KALIBRACJA ZMIENNOśCI LOKALNEJ Z WYKORZYSTANIEM METODY REGULARYZACJI TICHONOWA

MARCIN KOBIERZYCKI

Spis treści

1. Wprowadzenie

Raport ten dotyczy realizacji numerycznej w pakiecie GNU Octave zagadnienia kalibracji zmienności lokalnej w modelu Blacka-Scholesa z wykorzystaniem metody regularyzacji Tichonowa zaproponowanej w [1] oraz w [2]. W rozdziale drugim przedstawiona zostanie koncepcja i założenia zagadnienia kalibracji zmienności lokalnej, a także schemat numeryczny. Kolejny rozdział poświęcony jest bezpośredniej prezentacji i omówieniu poszczególnych funkcji i bloków załączonego programu oraz formatu danych wejściowych. Natomiast czwarty rozdział posłuży ilustracji działania programu, prezentacji wyników oraz wykresów.

2. Analiza zagadnienia

Punktem wyjścia, z którego rozpoczynamy prezentację zagadnienia jest powszechnie znany model Blacka-Scholesa, który zapisuje się poniższym stochastycznym równaniem:

$$\frac{dS}{S} = \mu dt + \sigma dW, \tag{1}$$

gdzie S, to wartość aktywa podstawowego, μ to współczynnik dryfu, σ zmienność aktywa, natomiast W jest standardowym procesem Wienera. Dane empiryczne wskazują, że założenie powyższego modelu o stałości współczynnika σ jest oderwane od rzeczywistości, czyniąc model ten adekwatnym tylko do wyceny najpłynniejszych instrumentów pochodnych (waniliowych opcji europejskich). Jednym z możliwych sposobów rozwiązania tego problemu jest ujęcie zmienności w tym modelu jako funkcji wartości aktywa podstawowego, oraz czasu. W takiej wersji model prezentuje się następująco:

$$\frac{dS}{S} = \mu dt + \sigma(S(t), t) dW. \tag{2}$$

Będąc postawionym przed problemem wyceny instrumentów pochodnych mniej płynnych niż waniliowe opcje, niezbędna jest identyfikcja parametrów tego modelu. Współczynnik dryfu jest dostępny bezpośrednio z kwotowań rynkowych. Natomiast niedostępnym bezpośrednio jest powierzchnia zmienności. Odtwarzana zostaje ona z danych rynkowych przy założeniu, że skoro uogólniony model (2) wycenia insturmenty pochodne zgodnie z rynkiem (tzn. nie prowadząc do arbitrażu), to w szczególności musi być on zgodny z modelem podstawowym (1) w wycenie najpłynniejszych instrumentów pochodnych, jakimi są waniliowe opcje europejskie. Zatem powierzchnia zmienności lokalnej odtwarzana jest z kwotowań waniliowych opcji (tutaj: opcji kupna), które w praktyce rynkowej przedstawiane są w postaci *zmienności implikowanej* przez model (1) (czyli dane wejściowe: $\sigma_{imp}(K,T)$, gdzie K, to cena wykonania opcji, natomiast T, to termin zapadalności opcji). Tak postawione zagadnienie nazywa się *kalibracją zmienności lokalnej* w modelu Blacka-Scholesa i przynależy do ogólnej klasy zagadnień odwrotnych (ang. inverse problems).

Problem ten ujmujemy matematycznie następująco: Niech $\sigma_{imp}(K,T)$ będzie wspomnianą powierzchnią zmienności implikowanej przez podstawowy model BS. Niech $C(K,T)$ będzie tablicą cen waniliowych opcji europejskich uzyskaną z tablicy $\sigma_{imp}(K,T)$ za pomocą wzoru Blacka-Scholesa (BSFormula,wikipedia), tj. zwartego wzoru na cenę waniliowej opcji kupna typu europejskiego wyprowadzonego z podstawowego modelu BS. Ponadto niech $u(\sigma)$ będzie rozwiązaniem *równania Dupire*, czyli równania sprzężonego do równania Blacka-Scholesa (zob. [1], lub [2]) reprezentującym ceny kontraktów opcyjnych w uogólnionym modelu BS dla pewnej powierzchni zmienności reprezentowanej przez σ. Z formalnego punktu widzenia szukamy takiej powierzchni $\sigma(K,T)$, aby zminimalizować poniższy funkcjonał:

$$I(\sigma) = ||u(\sigma(K,T)) - C(K,T)||^2_{L^2(\Omega\times[0,T^*])} \tag{3}$$

Dodatkowym problemem, który pojawia się przy tym zagadnieniu jest fakt tego, iż tak rozumiany problem jest źle postawiony w tym sensie, że minimum nie jest osiągane tylko dla jednej powierzchni σ. W szczególności podczas szukania minimum tak postawionego funkcjonału może się okazać, że raz następuje zbieżność w kierunku jednego z minimów, po to aby później odbić w kierunku drugiego minimum. Jednym z podejść do rozwiązywania zagadnień odwrotnych powyższego typu jest metoda regularyzacji Tichonowa: Funkcjonał I należy wzbogacić o dodatkowy czynnik $\alpha||\sigma(K,T)-\sigma^*(K,T)||^2_{H^1(\Omega\times[0,T^*])}$, gdzie $\alpha > 0$, natomiast σ^* jest pewnym wstępnym zgadnięciem szukanej powierzchni zmienności (u nas będzie to po prostu zmienność implikowana). Uzyskujemy funkcjonał:

$$J_\alpha(\sigma) = ||u(\sigma(K,T)) - C(K,T)||^2_{L^2(\Omega\times[0,T^*])} + \alpha||\sigma(K,T) - \sigma^*(K,T)||^2_{H^1(\Omega\times[0,T^*])} \tag{4}$$

Dodany czynnik jest czynnikiem regularyzującym, wprowadzonym po to, aby w trakcie działania algorytmu zapewnić zbieżność do tego minimum, które znajduje się najbliżej σ^*. Sam współczynnik α posiada następującą interpretację: im większy, tym zachowanie algorytmu będzie bardziej konserwatywne i większy nacisk przy optymalizacji będzie kładziony na zachowanie bliskości do zgadnięcia początkowego σ^*. Ponadto autorzy obu publikacji poczyniają założenie a priori odnośnie do postaci powierzchni zmienności lokalnej, a mianowicie:

$$\sigma(K,T) = \sigma_1(K)\cdot\sigma_2(T) \tag{5}$$

Przy tym założeniu zagadnienie kalibracji zmienności lokalnej rozbija się na dwa podzagadnienia: początkową **identyfikację struktury uśmiechu**, czyli identyfikację postaci krzywej $\sigma_1(K)$ przy T ustalonym na końcowe T^* oraz na późniejszą **identyfikację struktury terminowej**, czyli krzywej $\sigma_2(T)$. W ninejszym raporcie skoncentrujemy się, podobnie jak w obu publikacjach, przede wszystkim na identyfikacji struktury uśmiechu. Sama koncepcja działania algorytmu jest następująca: ustalamy pewien poziom dokładności δ. Działamy dopóki ceny zwracane dla kolejnych powierzchni a przez równanie Dupire nie jest dostatecznie bliskie wektorowi cen C co do δ. W kolejnych iteracjach algorytmu luzujemy więzy narzucone na współczynnik regularności α, biorąc $\alpha_i := 2^{1-i}$. Funkcja ZnajdzMinimumFunkcjonalu$J_{2^{(1-i)}}$ jest funkcją minimalizującą wykorzstującą nieskończenie wymiarową metodę największego spadku.

```
delta = 0.0001;
i := 1;
a := a^*;
dopóki ||u(a) - C||^2 > delta
    b := znajdz_minimum_funkcjonalu J_{2^(1-i)} rozpoczynając z a;
    a := b;
    i := i+1;
koniec pętli dopóki
```

Istotnym faktem jest to, iż w trakcie działania funkcji ZnajdzMinimumFunkcjonalu$J_{2^{(1-i)}}$ należy policzyć pochodną Frecheta funkcjonału $J_{2^{(1-i)}}$ względem powierzchni zmienności, a następnie zlinearyzować ją rozwiązując równanie różniczkowe zwyczajne na gradient g, które prezentuje się w sposób następujący:

$$-g_{yy} + g = 2\int_0^T (u_{yy} - u_y)dt + 2\alpha(-da_{yy} + da), \tag{6}$$

gdzie $da = a - a^*$ (a oraz a^* są tutaj odpowiednio σ oraz σ^*, korzystając z pierwszych oznaczeń). Zostało to omówione w [1] z małą rozbieżnością na postać równania na gradient. **Bardzo ważnym** spostrzeżeniem na ten moment odnośnie do powyższego równania jest następujący fakt, który implikuje warunki brzegowe:

$$g(y_{min}) = g(y_{max}) = 0 \tag{7}$$

Konkretniej: ustalając w funkcji solve_dupire (omówionej w następnym rozdziale) warunki brzegowe:

```
        u(1,j) = fwy(1);
        u(M, j)= fwy(M);

#       u(1,j) = max(S0 - exp(y_min),0);
#       u(M, j)= max(S0 - exp(y_max),0);#==0
```

czyli zapisując na twardo równość rozwiązania równania Dupire oraz wektora cen opcji kwotowanych na rynku dla wartości końcowych y_{min} oraz y_{max} możemy patrzeć na to zagadnienie jako na problem przybliżenia do siebie dwóch krzywych, których końce zaczepione są w dwóch, tych samych punktach, z czego jedna nie podlega żadnym modyfikacjom (krzywa cen rynkowych), natomiast w drugiej możemy manipulować pewnym parametrem (zmienność dla krzywej zwracanej przez równanie Dupire).

3. Implementacja numeryczna

Przedstawiwszy w ogólnym zarysie zagadnienie kalibracji zmienności lokalnej metodą regularyzacji Tichonowa oraz schemat algorytmu numerycznego przechodzimy do prezentacji poszczególnych części programu. Najpierw omówiona zostanie struktura pliku z danymi wejściowymi **locvol_data.txt**:

```
M -- liczba węzłów dla zmiennej przestrzennej
# name: M
# type: global scalar
110

N -- liczba węzłów dla zmiennej czasowej
# name: N
# type: global scalar
130

delta -- poziom precyzji dla głównej pętli algorytmu
# name: delta
# type: global scalar
0.0001

delta2 -- poziom precyzji dla pętli w funkcji minimalizującej
metodą gradientową
# name: delta2
# type: global scalar
0.0001

S0 -- wartość początkowa aktywa podstawowego
# name: S0
# type: global scalar
3.84926137

Oczywiste
# name: STOPA_PROC
# type: global scalar
0.05

Oczywiste
# name: ZMIEN_IMPL
# type: global scalar
0.1

ITER_ZEWN - Górne ograniczenie na liczbę iteracji w głównej
pętli algorytmu
# name: ITER_ZEWN
# type: global scalar
4

ITER_WEWN - Górne ograniczenie na liczbę iteracji w funkcji
```

```
minimalizującej metodą gradientową
# name: ITER_WEWN
# type: global scalar
3
```

Program standardowo działa przy założeniu stałej struktury terminowej stopy procentowej (r=0.05) oraz stałości zmienności implikowanej (imp_vol = 0.1). Przechodzimy do struktury pliku wykonywalnego **locvol.m**:

y to wektor reprezentujący siatkę przestrzenną dla zdyskontowanych cen wykonania K(zlogarytmowaną):

```
global y = log(K * DISC(T_y(10)));
global y_int = log(K_int * DISC(T_y(10)));
global T_y_int = interp1(1:10, T_y, linspace(1,10,N));
global y_min = min(y(:,10));
global y_max = max(y(:,10));
global dM = (y_max - y_min)/(M-1);
global dN = 1 /(N-1);
```

Funkcja licząca cenę waniliowej europejskiej opcji call w standardowym modelu Blacka-Scholesa:

```
#Wzor Blacka-Mertona-Scholesa
# Pamietac, ze strike to K, a nie y
# T w latach
function blackie = bs(S0, K, T, r, sigma)
    d1 = (log(S0/K) + (r+0.5*sigma^2)*T)/(sigma*sqrt(T));
    d2 = d1 - sigma*sqrt(T);
    blackie = normcdf(d1, 0, 1)*S0 - normcdf(d2, 0, 1)*K*exp(-r*T);
endfunction
```

Poniżej znajduje się funkcja rozwiązująca równanie Dupire zapisane w zmiennych logarytmicznych tak, jak w (9) w [2]**. a to krzywa w zmiennych logarytmicznych reprezentująca zmienność dla której rozwiązywane jest równanie. Wykorzystany został schemat implicit.**

```
#Funkcja rozwiazujaca rownanie Dupire schematem implicit. a to wektor 1..M
function z = solve_dupire(a)
    global S0;
    global M;
    global dM;
    global dN;
    global N;
    global y_min;
    global y_max;
    global T_y;
    global fwy;

    u = zeros(M, N);

    for i = 1:M
#        u(i, 1) = max(S0 - exp(y_min +(i-1)*dM)/DISC(T_y(10)),0);
        u(i, 1) = max(S0 - exp(y_min +(i-1)*dM),0);
```

```
    endfor;

    for j = 2:N
        u(1,j) = fwy(1);
#        u(1,j) = 1.62222493467445;
#        u(1,j) = max(S0 - exp(y_min),0);
#        u(M, j)= max(S0 - exp(y_max),0);#==0
        u(M, j)= fwy(M);
#        u(M, j)= 0.0104580210051421;#==0
#        u(1,j) = max(S0 - exp(y_min)/DISC(T_y(10)),0);
#        u(M, j)= max(S0 - exp(y_max)/DISC(T_y(10)),0);
    endfor;

#IMPLICIT

vec = u(2:(M-1),1);

for ny = 2:N
    Q = zeros(M-2,M-2);
    for i = 1:(M-3)
        Q(i,i) = (1 - dN*a(i+1)*(-(dM)^(-1)-2*(dM)^(-2)));
        Q(i,i+1) = -dN*a(i+1)*(dM)^(-2);
        Q(i+1,i) = -dN*a(i+2)*((dM)^(-1) + (dM)^(-2));
    endfor
    Q(M-2,M-2) = (1 - dN*a(M-1)*(-(dM)^(-1)-2*(dM)^(-2)));

    d = zeros(1,M-2);
    d(1) = dN*a(2)*((dM)^(-1) + (dM)^(-2)) * (u(1,ny));
    d(M-2) = dN*a(M-1)*(dM)^(-2) * (u(M,ny));
    d = d';
    vec_new = Q \ (vec + d);
    u(2:(M-1),ny) = vec_new;
    vec = vec_new;
endfor

z = u;

endfunction;
```

W funkcji solve_add_eqn rozwiazujemy sztuczne rownanie, ktore pojawilo sie przy liczeniu gradientu w (5.2) w [1]. Tutaj najpierw jest podstawienie S:=a(y)R(y) a później zamieniamy czas jak ponizej i w takiej postaci zwracane jest rozwiazanie. Pozniej przy liczeniu gradientu trzeba będzie pamiętać o tej zamianie czasu. Schemat implicit. init_f to funkcja r(y), jak w publikacji, natomiast a to zmienność w zmiennych logarytmicznych.

```
function zz = solve_add_eqn(init_f, a)
global S0 M N dM dN;
```

```
S = zeros(M, N);
S(:, 1) = init_f .* a;


vec = S(2:(M-1),1);

for ny = 2:N
    Q = zeros(M-2,M-2);
    for i = 1:(M-3)
        Q(i,i) = (1 - dN*a(i+1)*((dM)^(-1)-2*(dM)^(-2)));
        Q(i,i+1) = -dN*a(i+1)*(dM)^(-2);
        Q(i+1,i) = dN*a(i+2)*((dM)^(-1) - (dM)^(-2));
    endfor
    Q(M-2,M-2) = (1 - dN*a(M-1)*((dM)^(-1)-2*(dM)^(-2)));

    d = zeros(1,M-2);
    d(1) = -dN*a(2)*((dM)^(-1) - (dM)^(-2)) * (S(1,ny));
    d(M-2) = dN*a(M-1)*(dM)^(-2) * (S(M,ny));
    d = d';
    vec_new = Q \ (vec + d);
    S(2:(M-1),ny) = vec_new;
    vec = vec_new;
endfor

zz = S;

endfunction;
```

Ponizej znajduje sie funkcja zaczerpnieta z pakietu ode-bvp sluzaca do rozwiazywania rownan drugiego rzedu z podwojnym warunkiem brzegowym.

```
function sol = lfdif(v)

  global dM;
  global M;
  global N;
  global y_int y_min y_max;

  h = dM;
  x = y_int(2:(M-1), N)';
  #a = 2 + h^2 * q(x);
  a = 2 + h^2 * ones(1, M-2);
  b = -1 + (h / 2.) * zeros(1, (M-2));
  c = -1 - (h / 2.) * zeros(1, (M-2));

  A = spdiag(c(2 : (M-2)), -1) + spdiag(a) + spdiag(b(1 : M-3), 1);

  d(1) = -h^2 * v(1) + (1 + (h / 2.) * 0) * 0;
```

```
  d(2 : M-3) = -h^2 * v(2 : M-3);
  d(M-2) = -h^2 *v(M-2) + (1 - (h / 2.) * 0) * 0;

  x = vertcat(y_min, x', y_max);
#  y = vertcat(0, A\d', 0);
  y = vertcat(1, A\d', 0);

  sol = horzcat(x, y);
endfunction

# Kwadrat normy H1

function ution = H1norm2(b)
    global M;
    global dM;
    global y_min y_max;

    c = b;
    for i = 1:(M-1)
        c(i) = (b(i+1) - b(i))/dM;
    endfor;
    c(M) = (b(M) - b(M-1))/dM;

    ution = trapz(linspace(y_min, y_max, M), b.^2 + c.^2);

endfunction;

#pierwsza pochodna przestrzenna

function rep = diff1(b)
    global dM M;
    c = b;
    for i = 1:(M-1)
        c(i) = (b(i+1) - b(i))/dM;
    endfor;
    c(M) = (b(M) - b(M-1))/dM;

    rep = c;
endfunction;

#druga pochodna przestrzenna

function rep = diff2(b)
    global dM M;
    c = b;
    for i = 1:(M-1)
        c(i) = (b(i+1) - b(i))/dM;
    endfor;
```

```
    c(M) = (b(M) - b(M-1))/dM;

    for i = 1:(M-1)
        c(i) = (c(i+1) - c(i))/dM;
    endfor;
    c(M) = c(M-1);

    rep = c;
endfunction;
```

Ponizej znajduje sie kod dla odtworzenia struktury usmiechu. Zaczynamy od obliczenia wektora cen z danych rynkowych (wektor f) dla zmiennosci implikowanej przy koncowym czasie T. Tutaj cc oznacza wektor, ktory dla i-ej iteracji glownej jest wektorem minimalizujacym poprzedni funkcjonal $J_{\alpha-1}$. Dla pierwszej iteracji, jest to po prostu zmiennosc implikowana dla koncowego czasu.

```
f = zeros(1,M);
imp_vol(:, N) = ZMIEN_IMPL;

for i = 1:M
    f(i) = bs(S0, K_int(i, N), T_y_int(N),
              r_cont(r(T_y_int(N), T_y, r_d)), imp_vol(i, N));
endfor;

#plot(log(K_int(:, N)*DISC(T_y(10))),f, "r");

# Bierzemy odcinek [y_min, y_max] w zmiennych logarytmicznych (czyli w)
# i interpolujemy wektor cen rynkowych (wektor f) przy takich zmiennych.
# Wynik zapisujemy w fwy

w = linspace(y_min, y_max, M);
q = exp(w)/DISC(T_y(10));
q(1) = K_min;
q(M) = K_max;
fwy = interp1(K_int(:,N), f, q);
#plot(w, fwy)


#plot(0.5*imp_vol(:,N).^2)

dist = 3*delta;

############WAZNE###############
# W pierwszej publikacji co do wyznaczania ciagu wspolczynnikow
# regularyzujacych autorzy korzystali # z kryterium Morozova.
# W drugiej napisali, ze wystarczy wziac alfa_0 dowolne, a alfa_i
# = alfa_{i-1}/2. Tak tez to zaimplementowalem.
```

```
count = 1;
cc = 0.5*imp_vol(:,N).^2;
astar = cc;


#astar, to poczatkowy strzal. W naszym przypadku po prostu zmiennosc
#implikowana. astara nie zmieniamy w trakcie dzialania iteracji

#glowna petla minimalizujaca
while ((dist > delta) && count <=ITER_ZEWN) # (4,3) daje 1,41e-4 M110 N130;

    if count >=2
        odp_old = odp;
    endif

    figure(44+count);
    hold on;
    plot(w(2:(M-1)), cc(2:(M-1)), "g");
    hold off;

    odp = solve_dupire(cc);
    if count == 9999
        smile = cc;
        save "smilestr.txt" smile
    endif

# tutaj zapewniamy sobie graficzna ilustracje tego, czy i jak zbiegaja
# krzywe do danych rynkowych

    if count >=2
#     if count == 7744
        figure(count);
        hold on;
        plot(w, fwy, "r");
        plot(w, odp_old(:,N)', "b");
        plot(w, odp(:,N)', "g");
        hold off;
    endif

#dist - odleglosc w L^2 danych rynkowych od cen odpowiadajacych obecnej
# strukturze usmiechu

    dist = trapz(w, (odp(:,N)'-fwy).^2);
    NAJWAZNIEJSZA_ODLEGLOSC = dist

#incount -- licznik iteracji wewnetrznej petli
    incount = 0;
```

```
    x_1 = cc;

    printf("PETLA NUMER %d:\n", count);

#wewnetrzna petelka sluzaca do minimalizacji funkcjonalu J_{alpha_i}.
#Metoda: prosta metoda gradientowa. (steepest descent method)
    do
        x_0 = x_1;

        odp_x = solve_dupire(x_0);
        add = solve_add_eqn(odp_x(:,N)'-fwy, x_0');

# zeby policzyc gradient mamy (Ponizej wzoru (5.2) w pierwszej publikacji
# (u_yy i u_y)R = u_\tau * S / (a^2). (przy czym nalezy pamietac, ze S mamy
# w odwroconym czasie w programie)

        aa = x_0 .* x_0;

# ss - pochodna czasowa rozwiazania rnia Dupiry
        ss = odp_x;

        for i = 1:M
            ss(i,N) = (ss(i,N) - ss(i,N-1))/dN;
        endfor;

        for j = 1:(N-1)
            for i = 1:M
                ss(i,j) = (ss(i,j+1) - ss(i,j))/dN;
            endfor;
        endfor;

# s - wektor reprezentujacy to, co mamy w rniu zwyczajnym na gradient
# po prawej stronie. (Wzor o jeden ponizej od (5.2) w pierwszej publikacji)
        s = zeros(1, M);

        for i = 1:M
            s(i) = 2*trapz(linspace(0, 1, N), ss(i,:).*add(i, N:-1:1)/aa(i))
                   + 2*(2^(count-1))*(-diff2(x_0'-astar')(i) + x_0(i) - astar(i));
        endfor;

        grad = lfdif(s(2:(M-1)))(:,2);
        x_1 = x_0 - grad/100;#100

        funkcjonal_w_x_0 = trapz(w, ((solve_dupire(x_0)(:,N))' -fwy).^2) +
                           H1norm2(x_0' - astar')*(2^(count-1))
        funkcjonal_w_x_1 = trapz(w, ((solve_dupire(x_1)(:,N))' -fwy).^2) +
                           H1norm2(x_1' - astar')*(2^(count-1))
        odleglosc_x_0_i_x_1_w_H1 = H1norm2(x_0' - x_1')
```

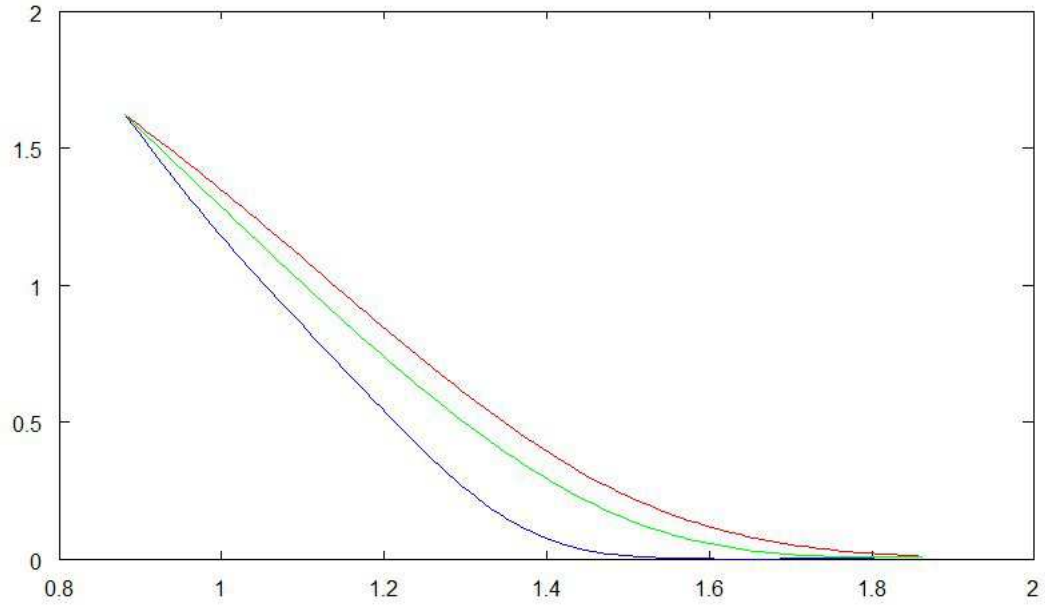
```
 #       if (count == 5)
 #           figure(count+incount);
 #           hold on;
 #           plot(w, grad, "b");
 #           hold off;
 #       end
        printf("-----------\n");

        incount++;
    until (H1norm2(x_0' - x_1') < delta2 || incount >ITER_WEWN) #3

    cc = x_1;
    ++count;
endwhile;

smile = cc;
```

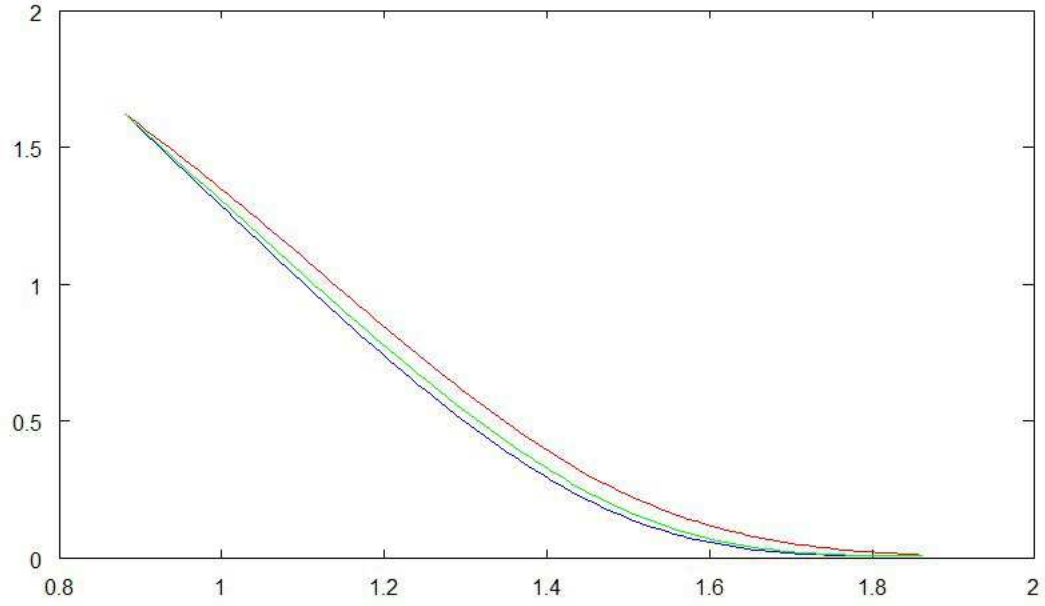
4. Prezentacja wyników

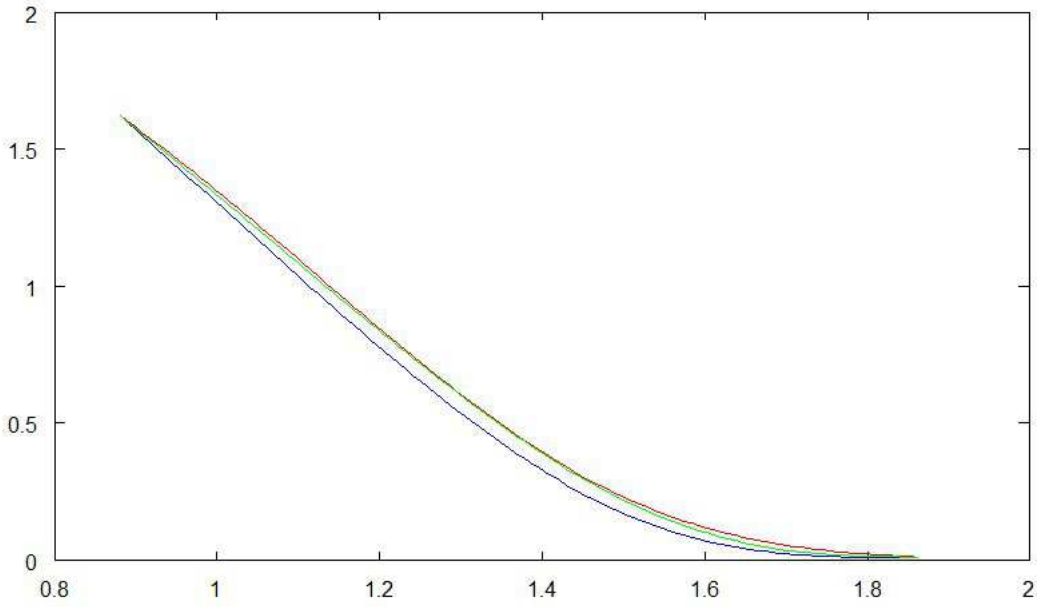
Program uruchomiony ze standardowymi parametrami **r = 0.05, zmienność implikowana = 0.1 (stała)** zwraca w kolejnych **4** iteracjach głównej pętli oraz górnym ograniczeniem na pętlę w metodzie gradientowej równym **3** i liczbą węzłów przestrzennych, **M = 110** oraz liczbą węzłów czasowych **N = 130** następujące wartości będące odległościami w L^2 pomiędzy danymi zwracanymi przez równanie Dupire oraz krzywą cen rynkowych:

```
0.0468147521633026
0.00578290435058766
0.00264793198908022
1.41846306691047e-004
```

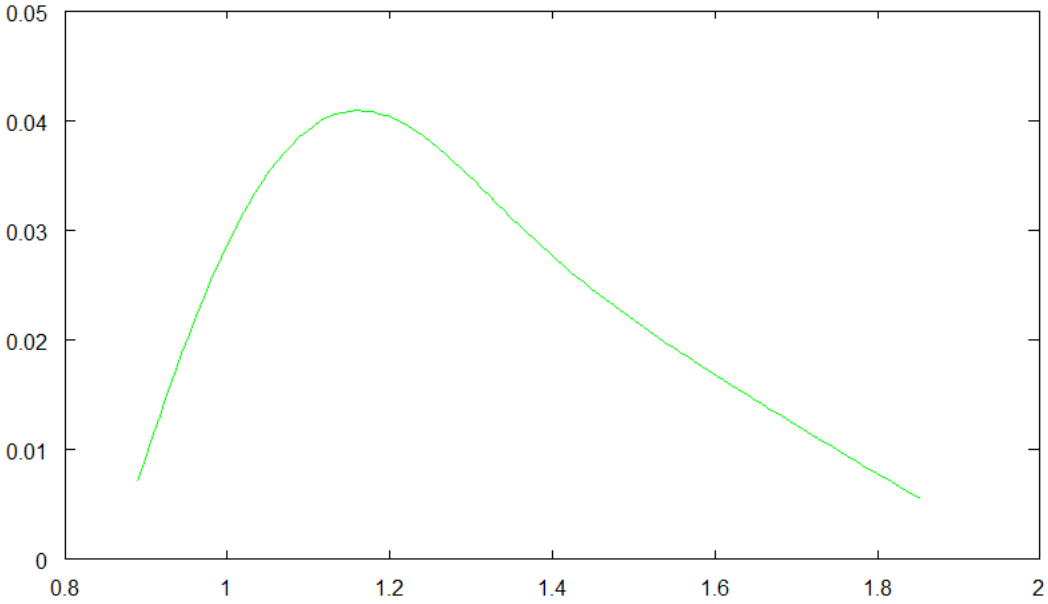
Zaproponowana metoda w [1] i [2] zapewnia zbieżność tylko do czwartej iteracji osiągając dokładność co do punktu bazowego, która to nie jest dostateczną w praktyce bankowej. W kolejnych iteracjach minimalizowana odległość nie maleje, a nawet rośnie. Manipulując liczbami węzłów M oraz N można co prawda poprawić ostatnią dokładność na przykład z 1.41846306691047e-004 na ok. 1.16e-004 biorąc M=200, N=220, jednak nie zmienia to faktu, iż zbieżność ma miejsce tylko do czwartej iteracji włącznie.
Poniższe wykresy prezentują kolejne cztery iteracje programu, gdzie kolorem czerwonym oznaczona została krzywa cen rynkowych C, niebieskim krzywa zwrócona przez równanie Dupire w poprzedniej iteracji, natomiast zielonym krzywa zwrócona przez równanie Dupire dla obecnej iteracji:

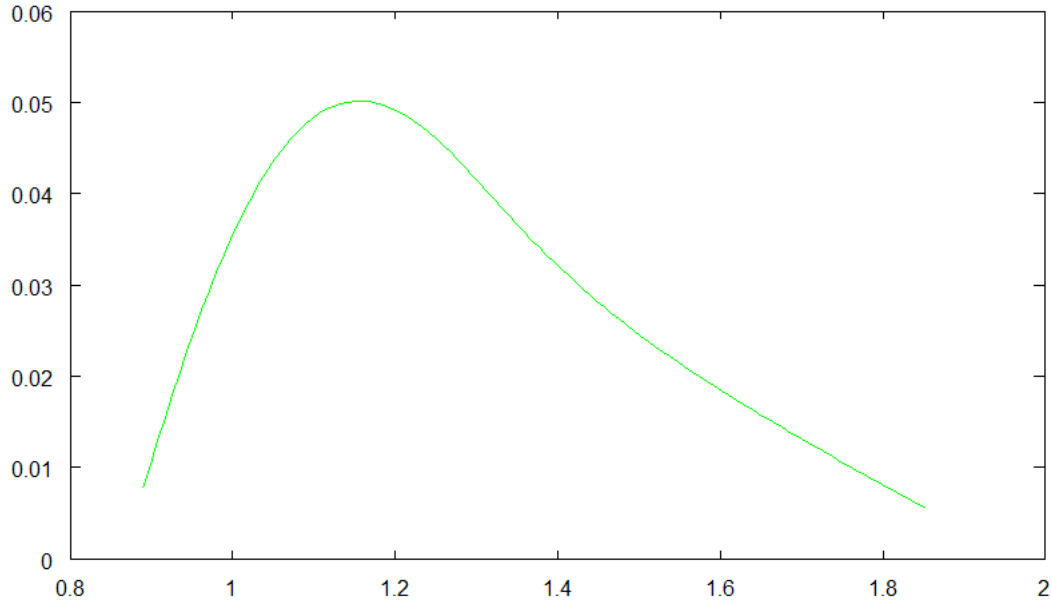

2
1.5
1
0.5
0
0.8
1
1.2
1.4
1.6
1.8
2

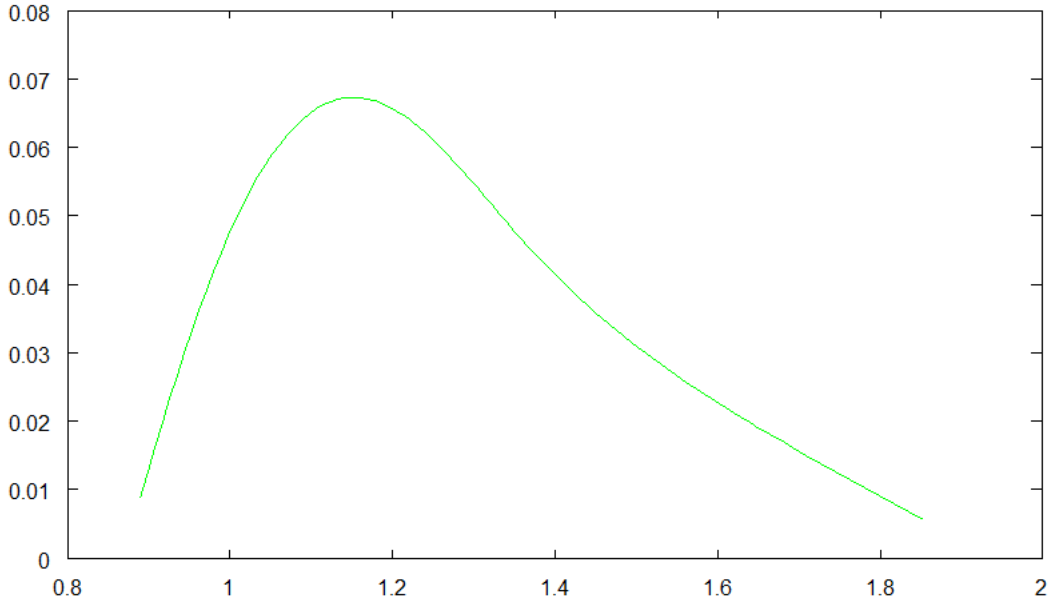

2
1.5
1
0.5
0
0.8
1
1.2
1.4
1.6
1.8
2



Pomijając pierwszą, stałą krzywą zmienności w kolejnych krokach krzywe struktury uśmiechu prezentują się następująco:







5. Lista załączników

(1) "locvol.m" - plik źródłowy programu
(2) "locvol_data.txt" - plik z danymi wejściowymi programu
(3) Dwie publikacje wspomniane w LITERATURA

Literatura

[1] Herbert Egger, Heinz W. Engl, "Tikhonov Regularization Applied to the Inverse Problem of Option Pricing: Convergence Analysis and Rates", Inverse Problems
[2] Herbert Egger, Torsten Hein, Bernd Hofmann "On decoupling of volatility smile and term structure in inverse option pricing", Inverse Problems