu jest to sygnał do zakupu akcji, a w przeciwnym wypadku do sprzedaży.

Jeśli wykresy $MACD$ i $SIGNAL$ przecinają się to różnica pomiędzy ich wartościami zmienia znak, w momentach przecięcia na wykresie danych rysowana jest pionowa linia - niebieska jeśli $SIGNAL$ przecina $MACD$ od góry i czerwona jeśli od dołu. Linie niebieskie wskazują moment sprzedaży akcji a czerwone zakupu.

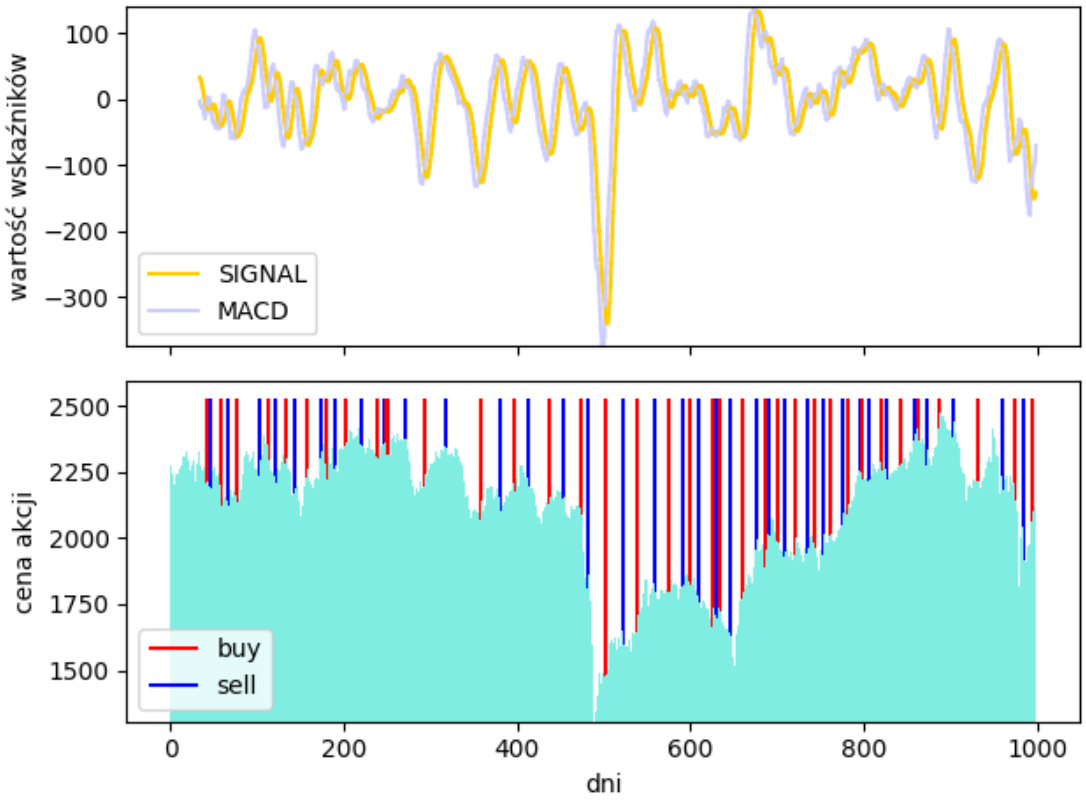
Funkcja zwraca wartość różnicy w danej iteracji oraz ilosć akcji i pieniędzy

[ ]:
```
def basicSim(actions, amount, ax, i, macd_arr, money, prev_dif, signal_arr,␣
↪y_max_1, y_min_1):
    dif = signal_arr[-1] - macd_arr[-1]
    if np.sign(dif) != np.sign(prev_dif) and i > 35:
        if dif - prev_dif > 0:
            ax[1].vlines(i, y_min_1, y_max_1, colors='b', zorder=1,␣
↪label="sell")
            actions, money = sell(actions, money, amount, data_vector[i])
        else:
            ax[1].vlines(i, y_min_1, y_max_1, colors='r', zorder=1, label="buy")
            actions, money = buy(actions, money, amount, data_vector[i])
    return dif, actions, money
```

Poniżej załączone są wykresy wygenerowane przy użyciu wyżej opisanej funkcji dwa wykresy dla różnych, 1000 elementowych wektorów danych oraz wyniki symulacji dla obu zestawów danych.




```
wartość startowa: 2395550.0, wartość końcowa : 1881655.0899999999, różnica: -513894.91000000015 ( 78.54793638204l7  % wartości startowej)
```




```
wartość startowa: 2395550.0, wartość końcowa : 1857823.1499999962, różnica: -537726.8500000038 ( 77.55309427897544  % wartości startowej)
```

1.2.2 Bardziej zaawansowana funkcja decyzji

Druga, bardziej zaawansowana funkcja opiera się na podobnym założeniu odnośnie wyznaczania momentów kupna / sprzedaży, aczkolwiek zamiast dokonywać transakcji w momencie przecięcia się wykresów można spróbować dokonać tego przed przecięciem się wykresów poprzez przewidzenie momentu ich przecięcia.

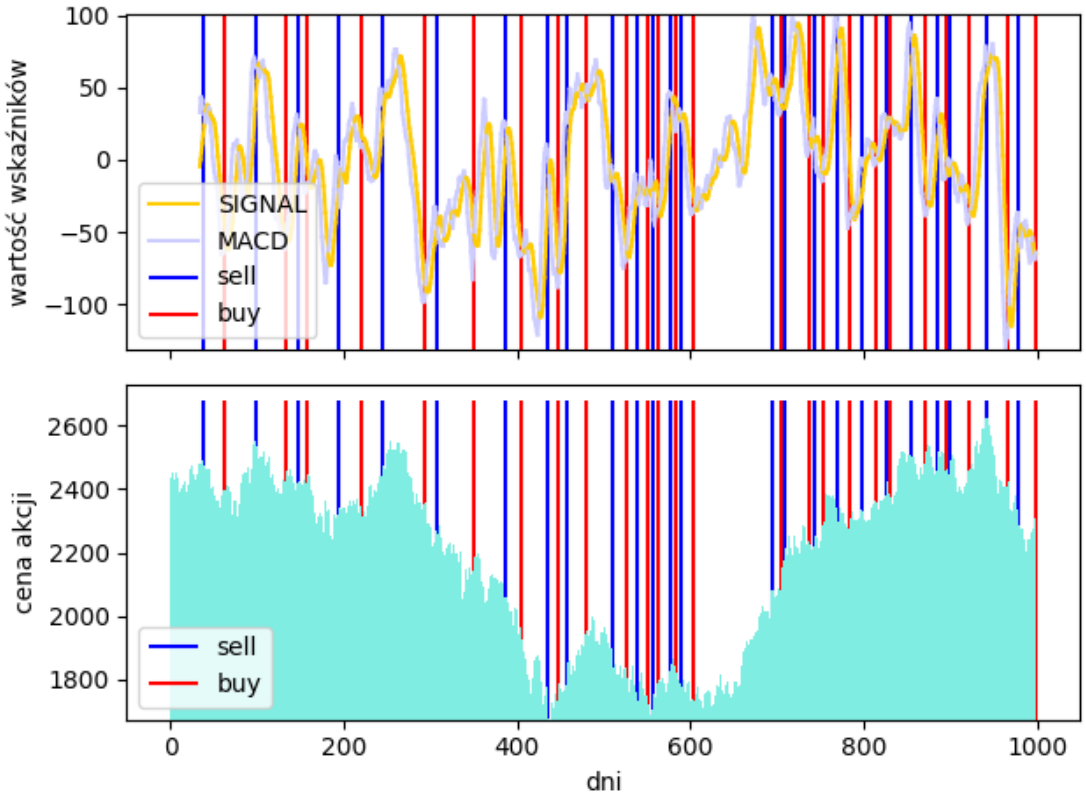
Predykcja momentu przecięcia odbywa się przy wykorzystaniu algorytmu lokalnie ważonej regresji liniowej i jest opisana w dalszej części sprawozdania.

Funkcja działa w oparciu o założenie, że zakup i sprzedaż odbywają się naprzemiennie (nie ma np. 2 zakupów pod rząd), zwraca wykonaną akcję oraz ilosć akcji i pieniędzy a podczas działania rysuje miejsce zakupu na wykresie danych jak i wskaźników.

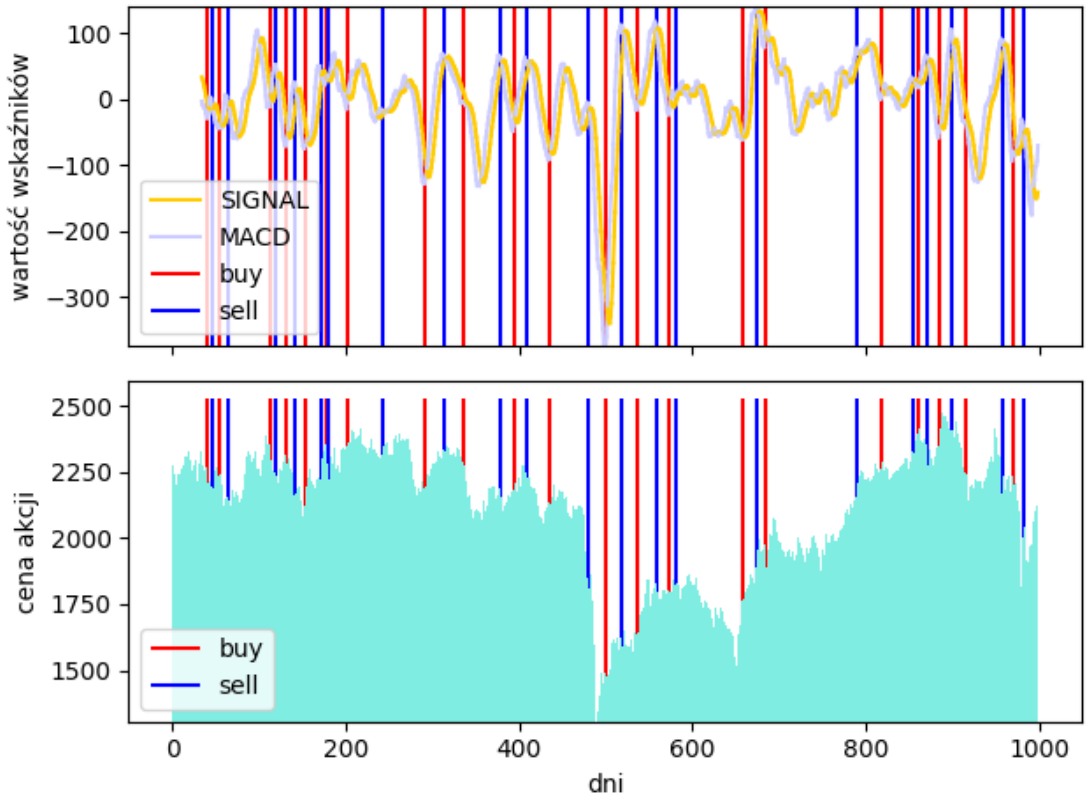
[ ]:
```
def advancedSim(actions, amount, ax, dist, i, last_action, macd_arr, money,␣
↪signal_arr, tau, y_max_1, y_min_1):
    what = predict(macd_arr, signal_arr, dist, tau, plot=False, ax=ax[0],␣
↪point=i)
    if last_action != "buy" and what == "buy":
        ax[1].vlines(i, y_min_1, y_max_1, colors='r', zorder=1, label="buy")
        ax[0].vlines(i, -1000000, 1000000, colors='r', zorder=1, label="buy")
```

```
        actions, money = buy(actions, money, amount, data_vector[i])
        last_action = "buy"

    elif last_action != "sell" and what == "sell":
        ax[1].vlines(i, y_min_1, y_max_1, colors='b', zorder=1, label="sell")
        ax[0].vlines(i, -1000000, 1000000, colors='b', zorder=1, label="sell")
        actions, money = sell(actions, money, amount, data_vector[i])
        last_action = "sell"
    return last_action, actions, money
```

Poniżej załączone są wykresy wygenerowane przy użyciu wyżej opisanej funkcji dwa wykresy dla różnych, 1000 elementowych wektorów danych oraz wyniki symulacji dla obu zestawów danych.




```
wartość startowa: 2395550.0, wartość końcowa : 2558321.6199999987, różnica: 162771.61999999871 ( 106.79474943123704  % wartości startowej)
```




```
wartość startowa: 2395550.0, wartość końcowa : 3134988.139999999, różnica: 739438.1399999992 ( 130.86715535054577  % wartości startowej)
```

1.2.3 Predykcja

Algorytm lokalnie ważonej regresji liniowej dopasowauje prostą do danych w konkretnym punkcie. Wagi innych punktów definiowane są poprzez odległość od wskazanego. Predykcja wykonywana jest dla najnowszego znanego punktu dancyh, co przy odpowiednim dobraniu parametru τ (określającego jak szybko wagi spadają w odległości od danego punktu) pozwala przewidzeić moment przecięcia się wykresów $MACD$ i $SIGNAL$.

Algorytm regresji opisany jest w dalszej części sprawozdania.

Funkcja przewidująca czy wykresy przetną się jako parametry oprócz obu tablic pobiera parametr ***dist*** określający jak daleko może być przewidziany punkt przecięcia od obecnego momentu w czasie aby funkcja przewidziała że wykresy przetną się oraz parametr ***tau*** określający w jakim tempie spadają wagi oddalonych punktów. **W przedstawionych przykładach parametry dobrane zostały eksperymentalnie.**

[ ]:
```
def predict(macd, signal, dist, tau, plot=False, ax=None, point=34,␣
 ↪s_color='g', m_color='g', pred_color='g'):
```

Najpierw zapewniana jest równa długość wektorów danych oraz ewentualne przycięcie ich do pewnego maksymalnego rozmiaru (tutaj określony na 100, w celu przyspieszenia obliczeń na dużych danych)

[ ]:
```
    width = min(len(macd), len(signal))
       if width > 100:
           width = 100
       m = macd[-width:]
       s = signal[-width:]

       m_b, m_a = fit(m, tau)
       s_b, s_a = fit(s, tau)
       if s_a == m_a:
           return "pass"
```

Punkt przecięcia się wykresów funkcji liniowych wyznaczany jest ze wzoru

$$x_{przecicia} = \frac{b_{MACD} - b_{SIGNAL}}{a_{SIGNAL} - a_{MACD}}$$

gdzie a, b są parametrami funkcji liniowych aproksymujących dane

[ ]:
```
    cross = (m_b - s_b)/(s_a - m_a)
```

Jeśli przecięcie znajduje się dalej na osi x niż punkt dla którego przewidujemy i w odległości nie większej niż ***dist*** określane jest która funkcja przecina drugą z góry i zwracany jest odpowiedni wynik. Ewentualnie zależnie od parametru ***plot*** rysowana jest linia oznaczająca przewidziany punkt przecięcia i odcinki prostych aproksymujących $MACD$ i $SIGNAL$

[ ]:
```
   if cross >= width and cross - width <= dist:
       # rysowanie przybliżonych linii aproksymujacych MACD i SIGNAL oraz␣
↪przewidzianego punktu przecięcia
       if plot:
           plot_prediction(m, s, m_a, m_b, s_a, s_b, dist)
           point_y = point - 34
           f_y_s_1 = s_a * point_y + s_b
           f_y_s_2 = s_a * (point_y + cross - width + 1) + s_b
           f_y_m_1 = m_a * point_y + m_b
           f_y_m_2 = m_a * (point_y+(cross - width) + 1) + m_b
           ax.plot([point, point+(cross - width) + 1], [f_y_s_1, f_y_s_2],␣
↪color=s_color, zorder=100)
           ax.plot([point, point +(cross - width) + 1], [f_y_m_1, f_y_m_2],␣
↪color=m_color, zorder=100)
           ax.vlines(point+(cross - width) + 1, -1000000, 1000000,␣
↪colors=pred_color, zorder=1,
                     label="przewidziane przecięcie")
       if abs(s_a) > abs(m_a):
           return "sell"
       else:
           return "buy"
   else:
       return "pass"
```

1.2.4 Lokalnie ważona regresja liniowa

W klasycznym algorytmie regresji liniowej funkcja kosztu zdefiniowana jest jako

$$J(\Theta) = \sum_{i=1}^{m} (y^{(i)} - \Theta^T x^{(i)})^2$$

Modyfikujemy funkcję kosztu, tak aby uwzględniała wagi poszczególnych elementów

$$J(\Theta) = \sum_{i=1}^{m} w^{(i)} (y^{(i)} - \Theta^T x^{(i)})^2$$

Funkcję określającą wagę i-tego elementu względem x możemy zdefiniować jako

$$w^{(i)} = exp(-\frac{(x^{(i)} - x)^2}{2\tau^2}) = exp(-\frac{(x^{(i)} - x)^T (x^{(i)} - x)}{2\tau^2})$$

Funkcja przyporządkowująca wagi elementom jest krzywą przypominającą krzywą Gaussa o odchyleniu standardowym τ i średniej x

Poniższa funkcja generuje macierz wag dla każdego x względem zadanego punktu

[ ]:
```
def weight_matrix(point, x, tau):
    n = x.shape[0]
    ret = np.eye(n)
    for i in range(n):
        ret[i, i] = np.exp(((x[i] - point).dot(np.transpose(x[i] - point))) /␣
 ↪(-2 * tau * tau))
    return ret
```

Klasyczną regresję liniową moglibyśmy dopasować przy uzyciu równania normalnej wyznaczając Θ jako :

$$\Theta = (X^T X)^{-1} (X^T y)$$

Ta sama metoda może być zaaplikowana dla naszego przypadku:

$$\Theta = (X^T W X)^{-1} (X^T W y)$$

gdzie W jest macierzą wag

Należy również pamiętać o dodaniu do wektora X kolumny jedynek aby wynikowy wektor Θ zawierał wyraz b dla równania prostej $ y = ax + b$, gdzie $ b = _0$ i $ a = _1 $

W tym przypadku jako punkt o najwyższej wadze określamy ostatni element wektora danych

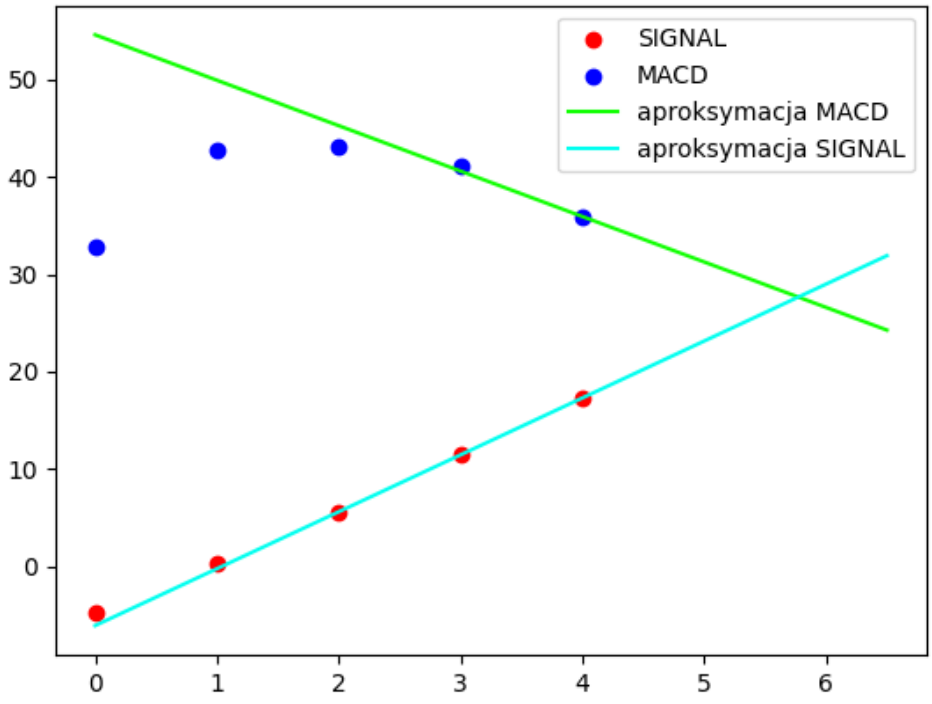
[ ]:
```
def fit(y, tau):
    n = y.shape[0]
    x = np.arange(0, n)
```

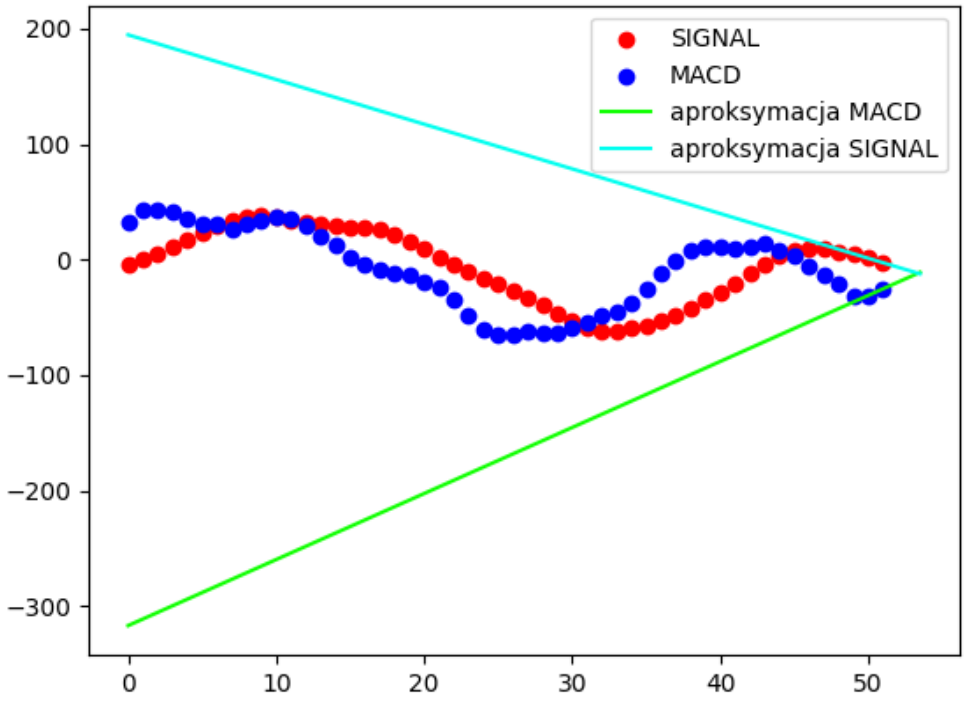
```
    x_a = np.append(np.ones(n).reshape(n, 1), x.reshape(n, 1), axis=1)
    point = np.array([1, n - 1])
    w = weight_matrix(point, x_a, tau)
    theta = np.linalg.pinv(np.transpose(x_a).dot(w.dot(x_a))).dot(np.
↪transpose(x_a).dot(w.dot(y)))

    # b, a in y = ax + b
    return theta[0], theta[1]
```

Dodatkowo zdefiniowana jest funkcja obrazująca aproksymację krzywych $MACD$ i $SIGNAL$ przez proste

[ ]:
```
def plot_prediction(m, s, m_a, m_b, s_a, s_b, dist):
    # zakładamy równą długość wektorów
    x_end = len(m) + dist
    x = np.arange(0, len(m))
    plt.figure()
    plt.scatter(x, s, color='#ff0000')
    plt.scatter(x, m, color='#0000ff')
    plt.plot([0, x_end], [m_b, m_a*x_end + m_b], color='#0fff00')
    plt.plot([0, x_end], [s_b, s_a*x_end + s_b], color='#00fff0')
```

Poniżej załączone są wyniki wyświetlane przez powyższą funkcję dla dwóch różnych zestawów danych

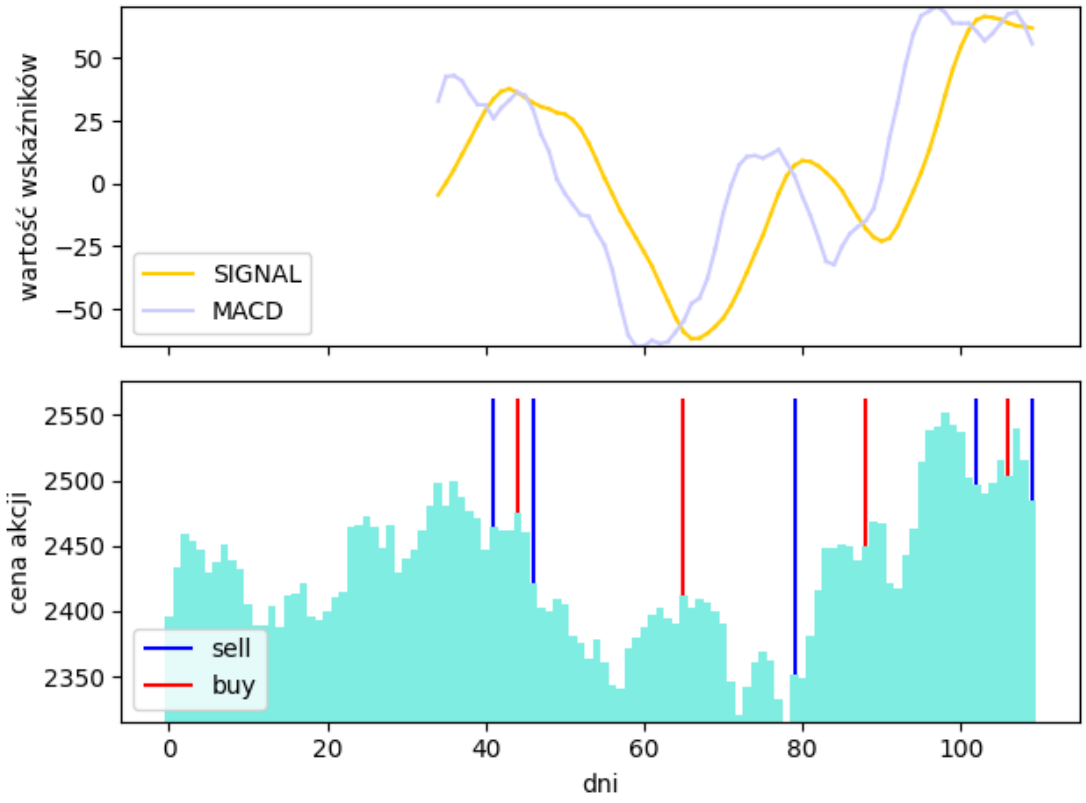

SIGNAL
MACD
aproksymacja MACD
aproksymacja SIGNAL
50
40
30
20
10
0
0
1
2
3
4
5
6



1.3 Ocena przydatności wskaźnika w analizie technicznej

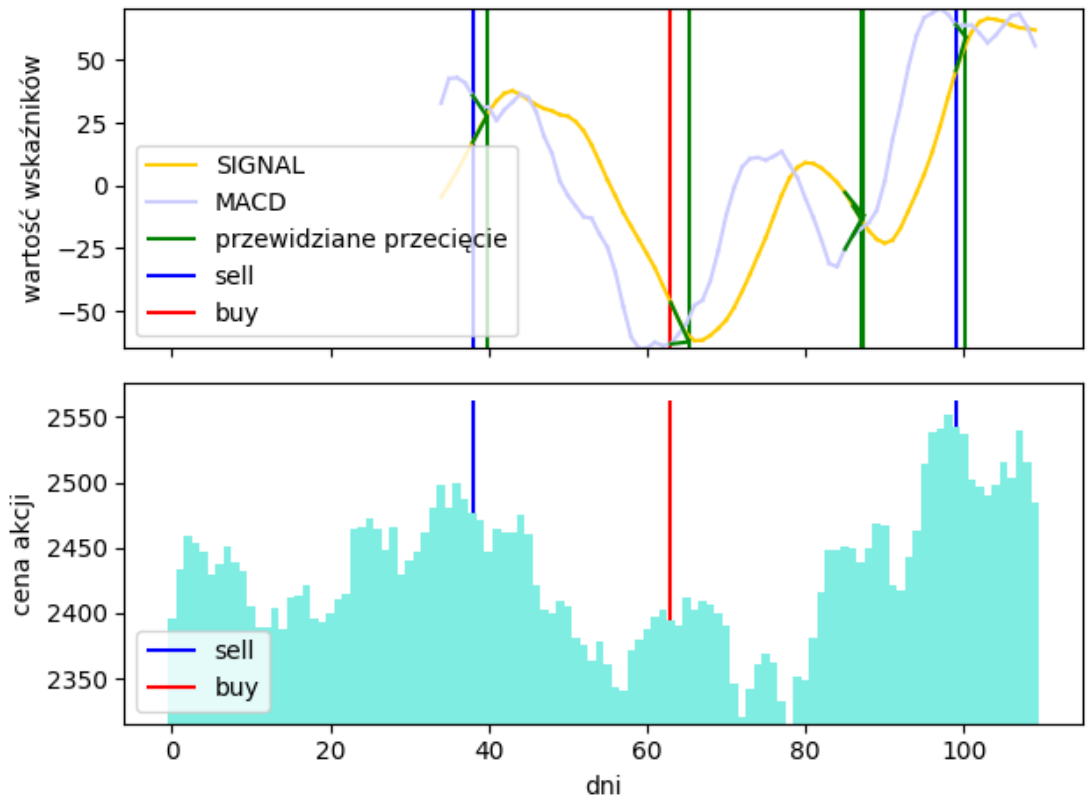
Ponizej przedstawione są wykresy 100 elementowego wektora danych (czytelniejszy wykres, więc prostszy w analizie) razem z wynikami dla obu opisanych wyżej algorytmów determinujących moment tranzakcji.

Wykres dla podstawowego algorytmu.




```
wartość startowa: 2395550.0, wartość końcowa : 2378831.21, różnica: -16718.790000000037 ( 99.30208970800025  % wartości startowej)
```

Wykres dla bardziej zaawansowanego algorytmu (wyrysowane są przewidziane punkty przecięcia).




```
wartość startowa: 2395550.0, wartość końcowa : 2630383.62, różnica: 234833.6200000001 ( 109.80291039635992  % wartości startowej)
```

Analizując wykres dla podstawowego algorytmu można stwierdzić, że decyzje o kupnie / sprzedaży są spóźnione, przez co przy dynamicznie zmieniającej się cenie akcji mogą prowadzić do strat. Punkty przecięcia się wykresów $MACD$ i $SIGNAL$ niewątpliwie niosą informację o dogodnym momencie tranzakcji aczkolwiek poprzez wspomniane opóźnienie dokonywanie zakupów / sprzedaży w momencie przecięcia się wykresów może przynieść zyski jedynie w dłuższej perspektywie i tylko wtedy kiedy w ogólnym rozrachunku cena akcji rośnie, przy cenie oscylującej wokół pewnej stałej wartości tranzakcje przeprowadzane w momentach przecięcia się wykresów prowadzą do strat co widać ma przedstawionych wcześniej wykresach dla 1000 elementowych wektorów danych.

Algorytm przewidujący do przodu moment przecięcia się radzi sobie lepiej od prostego algorytmu przynosząc lekkie zyski (wyżej przedstawione przykłady dla wektorów 1000 elementowych) pod warunkiem trafnego dobrania parametrów, gdyż elminuje on częściowo wspomniane wcześniej opóźnienia.

Podsumowując Wskaźnik $MACD$ zdecydowanie może być przydatny w analizie technicznej, a miejsca jego przecięcia z $SIGNAL$ zdecydowanie wskazują na zmianę trendu, aczkolwiek wskaźnik ten sam w sobie nie jest raczej podstawą do poprawnego wyznaczania momentów kupna / sprzedaży w czasie rzeczywistym.

[ ]: