České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Diplomová práce

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Diplomová práce

Dynamické projevy struktury fotonu v rámci kvantové chromodynamiky

Jiří Hejbal

Katedra fyziky
Akademický rok: 2003/2004
Vedoucí práce: Prof. Jiří Chýla, CSc., FZÚ AV ČR

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedeném v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Jiří Hejbal

Poděkování

Děkuji panu Prof. Chýlovi za ochotu a vstřícnost, kterou mi věnoval během všech konzultací a za velmi cenné rady a připomínky při tvorbě této práce.

Abstrakt

Předložená práce seznamuje s konceptem struktury fotonu v rámci kvantové chromodynamiky (QCD). Náš přistup bude obdobný popisu struktury hadronů v QCD, kterou se zabýváme v první tématické části (kapitola 2. - 5.). Na začátku budou vysvětleny základní pojmy partonového modelu, zavedeny budou distribuční funkce partonů v hadronech, kterými zmíněnou strukturu popisujeme. Značná pozornost bude věnována evolučním rovnicím QCD pro distribuční funkce - jejich odvození a metodám jejich řešení. Podrobněji se potom budeme zabývat řešením evolučních rovnic pro tzv. nesingletní distribuční funkce kvarků v nukleonech ve vedoucím řádu QCD metodou inverzní Mellinovy transformace. Získaná řešení a představené postupy užijeme v závěru této části k analýze experimentálních dat z rozptylu neutrin na nukleonech.

Pojmy a postupy představené v první části budou zásadní pro formulaci konceptu struktury fotonu (kapitola 6. - 8.). Nejprve bude představen popis struktury fotonu pomocí leptonových distribučních funkcí v tzv. Weizsäcker - Williamsově aproximaci, podobně bude představen popis kvarkové struktury fotonu - obojí v rámci kvantové elektrodynamiky. Věnovat se budeme extrakci zmíněných distribučních funkcí z informace o obecné srážce $e^{\pm}e^{-}$. Zvláštní pozornost pak bude věnována evolučním rovnicím pro distribuční funkce partonů ve fotonu, které se užívají pro popis struktury fotonu v rámci QCD. Získané výsledky a popsané procedury budou aplikovány na analýzu dat o struktuře fotonu.

Tento text je doplněn dokumentací o programu (napsaného v jazyce C++), umožnujícího řešit evoluční rovnice pro nesingletní distribuční funkce pomocí inverzní Mellinovy transformace (kapitola 9.). Zvláštní kapitola je věnována praktickým metodám při ověřování správnosti při takových výpočtech (kapitola 10.).

Obsah

Kapitola 1

Experimenty částicové fyziky

Mezníkem v moderní éře fyziky elementárních částic se stal objev elektronu J.J. Thomsonem v roce 1897. Byl to první přímý důkaz o existenci částic menších nežli atomy. Tato skutečnost byla podstatnou pro představu o elementárních částicích. Jestliže před rokem 1897 byly za základní částice tvořící hmotu považovány atomy, v průběhu 20. století tuto roli zastávala atomová jádra, potom protony, neutrony a další hadrony (π, ρ, K, Λ, ...)

Dnes jsou za elementární částice považovány kvarky a leptony, kdežto hadrony považujeme za složené z kvarků. Elektron považujeme za elementární částicí již více než 100 let a v dohledné době jí pro nás zůstane i nadále. Není všem vyloučeno, že by i tyto částice měly vnitřní strukturu, která by mohla být zjištěna na základě některého z budoucích experimentů.

Experimenty hrají ve fyzice klíčovou úlohu. Mezi hlavní druhy experimentů užívané ve fyzice elementárních částic patří srážky vysokoenergetických částic. Veškeré efekty a jevy spojené se srážkami se studují nepřímo pomocí detekce částic vzniklých během srážky. Rozvoj částicové fyziky pak jde ruku v ruce s technickou úrovní urychlovačů a detektorů. Důvod plyne z Heisenbergovy relace neurčitosti. Pro rozlišení malých objektů je potřeba velkých předaných energií. Z tohoto důvodu je pak částicová fyzika označována jako fyzika vysokých energií.

Ke studovaným procesům patří srážky leptonů (nabitých i neutrálních) s hadrony a leptonů a hadronů s hadrony. Elektron-protonové srážky s vysokým rozlišením patří k hlubokonepružným rozptylům a umožňují studovat vnitřní strukturu protonu. Tato znalost je velmi důležitá pro pochopení silné interakce mezi kvarky a gluony. Proton-antiprotonové reakce dnes poskytují největší energie, ale protože jde o srážku dvou částic s vnitřní strukturou, jsou také v mnohém rozmanité.

Urychlovače a detektory užívané v částicové fyzice jsou velmi nákladná a složitá zařízení. Z tohoto důvodu jsou experimenty soustředěny jen na několika místech světa. K nejvýznamnějším laboratořím patří CERN v Ženevě, Fermilab blízko Chicaga, SLAC ve Stanfordu a DESY v Hamburgu.

Kapitola 2

Standardní model

Úspěch částicové fyziky je z velké části zásluhou formulování Standardního modelu. Podle něj je veškerá hmota složena ze sady částic se spinem 1/2: kvarků a leptonů. Tyto částice navzájem interagují elektroslabě a silně, okrajově se zde podílí i gravitace. Jsou rozděleny do tří generací. Každý ze šesti druhů kvarků se odlišuje kvantovým číslem, zvaným vůně. Elementární částice mají jisté vlastnosti, které definují jejich chování při interakcích. Příkladem může být hmotnost, která je určující při gravitaci. Podobně je elektrický náboj zdrojem elektromagnetické interakce, slabý isospin souvisí se slabou interakcí a barevný náboj je zdrojem silné interakce.

První generace leptonů a kvarků se sestává z elektronu, elektronového neutrina a up a down kvarku. Její důležitost vyplývá z faktu, že veškerá hmota, která nás obklopuje je složena právě z těchto částic. Protony a neutrony v atomovém jádře jsou složeny z u and d kvarku, atomy jsou pak složeny z protonu nebo jádra obklopeného elektrony. Neutrina interagují velmi slabě a v běžném makroskopickém životě mají velmi slabý efekt. Velký význam však mají v některých jaderných procesech - lze pomocí nich popsat například beta rozpad. Dále mají význam v některých astrofyzikálních procesech za extrémních teplot a tlaků, například v centru hvězd. Když hvězda exploduje v supernovu, uvolní se velké množství energie vyzářené právě neutriny.

Druhé dvě generace nehrají tak důležitou roli jako generace předchozí, neboť se, s výjimkou neutrin, velmi rychle rozpadají. Nicméně, tyto dvě generace částic jsou velmi důležité v experimentech částicové fyziky (ověření platnosti standardního modelu).

2.1 Základní interakce

Základní síly ovlivňující interakci mezi částicemi jsou gravitace, elektroslabá a silná interakce. Gravitace byla popsána Einsteinovou obecnou teorií relativity a je důležitá při makroskopických procesech. Elementární částice ovlivňovala významně pouze při extrémně vysokých teplotách a tlacích v ranném stadiu vesmíru a v blízkosti velmi hmotných objektů (např. černé díry).

Elektromagnetická síla je v současnosti zahrnována do elektroslabé interakce.

Standardní model je kvantovou teorií pole, tedy silová pole jsou kvantována a interakce jsou zprostředkovány výměnou částic. Elektromagnetická interakce je zprostředkována fotony, slabá interakce bosony W^+, W^- a Z^0, silná interakce gluony. Všechny jsou bosony se spinem 1.

Slabá interakce popisuje rozpad leptonů a kvarků z druhé a třetí generace na své lehčí příbuzné z první generace. Podobně je slabou interakcí popsán i radioaktivní beta rozpad neutronů a jader. Intermediální bosony W a Z byly objeveny v ženevském CERN v roce 1983.

Na malých vzdálenostech jsou elektromagnetická a slabá interakce sjednocené a lze je popsat jednotnou teorií. Elektroslabá teorie předpovídá, že reálné částice získávají svou hmotnost mechanismem, v němž hraje důležitou roli Higgsova částice, která zatím nebyla objevena. Očekává se, že experimentálně bude otázka Higgsova bosonu definitivně rozřešena do roku 2010 na urychlovači LHC (Large Hadron Collider) v CERN.

Silná interakce se uplatňuje mezi částicemi s barevným nábojem. Kvarky mezi sebou silně interagují výměnou osmi gluonů. Protože i gluony jsou nositeli barevného náboje interagují silně i vzájemně mezi sebou.

Kvark každé vůně existuje ve třech různých stavech, nazývaných "barvy". Fyziky byly poeticky pojmenovány jako červený, zelený a modrý. Skládání "barevných" kvarků do stabilních systémů je totiž podobné skládání základních barev. Podle kvantové chromodynamiky odpovídají stabilní fyzikální stavy barevně neutrálním kombinacím. Antikvarky existují ve třech barevných stavech nazývaných antibarvami. Složením náboje a příslušného antináboje vznikne náboj barevně neutrální.

Jedním z problémů, který v rámci standardního modelu čeká na objasnění, je otázka proč doposud nebyly pozorovány volné kvarky. Bylo experimentálně ověřeno, že kvarky jsou vždy uzavřeny v hadronu.

2.2 Hluboko nepružný rozptyl leptonů

Jedním z nejvhodnějších prostředků ke zkoumání vnitřní struktury subatomárních částic jsou experimenty využívající hluboký nepružný rozptyl leptonů. Ze zjištěných energetických a uhlových rozdělení rozptýlených leptonů se pak rekonstruuje struktura zkoumané částice. Například v případě studia struktury protonu jde o proces

$$l(k) + \text{proton}(P) \rightarrow l^{'}(k^{'}) + X; \quad l, l^{'} = e, \mu, \nu_e, \nu_\mu, \tag{2.1}$$

kde X značí konečný stav, respektující zákony zachování, symboly v závorkách pak značí čtyřhybnosti příslušných částic. Speciálním případem (2.1) je pružný lepton-nukleonový rozptyl, kdy X =proton a $l = l^{'}$. Rozlišujeme dva typy hluboko nepružných rozptylů. Jedním jsou tzv. "neutrální proudy", které jsou zprostředkovány buď fotonem nebo (neutrálním) intermediálním vektorovým bosonem (IVB) Z. Druhým jsou potom "nabité proudy". Tyto prosesy jsou zprostředkovány výměnou nabitých intermediálních vektorových bosonů $W^{\pm}$. Nejčastěji používané proměnné, popisující procesy (2.1) jsou (při zanedbávání hmotnosti

leptonů):

$$S \equiv (k+P)^2 = M_p^2 + 2kP = M_p(2E_{lab} + M_p), \tag{2.2}$$

$$Q^2 \equiv -q^2 \equiv -(k-k')^2 = 2kk' = 4EE' \sin^2\left(\frac{\theta}{2}\right), \tag{2.3}$$

$$y \equiv \frac{qP}{kP} = \frac{E_{lab} - E'_{lab}}{E_{lab}}, \tag{2.4}$$

$$x \equiv -\frac{q^2}{2Pq} = \frac{Q^2}{Q^2 + W^2 - M_p^2}, \tag{2.5}$$

$$W^2 \equiv (q+P)^2 = \frac{Q^2(1-x)}{x} + M_p^2. \tag{2.6}$$

Proměnná W má jasnou interpretaci jako invariantní hmotnost hadronového systému X, vzniklého absorpcí fotonu nebo IVB terčovým nukleonem. Existují dvě základní referenční soustavy, ve kterých se proces (2.1) obvykle popisuje. Jednou z nich je laboratorní soustava, kde je ostřelovaný proton v klidu a těžišťová soustava, kde ostřelující lepton a proton mají opačné hybnosti téže velikosti. Poznamenejme, že všech pět veličin výše definovaných jsou relativistickými invarianty. V laboratorní soustavě mohou být vyjádřeny pomocí rozptylového úhlu θ_{lab} a energií E_{lab} *a* E'_{lab} nalétávajícího a rozptýleného leptonu. Veličina S určuje počáteční stav elektronu a protonu. Spolu s dalšími třemi veličinami, x, y a Q^2 a s azimutálním úhlem ϕ, popisují konečný stav elektronu. Pouze dvě z těchto veličin jsou nezávislé, zbylé lze vyjádřit pomocí ostatních. Můžeme tedy volit některou z dvojic proměných $(x,\ y)$, $(x,\ Q^2)$,$(y,\ Q^2)$ nebo $(E'_{lab},\ \theta_{lab})$, která pak společně s ϕ plně určuje konečný stav rozptýleného elektronu. Veškeré úvahy činíme za předpokladu, že částice v experimentu jsou nepolarizované. Pro pružný rozptyl elektronu na protonu je situace jednodušší, neboť v tomto případě $x = 1$ (v posledním výrazu v (2.5) totiž můžeme položit $W^2 = M_p^2$) a tedy pro určení koncového stavu postačuje jediná proměnná.

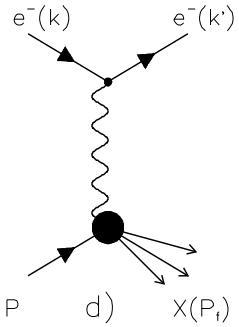
Pokud užíváme termín hluboce nepružný rozptyl, máme tím na mysli kinematickou oblast, kdy jsou invarianty Q^2 a Pq mnohem větší než kvadrát hmotnosti protonu M_p

$$Q^2 \gg M_p^2, \quad \nu \equiv Pq \gg M_p^2, \tag{2.7}$$

ale poměr $x = Q^2/2Pq$ zůstává konečný. Takové omezení se nazývá Bjorkenovo.

Nejjednodušší měřený typ účinného průřezu je takzvaný inkluzivní účinný průřez, který popisuje pouze elektron v koncovém stavu. V tomto případě půjde o vysčítané účinné průřezy přes všechny možné konečné stavy daného hadronu. Tento proces je popsán diagramem na obr. 2.1 Interakce je uskutečněna výměnou jednoho fotonu. Detailní struktura konečného stavu hadronu je v tomto diagramu symbolicky znázorněna "kapkou" v protonovém vrcholu. X symbolizuje sumu přes všechny možné hadronové stavy. Předpoklad o výměně fotonu, který je dobrou aproximací pro $Q^2 \ll M_W^2, M_Z^2$, znamená, že účinný průřez bude možné zapsat pomocí leptonového tensoru a tensoru popisující hadronový vrchol.

$$d\sigma \sim L_{\mu\nu} W^{\mu\nu}. \tag{2.8}$$



Obrázek 2.1: Feynamanův diagram popisující hluboko nepružný rozptyl elektronu na protonu

Leptonový tensor je velmi dobře popsán kvantovou elektrodynamikou (QED). Jeho tvar je dán vztahem:

$$L_{\mu\nu} = 2[k'_\mu k_\nu + k'_\nu k_\mu + (q^2/2)g_{\mu\nu}]. \tag{2.9}$$

Na druhé straně pro hadronový tenzor předpokládáme, že hlavní roli sehraje silná interakce. Pro diferenciální účinný průřez potom platí:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x\mathrm{d}y} = \frac{4\pi\alpha^2(2kP)}{Q^4}\left[\left(1-y-\frac{M_p^2}{S}\right)F_2(x,Q^2)+\frac{1}{2}y^2 2xF_1\left(x,Q^2\right)\right] \tag{2.10}$$

kde $S \equiv (k+P)^2$ je kvadrát celkové energie těžišťového systému. Neznámé funkce $F_i(x,Q^2)$ se nazývají nepružné formfaktory nebo častěji strukturní funkce protonu. Souvisí s obecným tvarem tenzoru $W_{\mu\nu}$:

$$W_{\mu\nu} = C_1(Q^2,pq)g_{\mu\nu} + C_2(Q^2,pq)p_\mu p_\nu + C_3(Q^2,pq)q_\mu q_\nu + C_4(Q^2,pq)(p_\mu q_\nu + p_\nu q_\mu) \tag{2.11}$$

jenž plyne z požadavku lorentzovské invariance a zachování parity. Podmínka kalibrační invariance nám dále dovoluje vyjádřit C_3 a C_4 pomocí C_1, C_2 a definovat strukturní funkce

$$F_1 \equiv C_1 \quad F_2 \equiv \frac{Pq}{M_p^2}C_2 \tag{2.12}$$

V rovnici (2.10) jsou vyjádřeny v proměnných $x,\ Q^2$, zatímco diferenciály na levé straně rovnice v proměnných $x,\ y$. Bylo tak učiněno s ohledem na to, že libovolnou z proměnných $x,\ y,\ Q^2$ lze, pro dané S, vyjádřit pomocí zbývajících dvou, přesněji $Q^2 = Sxy$.

Experimenty s hlubokým nepružným rozptylem elektronů na protonech prováděné ve SLAC od roku 1967 přinesly velmi překvapivé výsledky ukazující na to, že

- strukturní funkce $F_1(x,Q^2)$ i $F_2(x,Q^2)$ závisí, na rozdíl od pružných formfaktorů protonu, na Q^2 velmi slabě - jev zvaný "škálování",
- zhruba platí tzv. Callan-Grossova relace $F_2 \doteq 2xF_1$.

2.3 Partonový model

Pro tyto překvapivé experimentální výsledky navrhl Feynman jednoduché vysvětlení založené na představě, že proton je složen z menších částic zvaných partony. Podle Feynmana probíhá hluboký nepružný rozptyl na protonu tak, že se elektron pružně rozptýlí na jednom z partonů nesoucím zlomek x jeho celkové hybnosti p. Vzájemná interakce partonů v protonu se přitom zanedbává. Platnost Callan-Grossovy relace znamenala, že partony mají spin 1/2.

Pro každý parton uvažujeme distribuční funkci f_i(x), popisující pravděpodobnost nalezení partonu typu i mající zlomek hybnosti x. Počet partonů i-tého druhu, jejichž hybnost je mezi x a $x + \mathrm{d}x$ násobkem hybnosti protonu je potom dán výrazem $f_i(x)\mathrm{d}x$.

Celková hybnost nesená všemi partony by se měla rovnat celkové hybnosti protonu, tedy:

$$\sum_i \int_0^1 x f_i(x)\mathrm{d}x = 1 \tag{2.13}$$

kde suma probíhá přes všechny partony.

V partonovém modelu se účinné průřezy vyjadřují pomocí distribučních funkcí a účinných průřezů $\hat{\sigma}_i$ rozptylu na partonu typu i:

$$\sigma = \sum_i \int f_i(x)\hat{\sigma}_i(x)\mathrm{d}x \tag{2.14}$$

Partonový model lze s úspěchem aplikovat na všechny hadrony. Partonové distribuční funkce $f_i(x)$ jsou pak universální v tom smyslu, že pro daný hadron mohou být při všech interakcích použity tytéž distribuční funkce. Pochopitelně různé hadrony jsou popisovány různými distribučními funkcemi. Pro strukturní funkci F_2 výraz (2.14) znamená, že ji lze vyjádřit následovně:

$$F_2(x) = \sum_i e_i^2 x f_i(x) \tag{2.15}$$

kde e_i jsou elektrický náboj partonu i.

Z měření nejrůznějších hluboko nepružných rozptylů se ukázalo, že nabité partony nesou asi 50% hybnosti protonu. Zbytek hybnosti nesou neutrální partony. Neutrální v tom smyslu, že neinteragují elektromagneticky. Tento efekt kvantová chromodynamika popisuje zcela přirozeně tím, že neutrální partony ztotožňuje s gluony, zatímco elektricky nabité partony byly ztotožněny s kvarky. K tomuto tématu se ještě jednou vrátíme v závěru kapitoli 3.3.

Naše současné poznatky o původu antikvarků a gluonů se pokusíme shrnout následovně. Při procesech s malou rozlišovací schopností, například při rozptylu doprovázeném malým přenosem hybnosti, se proton chová jako systém tří konstituentních kvarků, držené pohromadě ve statickém potenciálu, který zhruba odpovídá výsledkům komplikovaných výměn gluonů, zprostředkujících síly mezi kvarky. V procesech s vyšším rozlišením, uskutečněném tvrdým rozptylem se obraz mění. Statické pole vyvolává gluony, které se přeměňují v pár kvark, antikvark, které opět září gluony a tak dále. Všechny vytvořené kvarky a gluony jsou virtuální a musí se tedy po nějakém čase rekombinovat. Pokud ovšem bude jejich virtualita velmi malá v porovnání s virtualitou Q^2 zkoumajícího fotonu, potom budou žít dostatečně dlouho na to, aby se na něm elektron rozptýlil jako na reálné částici.

Kapitola 3

Evoluční rovnice QCD

Kvarky a gluony považujeme před i po kolizi za volné částice a to i přes experimentálně podloženou skutečnost, že existují pouze uvnitř hadronu a za volné je tedy považujeme pouze pokud je zkoumáme na malých vzdálenostech. Evoluční rovnice, které se nyní pokusíme odvodit berou tento fakt do úvahy.

Odvození začneme zavedením tzv. "holých" distribučních funkcí partonů uvnitř hadronů, $q_0(x), G_0(x)$, které závisejí pouze na x a jsou interpretovány ve smyslu kvark partonového modelu. První případ takových funkcí představují takzvané nesingletní kvarkové distribuční funkce, které se ve vedoucím řádu (LO) poruchové kvantové chromodynamiky schodují s valenčními distribučními funkcemi kvarků. K uvažovanému základnímu procesu pružného rozptylu elektronu na kvarku přispívají i další procesy (obr. 3.1). Kvark uvnitř hadronu totiž může během uvažovaného procesu v důsledku silné interakce vyzářit jeden gluon nebo dva gluony atd.

Korekce uvažovaného základního rozptylu daná příspěvkem procesu $e^- + q \rightarrow e^- + q + g$ pochází ze započtení skutečnosti, že z nalétajícího kvarku se mohou vyzařovat gluony. Hlavní příspěvek v takovém procesu pochází z oblasti malých a vzájemně uspořádaných virtualit τ_i virtuálních partonů:

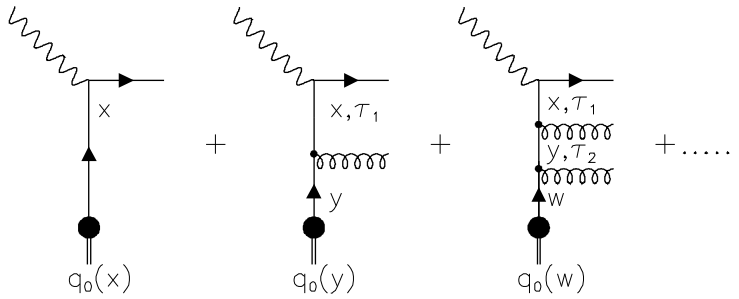
$$\tau_n \leq \tau_{n-1}, \cdots, \tau_2 \leq \tau_1 \leq Q^2. \tag{3.1}$$

Započtení příspěvků diagramů v obrázku 3.1 nám umožňuje definovat tzv. "oblečenou" nesingletní kvarkovou distribuční funkci, která zmíněné ekekty QCD bere do úvahy:

$$q_{NS}(x, M) \equiv q_{NS,0}(x) + \int_x^1 \frac{\mathrm{d}y}{y} \left[P_{qq}^{(0)}\left(\frac{x}{y}\right) \int_{m^2}^{M^2} \frac{\mathrm{d}\tau_1}{\tau_1} \frac{\alpha_s\left(\tau_1\right)}{2\pi} \right] q_{NS,0}(y) +$$
$$\int_x^1 \frac{\mathrm{d}y}{y} \int_y^1 \frac{\mathrm{d}\omega}{\omega} \int_{m^2}^{M^2} \frac{\mathrm{d}\tau_1}{\tau_1} \int_{m^2}^{\tau_1} \frac{\mathrm{d}\tau_2}{\tau_2} \frac{\alpha_s(\tau_1)}{2\pi} \frac{\alpha_s(\tau_2)}{2\pi} \left[P_{qq}^{(0)}\left(\frac{x}{y}\right) P_{qq}^{(0)}\left(\frac{y}{\omega}\right) \right] q_{NS,0}(\omega) + \ldots \tag{3.2}$$

kde $\alpha_s(\mu)$ je renormalizovaný barevný náboj, splňující rovnici (5.6) a nově zavedená škála M představuje horní hranici pro virtulitu kvarků vystupujích ve výše uvedených procesech. Funkce $P_{qq}^{(0)}$ popisuje emisi gluonu (více v kapitole 3.1). Derivováním obou stran rovnice podle $\ln M^2$ dostáváme rovnici:

$$\frac{\mathrm{d}q_{NS}(x, M)}{\mathrm{d}\ln M^2} = \frac{\alpha_s(M)}{2\pi} \int_x^1 \frac{\mathrm{d}y}{y} P_{qq}^{(0)}\left(\frac{x}{y}\right) [q_{NS,0}(y) +$$



Obrázek 3.1: Grafická reprezentace definice "oblečené" nesingletní distribuční funkce kvarků uvnitř protonu. τ_i značí absolutní hodnotu virtuality daného intermediálního stavu.

$$\int_y^1 \frac{\mathrm{d}\omega}{\omega} \int_{m^2}^{M^2} \frac{\mathrm{d}\tau_2}{\tau_2} \frac{\alpha_s(\tau_2)}{2\pi} P_{qq}^{(0)}\left(\frac{y}{\omega}\right) q_{NS,0}(\omega) + ...\Big], \tag{3.3}$$

tj.

$$\frac{\mathrm{d}q_{NS}(x,M)}{\mathrm{d}\ln M^2} = \frac{\alpha_s(M)}{2\pi} \int_x^1 \frac{\mathrm{d}y}{y} P_{qq}^{(0)}\Big(\frac{x}{y}\Big) q_{NS}(y,M) =$$
$$\frac{\alpha_s}{2\pi} \int \mathrm{d}z \int \mathrm{d}y P_{qq}^{(0)}(z) q_{NS}(y) \delta(x-yz) \equiv \frac{\alpha_s}{2\pi} P_{qq}^{(0)} \otimes q_{NS}, \tag{3.4}$$

která se nazývá Altarelli-Parisi evoluční rovnicí pro "oblečenou" nesingletní kvarkovou distribuční funkci $q_{NS}(x,M)$. Nově zavedenou operaci $\otimes$ budeme v celém textu nazývat konvolucí. Vidíme, že v rovnicích pro distribuční funkce, které zachycují efekty mnohonásobné emise gluonů se objevuje závislost na nové proměnné - škále M. Ta v partonovém modelu nevystupovala (viz např. 2.15). QCD nezmění původní rámec, pouze ji o tuto závislost obohatí.

Na závěr této kapitoli představíme systém vázaných evolučních rovnic pro singletní kvarkovou, antikvarkovou a gluonovou distribuční funkci $q_i(x,M)$, $\bar{q}_i(x,M)$ a $G(x,M)$:

$$\frac{\mathrm{d}q_i(x,M)}{\mathrm{d}\ln\ M} = \frac{\alpha_s(M)}{\pi}\Big[\int_x^1 \frac{\mathrm{d}y}{y} P_{qq}^{(0)}\Big(\frac{x}{y}\Big) q_i(y,M) + \int_x^1 \frac{\mathrm{d}y}{y} P_{qG}^{(0)}\Big(\frac{x}{y}\Big) G(y,M)\Big], \tag{3.5}$$

$$\frac{\mathrm{d}\bar{q}_i(x,M)}{\mathrm{d}\ln\ M} = \frac{\alpha_s(M)}{\pi}\Big[\int_x^1 \frac{\mathrm{d}y}{y} P_{qq}^{(0)}\Big(\frac{x}{y}\Big) \bar{q}_i(y,M) + \int_x^1 \frac{\mathrm{d}y}{y} P_{qG}^{(0)}\Big(\frac{x}{y}\Big) G(y,M)\Big], \tag{3.6}$$

$$\frac{\mathrm{d}G(x,M)}{\mathrm{d}\ln\ M} = \frac{\alpha_s(M)}{\pi}\left[\int_x^1 \frac{\mathrm{d}y}{y} P_{Gq}^{(0)}\left(\frac{x}{y}\right) \Sigma(x,M) + \int_x^1 \frac{\mathrm{d}y}{y} P_{GG}^{(0)}\left(\frac{x}{y}\right) G(y,M)\right]. \tag{3.7}$$

kde funkci $\Sigma(x,M)$ definujeme jako součet všech kvarkových a antikvarkových distribučních funkcí:

$$\Sigma(x,M) \equiv \sum_{i=1}^{n_f} (q_i(x,M) + \bar{q}_i(x,M)). \tag{3.8}$$

n_f značí počet vůní nehmotných kvarků.

V předcházejících evolučních rovnicích je M interpretováno jako horní mez partonových virtualit obsažených v definici "oblečených" partonových distribučních funkcí, bez nijaké specifikace jeho vztahu ke kinematickým proměnným nějakého fyzikálního procesu. Protože v případě hluboko nepružného procesu jsou příspěvky integrálů v (3.2) dominantní v oblasti malých virtualit τ_i, má volba horní meze na tyto integrály malý vliv. Nedopouštíme se velké chyby, pokud položíme $M^2 = Q^2$. Tuto volbu budeme užívat v celém následujícím textu.

3.1 Větvící funkce

Funkce $P_{qq}^{(0)}(x)$ představená v předcházející kapitole je příkladem tzv. větvící funkce, která v QCD popisuje větvení kvarku na kvark. Podobně existují funkce $P_{cb}^{(0)}(x)$, popisující obecné větvení partonů $a \to b + c$ v kolineární oblasti:

$$P_{qq}^{(0)}(x) = P_{\bar{q}\bar{q}}^{(0)}(x) = \frac{4}{3}\left[\frac{1+x^2}{1-x}\right]_+, \tag{3.9}$$

$$P_{Gq}^{(0)}(x) = P_{G\bar{q}}^{(0)}(x) = \frac{4}{3}\left[\frac{1+(1-x)^2}{x}\right], \tag{3.10}$$

$$P_{qG}^{(0)}(x) = P_{\bar{q}G}^{(0)}(x) = \left[\frac{x^2+(1-x)^2}{2}\right], \tag{3.11}$$

$$P_{GG}^{(0)}(x) = 6\left\{\left[\frac{x}{1-x}\right]_+ + \frac{1-x}{x} + x(1-x) + \left(\frac{33-2n_f}{36} - 1\right)\delta(1-x)\right\}. \tag{3.12}$$

Větvící funkce jsou stejné pro všechny kvarky $q_i, \bar{q}_i$. Pro vyjádření této skutečnosti bylo použito symbolů $q, \bar{q}$

Ve větvících funkcích (3.9) a (3.12) vystupuje "+" distribuce, která je definována:

$$[f(x)]_+ \equiv \lim_{\beta\to 0}\left(f(x)\theta(1-x-\beta) - \delta(1-x-\beta)\int_0^{1-\beta} f(y)\mathrm{d}y\right) \tag{3.13}$$

kde $\theta(x)$ a $\delta(x)$ jsou skoková funkce a Diracova δ -funkce.

Její působení na testovací funkci je následující:

$$\int_0^1 [f(x)]_+ g(x)dx \equiv \int_0^1 f(x)\left(g(x) - g(1)\right)dx. \tag{3.14}$$

3.2 Momenty distribučních funkcí a sumační pravidla

Tvar Altarelli-Parisiho rovnici je možné zjednodušit, pokud zavedeme tzv. moment funkce $f(x)$ definovaný jako:

$$f(n) \equiv \int_0^1 x^{n-1} f(x)dx \tag{3.15}$$

Taková transformace se nazývá Mellinova a o představených momentech se někdy mluví jako o Mellinových momentech. Podrobněji je popsána v dodatku A.

Použitím Mellinovy transformace v rovnicích (3.5) - (3.7) získáme místo konvolucí prosté násobení momentů:

$$\frac{\mathrm{d}q_i(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \frac{\alpha_s(Q^2)}{2\pi}\Big(P_{qq}^{(0)}(n)q_i(n,Q^2) + P_{qG}^{(0)}(n)G(n,Q^2)\Big), \tag{3.16}$$

$$\frac{\mathrm{d}\bar{q}_i(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \frac{\alpha_s(Q^2)}{2\pi}\Big(P_{qq}^{(0)}(n)\bar{q}_i(n,Q^2) + P_{qG}^{(0)}(n)G(n,Q^2)\Big), \tag{3.17}$$

$$\frac{\mathrm{d}G(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \frac{\alpha_s(Q^2)}{2\pi}\Big(P_{Gq}^{(0)}(n)\Sigma(n,M) + P_{GG}^{(0)}(n)G(n,Q^2)\Big). \tag{3.18}$$

Výrazně jednodušší rovnice platí pro momenty nesingletních kvarkových distribučních funkcí ($a \equiv \alpha_s/\pi$)

$$\frac{\mathrm{d}q_{NS}(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \frac{\alpha_s(Q^2)}{2\pi}P_{qq}^{(0)}(n)q_{NS}(n,Q^2). \tag{3.19}$$

Poznamenejme, že nultý moment (tj. prostý integrál) q_{NS} nezávisí na Q^2, neboť $P_{qq}^{(0)}(0) = 0$. Kombinací evolučních rovnic (3.16)-(3.18) můžeme napsat evoluční rovnici pro sumu

$$\bar{\Sigma}(n,Q^2) \equiv \sum_{i=1}^{n_f}(q_i(n,Q^2) + \bar{q}_i(n,Q^2)) + G(n,Q^2) = \Sigma(n,Q^2) + G(n,Q^2) \tag{3.20}$$

Rovnice má tvar:

$$\frac{\mathrm{d}\bar{\Sigma}(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \\ \frac{\alpha_s(Q^2)}{2\pi}\left[\Big(P_{qq}^{(0)}(n) + P_{Gq}^{(0)}(n)\Big)\Sigma(n,Q^2) + \Big(2n_f P_{qG}^{(0)}(n) + P_{GG}^{(0)}(n)\Big)G(n,Q^2)\right] \tag{3.21}$$

Rovnice (3.21) nabývá zvláštního významu pro $n = 1$, protože potom má $\sum(1,M)$ význam frakce hybnosti protonu nesené všemi partony. Jako takový ovšem musí být roven jedné při libovolné škále Q^2 a derivace (3.21) musí být nulová. To dává podmínku:

$$P_{qq}^{(0)}(1) + P_{Gq}^{(0)}(1) = \int_0^1 \mathrm{d}z z\Big(P_{qq}^{(0)}(z) + P_{Gq}^{(0)}(z)\Big) = 0, \tag{3.22}$$

$$2n_f P_{qG}^{(0)}(1) + P_{GG}^{(0)}(1) = \int_0^1 \mathrm{d}z z\Big(2n_f P_{qG}^{(0)}(z) + P_{GG}^{(0)}(z)\Big) = 0. \tag{3.23}$$

Přímý výpočet těchto integrálů použitím explicitních vyjádření (3.9)- (3.12) větvících funkcí ukazuje, že v QCD jsou tyto podmínky opravdu splněny.

3.3 Vlastnosti distribučních funkcí

V této části se blíže seznámíme s vlastnostmi partonových distribučních funkcí. Distribuční funkce kvarků uvnitř protonu označíme $q(x)$, $q = u, d, s, c$. Analogicky pro antikvarky

$\bar{q}(x),\ \bar{q} = \bar{u}, \bar{d}, \bar{s}, \bar{c}$. K jejich měření jsou k dispozici nejrůznější experimenty za pomoci svazků elektronů, mionů a (anti)neutrin a to užitím mnoha různých metod a technik. Získat distribuční funkce z experimentálních dat pro strukturní funkce je poměrně pracnou záležitostí. Důvodem je fakt, že strukturní funkce jsou vždy kombinací právě distribučních funkcí. Jako příklad uveďme, jak taková kombinace vypadá v případě strukturních funkcí protonu a neutronu (pro vyjádření distribučních funkcí kvarků v neutronu přitom využijeme předpoklad izospinové symetrie) :

$$F_2^{ep}(x) = x\left(\frac{4}{9}[u(x) + \bar{u}(x) + c(x) + \bar{c}(x)] + \frac{1}{9}[d(x) + \bar{d}(x) + s(x) + \bar{s}(x)]\right) \tag{3.24}$$

$$F_2^{en}(x) = x\left(\frac{4}{9}[d(x) + \bar{d}(x) + c(x) + \bar{c}(x)] + \frac{1}{9}[u(x) + \bar{u}(x) + s(x) + \bar{s}(x)]\right) \tag{3.25}$$

Zabývejme se nyní některými důležitými rysy kvarkových distribučních funkcí:

Pro $x \to 0$ se jak kvarková, tak antikvarková distribuční funkce chová zhruba jako $1/x$ a integrál

$$\int_0^1 q(x)\mathrm{d}x \tag{3.26}$$

tedy diverguje. Tuto skutečnost můžeme interpretovat tak, že počet nabitých partonů v protonu je nekoncový. Na první pohled je tato skutečnost značně znepokojivá, nicméně, jak se později ukáže lehce vysvětlitelná a v podstatě nepředstavující nijaký problém pro kvark partonový model. Pro vysvětlení nejprve uvažujme tzv. valenční distribuční funkci, definovanou jako rozdíl kvarkové a antikvarkové distribuční funkce

$$q_v \equiv q(x) - \bar{q}(x); \quad q = u, d, s, c \tag{3.27}$$

Valenční distribuční funkce poskytují spojení mezi kvark-partony v partonovém modelu a kvarky starého konstituentního kvarkového modelu. Takovéto distribuční funkce jsou integrovatelné a integrály

$$\int_0^1 u_v(x)\mathrm{d}x \doteq 2; \quad \int_0^1 d_v(x)\mathrm{d}x \doteq 1 \tag{3.28}$$

navíc souhlasí s předpověďmi konstituentního kvarkového modelu.

Vyjádříme nyní průměr (3.24) a (3.25):

$$F_2^{eN} \equiv \frac{1}{2}(F_2^{ep} + F_2^{en}) = \frac{5}{18}x\underbrace{[u(x) + d(x) + \bar{u}(x) + \bar{d}(x) + s(x) + \bar{s}(x) + c(x) + \bar{c}(x)]}_{\sum(x)} - \kappa(x) \tag{3.29}$$

kde $\kappa(x) \equiv x(s(x) + \bar{s}(x) - c(x) + \bar{c}(x))/6$ je číselně zanedbatelné vzhledem k $\sum(x)$.

Potom můžeme psát vztah:

$$\frac{18}{5}\int_0^1 F_2^{eN}(x)\mathrm{d}x \doteq \int_0^1 x\Sigma(x)\mathrm{d}x, \tag{3.30}$$

který udává zlomek celkové hybnosti protonu nesené $u,\ d,\ s\ a\ c$ kvarky a příslušnými antikvarky. Část hybnosti nesená $t\ a\ b$ kvarky je zcela zanedbatelná. Experimentální hodnota, zhruba 0.5 ukazuje na to, že v protonu musí být ještě další, elektricky neutrální partony, které nesou zbývající polovinu jeho hybnosti. Brzy po experimentálních měřeních (3.30) byly tyto partony identifikovány s gluony.

Kapitola 4

Metody řešení evolučních rovnic

Cílem této části bude představit některé z metod řešení evolučních rovnic pro kvarkové a gluonové distribuční funkce.

4.1 Metoda Laguerových polynomů

Tato metoda řeší evoluční rovnice (3.4) v analytické formě a to za použití rozvoje do sady Laguerreových polynomů. Tato metoda může být vpodstatě libovolně přesná, v praxi však musí být tento rozvoj samozřejmě koncový. Tato metoda byla používána v polovině osmdesátých let a to mnoha skupinami experimentátorů.

4.2 Numerické integrační metody

Poměrně záhy se začalo přistupovat k numerickým metodám řešení evolučních rovnic a to především zásluhou obrovského zvýšení výkonu a kapacit moderních počítačů. Tyto numerické postupy jsou obvykle vytvářeny tak, že jejich relativní jednoduchost umožňuje jejich použití i neodborníky, případně experimentátory. V následující části jednu takovou numerickou metodu představíme.

4.3 Metoda Jacobiho polynomů

Technika této metody je založena na rozvoji distribučních a strukturních funkcí do sady Jacobiho polynomů

$$\Theta_k^{\alpha\beta}(x) \equiv \sum_{j=0}^{k} c_{kj}^{\alpha\beta} x^j, \tag{4.1}$$

ortogonálních na intervalu $(0,1)$ s vahou $x^\alpha(1-x)^\beta$

$$\int_0^1 \mathrm{d}x x^\alpha (1-x)^\beta \Theta_k^{\alpha\beta}(x)\Theta_l^{\alpha\beta}(x) = \delta_{kl}. \tag{4.2}$$

Strukturní funkci $F(x, Q^2)$ můžeme rozvinout

$$q_{NS}(x, Q^2) = x^\alpha (1-x)^\beta \sum_{k=0}^{\infty} \Theta_k^{\alpha\beta}(x) a_k^{\alpha\beta}(Q^2) \tag{4.3}$$

v Jacobiho momentech $a_k^{\alpha\beta}(Q^2)$, které jsou dány lineárními kombinacemi momentů $F(j, Q^2)$:

$$a_k^{\alpha\beta}(Q^2) \equiv \int_0^1 \mathrm{d}x F(x, Q^2) \Theta_k^{\alpha\beta}(x) = \sum_{j=0}^{k} c_{kj}^{\alpha\beta} F(j, Q^2). \tag{4.4}$$

Tato cesta již vede k explicitnímu vyjádření $F(x, Q^2)$.

Kapitola 5

Metoda inverzní Mellinovy transformace

5.1 Řešení evoluční rovnice v nesingletním případě

V následující kapitole bude představena Mellinova transformace jako nástroj pro řešení evolučních rovnic kvantové chromodynamiky. Konkrétně tuto metodu předvedeme v případě nesingletní distribuční funkce.

Představme evoluční rovnici pro distribuční funkce $f(x, Q^2)$ ve tvaru:

$$Q^2 \frac{\partial f(x, Q^2)}{\partial Q^2} = P(x, Q^2) \otimes f(x, Q^2), \tag{5.1}$$

kde $P(x, Q^2)$ je tzv. Altarelli-Parisiho jádro a $\otimes$ značí konvoluci zavedenou v (3.4). Altarelli Parisiho evoluční jádro je doposud známo do druhého řádu poruchové QCD:

$$P(n, a_s) = \frac{\alpha_s}{\pi} P^{(0)}(n) + \left(\frac{\alpha_s}{\pi}\right)^2 P^{(1)}(n) + O(\alpha_s^3). \tag{5.2}$$

Rovnici (5.1) můžeme značně zjednodušit použitím Mellinových momentů (3.15). Získáme tak rovnici:

$$Q^2 \frac{\partial f(n, Q^2)}{\partial Q^2} = P(n, Q^2) \cdot f(n, Q^2). \tag{5.3}$$

Při řešení rovnice (5.3) existuje několik možných postupů. Všechny spočívají ve vhodných úpravách této rovnice. Předvedeme dva z nich. Přijměme nejprve značení $a_s = \alpha_s(Q^2)/\pi$. Potom můžeme rovnici (5.3) psát jako:

$$\frac{\partial f(n, Q^2)}{\partial a_s} = \frac{P(n, a_s)}{\beta(a_s)} f(n, Q^2), \tag{5.4}$$

kde

$$\beta(a_s) \equiv Q^2 \frac{\mathrm{d}a_s}{\mathrm{d}Q^2} \tag{5.5}$$

Funkce $a_s(Q^2)$, která hraje v kvantové chromodynamice významnou roli je dána rovnicí:

$$\frac{\mathrm{d}a_s(Q^2)}{\mathrm{d}\ln Q^2} = -\beta_0 a_s^2(Q^2) - \beta_1 a_s^3(Q^2). \tag{5.6}$$

kde

$$\beta_0 = \frac{33-2n_f}{6}; \quad \beta_1 = \frac{153-19n_f}{33-4n_f}. \tag{5.7}$$

Řešení rovnice (5.6) můžeme psát jako:

$$\frac{1}{a_s(Q^2)} + \frac{\beta_1}{\beta_0}\ln\left[\frac{\beta_1 a_s(Q^2)}{\beta_0+\beta_1 a_s(Q^2)}\right] = \beta_0 \ln\left(\frac{Q^2}{\Lambda^2}\right), \tag{5.8}$$

kde Λ představuje škálovací parametr, který může být pro tuto chvíli definován jako škála, při které se nuluje levá strana rovnice (5.8). Takovou rovnici je nutné řeší numericky. Zavedeme-li označení $L = \ln\frac{Q^2}{\Lambda^2}$, pak lze řešení (5.6) rozvinout do mocninné řady v $1/L$ ve tvaru:

$$a_s(Q^2) = \frac{1}{\beta_0 L}\left(1 - \frac{\beta_1}{\beta_0^2}\frac{\ln L}{L}\right) + \cdots \tag{5.9}$$

V těchto řádech je funkce β dána takto:

$$\beta(a_s) = -\beta_0 a_s^2 - \beta_1 a_s^3 + O(a_s^4). \tag{5.10}$$

Dosazením výrazů (5.2) a (5.10) do rovnice (5.4) dostáváme:

$$\frac{\partial f(n,Q^2)}{\partial a_s} = -\frac{(P^{(0)}(n) + a_s P^{(1)}(n))}{a_s(\beta_0+\beta_1 a_s)} f(n,Q^2) \tag{5.11}$$

jejíž řešení má tvar

$$f(n,Q^2) = E(n,Q^2,Q_0^2) f(n,Q_0^2) \tag{5.12}$$

kde funkce $f(n,Q_0^2)$ hraje roli počáteční podmínky a

$$E(n,\eta,a_s(Q^2),a_s(Q_0^2)) = \left(\frac{a_s(Q^2)}{a_s(Q_0^2)}\right)^{\frac{\gamma_0(n)}{2\beta_0}} \left(\frac{\beta_0+\beta_1 a_s(Q^2)}{\beta_0+\beta_1 a_s(Q_0^2)}\right)^{\frac{\gamma_1(n)}{2\beta_1}}, \tag{5.13}$$

je tzv. evoluční operátor. Ve výrazu (5.13) vystupují tzv. anomální dimenze $\gamma_0^{NS}(n)$ a $\gamma_1^{NS}(n)$, které souvisí s evolučním jádrem: $P_i(n) = -\frac{1}{2}\gamma_i^{NS}(n)$.

Dosadíme-li do rovnice (5.3) s ohledem na (5.9) vyjádření (5.2) a ponecháme-li tak vyjádření v proměnné Q^2, máme rovnici:

$$Q^2\frac{\partial f(n,Q^2)}{\partial Q^2} = \left(\frac{1}{\beta_0 L}\left(1-\frac{\beta_1}{\beta_0^2}\frac{\ln L}{L}\right)P^{(0)}(n) + \left(1-\frac{\beta_1}{\beta_0^2}\frac{\ln L}{L}\right)^2 P^{(1)}(n)\right) f(n,Q^2). \tag{5.14}$$

jejiž řešení lze vyjádřit pomocí evolučního operátoru

$$E(n,\eta,Q^2,Q_0^2) = \left(\frac{L_0}{L}\right)^{\frac{\gamma_0(n)}{2\beta_0}} \exp\left[-\frac{1}{2\beta_0^2}\left(\frac{1}{L_0}-\frac{1}{L}\right)\left(\gamma_1(n,\eta)-\frac{\beta_1}{\beta_0}\gamma_0(n)\right)+\right.$$
$$\frac{\beta_1}{2\beta_0^3}\left(\frac{\ln L_0}{L_0}-\frac{\ln L}{L}\right)\gamma_0(n)+\frac{\beta_1}{4\beta_0^4}\left(\frac{1+2\ln L_0}{L_0^2}-\frac{1+2\ln L}{L^2}\right)\gamma_1(z)$$
$$\left.-\frac{\beta_1^2}{54\beta_0^6}\left(\frac{2+6\ln L_0+9\ln^2 L_0}{L_0^3}-\frac{2+6\ln L+9\ln^2 L}{L^3}\right)\gamma_1(n)\right], \qquad (5.15)$$

kde $L_0 = \ln Q_i^2/\Lambda^2$.

Nyní se budeme soustředit na otázku zpětné transformace získaných momentů distribučních funkcí. Snahou bude přejít zpět k proměnným x a Q^2. Zpětná Mellinova transformace je dána výrazem:

$$f(x,Q^2) = \frac{1}{2\pi \mathrm{i}}\int_C \mathrm{d}n x^{-n} f(n,Q^2) \qquad (5.16)$$

zde C obíhá všechny singularity v integrandu.

Distribuční partonové funkce při zvolené počáteční škále Q_0^2 parametrizujeme:

$$xf(x,Q_0^2) = \sum_i A_i x^{\alpha_i}(1-x)^{\beta_i}. \qquad (5.17)$$

Kde A_i, α_i a β_i jsou volné parametry. Výraz (5.17) představuje pouze základní typ parametrizace. V praxi se zmíněná počateční podmínka popisuje více parametry. Po Mellinově tranformaci nabydou tvar:

$$f(n,Q_0) = \sum_i A_i B(n+\alpha_i-1, 1+\beta_i), \qquad (5.18)$$

kde $B(x,y)$ je Eulerova beta funkce. Vynecháme-li nyní argumenty Q^2 *a* Q_0^2 máme pro řešení (5.1):

$$f(x,Q^2) = \mathrm{Re}\frac{1}{\pi i}\int_{C_s} \mathrm{d}z E(n)R(n), \qquad (5.19)$$

kde

$$R(n) = x^{-n}\sum_i A_i B(z+\alpha_i-1, 1+\beta_i). \qquad (5.20)$$

5.2 Metody inverzní Mellinovy transformace

V předcházející kapitole byl naznačen postup při řešení evoluční rovnice tvaru (5.1). Rovnici jsme pomocí Mellinovy transformace převedli do N-prostoru, ve kterém ji lze řešit analyticky. Pro zpětnou transformaci do x-prostoru platí (5.16). V následující části předvedeme

dva způsoby postupu při této zpětné transformaci. Oba budou spočívat ve speciální volbě integrační křivky v integrálu (5.16). Poznamenejme ještě, že na volbě této křivky, pokud obíhá všechny singularity v integrandu, výpočet nezávisí. Situace se však změní, pokud budeme zmíněný integrál počítat přibližně. Tehdy už výsledek na volbě integračního oboru závisí.

5.2.1 Integrování po parabolické křivce

V následujícím popíšeme první z variant. Zvolíme integrační obor tak, že v něm bude možné integrand velmi dobře aproximovat sadou ortogonálních polynomů. Volí se parabolická křivka, která bude plně určená partonovou distribuční funkcí v počáteční škále Q_0^2. Získaný integrál je pak možné spočítat například pomocí Gauss-Laguerrovy metody pro výpočet určitého integrálu. Při této metodě je k dosažení vysoké přesnosti (0,02%) potřeba zhruba čtyř až pěti výpočtů integrandu.

Podívejme se blíže na tuto metodu v nesingletním případě. Křivka, podél které probíhá integrace se volí:

$$z(u) = z_0 + ic_2\sqrt{u} + \frac{1}{2}c_2^2 c_3 u, \qquad u = 0, \cdots, \infty \tag{5.21}$$

kde z_0 jsou minima funkce $f(n)$ na reálné ose. Parametry c_2 *a* c_3 mají tvar:

$$c_2 = \sqrt{\frac{2f(z_0)}{f''(z_0)}}, \quad c_3 = \frac{f'''(z_0)}{3f''(z_0)}. \tag{5.22}$$

$f'(n)$, $f''(n)$ *a* $f'''(n)$ jsou první tři derivace $f(n)$ podle z. S použitou parametrizací (5.21) jsou tyto derivace dány:

$$f' = \sum_i G_i' F_i, \tag{5.23}$$

$$f'' = \sum_i (G_i'' + G_i'^2) F_i, \tag{5.24}$$

$$f''' = \sum_i (G_i''' + 3G_i'' G_i' + G_i'^3) F_i, \tag{5.25}$$

$$\tag{5.26}$$

kde $F_i = x^{-z} A_i B(z + \alpha_i - 1, 1 + \beta_i)$ a

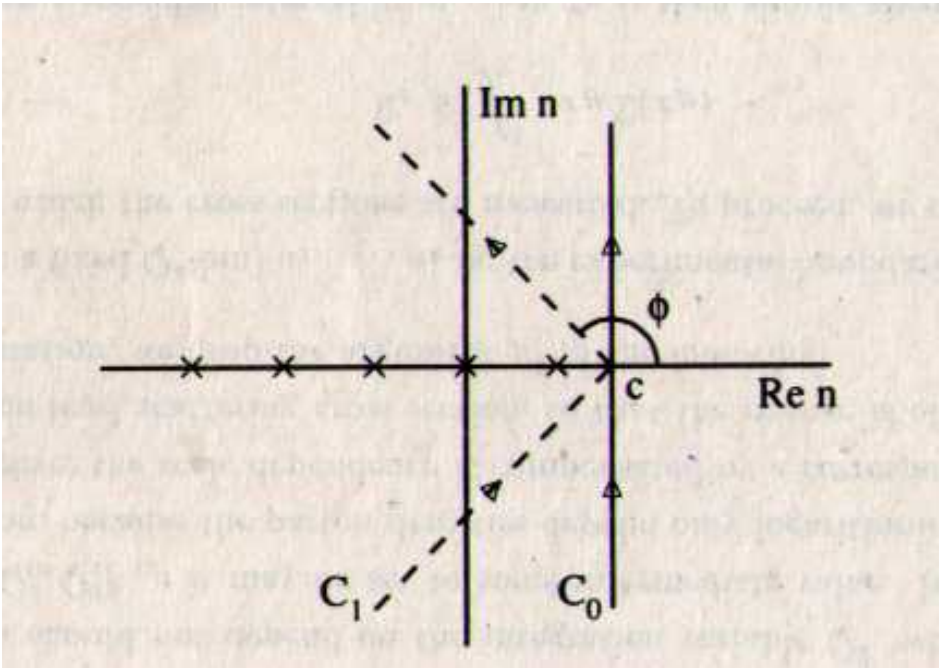
$$G_i = \ln(x^{-z} B(z + \alpha_i - 1, 1 + \beta_i)), \tag{5.27}$$

$$G_i' = -\ln x + \Psi(z + \alpha_i - 1) - \Psi(z + \alpha_i + \beta_i), \tag{5.28}$$

$$G_i'' = \Psi'(z + \alpha_i - 1) - \Psi'(z + \alpha_i + \beta_i), \tag{5.29}$$

$$G_i''' = \Psi'(z + \alpha_i - 1) - \Psi''(z + \alpha_i + \beta_i). \tag{5.30}$$

S touto parametrizací máme pro integrál (5.19):



Obrázek 5.1: Dvě možné volby integrační křivky při inverzní Mellinově tansformaci. Důležité je, že obě leží vpravo od oblasti singularit, které jsou v obrázku vyznačeny křížky.

$$I = \frac{c_2}{2\pi}\int_0^\infty \frac{\mathrm{d}u}{\sqrt{u}} e^{-u} \mathrm{Re}\left[e^u \left(1 - ic_2c_3\sqrt{u}\right) E(z(u))f(z(u))\right]. \tag{5.31}$$

Funkce $u^{-1/2}e^{-u}$ jsou vážené funkce Laguerrových polynomů $L_k^{-1/2}(x)$ a integrál můžeme aproximovat pomocí Gauss-Laggueova vzorce:

$$I \approx \frac{c_2}{2\pi}\sum_{j=1}^{k} w_j \mathrm{Re}\left[e^{u_j} \left(1 - ic_2c_3\sqrt{u_j}\right) E(z(u_j))f(z(u_j))\right]. \tag{5.32}$$

kde u_j jsou kořeny $L_k^{-1/2}(x)$ a váhy w_j jsou dány:

$$w_j = \frac{\Gamma(n+\frac{1}{2})}{n!(n+1)^2} \frac{u_j}{\left(L_{k+1}^{-1/2}(u_j)\right)^2}. \tag{5.33}$$

5.2.2 Integrace po přímce

Nyní přístoupíme ke druhé možnosti numerické realizace inverzní Mellinovy transformace. V následující kapitole bude tento postup aplikován na konkrétní fyzikální situaci.

Pokud funkce $f(x)$ rychle ubývá pro $x > 1$ a pokud je po částech spojitá pro $x > 0$, potom můžeme pro inverzní Mellinovu transformaci psát:

$$f(x) \ = \frac{1}{2\pi \mathrm{i}} \int_{c-i\infty}^{c+i\infty} \mathrm{d}n \ x^{-n} f(n), \tag{5.34}$$

kde reálné číslo c je zvoleno tak, aby byl integrál $\int_0^1 x^{c-1} f(x)$ absolutně konvergentní. To vyžaduje zvolit c tak, aby leželo napravo od singularity n_{max}, vzdálené na reálné ose nejvíc vpravo. Křivka užitá v integrálu (5.34) je zobrazena na obr. 5.1 a je označena jako C_0. Ve stejném obrázku je ilustrativně naznačena ještě další křivka, podél které je možné integraci vést.

Je užitečné přepsat rovnici (5.34) jako integraci přes reálnou proměnou. Budeme se soustředit na funkce splňující $f^*(n) = f(n^*)$, kde $*$ značí komplexní sdružení. Potom je snadné ukázat, že funkce $f(x)$ splňuje pro integrační obor určený souřadnicí c a úhlem ϕ, následující rovnost:

$$f(x) \ = \frac{1}{\pi} \int_0^\infty \mathrm{d}z \ \mathrm{Im} \left[\exp(i\phi) x^{-c-z \ \exp(i\phi)} f(n = c + z \ \exp(i\phi)) \right]. \tag{5.35}$$

Tento integrál nezávisí ani na c ani ϕ. Nicméně pro numerický výpočet může být vhodná volba těchto parametrů velmi užitečná. Při konkrétních požadavcích na přesnost výpočtu je tato vhodná dokonce nezbytná.

Ve zcela obecném případě, kdy je třeba momenty partonových distribučních funkcí počítat numericky, je výhodné volit integrační křivku C_0, tedy $\phi = \pi/2$. Tehdy rovnice (5.35) vede na:

$$f(x) \ = \frac{1}{\pi} \int_0^\infty \mathrm{d}z \ \mathrm{Re} \left[x^{-c-iz} f(n = c + iz) \right]. \tag{5.36}$$

Při výpočtu integrálu (5.34) tedy volíme integraci po přímce kolmé na reálnou osu a procházející na této ose bodem c.

5.3 Numerický výpočet řešení evoluční rovnice pro nesingletní distribuční funkci

V následující části aplikujeme metodu inverzní Mellinovy transformace na konkrétní příklad. Uvažujme reakci $\nu_\mu(\bar{\nu}_\mu) + N \to \mu^- + X$. Ta odpovídá rozptylu mionového neutrina (antineutrina) na nukleonu N. Diferenciální účinný průřez této reakce je dán následující formulí:

$$\frac{\mathrm{d}\sigma^{(\nu/\bar{\nu})p}}{\mathrm{d}x\mathrm{d}y} = \frac{G_F^2 S}{2\pi} \left[F_2^{(\nu/\bar{\nu})p}(x, Q^2) \left(\frac{1 + (1-y)^2}{2} \right) \pm x F_3^{(\nu/\bar{\nu})p}(x, Q^2) \left(\frac{1 + (1-y)^2}{2} \right) \right] \tag{5.37}$$

kde G_F je Fermiho vazbová konstanta. Srovnáním této formule s diferenciálními účinnými průřezy tohoto procesu v partonového modelu dostáváme pro strukturní funkci F_3:

$$xF_3^{\nu p} \ = \ 2x[s(x) + d(x) - \bar{u}(x) - \bar{c}(x)] \tag{5.38}$$

$$xF_3^{\bar{\nu} p} \ = \ 2x[u(x) + c(x) - \bar{s}(x) - \bar{d}(x)] \tag{5.39}$$

Uvažujme následující výraz:

$$\frac{1}{2}\left(F_3^{\nu p}+F_3^{\bar{\nu}p}\right)=(u+d+s+c-\bar{u}-\bar{d}-\bar{s}-\bar{c})\doteq\frac{1}{2}(F_3^{\nu p}+F_3^{\nu n})\equiv F_3^{\nu N} \tag{5.40}$$

Ten lze určit experimentálně neboť je kombinací diferenciálních učinných průřezů:

$$F_3^{\nu N}\sim\frac{\mathrm{d}\sigma^{\nu N}}{\mathrm{d}x\mathrm{d}y}-\frac{\mathrm{d}\sigma^{\bar{\nu}N}}{\mathrm{d}x\mathrm{d}y}. \tag{5.41}$$

Je patrné, že výraz $F_3^{\nu N}$ je kombinací valenčních distibučních funkcí. Z linearity evoluční rovnice pro nesingletní distribuční funkce vyplývá, že i funkce $F_3(x,Q^2)$ je jejím řešením. Tedy:

$$\frac{\mathrm{d}F_3(x,Q^2)}{\mathrm{d}\ln Q^2}=\frac{\alpha_s(Q^2)}{2\pi}\int_x^1\frac{\mathrm{d}y}{y}P_{qq}^{(0)}\left(\frac{x}{y}\right)F_3(y,Q^2). \tag{5.42}$$

V následující části naznačíme základní kroky při řešení takové rovnice. Tento postup bude analogický s algoritmem řešení popsaném v kapitolách 5.1 a 5.2.2. Ještě předtím ale stručně připomeneme základní schéma postupu při řešení evolučních rovnic.

K tomu abychom mohli řešit rovnici typu (5.42) je v prvé řadě zapotřebí znalost hraniční podmínky. Ta zahrnuje informace o závislosti hledané funkci na x při nějaké počáteční škále Q_0^2. Výběr této počáteční škály je poměrně delikátní záležitostí, neboť hledané řešení rovnice na výběru Q_0^2 závisí (viz. dodatek C). Obvykle se Q_0^2 volí v rozsahu několika GeV. Poruchová teorie tyto počáteční podmínky neumožňuje vypočítat, musí být získány z experimentálních dat.

Standardní analýza experimentálních dat se provádí v následujících krocích:

- Provede se volba počáteční škály Q_0^2, zvolí se vhodná parametrizace počáteční podmínky tj. funkce $f(x,Q_0^2)$. Příkladem může být:

$$f(x,Q_0^2)=Ax^{\alpha}(1-x)^{\beta}(1+\gamma x^{\eta}) \tag{5.43}$$

  kde $A,\alpha,\beta,\gamma,\eta$ jsou volné parametry.

- Pro zvolenou sadu těchto parametrů spolu s hodnotou hlavního parametru QCD Λ, vstupujícím do $\alpha_s(Q/\Lambda)$ řešíme rovnici (5.42)

- Získaná řešení se pak porovnávají s experimentálními daty a na základě těchto srovnání se hodnoty uvedených parametrů dále zpřesňují. Výsledkem je sada parametrů $A,\alpha,\beta,\gamma,\eta$ určující hraniční podmínku tak, že řešením evolučních rovnic s touto podmínkou jsou hledané distribuční funkce

Obraťme nyní pozornost na řešení evoluční rovnice (5.42). Z důvodu zachování značení přeznačíme řešení takové rovnice na $f(x,Q^2)$. Uvažujme tedy evoluční rovnici:

$$\frac{\mathrm{d}f(x,Q^2)}{\mathrm{d}\ln\ Q^2}=\frac{\alpha_s(Q^2)}{2\pi}P_{qq}^{(0)}(x)\otimes f(x,Q^2), \tag{5.44}$$

kde

$$P_{qq}^{(0)}(x) = \frac{4}{3}\left[\frac{1+x^2}{1-x}\right]_+ . \tag{5.45}$$

Po aplikaci Mellinovy transformace máme:

$$\frac{\mathrm{d}q(n,Q^2)}{\mathrm{d}\ln\ Q^2} = \frac{\alpha_s(Q^2)}{2\pi} P_{qq}^{(0)}(x) \cdot q_{NS}. \tag{5.46}$$

Tuto rovnici již umíme vyřešit analyticky. Takové řešení, včetně hraničních podmínek hledejme ve tvaru:

$$f(n,Q^2) = E(n,Q^2,Q_0^2) f(n,Q_0^2) \tag{5.47}$$

V rámci našeho numerického výpočtu se omezíme na nejnižší řády známých rozvojů všech veličin. V takovém případě pro evoluční operátor $E(n,Q^2,Q_0^2)$ máme (viz.(5.13)):

$$E(n,a_s(Q^2),a_s(Q_0^2)) = \left(\frac{a_s(Q^2)}{a_s(Q_0^2)}\right)^{-\frac{P^{(0)}(n)}{\beta_0}} . \tag{5.48}$$

V evolučním operátoru vystupuje moment větvící funkce $P_{qq}^{(0)}$. Odbočíme tedy na chvíli a pokusíme se nalézt momenty pro úplnost všech větvících funkcí (3.9) - (3.12).

Pomocí formule (3.14) a definice Mellinových momentů (3.15) snadno najdeme momenty obecné mocniny x^r:

$$F(n) = \int_0^1 x^{n-1} x^r \mathrm{d}x = \left[\frac{x^{n+r}}{n+r}\right]_0^1 = \frac{1}{n+r}, \quad \mathrm{Re}\ n > -r \tag{5.49}$$

a libovolné "+" distribuce tvaru $[x^r/(1-x)]_+$:

$$F(n) = \int_0^1 x^{n-1} \left[\frac{x^r}{(1-x)}\right]_+ \mathrm{d}x = \int_0^1 \frac{x^{n+r-1}-x^r}{1-x}\mathrm{d}x. \tag{5.50}$$

S využitím vztahu:

$$\int_0^1 \frac{1-t^{z-1}}{1-t}\mathrm{d}t = \psi(z) + \gamma_E, \qquad \mathrm{Re}\ z > 0, \tag{5.51}$$

kde γ_E je Eulerova konstanta,

můžeme po jednoduché úpravě pro momenty v (5.50) psát:

$$F(n) = \int_0^1 \left(\frac{1-x^r}{1-x} - \frac{1-x^{n+r-1}}{1-x}\right)\mathrm{d}x = \psi(r+1) - \psi(n+r), \quad \mathrm{Re}\ n > -r. \tag{5.52}$$

S použitím těchto výrazů pak pro momenty větvících funkcí máme:

$$P_{qq}^{(0)}(n) = \frac{4}{3}\left(\psi(1) - \psi(n) + \psi(3) - \psi(n+2)\right), \tag{5.53}$$

$$P_{Gq}^{(0)}(x) = \frac{4}{3}\left(\frac{2}{n-1} - \frac{2}{n} + \frac{1}{n+1}\right), \tag{5.54}$$

$$P_{qG}^{(0)}(x) = \frac{1}{n+2} - \frac{1}{n+1} + \frac{1}{2n}, \tag{5.55}$$

$$P_{GG}^{(0)}(x) = 6\left[\psi(2) - \psi(n+1) + \frac{1}{n-1} - \frac{1}{n} + \frac{1}{n+1} - \frac{1}{n+2} + \frac{33-2n_f}{36} - 1\right]. \tag{5.56}$$

Vraťme se nyní zpět k popisu algoritmu inverzního transformování. Nyní již můžeme zapsat momenty hledané distribuční funkce:

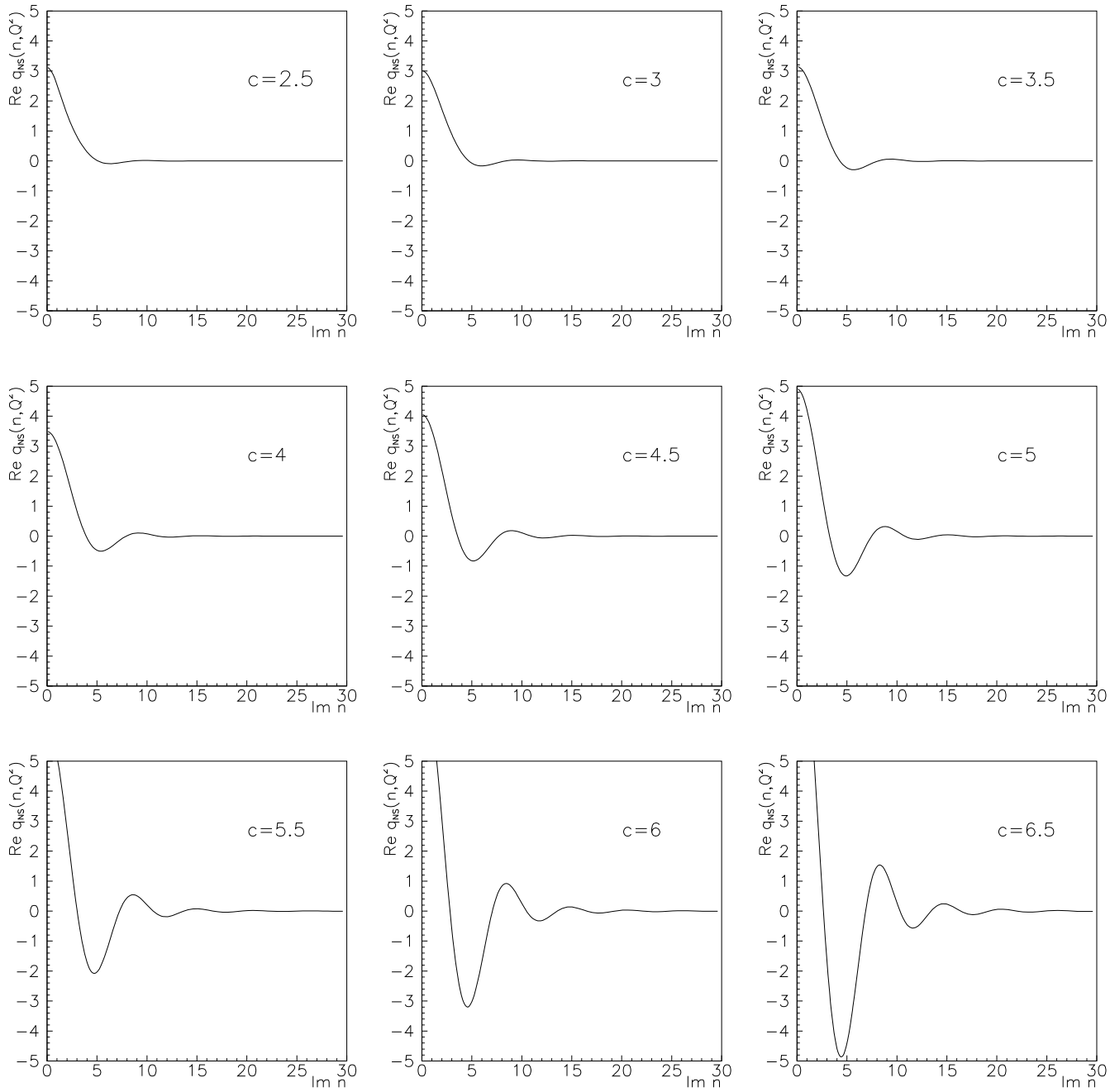
$$f(n,Q^2) = A\left(\frac{a_s(Q^2)}{a_s(Q_0^2)}\right)^{-\frac{P^{(0)}(n)}{\beta_0}} B(n+\alpha-1, 1+\beta) + \gamma B(n+\alpha+\eta-1, 1+\beta). \tag{5.57}$$

Vše je tedy připraveno pro zpětnou transformaci. Jejím provedením pak pro hledané distribuční funkce nalezneme:

$$f(x,Q^2) = \frac{1}{\pi}\int_0^\infty \mathrm{d}z \mathrm{Re}\left[x^{-c-iz} f(c+iz, Q^2)\right] \tag{5.58}$$

Do tohoto integrálu vstupuje celkem sedm parametrů: $A, \alpha, \beta, \gamma, \eta, Q_0^2$ a c. Prvních šest parametrizují hraniční podmínku (5.43). Jejich určení je pro řešení evoluční rovnice klíčové. Poruchová teorie je neumožňuje spočítat, zjištěny jsou na základě srovnání s experimentálními daty. Pokud jde o parametr c, na něm výpočet integrálu (5.58) nezávisí, pokud je zvolen tak, že leží napravo od singularity umístěné na reálné ose nejvíc vpravo. Singulárními body integrandu v (5.58) jsou realná čísla $\{..., -3, -2, -1, 0\}$. Zodpovědný je za ně moment větvící $P_{qq}^{(0)}(x)$. ψ-funkce v něm obsažená totiž v těchto bodech není definována. V případě naší rovnice má tedy tento parametr význam vzdálenosti integrační křivky od nejkrajnější, vpravo umístěné singularity. Obecně je to bod ve kterém integrační křivka v komplexní rovině protíná reálnou osu.

Na poloze integrační křivky za výše zmíněných předpokladů výpočet integrálu (5.34) nezávisí. To je ovšem pravda jen do té doby, do kdy výpočet tohoto integrálu neprovádíme numericky. Při numerickém výpočtu už výsledek na volbě c závisí. Přivede nás k tomu zkoumání vlivu parametru c na integrovanou funkci. Průběhy integrandu pro různá c jsou znázorněna obr. 5.2. Parametrizace okrajové podmínky byla zvolena následující: $A = 8.6$, $\alpha = 0.85$, $\beta = 3.7$, $Q_0^2 = 5$ GeV^2, $\Lambda = 0.377$ MeV, $\gamma = 0$. Integrací bychom obdrželi distribuční funkci v bodě $x = 0.3$ a $Q^2 = 50$ GeV^2. Je patrné, že se vzrůstajícím c integrovaná funkce značně osciluje a to na stále větší oblasti definičního oboru. Je jasné, že pro velká c je potřeba volit úměrně velký integrační obor, což u numerického výpočtu naráží na problém časové náročnosti. Navíc integrací takto oscilujících funkcí značně narůstá chyba výpočtu. Abychom tuto chybu co nejvíce eliminovali, budeme volit c co nejblíže singularity. Tehdy



Obrázek 5.2: Integrandy inverzní Mellinovy transformace pro různé polohy integrační křivky.

je průběh integrované nejméně komplikovaný – dochází zde nejméně ke vzájemnému rušení kladných a záporných příspěvků k počítanému integrálu. Pro numerický výpočet je tedy tato volba nejschůdnější.

Při numerickém výpočtu je nutné nahradit nekonečno v integrálu nějakou konečnou hodnotou. Musíme ovšem počítat s tím, že se tím dopouštíme určité chyby. Při volbě horní meze je možné využít zjištěných průběhů integrandu. Volíme takovou horní mez, abychom integrovaly jen přes relevantní část integrandu. Integrovaná funkce totiž poměrně rychle ubývá a nevhodnost zatížení výpočtu integrací v oblasti, kde jsou příspěvky k integrálu velmi malé je nasnadě. Příslušná volba horní meze musí samozřejmě korespondovat s volbou c.

5.4 Výsledky numerických výpočtů

V následující kapitole shrneme výsledky, které byly získány pomocí vypracovaného programu (více kapitola 9). Nejprve předvedeme některá řešení evoluční rovnice (5.44), pote výsledky fitování těchto řešení na experimentální data.

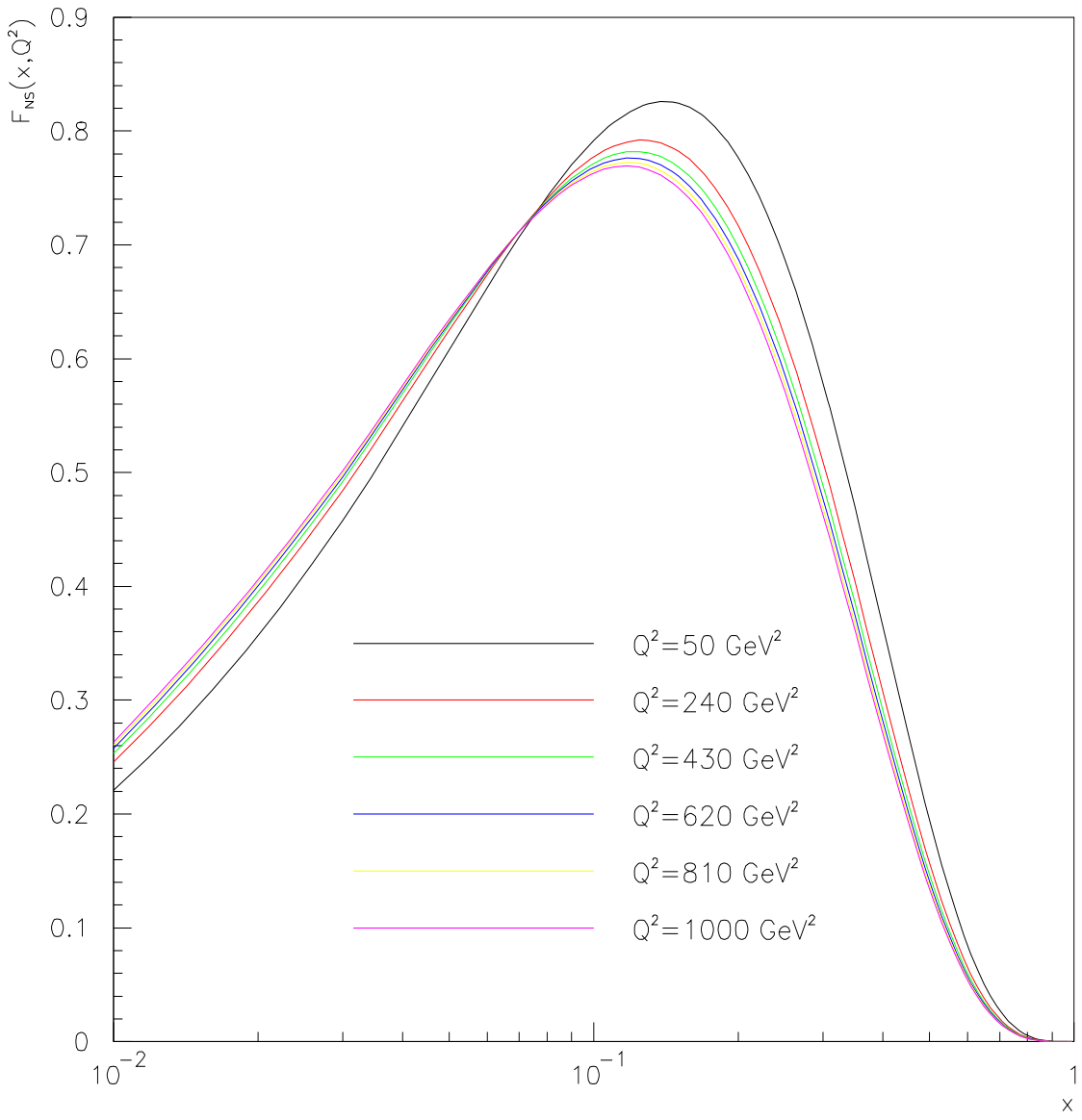
Program umožňuje pro zadanou počáteční podmínku numericky napočítat hodnotu distribuční funkce pro libovolné x a Q^2. Výpočtem dostatečného počtu takových hodnot lze získat představu o průběhu funkcí řešících příslušnou evoluční rovnici. Nyní si představíme konkrétní výsledky. Na obr. 5.3 je znázorněna závislost distribučních funkcí řešících rovnici (5.44) na x pro různé hodnoty Q^2, na obr. 5.4 jsou potom vyznačeny závislosti na Q^2 pro různá x. Zvolený rozsah x i Q^2 je pod možnostmi současných urychlovačů, nicméně mnou zjištěné výsledky se snaží mít pouze ilustrativní charakter a pro ten účel jsou dostačující.

Důležité je, že je z těchto výsledků patrná jedna zajímavá vlastnost distribučních funkcí. Pro malá x jsou to klesající funkce v Q^2, kdežto pro velká x jsou v Q^2 funkcemi klesajícími. Fyzikální vysvětlení této skutečnosti je následující. S rostoucím předaným kvadrátem čtyřhybnosti Q^2 se zvětšuje rozlišovací schopnost experimentu. To znamená, že to, co se nám při menším Q^2 jevilo jako jeden parton nesoucí např. 10 GeV^2 se nám při větším předaném čtyřimpulsu jeví jako shluk například dvou partonů, z nichž každý nese energii 5 GeV^2. Je tedy zřejmé, že výskyt partonů nesoucích menší podíly celkové čtyřhybnosti nukleonu s rostoucím Q^2 roste, kdežto výskyt partonů nesoucích větší podíly celkové čtyřhybnosti s rostoucím Q^2 klesá.

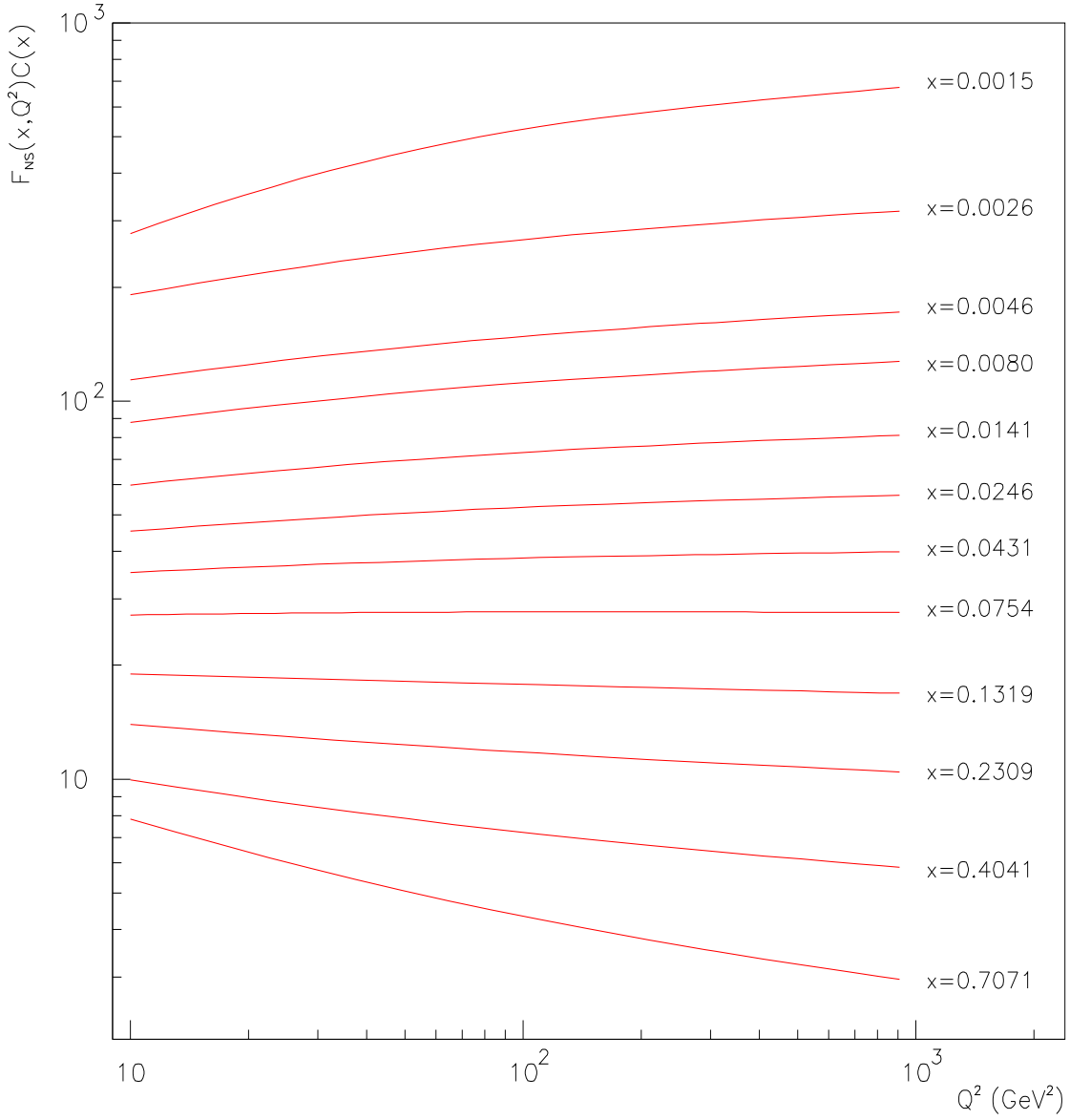
Nyní obrátíme pozornost na výsledky fitů. Experimentální data byla získána z adresy [20]. Pro minimalizaci funkce $\chi^2(A,\ \alpha,\ \beta,\ \gamma,\ \eta,\ \Lambda)$ byly použity dvě metody, Nelderova a Meadova simplexní metoda (více kapitola B.4) a metoda konjugovaných gradientů (více kapitola B.5). Postup vývoje minimalizace použitím první metody je zaznamenán v dodatku D. Jsou v nich zaznamenány jednotlivé iterační kroky. V každém z nich dochází k nalezení nových hodnot parametrů, které snižují hodnotu funkce χ^2. V obou metodách se ukázalo velmi užitečné předefinovat parametr A ve výrazu (5.43) na A^2. Minimalizovaná funkce se stává citlivější na změnu tohoto parametru. V případě metody konjugovaných gradientů navíc tento parametr vstupuje do všech složek gradientu (původně byla jedna složka gradientu na tomto parametru nezávislá). To všechno přispělo k výraznému zlepšení výsledků minimalizací v případě jedné i druhé metody. Oběma metodama bylo dosaženo takřka stejné minimální hodnoty $\chi = 70/64$.

Odlišnost je jen v hodnotách parametrů, v nichž je toto minimum nabyto.

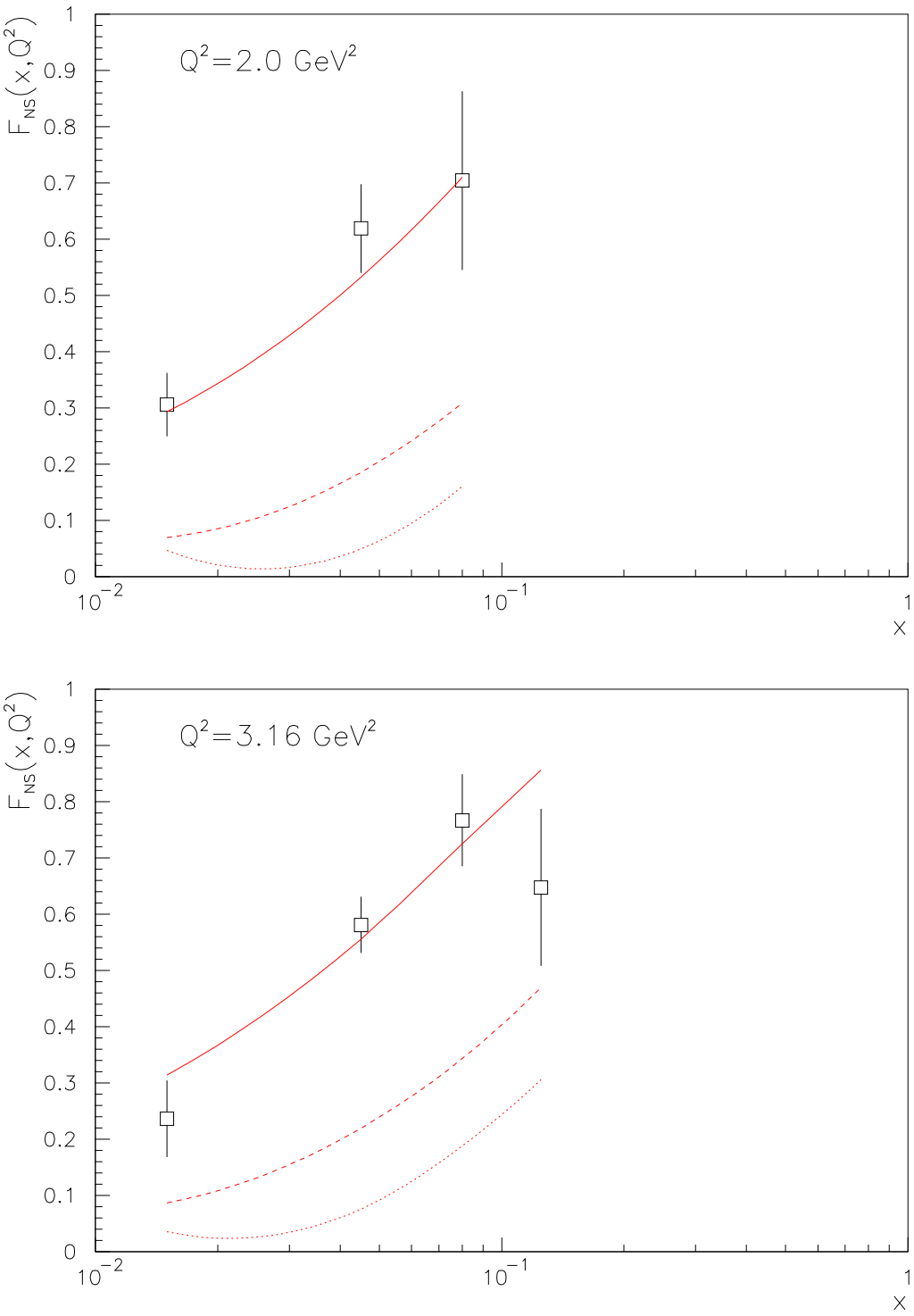
Na sérii obr. 5.5-5.9 je proveden fit řešení evoluční rovnice na experimentální data pro různé hodnoty Q^2. Byly použity hodnoty parametrů zjištěné metodou konjugovaných vektorů, tedy $A = 2.63$, $\alpha = 0.50$, $\beta = 3.41$, $\gamma = 1.53$, $\eta = 0.81$ a $\Lambda = 567$ MeV; $Q^2 = 5$ GeV^2. V těchto grafech je také zaznamenán vývoj při procesu fitování. Je znázorněn přerušovanými křivkami, které představují řešení evoluční rovnice odpovídající parametrům zvoleným na počátku minimalizace a napočítaným parametrům z 11-té iterace ($\chi^2 = 2585$).



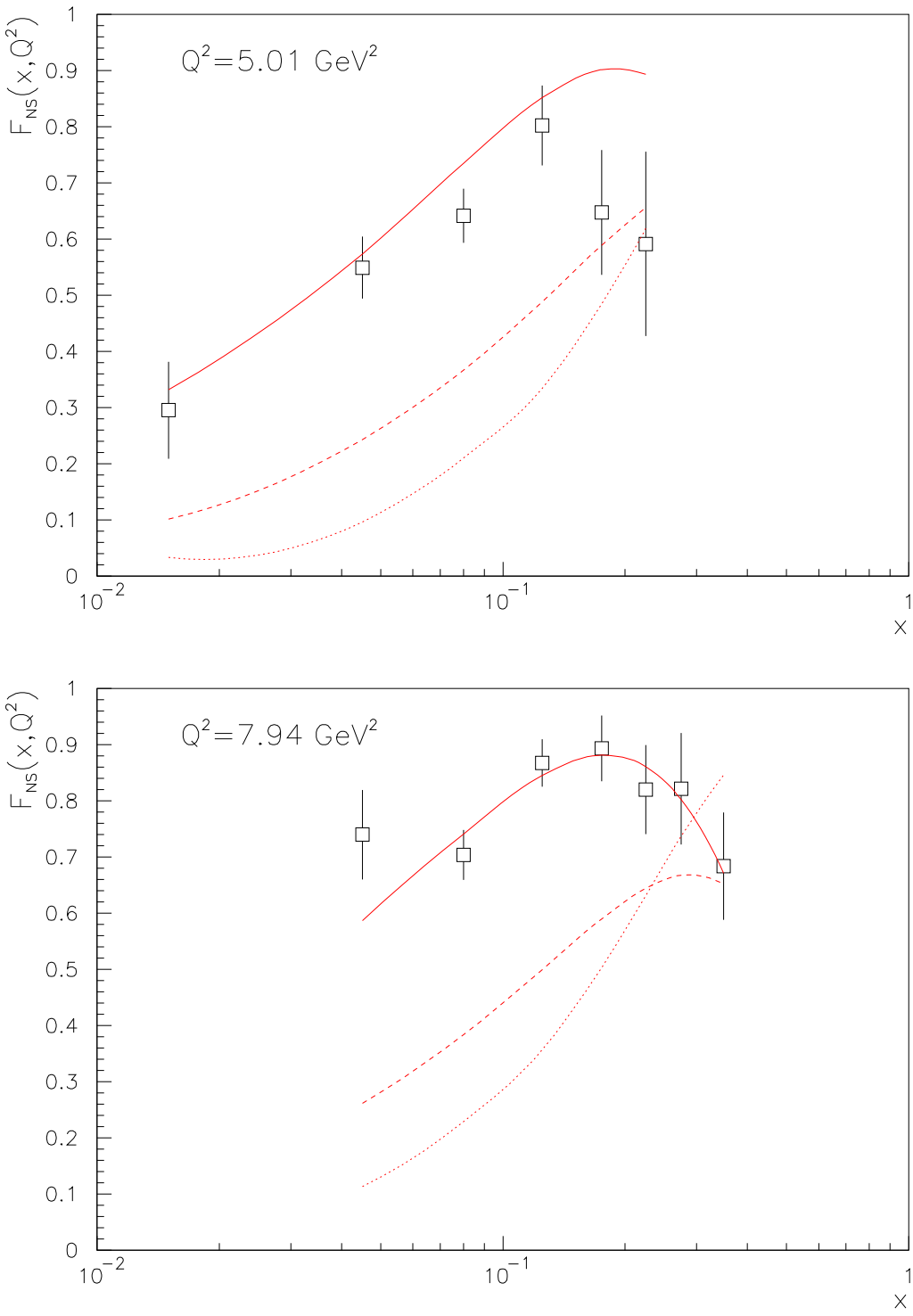
Obrázek 5.3: Vypočtené závislosti hadronové nesingletní distribuční funkce na x pro různé hodnoty Q^2.



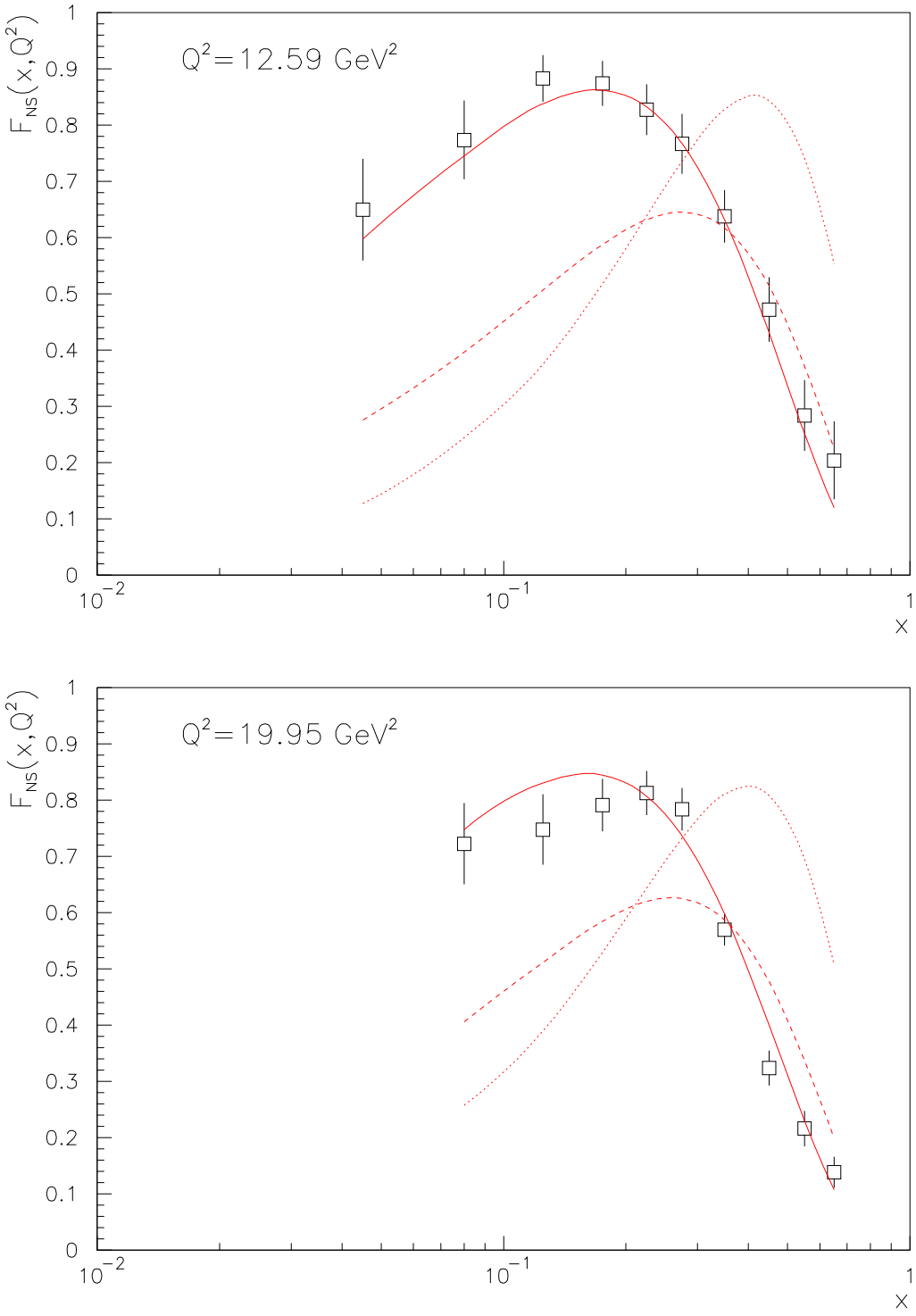
Obrázek 5.4: Vypočtené závislosti hadronové nesingletní distribuční funkce na Q^2 pro různé hodnoty x.



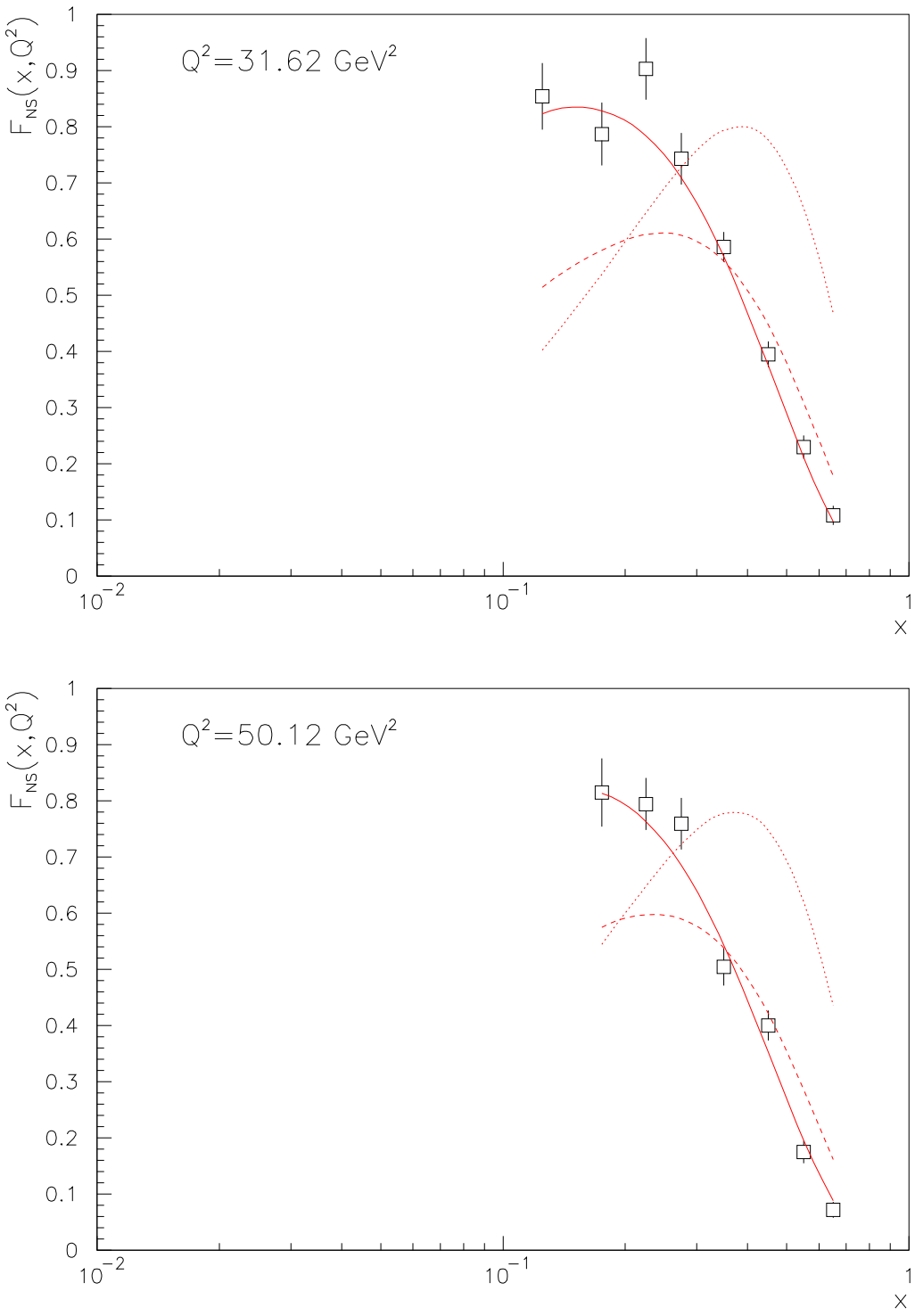
Obrázek 5.5: Srovnání experimentálních dat pro F_{NS} při $Q^2 = 2\ GeV^2$ a $Q^2 = 3.16\ GeV^2$ s hodnotami vypočtenými řešením evolučních rovnic s postupně zpřesňovanými parametry $A,\ \alpha,\ \beta,\ \gamma,\ \eta\ a\ \Lambda$.



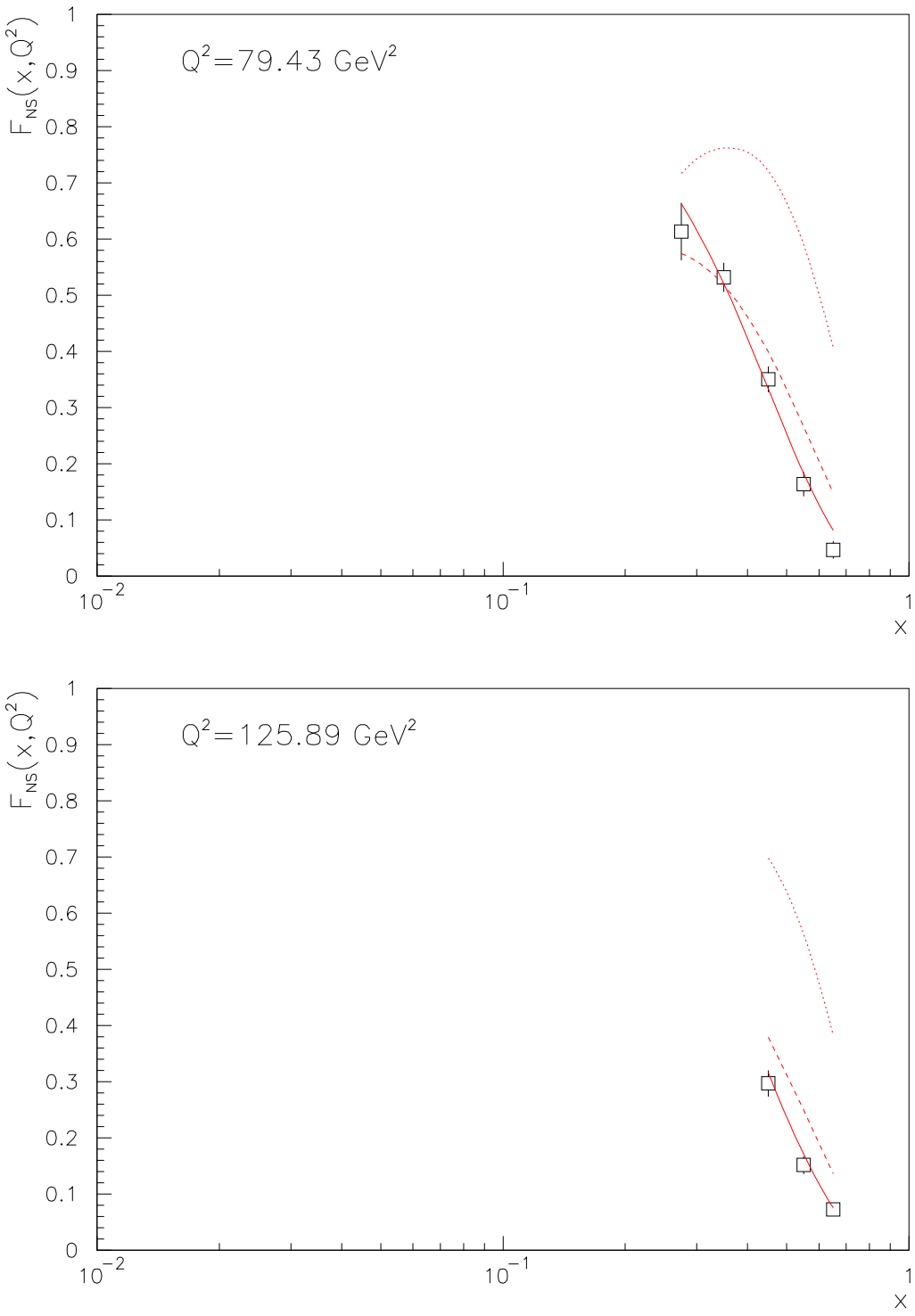
Obrázek 5.6: Srovnání experimentálních dat pro F_{NS} při $Q^2 = 5.01\ GeV^2$ a $Q^2 = 7.94\ GeV^2$ s hodnotami vypočtenými řešením evolučních rovnic s postupně zpřesňovanými parametry $A,\ \alpha,\ \beta,\ \gamma,\ \eta\ a\ \Lambda$.



Obrázek 5.7: Srovnání experimentálních dat pro F_{NS} při $Q^2 = 12.59\ GeV^2$ a $Q^2 = 19.95\ GeV^2$ s hodnotami vypočtenými řešením evolučních rovnic s postupně zpřesňovanými parametry A, α, β, γ, η *a* Λ.



Obrázek 5.8: Srovnání experimentálních dat pro F_{NS} při $Q^2 = 31.62\ GeV^2$ a $Q^2 = 50.12\ GeV^2$ s hodnotami vypočtenými řešením evolučních rovnic s postupně zpřesňovanými parametry A, α, β, γ, η *a* Λ.



Obrázek 5.9: Srovnání experimentálních dat pro F_{NS} při $Q^2 = 79.43\ GeV^2$ a $Q^2 = 125.89\ GeV^2$ s hodnotami vypočtenými řešením evolučních rovnic s postupně zpřesňovanými parametry $A,\ \alpha,\ \beta,\ \gamma,\ \eta\ a\ \Lambda$.

Kapitola 6

Struktura fotonu

Při zkoumání fotonu na velkých vzdálenostech se foton projevuje jako bezstrukturní objekt podlehající zákonům kvantové elektrodynamiky. Nicméně táž částice při zkoumání na malých vzdálenostech vykazuje jisté vlastnosti charakteristické pro hadrony. Tato "fotonová struktura" je kvantifikována, podobně jako v případě hadronů pomocí distribučních funkcí, které splňují jisté evoluční rovnice. Kvůli možnosti větvení $\gamma \to q\bar{q}$ jsou tyto rovnice, narozdíl od případu hadronů, nehomogenní.

Koncept struktury světelného kvanta je založen na principech kvantové teorie pole. V rámci tohoto popisu lze foton (ostatně i jiné elementární částice) vnímat jako častici fluktující v rozličných (virtuálních) stavech částic např. leptonů, kvarků, $W^{\pm}$ bosonů, hadronů atd. Přechod k těmto stavům a možnost jejich pozorování je realizováno konkrétními fyzikálními procesy díky interakcím s jinými objekty. V tomto pojetí můžeme foton v interakci s Coulombovým polem vnímat jako částici projevující se jako elektron-pozitronový pár $\gamma \to e^+e^-$. Tento jev byl vůbec prvním dokladem fotonové struktury.

Podobně může foton při vysokých energiích v interakci s hadrony fluktuovat na pár $q\bar{q}$. V rámci poruchové QCD tak lze dospět k partonové struktuře fotonů.

Protože vysokoenergetické svazky reálných fotonů neexistují, využívají se při zkoumání struktury reálných fotonů srážky e^+e^- nebo $e^{\pm}$ p. Ve všech těchto případech vzniká z elektronového nebo pozitronového paprsku proud "skoro" reálných fotonů. V jistém přiblížení lze ovšem s takovými fotony pracovat jako se zcela reálnými.

6.1 Účinný průřez virtuálních fotonů

Zabývejme se nejprve popisem "srážky" virtuálního fotonu s protonem. Představen bude účinný průřez takové reakce. Na rozdíl od reálného fotonu, který se může nalézat ve dvou spinových stavech, kterým odpovídají dva polarizační čtyřvektory ϵ_μ, u virtuálního fotonu rozlišujeme čtyři nezávislé spinové stavy, popisovanými polarizačními čtyřvektory $\epsilon_\mu^{(i)}$, $i = 1, 2, 3, 4$.

Zacházení s fotony, ať už s reálnými nebo virtuálními, vyžaduje zvolit konkrétní kalibraci. Ta potom vede k dodatečné podmínce na polarizační čtyřvektory ϵ_μ. Použití takzvané Lorentzovy kalibrace nás přivádí k podmínce $\epsilon(q)q = 0$. Takové podmínce můžeme nyní

vyhovět třemi nezávislými polarizačními vektory. Zvolíme-li souřadný systém tak, aby měla hybnost fotonu směr třetí souřadné osy a označíme-li $q = (q_0, q_1, q_2, q_3)$, můžeme je zvolit například takto:

$$\epsilon_T^{(1)} = (0,1,0,0); \quad \epsilon_T^{(2)} = (0,0,1,0); \quad \epsilon_L = \frac{1}{\sqrt{Q^2}}(q_3,0,1,q_0). \tag{6.1}$$

První dva popisují transverzální a třetí longitudiální polarizační vektory virtuálního fotonu se čtyřhybností q. Tyto vektory navíc vyhovují následující normalizační podmínce $(\epsilon_T^{(1,2)})^2 = -1, (\epsilon_L)^2 = 1$.

Nyní zapíšeme totální účinný průřez srážky virtuálního fotonu se čtyřhybností q a polarizačním čtyřvektorem $\epsilon_\mu = (\epsilon_0, \vec{\epsilon})$ s protonem, který je před srážkou v klidu, (označíme-li P jako čtyřhybnost fotonu, potom $P = (M,0,0,0)$):

$$\sigma(\gamma^* p) = C\epsilon^\mu W_{\mu\nu}(P,q)\epsilon^\nu = C(-W_1\epsilon^2 + \epsilon_0^2 W_2), \tag{6.2}$$

kde $W_{\mu\nu}$ je hadronový tenzor (2.11). Pro tři polazizační stavy virtuálních fotonů nyní následujícím způsobem zavedeme longitudiální a transversální účinné průřezy:

$$\sigma_T(\gamma^* p) \equiv \frac{1}{2}\left(\sigma_T^{(1)} + \sigma_T^{(2)}\right) = CW_1, \tag{6.3}$$

$$\sigma_L(\gamma^* p) = C\left(-W_1 + \frac{q_3^2}{Q^2}W_2\right) = C\left(-W_1 + \left(1 + \frac{Q^2}{4M_p^2x^2}\right)W_2\right), \tag{6.4}$$

$$\sigma(\gamma^* p) \equiv (\sigma_T + \sigma_L) = C\left(1 + \frac{Q^2}{4M_p^2x^2}\right)W_2. \tag{6.5}$$

Odtud tedy můžeme psát:

$$CW_1 = \sigma_T \Rightarrow F_1 = \frac{1}{C}\sigma_T \tag{6.6}$$

$$CW_2 = (\sigma_T + \sigma_L)\frac{4M_p^2x^2}{Q^2 + 4M_p^2x^2} \Rightarrow F_2 = \frac{2x}{C}\left[\frac{Q^2}{Q^2 + 4M_p^2x^2}\right](\sigma_T + \sigma_L). \tag{6.7}$$

Vedle funkcí F_1 a F_2 se ještě obvykle zavádějí tzv. transversální a longitudiální strukturní funkce:

$$F_T(x,Q^2) \equiv 2xF_1(x,Q^2) \tag{6.8}$$

$$F_L(x,Q^2) \equiv F_2(x,Q^2)\left(1 + \frac{4M_p^2x^2}{Q^2}\right) - 2xF_1(x,Q^2) \tag{6.9}$$

Z rovnic (6.6) - (6.7) můžeme vyjádřit σ_T a σ_L pomocí strukturních funkcí F_1 a F_2:

$$\sigma_T(x,Q^2) = CF_1(x,Q^2), \tag{6.10}$$

$$\sigma_L(x,Q^2) = \frac{C}{2x}\left[F_2(x,Q^2)\left(1+\frac{4M_p^2x^2}{Q^2}\right)-2xF_1(x,Q^2)\right] = C\frac{F_L(x,Q^2)}{2x}. \tag{6.11}$$

Ve výše uvedených výrazech byl účinný průřez reakce $\gamma^* p$ určen až na konstantu C. V následujícím se tuto konstantu pokusíme určit. V případě reálných fotonů s polarizačním vektorem ϵ je totální účinný průřez reakce γp dán formulí:

$$\sigma(\gamma p) = \frac{4\pi^2\alpha}{E_\gamma M_p}\epsilon^\mu\left(-W_1(W,Q^2=0)g_{\mu\nu}+\frac{P_\mu P_\nu}{M^2}W_2(W,Q^2=0)\right)\epsilon^\nu, \tag{6.12}$$

odkud pro reálný foton máme $C = 4\pi^2\alpha/E_\gamma M_p$. Použití stejné formule pro virtuální fotony s využitím vztahu $E_{\gamma^*} = (W^2 - M^2 + Q^2)/(2M)$ vede k přímému zobecnění:

$$C = \frac{8\pi^2\alpha}{W^2 - M_p^2 + Q^2}. \tag{6.13}$$

Pro virtuální fotony se nicméně za standartní definici C užívá (6.13) ale bez Q^2 ve jmenovateli:

$$C = \frac{8\pi^2\alpha}{W^2 - M_p^2} = \frac{8\pi^2\alpha x}{Q^2(1-x)}. \tag{6.14}$$

Odtud již můžeme psát vztah mezi F_2 a $\sigma(\gamma^* p)$:

$$F_2(x,Q^2) = \sigma(\gamma^* p, W, Q^2)\left[\frac{Q^2}{Q^2+4M_p^2x^2}\right]\frac{Q^2(1-x)}{4\pi^2\alpha}, \tag{6.15}$$

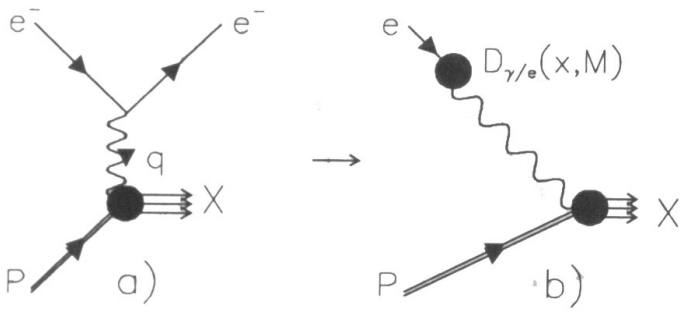
$$\sigma(\gamma^* p, W, Q^2) = \frac{F_2(x,Q^2)}{Q^2}\left[\frac{Q^2+4M^2x^2}{Q^2}\right]\frac{4\pi^2\alpha}{1-x}. \tag{6.16}$$

6.2 Aproximace ekvivalentních fotonů

V následující kapitole bude diskutován jeden z konceptů popisu struktury fotonu. Bude popisován v rámci kvantové elektrodynamiky (QED). V kapitole 2.2 byl popsán základní kinematický popis srážky e+p → cokoliv. Uvedli jsem, že diferenciální účinný průřez takové srážky je dán formulí (2.10). Zkusme se nyní zabývat stejným procesem, tentokrát však v kinematické oblasti, ve které je $Q^2 << W^2$. Ukážeme, že v takovém případě lze formuli pro účinný průřez zjednodušit a vyjádřit totální účinný průřez $\sigma_{tot}(ep)$ pomocí totálního účinného průřezu $\sigma_{tot}(\gamma p)$ reakce reálného fotonu na protonu:

$$\gamma + p \to cokoliv. \tag{6.17}$$

Takový krok můžeme učinit za předpokladu, že lze v reakci fotonů s protonem nahradit fotony s malou virtualitou fotony reálnými. Ve fyzikálním smyslu malá virtualita znamená,



Obrázek 6.1: Vztah mezi účinnými průřezy srážek ep (a) a γp (b) v oblasti malých Q^2.

že se foton v CMS chová jako skoro reálný a pohybuje se skoro rovnoběžně se směrem nalétajících elektronů. Nabízí se přirozená představa proudu elektronů, který je obklopen chomáčem fotonů s určitým rozdělením hybnosti. Takové rozdělení nyní odvodíme.

Vyjdeme z výrazu pro účinný průřez procesu e+p $\rightarrow$ cokoliv. Bez újmy na obecnosti sledujme tento proces v soustavě spojené s protonem,

$$d\sigma = \frac{(2\pi)^4 e^4}{2E2M} L^{\mu\nu} W_{\mu\nu}(q,p) \frac{d^3k'}{(2\pi)^3 2E_{k'}} \frac{1}{Q^4}, \tag{6.18}$$

kde

$$L^{\mu\nu} \equiv \frac{1}{2} Tr[(\not{k}' + m)\gamma^\mu(\not{k} + m)\gamma^\nu] = 2[k^\mu k'^\nu + k^\nu k'^\mu - g^{\mu\nu}(kk' - m^2)] \tag{6.19}$$

je leptonový tenzor asociovaný s horním elektronovým vertexem. $W_{\mu\nu}(q,p)$ je hadronový tenzor asociovaný s dolním vertexem (obr. 6.1a). V něm je reflektována struktura příslušného hadronu - v našem případě protonu. Jeho konkrétní tvar představíme o něco později.

Ještě předtím zapíšeme formuli pro proces (6.17):

$$\sigma_{tot}(\gamma p) = \frac{(2\pi)^4 e^2}{2E_\gamma 2M} \left(-\frac{1}{2} g^{\mu\nu}\right) W_{\mu\nu}(Q^2 = 0, pq), \tag{6.20}$$

kde $W_{\mu\nu}$ je stejný hadronový tenzor jako v (6.18). Faktor 1/2 vznikl středováním přes spinové stavy přicházejícího fotonu.

Připomeňme tvar nejobecnějšího symetrického hadronového tenzoru pro rozptyl nepolarizovaného elektronu na nepolarizovaném protonu:

$$W_{\mu\nu} = C_1(Q^2, pq) g_{\mu\nu} + C_2(Q^2, pq) p_\mu p_\nu + C_3(Q^2, pq) q_\mu q_\nu + C_4(Q^2, pq)(p_\mu q_\nu + p_\nu q_\mu) \tag{6.21}$$

kde $C_i(Q^2, pq)$ jsou funkce kvadrátu přeneseného čtyřimpulsu z elektronu na proton Q^2 a součinu čtyřhybností pq. Tyto funkce v sobě nesou úplnou informaci o struktuře nepolarizovaného protonu. Z požadavku kalibrační invariance lze odvodit, že jen dvě z funkcí C_i jsou nezávislé:

$$C_4(Q^2,pq) = -C_1(Q^2,pq)\frac{1}{(pq)} - C_3(Q^2,pq)\frac{q^2}{(pq)} \tag{6.22}$$

$$C_2(Q^2,pq) = C_1(Q^2,pq)\frac{q^2}{(pq)^2} + C_3(Q^2,pq)\frac{q^4}{(pq)^2} \tag{6.23}$$

Předpokládejme nyní, že všechny funkce C_i jsou regulární v bodě $Q^2 = 0$. Do učinného průřezu uvažované reakce (6.17) s reálným fotonem vstupuje po kontrakci hadronového tenzoru $W_{\mu\nu}$ s metrickým tenzorem $g^{\mu\nu}$ pouze funkce $C_1(Q^2 = 0, pq)$:

$$-\frac{1}{2}g^{\mu\nu}W_{\mu\nu}(Q^2 = 0, pq) = -C_1(Q^2 = 0, pq). \tag{6.24}$$

Naproti tomu do výrazu (6.18) ve které vystupují fotony s malou, ovšem nenulovou virtualitou přispívají skrze kontrakci hadronového tenzoru $W_{\mu\nu}$ s leptonovým tenzorem $L^{\mu\nu}$ členy úměrné funkcím C_1 a C_2:

$$L^{\mu\nu}W_{\mu\nu} = 2[-2C_1(Q^2,pq)(kk' - 2m^2) + C_2(Q^2,pq)(2(kp)(k'p) - M^2(kk'))]. \tag{6.25}$$

Dosaďme nyní do posledního výrazu za funkci $C_2(Q^2, pq)$ z (6.23) a v takto získaném výrazu - vzhledem ke zkoumané kinematické oblasti - ponechme pouze členy úměrné Q^2, vyšší řády zanedbejme. Ještě předtím ve výrazu (6.25) vyjádříme kinematické veličiny v nich vystupující pomocí veličin Q^2 resp. q. Z definice (2.3) snadno najdeme, že

$$kk' = Q^2/2 \quad a \quad (kp)(k'p) = (kp)((k-q)p) = (kp)^2 - (kp)(pq) \tag{6.26}$$

Nyní už můžeme psát:

$$L^{\mu\nu}W_{\mu\nu} = -2C_1(Q^2,pq)Q^2\left[1 + 2\left(\frac{kp}{qp}\right)^2 - 2\left(\frac{kp}{qp}\right) - \frac{2m^2}{Q^2}\right] \tag{6.27}$$

$$= -2C_1(Q^2,pq)Q^2\left[\frac{1+(1-y)^2}{y^2} - \frac{2m^2}{Q^2}\right], \tag{6.28}$$

kde $y \equiv (qp)/(kp)$. V klidové soustavě terčíkové částice y fyzikálně interpretujeme jako frakci energie naletávajícího elektronu, nesenou vyměňovaným, virtuálním fotonem.

Pokračujme nyní v úpravě výrazu (6.18). Připomeňme snahu vyjádřit totální účinný průřez reakce ep pomocí účinného průřezu reakce γp. Za tím účelem bude potřeba ve výrazu (6.18) integrovat přes hybnost koncového stavu. Vzhledem k použitým proměnným ve výrazu (6.28) přejdeme od diferenciálů d^3k' k diferenciálům dy a k příčné složce čtyřhybnosti q - dq_T. V zásadě půjde o přechod k cylindrickým souřadnicím. Všechny úvahy budeme provádět v přiblížení, kdy $q_T << q_\parallel$, tj. vyzářený foton se pohybuje téměř stejným směrem jako nalétavající elektron (tzv. kolineární kinematika):

$$\frac{d^3k'}{2E_{k'}} = \frac{q_T dq_T dq_\parallel d\phi}{2E_{k'}} = \frac{dq_T^2 dq_\parallel d\phi}{4E_k(1-y)} \doteq \frac{d\phi}{4(1-y)} dy dq_T^2 \tag{6.29}$$

V posledním výrazu jsme užili aproximace $y = q_\parallel / E_k$. V kolineární kinematice také můžeme psát:

$$Q^2 = 4E_k E_{k'} \sin^2(\theta/2) \doteq \frac{q_T^2}{1-y}. \tag{6.30}$$

Použitím posledních dvou vyjádření spolu s (6.28) v (6.18) získáváme vyjádření totálního účinného průřezu reakce ep pro malá Q^2:

$$\sigma_{ep} = \int\int dy \frac{dq_T^2}{q_T^2} \left[\frac{\alpha}{2\pi} \left(\frac{1+(1-y)^2}{y} - \frac{2m^2 y}{Q^2} \right) \right] \sigma_{\gamma p}. \tag{6.31}$$

V posledním výrazu, představující hledaný vztah mezi výše představenými účinnými průřezy je vzhledem k použitým přiblížením nutné provádět integraci přes takové hodnoty y a q_T, které odpovídají kinematické oblasti $Q^2 < Q^2_{max}$.

Funkci

$$\frac{\alpha}{2\pi} \frac{1}{Q^2} \left(\frac{1+(1-y)^2}{y} - \frac{2m^2 y}{Q^2} \right) \tag{6.32}$$

je možné vzhledem k výrazu (6.31) interpretovat jako pravděpodobnost nalezení fotonu s virtualitou Q^2 a hybností yP, kde P značí hybost elektronu.

Pokusme se nyní dále upravit výraz (6.31). Přejděme v něm nejprve pomocí vztahu (6.29) od proměnné q_T^2 k proměnné Q^2:

$$\sigma_{ep} = \int\int dQ^2 \frac{dq_T^2}{q_T^2} \frac{\alpha}{2\pi} \left(\frac{1+(1-y)^2}{y} \frac{1}{Q^2} - \frac{2m^2 y}{Q^4} \right) \sigma_{\gamma p}. \tag{6.33}$$

a integrujme přes tuto proměnnou v intervalu $(Q^2_{min},\ Q^2_{max})$, kde spodní hranice Q^2_{min} je daná kinematikou:

$$Q^2_{min} = \frac{m^2 y^2}{1-y} \tag{6.34}$$

Obdržíme tak následující výraz:

$$\sigma_{ep} \doteq \int dy \frac{\alpha}{2\pi} \left(\left[\frac{1+(1-y)^2}{y} \right] \ln \frac{Q^2_{max}}{Q^2_{min}} - 2m^2 y \left[\frac{1}{Q^2_{min}} - \frac{1}{Q^2_{max}} \right] \right) \sigma_{\gamma p} \tag{6.35}$$

Označme funkci vystupující pod integrálem v součinu se $\sigma_{\gamma p}$ jako $f_{\gamma/e}(y, Q^2_{max}, Q^2_{min})$ a dosaďme do ní za Q^2_{min} z (6.34). Pokud nyní přeznačíme Q^2_{max} na volný parametr Q^2, dostáváme:

$$f_{\gamma/e}(y,Q^2) = \frac{\alpha}{2\pi}\left[\frac{1+(1-y)^2}{y}\ln\frac{Q^2(1-y)}{m^2y^2} - \frac{2(1-y)}{y} + O(m^2/Q^2)\right] \quad (6.36)$$

$$= \frac{\alpha}{2\pi}\left[\frac{1+(1-y)^2}{y}\ln\frac{Q^2}{m^2} + \frac{1+(1-y)^2}{y}\ln\frac{1-y}{y^2} - \frac{2(1-y)}{y}\right] \quad (6.37)$$

Tato funkce se interpretuje jako distribuční funkce fotonů nesoucích frakci y energie elektronu a mající virtualitu nejvýše Q^2. V praxi bývá y vzdálené bodům 0 a 1, blízko kterých může být druhý logaritmický člen obrovský. Proto je ve zmíněném výrazu obvyklé ponechávat pouze první logaritmický člen.

Analogický postup lze užít v situaci, kdy nalétávající částicí je (v našem přiblížení reálný) foton, větvící se na pár e^+e^-. Distribuční funkce elektronů (nebo pozitronů) uvnitř fotonu, nesoucí zlomek y jeho energie a mající virtualitu nejvýše Q^2 je dána výrazem:

$$f_{e/\gamma}(y,Q^2) = \frac{\alpha}{2\pi}\left[y^2+(1-y)^2\right]\ln\frac{Q^2}{m^2}. \quad (6.38)$$

kde jsme ze stejných důvodů jako v předcházejícím případě zanedbaly členy typu $\ln y$ a $\ln(1-y)$.

6.3 Popis srážky $e^-\gamma$

Jak již bylo řečeno, současných experimentů na nichž je založené studium "struktury" reálných fotonu se žádné svazky vysokoenergetických reálných fotonů neúčastní. Pohled na jejich strukturu nám může přinést například studium srážky e^+e^-, v rámci kterých zkoumáme hlubokonepružný rozptyl elektronu na skoro reálném fotonu. V takovém případě zkoumané "reálné" fotony pocházejí z elektronových nebo pozitronových svazků. Rozdělovací funkcí fotonů v elektronu (příp. pozitronu) jsme se zabývali v předcházející kapitole. Cílem této části bude představit obdobnou veličinu, distribuční funkci kvarků ve fotonu.

Uvažujme nyní hluboko nepružný rozptyl:

$$e(k)\gamma(p) \to e(k')hadrony, \quad (6.39)$$

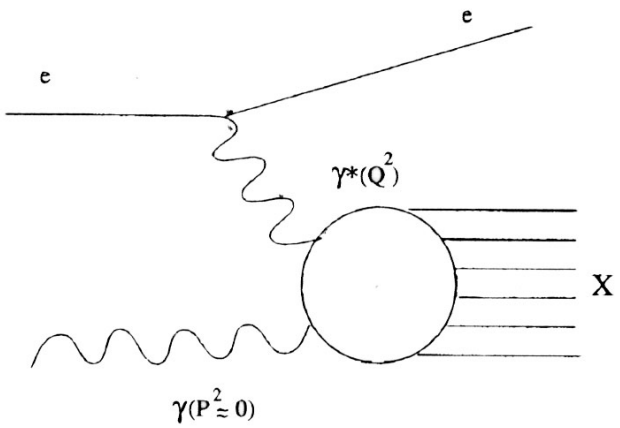
při srážce e^+e^- (obr. 6.2). V takovém případě považujeme za terč reálný foton (ve skutečnosti skoro reálný, $p^2 \sim 0$) vyzářený vstupujícím elektronem.

Při srážce (6.39) je reálný foton se čtyřhybností p a virtualitou $P^2 = -p^2 = 0$ zkoumán vysoce virtuálním fotonem se čtyřhybností $q = k - k'$, kde k a k' jsou čtyřhybnosti vstupujícího elektronu před a po srážce, přičemž $Q^2 = -q^2 > 0$.

Diferenciální účinný průřez reakce (6.39) je dán následující formulí:

$$\frac{d\sigma(e\gamma \to eX)}{dxdy} = \frac{2\pi\alpha^2}{Q^4}\left[\left(1+(1-y)^2\right)F_2^\gamma - y^2F_L^\gamma\right]. \quad (6.40)$$

Proměnné x a y jsou standardními proměnnými popisujícími hluboko nepružný rozptyl:



Obrázek 6.2: Hluboký nepružný rozptyl na (skoro) reálném fotonu eγ ($P^2 \approx 0$) $\rightarrow eX$. Zkoumající foton má ve srovnání se zkoumaným fotonem velmi velkou virtualitu.

$$x = \frac{Q^2}{2pq}, \qquad y = \frac{pq}{pk} \tag{6.41}$$

Připomeňme, že v případě masivního terče (např. protonu) má proměnná y v klidové soustavě terčíkové částice význam frakce energie nalétavajícího elektronu nesenou vyměňovaným fotonem. V praxi, při uvažovaném hluboko nepružném rozptylu $e\gamma$ bývá proměnná y velmi malou, proto se někdy ve výrazu (6.40) člen úměrný F_L^γ zanedbává.

V rámci čisté QED je strukturní funkce F_2^γ dána tzv. Bethe-Heitlerovu formuli:

$$F_2^\gamma(x,Q^2) = \frac{\alpha}{\pi} N_c \sum_{i=1}^{N_f} e_{q_i}^4 x \left[\left(-1 + 8x(1-x) - x(1-x)\frac{4m_{q_i}^2}{Q^2} \right) \beta \right. \tag{6.42}$$

$$\left. + \left[x^2 + (1-x)^2 + x(1-3x)\frac{4m_{q_i}^2}{Q^2} \right] \ln \frac{1+\beta}{1-\beta} \right]. \tag{6.43}$$

kde

$$\beta = \sqrt{1 - \frac{m_{q_i}^2 x}{Q^2(1-x)}}. \tag{6.44}$$

Pro $Q^2 \gg m_{q_i}^2$ lze logaritmický člen v (6.43) aproximovat jako

$$\ln \frac{1+\beta}{1-\beta} \approx \ln \frac{Q^2(1-x)}{m_{q_i}^2 x}. \tag{6.45}$$

Strukturní funkci F_2^γ pak v takovém případě můžeme vyjádřit následovně:

$$F_2^\gamma = \frac{\alpha}{\pi} N_c \sum_{i=1}^{N_f} e_{q_i}^4 x \left[\left[x^2 + (1-x)^2 \right] \ln \frac{Q^2}{m_{q_i}^2} + (x^2 + (1-x)^2) \ln \frac{1-x}{x} + 8x(1-x) - 1 \right]. \tag{6.46}$$

Protože v rámci partonového modelu platí mezi strukturní funkcí F_2^γ a distribučními funkcemi kvarků ve fotonu poněkud jiný vztah než u hadronů:

$$F_2^\gamma = x \sum_{i=1}^{2N_f} e_{q_i}^2 \left[q_i^\gamma(x, Q^2) + (x^2 + (1-x)^2) \ln \frac{1-x}{x} + 8x(1-x) - 1 \right] \tag{6.47}$$

implikuje (6.46) následující vztah

$$q_i^\gamma(x, Q^2) = \frac{\alpha}{\pi} N_C e_{q_i}^2 [x^2 + (1-x)^2] \ln \frac{Q^2}{m_{q_i}^2}. \tag{6.48}$$

Je důležité poznamenat, že fotonová strukturní funkce F_2^γ je v partonovém modelu narozdíl od strukturní funkce hadronu, t.j. nukleonové strukturní funkce F_2^N spočitatelná. Ponecháme-li ve výrazu (6.46) pouze dominantní člen $\ln Q^2/m_{q_i}^2$, můžeme pro kvarkovou distribuční funkci ve fotonu psát:

Tento výraz je obdobný distribuční funkci elektronu ve fotonu (6.38) - jen místo jednotkového elektrického náboje zde vystupuje faktor $N_c e_{q_i}^2$.

Kapitola 7

Popis srážek dvou fotonů

V předcházející kapitole byla představena elektron-pozitronová a kvark-antikvarková distribuční funkce fotonu. Obě byly odvozeny v rámci kvantové elektrodynamiky. Cílem této kapitoli bude představit tyto funkce v poněkud obecnějším kontextu. Výchozí situací pro nás bude srážka e^-e^-, kterou se budeme následně zabývat v takových kinematických oblastech, ve kterých bude možné kvantifikovat pojem struktury fotonu právě v termínech těchto distribučních funkcí.

Existuje celá řada fyzikálních problémů která je spojená se dvou fotonovými reakcemi. Snahou této části bude odvození diferenciálního účinného průřezu reakce, schematicky znázorněné na obr. 7.1. Taková reakce je významným zdrojem poznání struktury fotonu. Popis takové struktury, vyžaduje schopnost extrahovat z diferenciálního účinného průřezu procesu $e^{\pm}e$ příspěvek k podprocesům se zkoumanými fotony. V našem případě se budeme zajímat - prozatím ve vší obecnosti - na popis přechodu

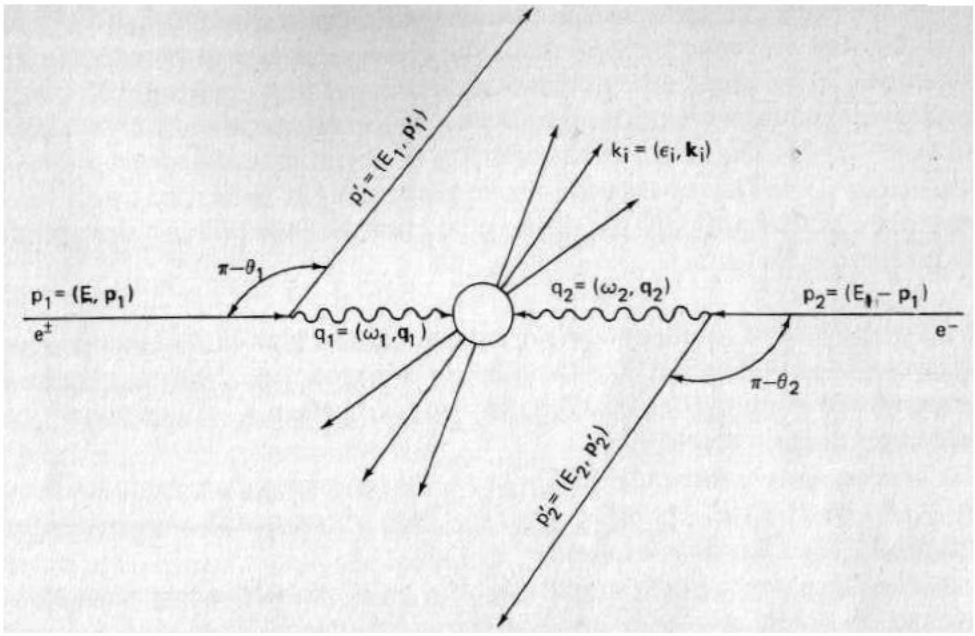
$$\gamma\gamma \to f \tag{7.1}$$

kde f je koncový blíže neurčený hadronový stav. Jak se ukáže, matice přechodu takového procesu souvisí s účinným průřezem základního procesu $e^{\pm}e$ poměrně komplikovaně. Nicméně, existují kinematické oblasti, ve kterých se dané formule výrazně zjednoduší. Budeme se jimi podrobně zajímat. Námi získané formule budou použitelné pro celou třídu srážek, kterých se učastní dva fotony. Navzájem se mohou lišit druhem základních srážejících se částic nebo produkovaným systémem f. Zároveň taková vyjadření umožňují, kromě extrakce informace o procesu $\gamma\gamma \to f$, případně výpočtu účinného průřezu, pokud je amplituda takového procesu známa, snadno přecházet do různých aproximativních režimů.

Účinný průřez pro dvou fotonovou produkci v procesu srážky e^-e^- můžeme pomocí amplitudy $M^{\mu\nu}$ procesu $\gamma\gamma \to f$ zapsat následujícím způsobem:

$$d\sigma = \frac{(4\pi\alpha)^2}{q_1^2 q_2^2}\rho_1^{\mu\mu'}\rho_2^{\nu\nu'}M^{*\mu'\nu'}M^{\mu\nu}\frac{(2\pi)^4\delta(q_1+q_2-k)d\Gamma}{4\{(p_1p_2)^2-m_1^2m_2^2\}^{1/2}}\frac{d^3p_1'd^3p_2'}{2E_1 2E_2(2\pi)^6}. \tag{7.2}$$

Zde $p_{1,2}$ jsou hybnosti srážejících se elektronů, $q_i = p_i - p_i'$ jsou hybnosti virtuálních fotonů, $k = \sum_i k_i = q_1 + q_2$ je celková hybnost produkovaného systému f s hmotou $W = \sqrt{k^2}$ a



Obrázek 7.1: Diagram ppisující srážku $e^{\pm}e^{-}$

fázový objem je $d\Gamma = \prod_j d^3k_j/2\epsilon_j(2\pi)^3$. Matice ρ_i má význam matice hustoty pro virtuální foton, generovaný i-tou částicí. V případě elektronu máme:

$$\rho_i^{\alpha\beta} = -\left(g^{\alpha\beta} - \frac{q_i^\alpha q_i^\beta}{q_i^2}\right) - \frac{(2p_i - q_i)^\alpha (2p_i - q_i)^\beta}{q_i^2}. \tag{7.3}$$

Po integraci přes možné stavy systému f vstupuje do formule (7.2) výraz:

$$W^{\mu'\nu',\mu\nu} = \frac{1}{2}\int M^{*\mu'\nu'} M^{\mu\nu} (2\pi)^4 \delta(q_1 + q_2 - k) d\Gamma \tag{7.4}$$

Veličinu $W^{\mu'\nu',\mu\nu}$ můžeme zapsat pomocí tenzorů, konstruovaných pomocí čtyřvektorů q_1, q_2 a tenzoru $g^{\alpha\beta}$. Od toho výrazu vyžadujeme Lorentzovskou invarianci, T-invarianci (symetrie v záměně $\mu'\nu' \leftrightarrow \mu\nu$) a kalibrační invarianci. Tyto požadavky jsou shrnuty v následujících podmínkách:

$$q_1^\mu W^{\mu'\nu',\mu\nu} = q_1^{\mu'} W^{\mu'\nu',\mu\nu} = q_2^\nu W^{\mu'\nu',\mu\nu} = q_2^{\nu'} W^{\mu'\nu',\mu\nu} = 0. \tag{7.5}$$

Zmiňované tenzory zapíšeme lineárními kombinacemi q_1, q_2 a $g^{\alpha\beta}$:

$$Q_1 = \sqrt{\frac{-q_1^2}{X}}\left[q_2 - q_1\frac{(q_1q_2)}{q_1^2}\right]; \quad Q_2 = \sqrt{\frac{-q_1^2}{X}}\left[q_1 - q_2\frac{(q_1q_2)}{q_1^2}\right] \tag{7.6}$$

$$R^{\alpha\beta} = R^{\beta\alpha} = -g^{\alpha\beta} + X^{-1}[(q_1q_2)(q_1^\alpha q_2^\beta + q_1^\beta q_2^\alpha) - q_1^2 q_2^\alpha q_2^\beta - q_2^2 q_1^\alpha q_1^\beta] \tag{7.7}$$

kde

$$X = (q_1 q_2)^2 - q_1^2 q_2^2 \tag{7.8}$$

Výhodnost takové volby spočívá v tom, že jednotkové vektory Q_i jsou ortogonální k vektorům q_i a symetrický tenzor $R^{\alpha\beta}$ je ortogonální ke q_1 a q_2 (resp. Q_1, Q_2):

$$q_1 Q_1 = q_2 Q_2 = 0; \quad q_i^\alpha R^{\alpha\beta} = Q_i^\alpha R^{\alpha\beta} = 0; \quad Q_1^2 = Q_2^2 = 1; R^{\alpha\beta} R^{\alpha\beta} = 2. \tag{7.9}$$

Použití takové sady tenzorů nám pozdější zacházení s veličinou $W^{\mu'\nu',\mu\nu}$ v mnohém ulehčí. Navíc pro funkce stojící u kombinací použitých tenzorů Q_1,Q_2 a $R^{\alpha\beta}$ při vyjádření veličiny $W^{\mu'\nu',\mu\nu}$ bude možné nalézt jednoduchou fyzikální interpretaci:

$$W^{\mu'\nu',\mu\nu} = R^{\mu\mu'} R^{\nu\nu'} W_{TT} + R^{\mu\mu'} Q_2^\nu Q_2^{\nu'} W_{TS} + Q_1^\mu Q_1^{\mu'} R^{\nu\nu'} W_{ST} + Q_1^\mu Q_1^{\mu'} Q_1^\nu Q_1^{\nu'} W_{SS} \tag{7.10}$$

Nově představené funkce W_{ab} závisejí pouze na invariantech $W^2 = (q_1 + q_2)^2, q_1^2$ a q_2^2. Je možné vyjádřit je pomocí účinných průřezů σ_{ab} ($ab \equiv S$ pro skalární fotony a $ab \equiv T$ pro transversální fotony):

$$W_{TT} = 2\sqrt{X}\sigma_{TT} \quad W_{SS} = 2\sqrt{X}\sigma_{SS} \tag{7.11}$$

$$W_{TS} = 2\sqrt{X}\sigma_{TS} \quad W_{ST} = 2\sqrt{X}\sigma_{ST} \tag{7.12}$$

Protože tenzor $W^{\mu'\nu',\mu\nu}$ je regulární v $q_i^2 \to 0$, účinný průřez skalárních fotonů vymizí. Veličina σ_{TT} se potom transformuje na veličinu reálného fotoprocesu. Konkrétně, při $q_i^2 = 0$ představuje σ_{TT} účinný průřez srážky $\gamma\gamma \to f$ dvou nepolarozovaných reálných fotonů.

Použitím výrazu (7.10) v definujícím výraze (7.2) pro diferenciální účinný průřez základního procesu dostáváme:

$$\begin{aligned} d\sigma = \frac{\alpha^2}{16\pi^4 q_1^2 q_2^2} \left[\frac{(q_1 q_2)^2 - q_1^2 q_2^2}{(p_1 p_2)^2 - m_1^2 m_2^2}\right]^{1/2} & [4\rho_1^{++}\rho_2^{++}\sigma_{TT} \\ & + 2\rho_1^{++}\rho_2^{00}\sigma_{TS} + 2\rho_1^{00}\rho_2^{++}\sigma_{ST} + \rho_1^{00}\rho_2^{00}\sigma_{SS}] \end{aligned} \tag{7.13}$$

Veličiny ρ_i^{ab} představují kontrakce matic $\rho_i^{\alpha\beta}$ s tenzory ve výrazu (7.2). Všechny veličiny jsou tedy vyjádřeny pomocí měřitelných hybností p_i a p_i'. Přesněji:

$$\begin{aligned} & 2\rho_1^{++} = 2\rho_1^{--} = \rho_1^{\alpha\beta} R^{\alpha\beta} = X^{-1}(2p_1 q_2 - q_1 q_2)^2 + 1 + 4m_e^2/q_1^2; \\ & \rho_1^{00} = \rho_1^{\alpha\beta} Q_1^\alpha Q_1^\beta = X^{-1}(2p_1 q_2 - q_1 q_2)^2 - 1; \\ & |\rho_2^{ab}| = |\rho_1^{ab}(1 \leftrightarrow 2)|; \quad |\rho_i^{+-}| = \rho_i^{++} - 1; \quad |\rho_i^{+0}| = \sqrt{(\rho_i^{00} + 1)|\rho_i^{+-}|} \end{aligned} \tag{7.14}$$

Všechny výše uvedené platí pro nepolarizované elektrony.

7.0.1 Dvoufotonová produkce ve vybraných kinematických oblastech

Pokud jsou známy hybnosti rozptýlených elektronů, je rovnicemi (7.13) a (7.14) kompletně vyřešena otázka extrakce informace o procesu $\gamma\gamma \to h$ z experimentálních dat. Měřením úhlových a energetických rozdělení rozptýlených elektronů pomocí (7.13) a (7.14) umožňuje nalézt čtyři amplitudy procesu $\gamma\gamma \to h : \sigma_{TT}, \sigma_{TS}, \sigma_{ST}, \sigma_{SS}$ závisejících na W^2, q_1^2, q_2^2. Koeficienty ρ_i^{ab} před těmito funkcemi závisejí pouze hybnostech elektronů. V tomto smyslu jsou plně určeny výrazem (7.14). Existuje několik kinematických oblastí ve kterých lze zmíněné formule značně zjednodušit. Těmi se budeme v následující části zabývat.

A. Hlavní příspěvek k účinnému průřezu pochází z kinematické oblasti velmi malých úhlů θ_i rozptýlených elektronů. Tomu odpovídají i malé virtuality q_i. Předpokládejme tedy, že $m_e^2 \ll |q_i^2| \ll W^2$. Potom z (7.13) dostáváme:

$$\frac{d\sigma}{dE_1 dE_2 d\Omega_1 d\Omega_2} = \left(\frac{\alpha}{8\pi^2}\right) \frac{(E^2+E_1^2)(E^2+E_2^2)}{E^4(E-E_1)(E-E_2)\sin^2\frac{1}{2}\theta_1 \sin^2\frac{1}{2}\theta_2} \sigma^{exp}_{\gamma\gamma\to h} \tag{7.15}$$

kde

$$\sigma^{exp}_{\gamma\gamma\to h} = \sigma_{TT} + \xi_2\sigma_{TS} + \xi_1\sigma_{ST} + \xi_1\xi_2\sigma_{SS}. \tag{7.16}$$

V této aproximaci je

$$\xi_i = \frac{2EE_i}{E^2+E_i^2} \tag{7.17}$$

B. Soustřeďme se nyní na situaci, kdy je jen jeden z úhlů rozptylu θ_i (řekněme např. θ_2) velmi malý a tedy $Q_2^2 \ll Q_1^2$. Tehdy můžeme podproces, v němž vystupují fotony interpretovat jako rozptyl vysoce virtuálního fotonu na fotonu reálném. Ve studovaných experimentech je za takových okolností obvyklé měřit energie a rozptylové úhly obou elektronů. Odpovídající formule pro účinný průřez takového procesu je potom následující:

$$\frac{d\sigma}{dE_1 dE_2 d\cos\theta_1} = \frac{\alpha}{4\pi} \frac{(E^2+E_1^2)N(E_2,\theta_m)}{E^2(E-E_1)(E-E_2)\sin^2\frac{1}{2}\theta_1} (\sigma_{TT} + \xi_1\sigma_{ST}); \tag{7.18}$$

kde

$$N(E_2,\theta_m) = \frac{\alpha}{\pi}\left[\frac{E^2+E_2^2}{E^2}\ln\frac{EE_2\theta_2}{m_e(E-E_2)} - \frac{E_2}{E}\right]. \tag{7.19}$$

$$\sigma_{TT} = \quad \pi\alpha^2\left[\left(\frac{2}{q_1q_2} + \frac{q_1^2W^2}{(q_1q_2)^3}\right) 2\ln\left(\frac{W}{2m} + \sqrt{\frac{W^2}{4m^2}-1}\right)\right.$$

$$- \left(\frac{2}{q_1 q_2} + \frac{2q_1^2 W^2}{(q_1 q_2)^3} \right) \sqrt{1 - \frac{4m^2}{W^2}} \Bigg] \tag{7.20}$$

$$\sigma_{ST} = \quad -2\pi\alpha^2 \left(\frac{q_1^2 W^2}{(q_1 q_2)^3} \right) \sqrt{1 - \frac{4m^2}{W^2}} \tag{7.21}$$

7.0.2 Extrakce fotonových distribučních funkcí

V následující kapitole se budeme poněkud detailněji zabývat procesem $e^{\pm}e^{-}$ v kinematické oblasti popsané v části B. předchozí kapitoly. Snahou bude zapsat formuli (7.18) popisující takový proces pomocí distribučních funkcí partonů ve fotonu.

Soustřeďme se nejprve na logaritmický výraz $N(E_2, \theta_m)$. Ukážeme, že tento výraz souvisí s dříve představenenou distribuční funkcí (6.38) fotonů ve druhém elektronu ve Weitzakerově-Wiliamsově aproximaci. Připomeňme nejprve veličinu y definovanou v (2.4) a aplikujme ji na vrchol s druhým elektronem. Dostáváme:

$$y_2 \equiv \frac{q_2 p_1}{p_2 p_1} = \frac{\omega_2 E - q_2 E \cos\theta_2}{E^2} = \frac{\omega_2 (1 + \sqrt{\frac{Q_2^2}{\omega_2}} \cos\theta_2)}{E} \doteq \frac{\omega_2}{E} \tag{7.22}$$

V předchozím vztahu jsme v souladu s dřívějším značením použili značení $Q_i^2 = -q_i^2$. Virtualitu druhého, zkoumaného fotonu Q_2^2 považujeme v porovnání s ostatními kinematickými veličinami za velmi malou. Z kinematických důvodů pak musí být malý i úhel θ_2. Tím je objasněna aproximace v posledního kroku posledního výrazu.

Člen stojící ve výrazu $N(E_2, \theta_m)$ před logaritmem potom můžeme zapsat jako $1+(1-y_2)^2$. Pro úpravu argumentu logaritmu nejprve připomeňme veličiny Q^2_{min} a Q^2_{max} vymezující oblast hodnot virtuality fotonu z druhého vrcholu v režimu Weitzaker-Wiliamsově aproximace:

$$Q^2_{min} = \frac{m_e^2 y^2}{1-y} = \frac{m_e^2 (E - E_2)^2}{E E_2} \tag{7.23}$$

$$Q^2_{max} = 2EE_2(1 - \cos\theta_2) \tag{7.24}$$

Pro jejich podíl máme:

$$\frac{Q^2_{max}}{Q^2_{min}} = \frac{E^2 E_2^2 \theta_2^2}{m_e^2 (E - E_2)^2}, \tag{7.25}$$

kde jsme použili $1 - \cos\theta \approx \frac{1}{2}\theta^2$ (platí pro malá θ). Vidíme tedy, že výraz v logaritmu v (7.19) představuje podíl Q_{max}/Q_{min}. Upravíme-li uvedený podíl na kvadrát a ponecháme člen úměrný $1/m_e^2$ (viz WWA), můžeme výraz (7.19) přepsat na tvar:

$$N(Q^2, y) = \frac{\alpha}{2\pi} \left[1 + (1 - y_2)^2 \ln \frac{Q^2}{m_e^2} \right]. \tag{7.26}$$

Podobně jako v (7.22) zavedeme veličinu y_1:

$$y_1 \equiv 1 - \frac{E_1}{E}, \tag{7.27}$$

která nyní představuje frakci energie prvního elektronu nesenou vysoce virtuálním fotonem.

Použitím veličin y_1 a y_2 spolu s formulí (7.26) ve výrazu (7.18) můžeme diferenciální účinný průřez (7.18) psát jako:

$$\frac{d\sigma}{dxdy_1dy_2} = \frac{\alpha}{2\pi}\left(1 + (1-y_2)^2 \ln\frac{Q^2}{m_e^2}\right)\frac{2p_1q_2}{q_1^2}[1 + (1-y_1)^2](\sigma_{TT} + \sigma_{ST}). \tag{7.28}$$

V kapitole 6.1 jsme se zabývali popisem srážky virtuálního fotonu s protonem. Vyústěním byla formule (6.15) pro účinný průřez takové reakce. Pro její vyjádření byla užita strukturní funkce protonu F_2. Podobně můžeme vyjádřit učinný průřez srážky virtuálního fotonu s fotonem reálným. Namísto strukturní funkce F_2 v něm však bude představena strukturní funkce fotonu F_2^γ:

$$\sigma(\gamma\gamma^*) \equiv \sigma_{TT} + \sigma_{ST} = 4\pi^2\alpha\frac{F_2^\gamma(x, Q_1^2)}{Q_1^2} \tag{7.29}$$

Bude tedy nejprve zapotřebí přepsat účinné průřezy (7.20) a (7.21) do proměnných x a Q^2:

$$\sigma_{TT} = \pi\alpha^2\left[\left(\frac{4x}{Q_1^2} - \frac{8x^2(1-x)}{Q_1^2}\right)\ln\frac{1+\beta}{1-\beta} - \left(\frac{2x}{Q_1^2} - \frac{8x^2(1-x)}{Q_1^2}\right)\sqrt{1 - \frac{m^2x}{Q_1^2(1-x)}}\right] \tag{7.30}$$

$$\sigma_{ST} = \pi\alpha^2\left(\frac{4x^2(1-x)}{Q_1^2}\right)\sqrt{1 - \frac{m^2x}{Q_1^2(1-x)}}, \tag{7.31}$$

kde jsme stejně jako v (6.44) označili:

$$\beta = \sqrt{1 - \frac{m^2x}{Q^2(1-x)}}. \tag{7.32}$$

Při úpravách výrazů (7.30) a (7.31) jsem využili následujících výrazů:

$$x = \frac{Q_1^2}{Q_1^2 + W^2} = \frac{-q_1^2}{-q_1^2 + q_1^2 + 2q_1q_2} \quad \Rightarrow \quad q_1q_2 = -\frac{q_1^2}{2x} \tag{7.33}$$

$$W^2 = (q_1 + q_2)^2 = q_1^2 - \frac{q_1^2}{x} = \frac{-q_1^2(1-x)}{x} \tag{7.34}$$

To nám dovoluje zapsat výraz (7.28) následujícím způsobem:

$$\frac{d\sigma}{dxdy_1dy_2} = \underbrace{\frac{\alpha}{2\pi}\left(1 + (1-y_2)^2\ln\frac{Q^2}{m_e^2}\right)}_{WWA}\underbrace{\frac{2\pi\alpha^2(2p_1q_2)}{q_1^4}[1 + (1-y_1)^2]F_2^\gamma(x, Q_1^2)}_{\sigma(e\gamma\to ee)} \tag{7.35}$$

kde

$$F_2^\gamma(x, Q_1^2) = \frac{\alpha}{\pi} x \left[[x^2 + (1-x)^2] \ln \frac{Q^2}{m_e^2} + 8x(1-x) - 1 \right] \tag{7.36}$$

Dospíváme tak k formuli, ve které lze vyčlenit část popisující extrakci (skoro) reálného fotonu z elektronu (Weitzacker-Williamsova aproximace - viz. kapitola 6.2) a rozptyl $e\gamma \to ee$ podobný rozptylu $e\gamma \to eX$ popsaný v kapitole 6.3 (srovnejte (7.36) a (6.46)).

Kapitola 8

Evoluční rovnic pro partony ve fotonu

8.1 Odvození evoluční rovnic pro partony ve fotonu

V této kapitole představíme postup při odvození evolučních rovnic pro partonové distribuční funkce pro foton. Tyto rovnice budou velmi podobné evolučním rovnicím pro distribuční funkce partonů v hadronech. Lišit se budou přítomností dodatečného, nehomogenního členu.

Odvození začněme představením tzv. "bodové" nesingletní distribuční funkce $q_{NS}^{PL}(x, M_0, M)$, která je výsledkem resumace řady diagramů na obr. 8.1:

$$q_{NS}^{PL}(x, M_0, M) \equiv \tag{8.1}$$

$$\equiv \frac{\alpha}{2\pi} k_{NS}^{(0)}(x) \int_{M_0^2}^{M^2} \frac{d\tau}{\tau} + \int_x^1 \frac{dy}{y} P_{qq}^{(0)}\left(\frac{x}{y}\right) \int_{M_0^2}^{M^2} \frac{d\tau_1}{\tau_1} \frac{\alpha_s(\tau_1)}{2\pi} \frac{\alpha}{2\pi} k_{NS}^{(0)}(y) \int_{M_0^2}^{\tau_1} \frac{d\tau_2}{\tau_2} + \tag{8.2}$$

$$\int_x^1 \frac{dy}{y} P_{qq}^{(0)}\left(\frac{x}{y}\right) \int_y^1 \frac{dw}{w} P_{qq}^{(0)}\left(\frac{y}{w}\right) \int_{M_0^2}^{M^2} \frac{d\tau_1}{\tau_1} \frac{\alpha_s(\tau_1)}{2\pi} \times \tag{8.3}$$

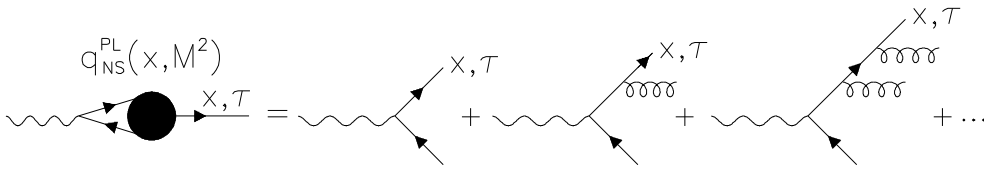
$$\times \int_{M_0^2}^{\tau_1} \frac{d\tau_2}{\tau_2} \frac{\alpha_s(\tau_2)}{2\pi} \frac{\alpha}{2\pi} k_{NS}^{(0)}(w) \int_{M_0^2}^{\tau_2} \frac{d\tau_3}{\tau_3} + \cdots, \tag{8.4}$$

kde $k_{NS}(x) = \delta_{NS} k_q^{(0)}(x)$ a

$$\delta_{NS} = 6 n_f \left(\left\langle e^4 \right\rangle - \left\langle e^2 \right\rangle^2 \right) \quad k_q^{(0)}(x) = (x^2 + (1-x)^2). \tag{8.5}$$

Ve výše představené řadě vycházíme z čistě QED vrchlolu $\gamma \to q\bar{q}$. K němu jsme připočetly další procesy, popisované v rámci QCD. Virtuální kvark vzniklý větvením $\gamma \to q\bar{q}$ totiž může v důsledku silné interakce vyzářit další kvarky a gluony. Ve zmíněném případě započítáváme příspěvky diagramů, kdy se z kvarku vyzáří jeden gluon, dva gluony, atd. Uvedená definice je nápadně podobná konstrukci oblečené nesingletní kvarkové distribuční funkci (3.2). Podobně derivací podle $\ln M^2$ dospíváme k rovnici pro tzv. nesingletní distribuční funkci pro fotony:

$$\frac{q_{NS}(x, M^2)}{d \ln M^2} = \frac{\alpha}{2\pi} \delta_{NS} k_q^{(0)}(x) + \frac{\alpha_s}{2\pi} \int_x^1 \frac{dy}{y} P_{qq}\left(\frac{x}{y}\right) q_{NS}, \tag{8.6}$$



Obrázek 8.1: Diagram popisující bodovou část nesingletní distribuční funkce

která se od hadronové nesingletní distribuční funkce liší přítomností členu $(\alpha/2\pi)\delta_{NS}k_q^{(0)}$. Doposud jsme v konstrukci fotonových distribučních funkcí uvažovali pouze příspěvky multigluonové emise. V případě obecné kvarkové, antikvarkové a gluonové distribuční funkce je nutné započíst i příspěvky dalších procesů - větvení gluon na kvark v případě kvarkové a gluon na gluon a kvark na gluon v případě gluonové distribuční funkce. Evoluční rovnice pro zmíněné distribuční funkce mají tvar:

$$\frac{dq_i^\gamma(x,M^2)}{d\ln M^2} = \frac{\alpha}{2\pi}e_i^2 P_{q\gamma} + \frac{\alpha_s}{2\pi}\int_x^1 \frac{dy}{y}\left[P_{qq}\left(\frac{x}{y}\right)q_i^\gamma(y,M^2) + P_{qg}\left(\frac{x}{y}\right)G^\gamma(y,M^2)\right], \quad (8.7)$$

$$\frac{d\bar{q}_i^\gamma(x,M^2)}{d\ln M^2} = \frac{\alpha}{2\pi}e_i^2 P_{q\gamma} + \frac{\alpha_s}{2\pi}\int_x^1 \frac{dy}{y}\left[P_{qq}\left(\frac{x}{y}\right)\bar{q}_i^\gamma(y,M^2) + P_{qg}\left(\frac{x}{y}\right)G^\gamma(y,M^2)\right], \quad (8.8)$$

$$\frac{dG^\gamma(x,M^2)}{d\ln M^2} = \frac{\alpha_s}{2\pi}\int_x^1 \frac{dy}{y}\left[P_{gq}\left(\frac{x}{y}\right)\Sigma(y,M^2) + P_{gg}\left(\frac{x}{y}\right)G^\gamma(y,M^2)\right] \quad (8.9)$$

S použitím kvarkové a antikvarkové distribuční funkce se zavádí nesingletní distribuční funkci $q_{NS}(x,M^2)$, řešící výše představenou rovnici (8.6):

$$q_{NS}(x,M) \equiv \sum_{i=1}^{n_f}(e_i^2 - < e^2 >)\left(q_i(x,M) + \bar{q}_i(x,M)\right). \quad (8.10)$$

kde e^2 je střední hodnota e_i^2.

8.2 Řešení evolučních rovnic pro fotony v nesingletním případě

V této kapitole popíšeme užití inverzní Mellinovy transformace k řešení evolučních rovnic pro distribuční funkce partonů ve fotonu.

Obecná řešení soustavy rovnic (8.7)-(8.9) lze psát jako součet partikulárního řešení nehomogenní rovnice a obecného řešení homogenní rovnice, nazývaného jako "hadronová" část. Zmíněná homogenní rovnice je totiž identická s rovnicí pro nesingletní distribuční funkci partonů v hadronu. Podmnožinu partikulárních řešení budeme nazývat "bodovým" a značit "PL" (z angl. pointlike) řešením.

Obecně pak můžeme psát $(D = q, \bar{q}, G)$

$$D(x, M) = D^{PL}(x, M) + D^{HAD}(x, M). \tag{8.11}$$

V dalším textu se budeme soustředit výhradně na evoluční distribuční funkce (8.6) pro nesingletní distribuční funkci $q_{NS}(x, M^2)$.

Po Mellinově transformaci můžeme zmíněnou rovnici psát jako:

$$\frac{dq_{NS}(n, M^2)}{d\ln M^2} = \frac{\alpha}{2\pi}\delta_{NS}k_q^{(0)}(n) + \frac{\alpha_s(M^2)}{2\pi}P_{qq}^{(0)}(n)q_{NS}(n, M^2) \tag{8.12}$$

kde

$$k_q^{(0)}(n) = \left(\frac{2}{n+2} - \frac{2}{n+1} + \frac{1}{n}\right) \tag{8.13}$$

Bodové řešení jsme konstruktivně představili již dříve výrazem (8.4). V Mellinových momentech ho můžeme zapsat jako:

$$q_{NS}^{PL}(n, M_0, M) = \frac{4\pi}{\alpha_s(M)}\left[1 - \left(\frac{\alpha_s(M)}{\alpha_s(M_0)}\right)^{1-2P_{qq}^{(0)}(n)/\beta_0}\right]a_{NS}(n), \tag{8.14}$$

kde

$$a_{NS}(n) \equiv \frac{\alpha}{2\pi\beta_0}\frac{k_{NS}^{(0)}(n)}{1 - 2P_{qq}^{(0)}(n)/\beta_0}. \tag{8.15}$$

Nyní již můžeme zapsat obecné řešení evoluční rovnice (8.12):

$$\begin{aligned} q_{NS}(n, M^2) = \quad & \left(\frac{a_s(Q^2)}{a_s(Q_0^2)}\right)^{-\frac{P_{qq}^{(0)}}{4\beta_0}} \\ & + \frac{2\alpha k_{NS}^{(0)}(n)}{\alpha_s(M)\left(\beta_0 - 2P_{qq}^{(0)}(n)\right)}\left[1 - \left(\frac{\alpha_s(M)}{\alpha_s(M_0)}\right)^{1-2P_{qq}^{(0)}(n)/\beta_0}\right] \end{aligned} \tag{8.16}$$

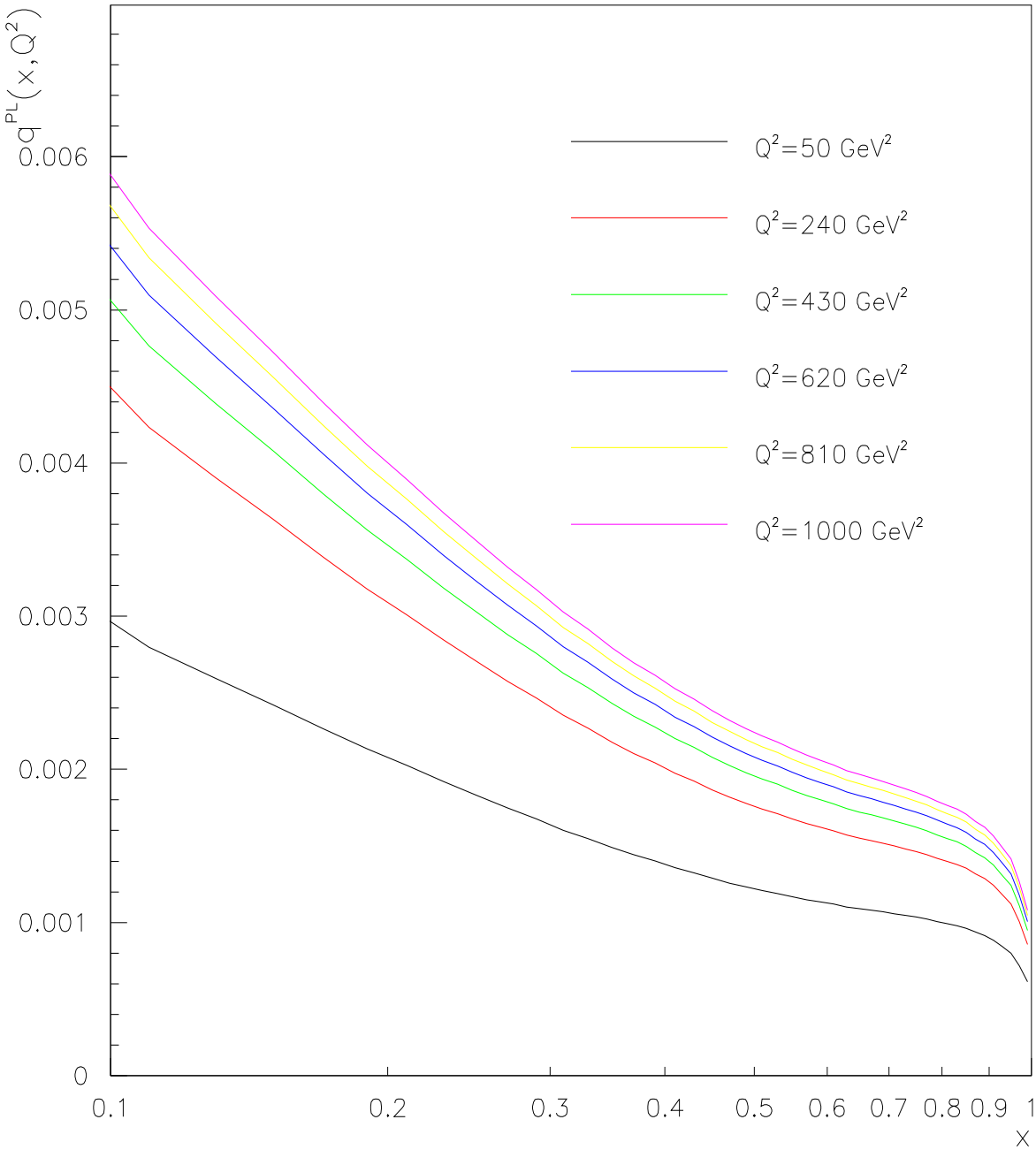
Známé je tedy řešení evoluční rovnice pro nesingletní distribuční funkci q_{NS} v termínech Mellinových momentů. Řešení $q_{NS}(x, M)$ získáme použitím inverzní Mellinovy transformace. Tento krok byl podrobně popsán v kapitole 5.2.

8.3 Analýza experimentálních dat o struktuře fotonu

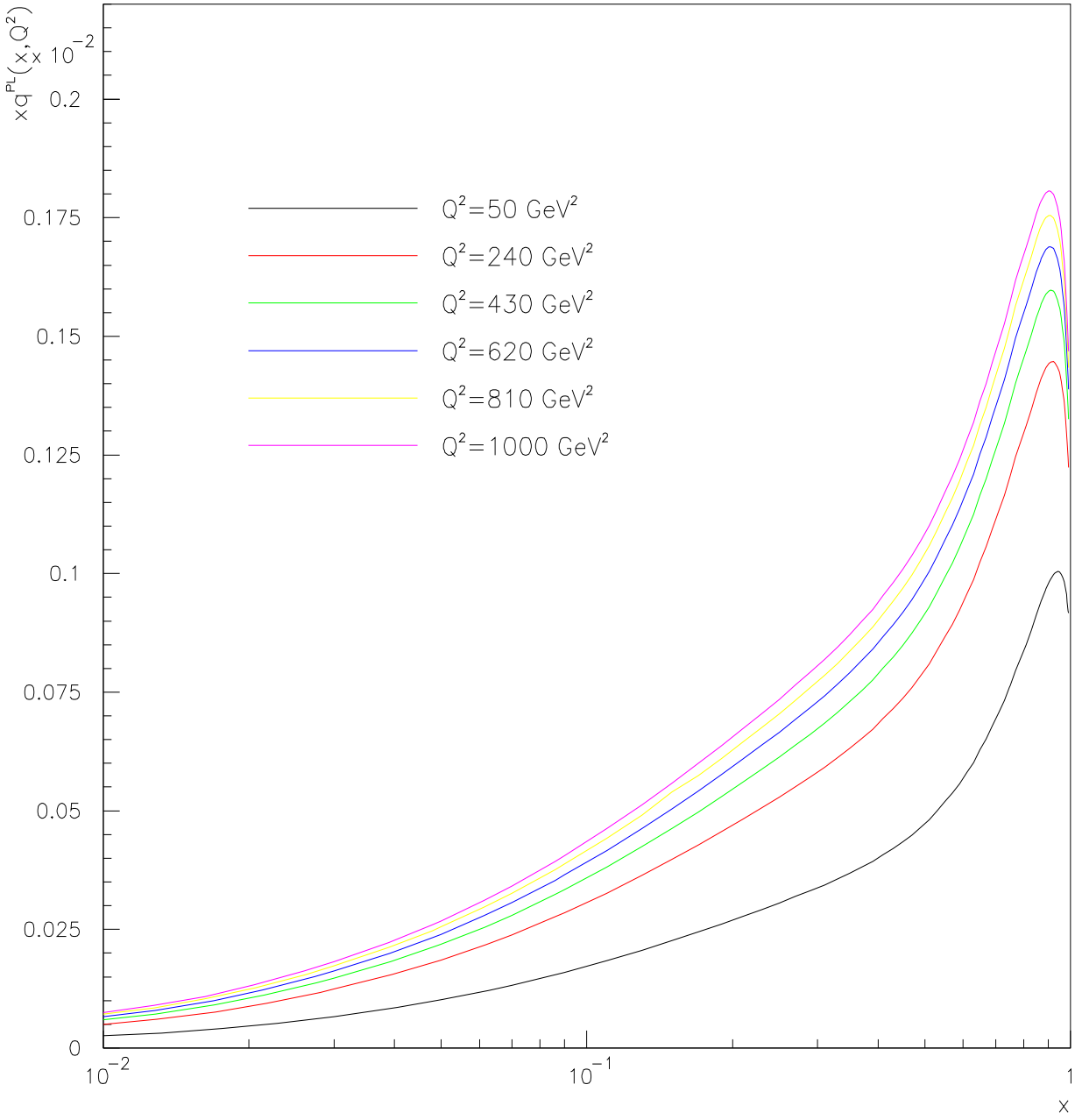
V zásadě budeme postupovat stejně jako při analýze dat o struktuře hadronu provedené v kapitole 5.4. Především budou představeny výsledky fitů řešení fotonové nesingletní evoluční rovnice na experimentální data. Ještě předtím graficky předvedeme tvar bodových řešení

$q^{PL}(x, Q^2)$, která v součtu s hadronovým řešením představují obecné řešení uvažované rovnice. Na obr. 8.2 a 8.3 je zachycen průběh funkcí $q^{PL}(x, Q^2)$ a $xq^{PL}(x, Q^2)$ v závislosti na x pro různá Q^2. Pro všechna řešení byla zvolena stejná počáteční podmínka $Q_0^2 = 5$ Gev^2.

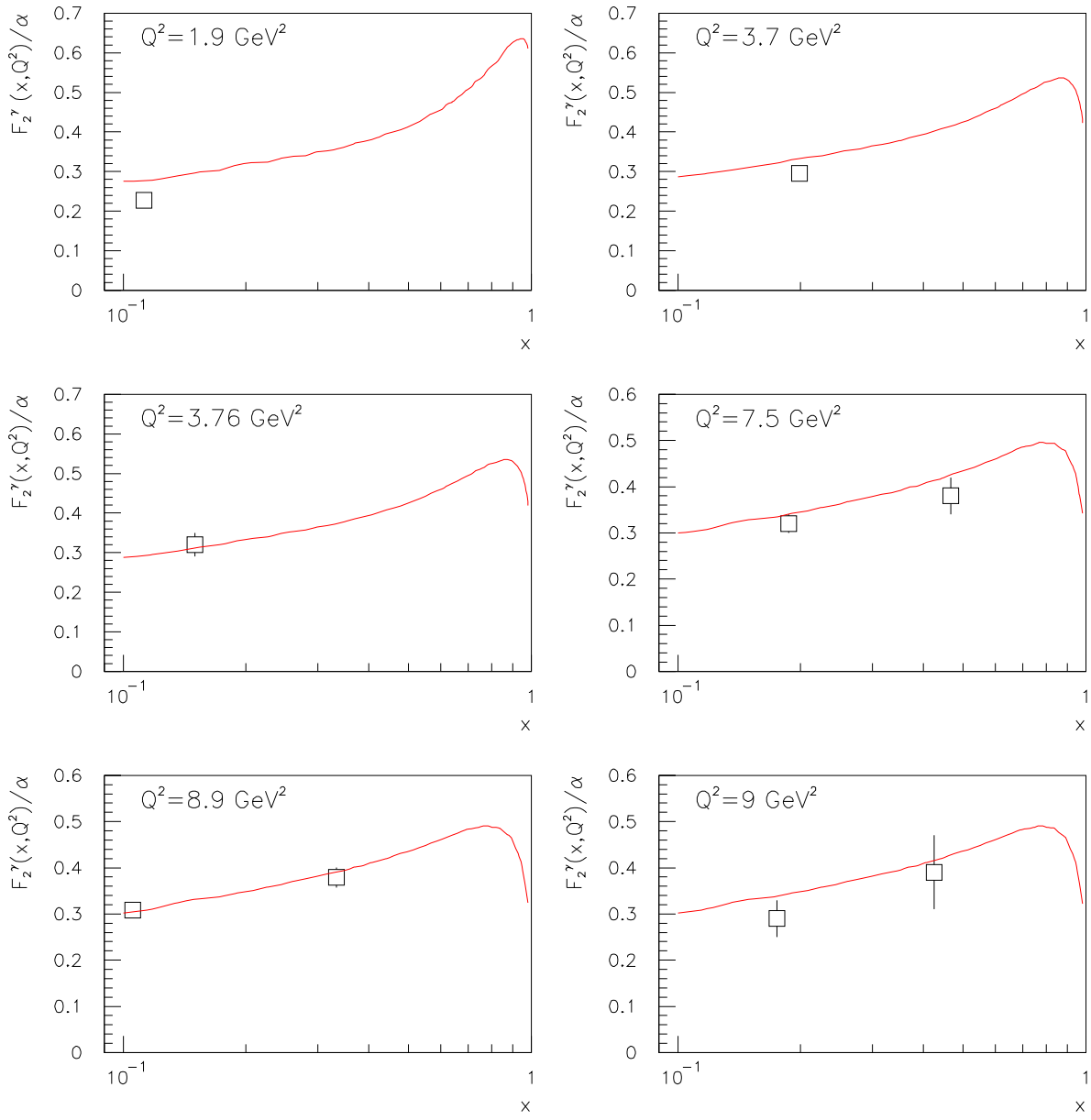
Pro fitování byla použita experimentální data publikovaná v [6]. Obecně není experimentálních dat zachycujících strukturu fotonu příliš mnoho. Za tím účelem byla použita data hned z několika experimentů, jmenovitě OPAL, L3 a ALEPH. Fitovalo se souhrně na všechna experimentální data. Výsledky jsou pro přehlednost opět vykresleny pro jednotlivá Q^2. K nalezení jsou na sérii obr. 8.4 - 8.7. Při minimalizaci funkce χ^2 byla volena simplexní metoda (vývoj parametrů je zaznámen v dodatku E), kombinovaná metodou konjugovaných gradientů. Při inverzi Mellinových momentů fotonových distribučních funkcí je totiž potřeba postupovat poněkud jemněji. Osvědčilo se mít všechny výpočty neustále pod kontrolou a ve všech situacích pečlivě zkoumat chování integrandů a adekvátně volit polohu integrační křivky i horní mez integrálu. V zásadě pak platilo, že všechny integrály při inverzní Mellinově transformaci bylo potřeba na rozdíl od inverze hadronových řešení počítat s výrazně vyšší přesností. A protože tato skutečnost přináší mnohem vyšší časové nároky, byla při minimalizaci upřednostněna rychlejší simplexní metoda. Teprve v blízkosti minima byla doplněna časově náročnější, ale jemnější metodou konjugovaných gradientů. Takovým postupem bylo dosaženo $\chi^2 = 138/49$.



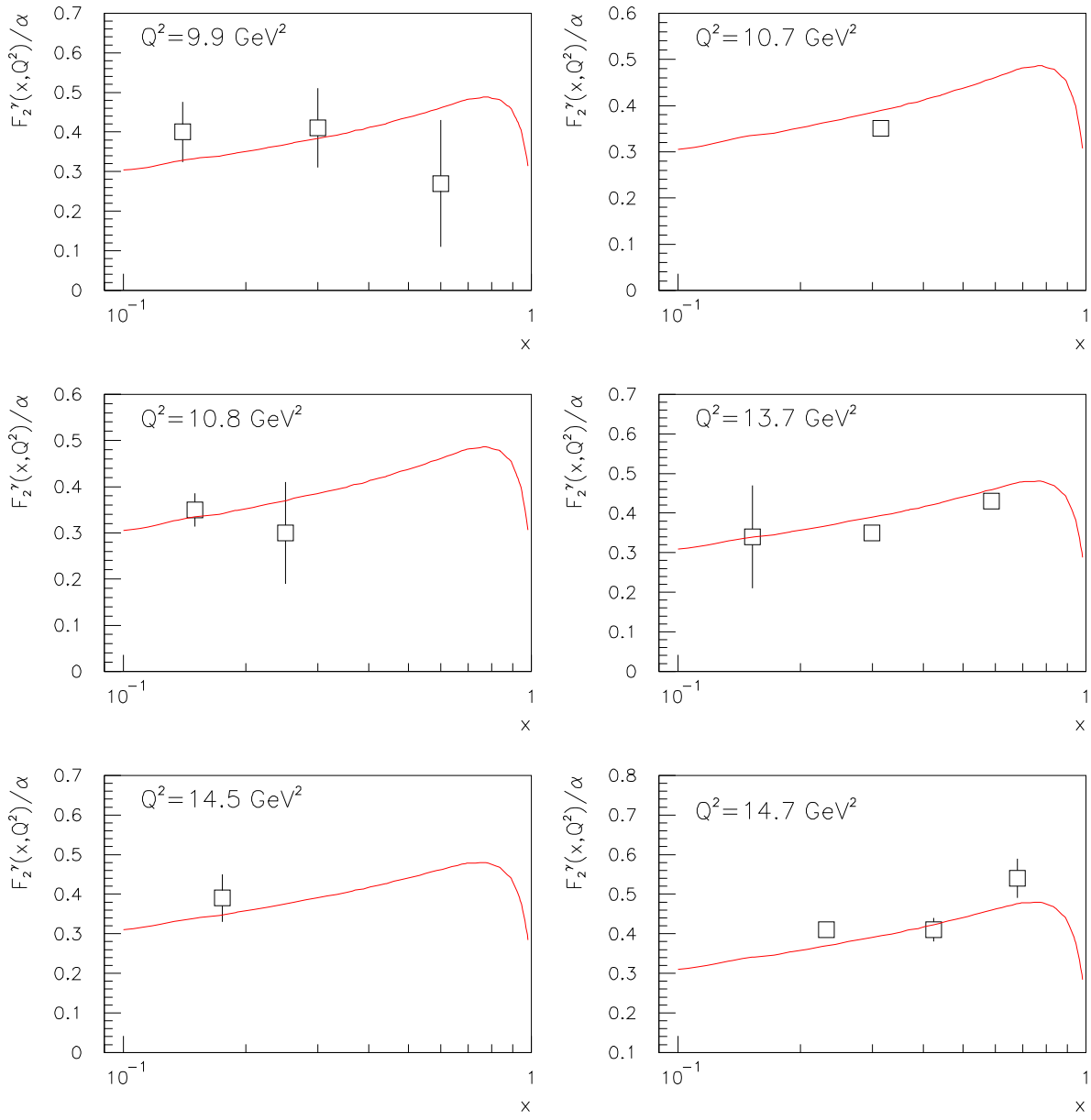
Obrázek 8.2: Vypočtené závislosti bodové časti nesingletní distribuční funkce $q^{PL}(x, Q^2)$ na x pro různé hodnoty Q^2.



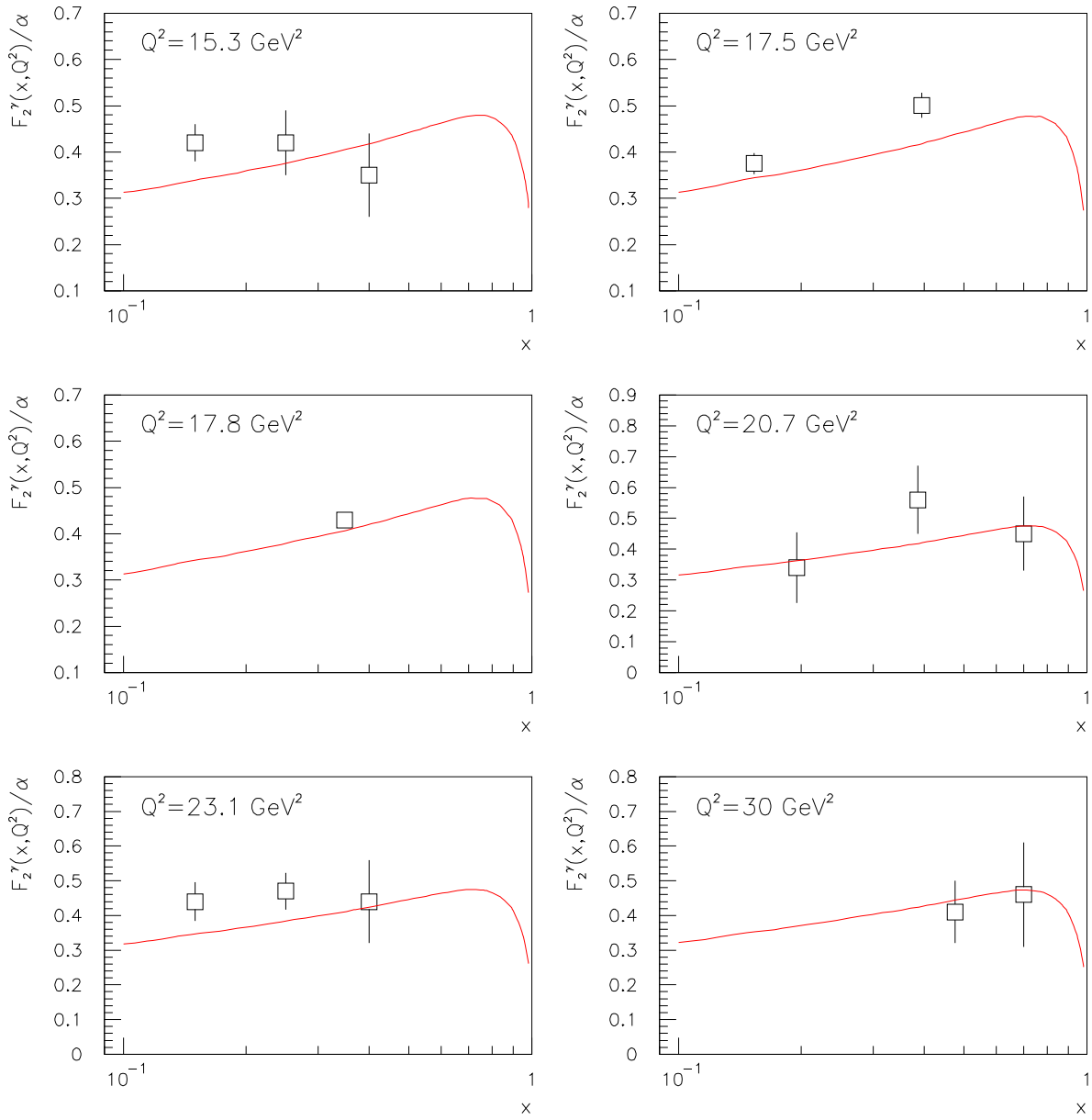
Obrázek 8.3: Vypočtené závislosti bodové časti nesingletní distribuční funkce $xq^{PL}(x, Q^2)$ na x pro různé hodnoty Q^2.



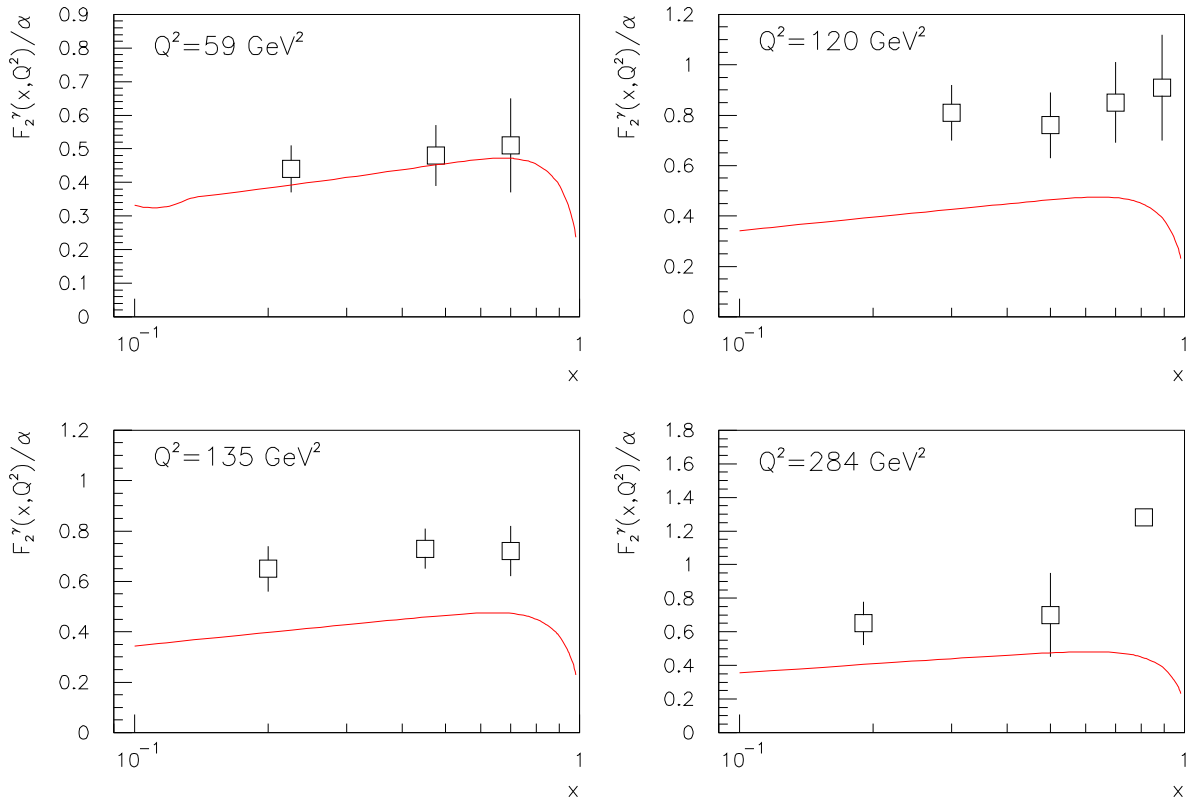
Obrázek 8.4: Srovnání experimentálních dat pro F_2^γ s hodnotami vypočtenými řešením nesingletní evoluční rovnice pro fotony.



Obrázek 8.5: Srovnání experimentálních dat pro F_2^γ s hodnotami vypočtenými řešením nesingletní evoluční rovnice pro fotony.



Obrázek 8.6: Srovnání experimentálních dat pro F_2^γ s hodnotami vypočtenými řešením nesingletní evoluční rovnice pro fotony.



Obrázek 8.7: Srovnání experimentálních dat pro F_2^γ s hodnotami vypočtenými řešením nesingletní evoluční rovnice pro fotony.

Kapitola 9

Popis programu užitého při výpočtu evolučních rovnic

V následujícím textu bude popsána činnost programu, který umožňuje numericky řešit evoluční rovnice QCD - přesněji evoluční rovnici pro hadronovou a fotonovou nesingletní funkci. Program byl napsán v jazyce C++, zdrojový kód je připojen v dodatku F. Jeho činnost spočívá ve dvou základních dovednostech. První z nich je výpočet distribuční funkce pro libovolné x a Q^2, druhou je pak fitování obecného řešení evoluční rovnice na naměřené hodnoty distribuční funkce, dodané experimentem. Obě tyto činnosti nyní popíšeme trochu podrobněji.

- Princip a činnost části programu, umožňující výpočet distribuční funkce je velmi jednoduchý. Program si žádá sadu parametrů $A, \alpha, \beta, \gamma, \eta, Q_0$, polohu integrální křivky c, parametr Λ a pochopitelně hodnotu x a Q^2 (v případě fotonových distribučních funkcí invertujeme zvlášť hadronové a bodové řešení, pro každé z nich je tedy možné volít obecně různá c a Q_0^2). Vnitřními parametry jsou potom horní mez integrálu zpětné transformace a velikost absolutní a relativní chyby výpočtu. Tyto hodnoty se během základních početních operací většinou nemění. Po dodání všech těchto parametrů se spustí algoritmus, jehož základní kroky byly popsány v kapitole 5.1 a 8.2. Výsledkem je hodnota distribuční funkce v žádaných bodech bodech x a Q^2. Při analýze výsledku je obvyklé zpracovat grafy s naznačenými průběhy jedné proměnné při fixování druhé. Tento program umožňuje dodat dostatečný počet příslušných funkčních hodnot pro tvorbu těchto grafů v obou variantách. (tj. průběh distribuční funkce v proměnné x při pevném Q^2 a naopak). Všechny výsledky jsou ukládány do souborů a uchovány tak pro jejich případně zpracování dalšími aplikacemi.

  Součástí vypracovaného programu je i několik testů ověřujících správnost numerického výpočtu inverzní Mellinovy transformace. Jejich popis fungování spolu s výsledky budou popsány v kapitole 10.

- Druhá významná součást programu má na starosti fitování obecného řešení evoluční rovnice na konkrétní experimentální data. To se děje systematickým vyhledáváním a postupným zpřesňováním parametrů A, α, β, γ, η a Λ. Snahou je nalézt takové

parametry, aby řešení rovnice s hraniční podmínkou určenou prvními pěti parametry byla těmi řešeními, která v příslušných x a Q^2 odpovídají experimentálním hodnotám. Do vyhledávacích manévrů stále zahrnujeme i volný parametr Λ. Princip vyhledávání těchto parametrů je postaven na minimalizaci funkce

$$\chi^2(A,\alpha,\beta,\gamma,\eta,\Lambda) = \sum_{k=1}^{K} \left(\frac{e_k - f(A,\alpha,\beta,\gamma,\eta,\Lambda)}{\sigma_k} \right)^2 , \tag{9.1}$$

kde e_k a σ_k jsou naměřené hodnoty a jejich chyby, $f(A,\alpha,\beta,\Lambda)$ jsou napočítané hodnoty distribuční funkce s příslušnými parametry. Hodnoty x a Q^2 použité při výpočtu distribuční funkce si v každém sčítanci odpovídají s hodnotami, za jakých byly naměřeny hodnoty e_k. Počet měření K musí být mnohem větší, než je počet parametrů n vstupujících do výpočtu hodnot $xf(x,Q^2)$.

Je tedy potřeba zvolit počáteční sadu parametrů A, α, β, Q_0 a Λ a pro ni napočítat χ^2. Program potom funkci χ^2 iterativně minimalizuje. V každém kroku dochází ke zpřesnění všech parametrů. Program umožňuje vývoj funkce χ^2 i příslušné hodnoty parametrů v každém kroku sledovat.

Kapitola 10

Ověření správnosti inverzní Mellinovy transformace

Po sepsání výše zmíněného programu bylo vhodné ověřit, že skutečně počítá to co počítat má. Existuje několik postupů, kterými lze správnost výpočtů kontrolovat nebo přímo ověřovat.

Pokud jde o přímé testování správnosti inverzního integrování, provedeme srovnání nějaké zvolené funkce s výsledkem inverzní Mellinovy transformace momentů této funkce. Očekávat budeme stejné hodnoty. Za testovací funkci byla zvolena parametrizovaná hraniční podmínka (5.43) s parametry zvolenými tak, aby práce s touto funkcí simulovala výpočty prováděné při řešení "hadronové" evoluční rovnice. Výhodnost takové volby ve vztahu k našemu řešenému problému se ozřejmí, pokud si uvědomíme, že volbou $Q^2 = Q_0^2$ ve výrazu (5.58) dostáváme:

$$f(x, Q_0^2) = \frac{1}{\pi} \int_0^\infty dz \mathrm{Re}\left[x^{-c-iz} AB(n + \alpha - 1, 1 + \beta)\right]. \tag{10.1}$$

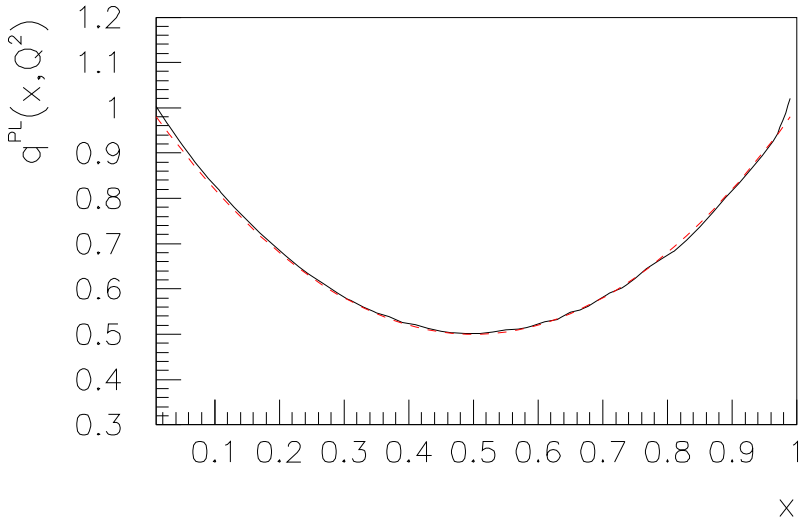
Inverzní transformování momentů hraniční podmínky evoluční rovnice s vhodnými parametry je tedy speciálním případem obecné inverzní transformace (5.58). Toto konstatování lze dokonce rozšířit na případ inverze momentů fotonových distribučních funkcí (8.16), neboť při $Q^2 = Q_0^2$ bodové řešení vymizí a zpětně tedy invertujeme pouze momenty hadronového řešení. Pro náš test byla zvolena funkce:

$$g(x) = 8.6x^{8.5}(1 - x)^{3.94} \tag{10.2}$$

Při výpočtu srovnávací funkce tedy bylo potřeba zvolit následující parametry: $A = 8.64$, $\alpha = 8.5$, $\beta = 3.94$, $\gamma = 0$. Dále bylo voleno $c = 2.5$ a $Q^2 = Q_0^2$. Výsledky jsou zaznamenány v tab. 10.1. V prvním sloupci jsou ve vybraných bodech hodnoty funkce $g(x)$, ve druhém jsou pak v odpovídajících bodech invertované momenty funkce $g(x)$. Odhadované chyby těchto výsledků jsou k nalezení ve třetím sloupci. V posledním sloupci jsou pak skutečné chyby, kterých jsme se při inverzní transformaci dopustili tj. |Výsledek 1-Výsledek 2|.

Uvedeme ještě jeden způsob možnosti ověření, tentokrát jen dílčího výpočtu při inverzní Mellinově transformaci. Týká se napočítaných momentů (5.57). Ještě než dojde k jejich zpětné transformaci, je vhodné ověřit, že mají odpovídající vlastnosti.

Z definice větvící funkce $P_{qq}^{(0)}(x)$ snadno zjistíme, že:



Obrázek 10.1: Tvar bodové části nesingletní distribuční funkce fotonů pro $Q^2 \approx Q_0^2$ (QED limita). Podrobný popis je v textu.

$$P_{qq}^{(0)}(1) = \int_0^1 P_{qq}^{(0)}(x)dx = 0. \tag{10.3}$$

První moment větvící funkce $P_{qq}^{(0)}(x)$ je tedy nulový, tudíž první moment distribuční funkce by měl být konstantní.

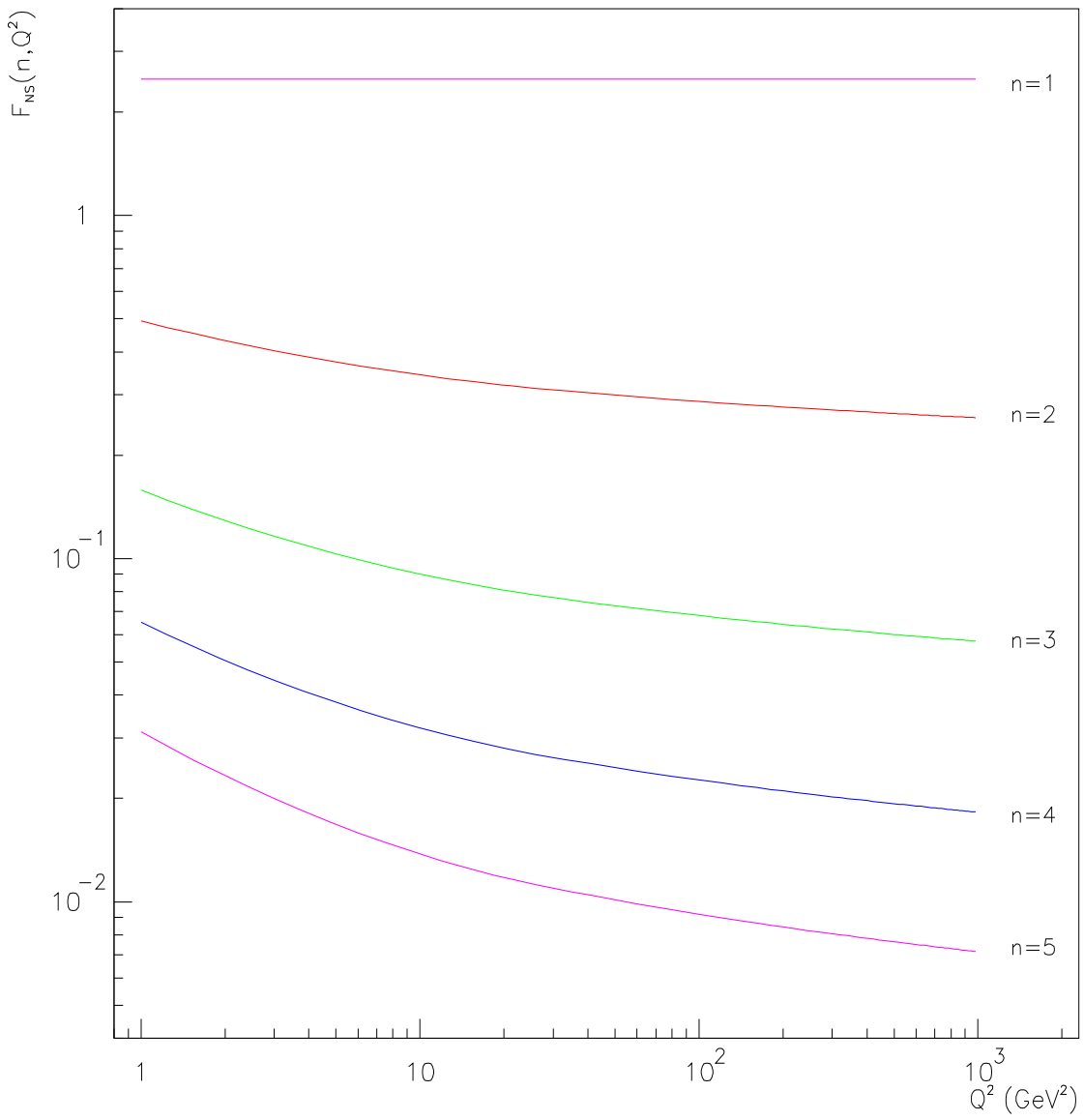
Pro přirozené momenty této větvící funkce větší než jedna snadno nalezneme:

$$P_{qq}^{(0)}(n) = \frac{4}{3}\left(1 + \frac{1}{2} - \sum_{i=1}^{n-1}\frac{2}{i} - \frac{1}{n} - \frac{1}{n+1}\right). \tag{10.4}$$

Tento výraz je vždy záporný, tedy exponent výrazu $a_s(Q^2)/a_s(Q_0^2)$ v rovnici (5.57) je kladný. Vzhledem k tomu, že je funkce $a_s(Q^2)/a_s(Q_0^2)$ v proměnné Q^2 klesající (viz (5.9)), měly by být (pro všechna přirozená $n > 1$) v proměnné Q^2 klesající i momenty distribučních funkcí.

Přesně tyto vlastnosti mají mnou zjištěné výsledky. Graficky jsou shrnuty v obr. 10.2. Zde je vyneseno prvních pět přirozených momentů distribuční funkce. Při výpočtu byly zvoleny následující parametry: $A = 8.6$, $\alpha = 0.85$, $\beta = 3.94$, $Q_0^2 = 5$ GeV^2, $\Lambda = 0.337$ MeV.

Soustřeďme se nyní na "bodové" řešení (8.14) evoluční rovnice pro fotonovou nesingletní distribuční funkci. Zabývejme se situací, kdy bude horní mez virtualit M^2 blízká zvolené počáteční škále M_0^2. Fyzikálně to znamená, že dojde k potlačení efektů kvantové chromodynamiky, v našem případě k potlačení příspěvků procesů, kdy se z virtuálního kvarku vyzáří jeden gluon, dva gluony, atd. Taková interpretace je zřejmá z definice z bodové distribuční funkce (8.4). Její konstrukce se prováděla tak, aby zachycovala právě takové procesy. Podobně bychom mohli provést limitu $\alpha_s \to 0$ (nebo ekvivaletně $\Lambda \to 0$). V obojím případě dospíváme k dominujícímu prvnímu členu:



Obrázek 10.2: Prvních pět přirozených momentů hadronové nesingletní distribuční funkce. Podrobný popis je v textu.

$$q_{NS}^{PL}(x, M, M_0) \to \frac{\alpha}{2\pi} k_{NS}^{(0)}(x) \ln \frac{M^2}{M_0^2}, \tag{10.5}$$

který odpovídá ryzímu efektu QED popisující větvení $\gamma \to q\bar{q}$. Ověřit takové chování v rámci numerického výpočtu je vhodným testem správného fungování vypracovaného programu. Pro ověření byly zvoleny parametry $Q_0 = 5$ GeV a $Q^2 = 5.5$ GeV. Pro grafickou reprezentaci výsledku se napočtené invertované momenty ještě vydělily členem $(\alpha\delta_{NS}/2\pi)\ln(M^2/M_0^2)$. Podle teoretické předpovědi by měl tento výsledek odpovídat větvící funkci $x^2 + (1-x)^2$. Grafické srovnání je k nalezení na obr. 10.1. Přerušovaná křivka představuje graf funkce $x^2 + (1-x)^2$, plná potom numerický výsledek. S potěšením lze konstatovat, že výsledky numerického výpočtu jsou ve velmi dobré shodě s očekáváním.

Výsledek 1	Výsledek 2	Odhadovaná chyba	Skutečna chyba
33.9605	34.1589	0.0000	0.1985
18.4426	18.4489	0.0338	0.0063
13.4200	13.4200	0.0002	0.0000
10.6406	10.6366	0.0000	0.0040
8.7917	8.7893	0.0000	0.0024
7.4395	7.4380	0.0000	0.0014
6.3923	6.3935	0.0177	0.0012
5.5498	5.5491	0.0055	0.0007
4.8538	4.8543	0.0024	0.0006
4.2672	4.2667	0.0020	0.0006
3.7656	3.7662	0.0019	0.0005
3.3318	3.3316	0.0020	0.0002
2.9531	2.9529	0.0020	0.0002
2.6203	2.6206	0.0021	0.0003
2.3262	2.3262	0.0022	0.0000
2.0651	2.0648	0.0022	0.0003
1.8325	1.8325	0.0023	0.0000
1.6247	1.6249	0.0024	0.0002
1.4388	1.4388	0.0025	0.0000
1.2721	1.2719	0.0026	0.0002
1.1226	1.1225	0.0028	0.0001
0.9885	0.9885	0.0029	0.0001
0.8681	0.8682	0.0000	0.0002
0.7601	0.7601	0.0000	0.0001
0.6633	0.6632	0.0000	0.0001
0.5766	0.5765	0.0000	0.0001
0.4991	0.4991	0.0000	0.0001
0.4301	0.4301	0.0000	0.0001
0.3686	0.3687	0.0000	0.0001
0.3140	0.3141	0.0000	0.0001
0.2658	0.2658	0.0000	0.0000
0.2234	0.2233	0.0000	0.0001
0.1861	0.1860	0.0000	0.0001
0.1537	0.1536	0.0000	0.0001
0.1255	0.1255	0.0000	0.0000
0.1013	0.1014	0.0000	0.0001
0.0806	0.0807	0.0000	0.0001
0.0631	0.0632	0.0000	0.0001

Tabulka 10.1: Ověření správnosti inverzní Mellinovy transformace. Podrobný popis je v textu.

Dodatek A

Vlastnosti Mellinovy transformace

Mějme funkci $f(x)$ definovanou na intervalu $(0,1)$, pro kterou existuje takové reálné číslo p, že $x^p f(x) \in L(0,1)$. Mellinovu transformaci funkce $f(x)$ definujeme vztahem:

$$F(n) = \int_0^1 x^{n-1} f(x) \mathrm{d}x, \tag{A.1}$$

kde n je komplexní číslo, jehož reálná část je větší nebo rovna $p+1$. Mellinovy obrazy $F(n)$ funkce $f(x)$ se nazývají Mellinovy momenty funkce $f(x)$.

Pokud v definičním vztahu Mellinovy transformace provedeme substituci $x = e^{-t}$, potom pro momenty $F(n)$ máme:

$$F(n) = \int_0^\infty x^{-nt} f(e^{-t}) \mathrm{d}t. \tag{A.2}$$

To tedy znamená, že Mellinova transformace funkce $f(x)$ je rovna Laplacově transformaci funkce $f(e^{-t})$. Vlastnosti Mellinovy transformace tak můžeme získat z obdobných vlastností Laplaceovy transformace.

Inverze Laplaceovy transformace (A.2) je dána formulí

$$f(e^{-t}) = \frac{1}{2\pi i} \int_{\xi - i\infty}^{\xi + \infty} F(n) e^{nt} \mathrm{d}n, \tag{A.3}$$

kde $\xi \geq p+1$. Substitucí $t = -\ln x$ tak získáme vztah pro inverzní Mellinovu transformaci ve tvaru:

$$f(e^{-t}) = \frac{1}{2\pi i} \int_{\xi - i\infty}^{\xi + \infty} F(n) x^{-n} \mathrm{d}n, \quad \xi \geq p+1. \tag{A.4}$$

Je -li funkce $f(x)$ reálná, potom z formule (A.1) plyne $F(n^*) = F^*(n)$. Předchozí vztah lze pak zjednodušit na

$$f(e^{-t}) = \frac{1}{\pi} \int_0^\infty Re(F(\xi + i\eta) x^{-\xi - i\eta}) d\eta, \quad \xi \geq p+1. \tag{A.5}$$

Toto vyjádření je zejména vhodné pro numerický výpočet inverzní Mellinovy transformace.

Mezi důležité vlastnosti Mellinovy transformace patří:

- Mellinova transformace a její inverze jsou lineární.
- Momenty $F(n)$ jsou holomorfní funkce v polorovině $\mathrm{Re}\, n > p + 1$
- $F(n) \to 0$ pro $\mathrm{Re}\, n \to +\infty$, přičemž stejnoměrně vzhledem k $\mathrm{Im}\, n$.
- $F(n) \to 0$ pro $\mathrm{Im}\, n \to \pm\infty$ při každém pevném $\mathrm{Re}\, n \geq p + 1$.

Dodatek B

Krokové metody minimalizace v mnoha proměnných

Metody minimalizace ve více dimenzích jsou většinou zobecněnými variantami jednodimenzionálních případů. Nezřídka však tato zobecnění sebou přinášejí problémy, které nás nutí volit jiné cesty.

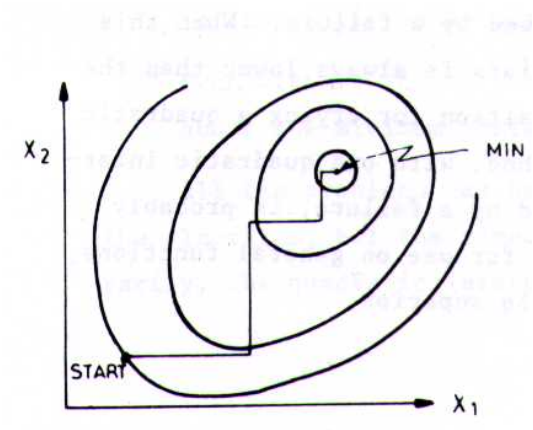
B.1 Síťové hledání a náhodné hledání

Příkladem enormního zvýšení složitosti při přechodu do prostoru vyšších dimenzí je použití metody uzlů pro funkce více proměnných. Pro lokalizaci minima s přesností 1% v případě jedné proměnné je potřeba 100 výpočtů funkce. V případě funkce 10 proměnných je už vyžadováno 10^{20} napočtených bodů. U minimalizací funkcí platí obecně, že metoda, která se osvědčila v jednodimenzionálním případě už nemusí být rozšířením do více rozměrů přijatelná. Dokonce ani pokud tyto metody vylepšíme o zpřesnění vyhledávacích oblastí v jednotlivých proměnných. Tehdy se jejich efektivita zvyšuje jen velmi pomalu a proto je vhodnější obrátit pozornost na jiné, vhodnější techniky.

B.2 Jednoparametrická variace

Jelikož námi hledaný extrém považujeme za stacionární bod v n proměnných x_i, bude nulovat všech n prvních derivací $\partial F/\partial x_i$. Bylo by možné pokusit se při jeho hledání nulovat každou derivaci zvlášť.

Takový postup představuje poměrně starou metodu jednoparametrické variace, kdy se pokoušíme hledat minimum vzhledem k jedné proměnné a to za pomoci některé jednodimenzionální techniky. Po skončení minimalizace vzhledem k jedné proměnné pochopitelně nenacházíme minimum vzhledem k jiným proměnným a proto je potřeba celou proceduru stále opakovat. Takový algoritmus nicméně obvykle konverguje. Ilustrovat to můžeme na příkladu funkce dvou proměnných. Její schéma je znázorněné na obr. B.1. Uzavřené křivky

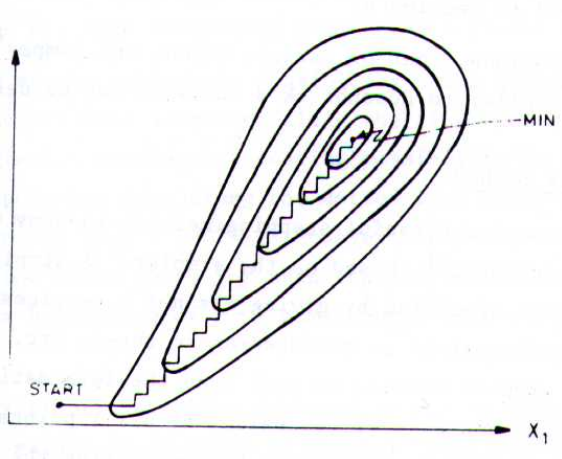


Obrázek B.1: Schématické znázornění postupu při minimalizaci metodou jednoparametrické variace v případě funkce dvou proměnných.

značí oblasti se stejnou funkční hodnotou, rovné úseky značí postup při minimalizaci v jednotlivých krocích. V prvním kroku hledáme minimum vzhledem k proměnné x_1, ve druhém vzhledem k proměnné x_2, ve třetím opět vzhledem k x_1 atd. V tomto případě metoda konverguje po pouhých čtyřech jednoparametrických minimalizacích.

Uvažujme nyní funkci reprezentovanou schématem na obr. B.2. V tomto případě naše metoda postupuje velmi pomalu. Je to způsobeno "rovnými" úseky spojující body se stejnými funkčními hodnotami. Takové chování při procesu minimalizace může být považováno za neúnosně pomalé.

V následují části popíšeme několik úspěšných vylepšení zaměřených na eliminaci právě takového chování.



Obrázek B.2: Schematické znárornění méně přiznivé varianty při minimalizaci metodou jednoparametrické variace.

B.3 Rosenbrockova metoda

Rosenbrockův algoritmus začíná v podstatě úplně stejně jako jednoparametrická minimalizace. Poté co proběhne úplný cyklus se všemi proměnnými, zavede se nový výchozí souřadný systém a to tak, že jedna z os bude určena vektorem z počátečního bodu minimalizace do koncového bodu jednoho cyklu. Směr tohoto vektoru představuje pro pozdější výpočet vhodnou volbu. V případě výše uvedeného "úzkého údolí" probíhají jednotlivé kroky minimalizace víceméně podél stěn a úspěšně se tak vyhýbá "cik-cak" chování. V dalším cyklu je jednoparametrická minimalizace provedena se sadou proměnných nového souřadného systému.

Rosenberckova metoda je vcelku účinná a bývá stabilní. Se vzrůstajícím počtem proměnných, jeho výkonnost klesá. Navíc ve srovnání s metodami popsanými později konverguje pomaleji.

B.4 Metoda simplexu

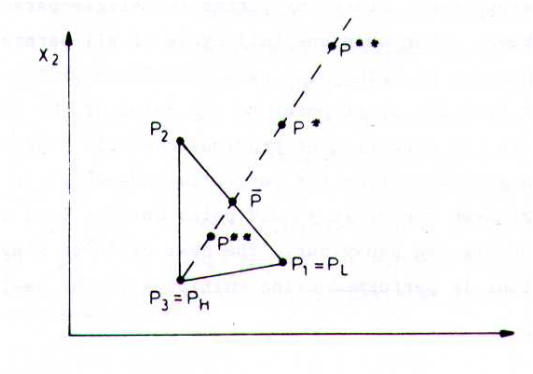
Jedna z nejúspěšnějších metod užívaná v případě mnoha proměnných pochází z rukou Neldera a Meada a je založena na útvaru simplexu. Simplex je n-dimensionální útvar určený svými $n+1$ vrcholy. Je to kupříkladu trojúhelník ve dvou dimenzích, čtyřstěn ve třech dimenzích apod. Simplex dal název této metodě proto, že s funkcí v každém kroku pracujeme v jejích $n+1$ hodnotách. Pro lepší představu se blíže podívejme na užití této metody ve dvou dimenzích. Zvolme tři simplexní body (třeba i náhodně) a vypočítejme hodnotu funkce v těchto bodech. Označme P_H bod s největší funkční hodnotou (pro náš případ minimalizace bod nejméně přijatelný) a P_L bod naopak s nejnižší hodnotou. Budiž $\overline{P}$ těžištěm všech bodů simplexu kromě bodu P_H, tedy:

$$\overline{P} = \frac{1}{n}\left(\sum_{i=1}^{n+1} P_i - P_H\right) \tag{B.1}$$

Z původního simplexu vytvoříme nový záměnou bodu P_H nějakým vhodnějším bodem. Pokusíme se tento bod nalézt zobrazením bodu P_H ve středové symetrii podle bodu $\overline{P}$, tedy $P^* = \overline{P} + (\overline{P} - P_H)$. Pokud $F(P^*) < F(P_L)$, volíme bod $P^{**} = \overline{P} + 2(\overline{P} - P_H)$. Pokud $F(P^*) > F(P_H)$, volíme bod $P^{**} = \overline{P} - 1/2(\overline{P} - P_H)$. Nejvhodnější bod potom v simplexu pro další krok nahrazuje bod P_H. Pokud takový nenalezneme, vytvoříme kolem bodu P_L nový simplex.

Tuto metodu je možné při hledání nového bodu na přímce $P_H\overline{P}$ upravovat volbou číselného faktoru vystupujícím ve výrazu pro P^{**}. Důležité je, aby tento nový bod nebyl příliš blízko $\overline{P}$. V tom případě by náš simplex kolapsoval do přímky (nebo obecně do nadplochy v n rozměrném prostoru), což by znemožnilo další iterace.

Simplexní algoritmus je vytvořen tak, aby postupoval v co největších krocích. Je tedy poněkud méně citlivý v případě mělkých minim. Výhodou je však nutnost jen několika výpočtů funkce, obvykle jednoho nebo dvou v každé iteraci. V každém kroku se postupuje v nejvhodnějším směru - od nejvyšší hodnoty do průměru nejnižších hodnot.



Obrázek B.3: Znázornění postupu při minimalizaci metodou simplexu. Podrobný popis je v textu.

Vhodným kritériem konvergence pro simplexní metodu je rozdíl $F(P_H) - F(P_L)$. Iterace jsou ukončeny pokud je tento rozdíl menší než nějaká předvolená hodnota. Koncovým krokem výpočtu je potom výpočet funkce v $\overline{P}$.

B.5 Gradientní metody

Výpočet derivací

Za gradientní metody budeme považovat takové, které užívají informace z velmi malé oblasti proměnných k určení relativně vzdálených bodů. Nemusí to nutně znamenat, že sledují směr gradientu, ale gradient nebo vyšší derivace jsou při výpočtu nějakým způsobem užity. Mnoho výkonných algoritmů níže uvedených vyžadují znalost derivace funkce, minimalizační program tedy musí být schopen na základě koncových rozdílů odhadnout derivaci minimalizované funkce.

První derivace může být odhadnuta z:

$$\frac{\partial F}{\partial x}|_{x_0} \approx \frac{F(x_o + d) - F(x_o)}{d} \tag{B.2}$$

kde d je malé. Chyba dána nejnižším rozvojem Taylorova rozvoje pak bude dána:

$$\delta \approx \frac{d}{2}\frac{\partial^2 F}{\partial x^2}|_{x_0}. \tag{B.3}$$

Pochopitelně je výhodné uvažovat d co nejmenší, ale zároveň tak velké, aby chyba při výpočtu F nebyla větší než právě představené δ. Protože druhé derivace nejsou známy, není možné nalézt optimální d. Je tedy nutné zapojit intuici a šikovně ho odhadnout.

Mnohem bezpečnější bude zvolit body symetricky na každé straně x_0:

$$\frac{\partial F}{\partial x}|_{x_0} \approx \frac{F(x_o + d) - F(x_o - d)}{2d}, \tag{B.4}$$

v tomto případě je chyba δ ve druhém řádu nulová a člen nejnižšího řádu rozvoje je úměrný třetí derivaci. Nevýhodou této metody je, že vyžaduje $2n$ volání funkce pro odhad prvních n derivací narozdíl od asymetrické metody, kdy jich je ke stejnému účelu zapotřebí jen $n+1$ (nebo jen n, pokud již známe $F(x_0)$). Výhodou je naopak fakt, že druhou derivaci dostáváme jako mezivýsledek výpočtu první derivace:

$$\frac{\partial F}{\partial x}|_{x_0} \approx \frac{F(x_o+d)+F(x_o-d)-2F(x_0)}{d^2}, \tag{B.5}$$

Nejstrmější spád

Pokud známe první derivace minimalizované funkce, je přirozené vydat se při hledání minima ve směru záporně vzatého gradientu, neboť to je směr ve kterém funkce klesá nejrychleji. Tento postup použil již před 130 lety Cauchy.

Tato metoda sestává ze série jednodimenzionálních minimalizací každá ve směru nejstrmějšího poklesu funkce v bodě, kde se hledání počíná. Směr gradientu pochopitelně nezůstává konstantní, je tedy nutné počítat s mnoha iteracemi.

Newtonova metoda

Uvažujme obecnou kvadratickou funkci. Tu můžeme minimalizovat jen pokud budeme mít k dispozici hodnoty funkce, její první i druhou derivaci. Kvadratickou funkci napíšeme jako:

$$F(\underline{x}) = F(\underline{x}_0) + g^T \cdot (\underline{x}-\underline{x_0}) + \frac{1}{2}(\underline{x}-\underline{x}_0)^T G(\underline{x}-\underline{x}_0), \tag{B.6}$$

kde gradient g je vypočten v bodě $\underline{x}_0$ a matice G druhé derivace je konstanta. Potom je minimum dáno:

$$\underline{x}_m = \underline{x}_0 - G^{-1}\underline{g} = \underline{x}_0 - V\underline{g}, \tag{B.7}$$

kde jsme inverzní matici druhé derivace označili V. To je v podstatě vícedimensionální ekvivalent kvadratické interpolace. K výhodám patří fakt, že jednotlivé iterační kroky nemohou být libovolně dlouhé, jsou totiž přesně určeny touto metodou. Výhodou je i skutečnost, že směr při hledání není vždy pevně daný gradientem, ale bývá korelován smíšenými druhými derivacemi.

K záporům patří skutečnost, že metoda diverguje, pokud je matice G (nebo V) pozitivně definitní. Přesto je to velmi účinný nástroj a mnoho užitečných algoritmů je na Newtonově metodě založena.

Konjugované směry

Vektory $\underline{d}_i$ a $\underline{d}_j$ nazveme konjugované vzhledem k pozitivně definitní symetrické matici A, pokud:

$$\underline{d}_i^T A \underline{d}_j = 0 \qquad pro\ i \neq j. \tag{B.8}$$

Pokud by A byla jednotková matice, byly by tyto konjugované vektory ortogonální. Konjugaci tedy můžeme považovat za zobecnění ortogonality. Množina n konjugovaných vektorů generuje n-dimensionální prostor, tedy libovolný vektor toho toho prostoru můžeme zapsat jako lineární kombinaci n konjugovaných vektorů. Sadu konjugovaných vektorů můžeme zkonstruovat pomocí Gram-Smidtova ortogonalizačního procesu. Zvolme si libovolný vektor $\underline{d}_1$. Potom je snadné ověřit, že vektor

$$\underline{d}_2 = A\underline{d}_1 - \frac{\underline{d}_1^T AA\underline{d}_1}{\underline{d}_1^T A\underline{d}_1}\underline{d}_1 \tag{B.9}$$

je konjugovaný k vektoru $\underline{d}_1$. Tento proces můžeme zopakovat a získat tak vektor $\underline{d}_3$, který bude konjugovaný k $\underline{d}_1$ *i* $\underline{d}_2$. Takto bychom mohli pokračovat. Získali bychom n konjugovných vektorů $\{\underline{d}_1, \underline{d}_2, ..., \underline{d}_n\}$.

Takové vektory pro nás budou z hlediska minimalizace zajímavé, pokud budou konjugované vzhledem k matici druhé derivace G. V takovém případě se dá ukázat, že posloupnost linárních minimalizací v každém z n konjugovaných směrů minimalizuje obecnou kvadratickou funkci n proměnných. Tuto skutečnost můžeme snadno předvést. Uvažujme kvadratickou fukci:

$$F(\underline{x}) = F(\underline{0}) + \underline{g}^T\underline{x} + \frac{1}{2}\underline{x}^T G\underline{x} \tag{B.10}$$

a n vektorů konjugovaných vzhledem ke G:

$$\underline{d}_i^T G\underline{d}_j = 0, \qquad i \neq j. \tag{B.11}$$

Vektory $\underline{x}$ a $\underline{g}$ můžeme vyjádřit jako lineární kombinace

$$\underline{x} = \sum_i y_i\underline{d}_i \tag{B.12}$$

$$\underline{g} = \sum_i c_i\underline{d}_i. \tag{B.13}$$

Naší obecnou kvadratickou funkci tedy můžeme nyní psát jako:

$$F(\underline{x}) = F(\underline{0}) + \left(\sum_i c_i\underline{d}_i^T\right)\left(\sum_j y_j\underline{d}_j\right) + \frac{1}{2}\left(\sum_i y_i\underline{d}_i^T\right)G\left(\sum_j y_j\underline{d}_j\right) \tag{B.14}$$

Pokud nyní poslední člen přeskupíme do dvojné sumy, vypadnou kvůli konjugační podmínce členy, pro které $i \neq j$. Poslední výraz tedy můžeme zjednodušit:

$$F(\underline{x}) = F(\underline{0}) + \sum_i\sum_j c_i\underline{d}_i^T\underline{d}_j y_j + \frac{1}{2}\sum_j y_j^2\underline{d}_j^T G\underline{d}_j \tag{B.15}$$

$$= \sum_j \left(b_j y_j + b_j' y_j^2\right) \tag{B.16}$$

kde

$$b_j = \sum_i c_i \underline{d}_i^T \underline{d}_j \tag{B.17}$$

a

$$b_j' = \sum \underline{d}_j^T G \underline{d}_j \tag{B.18}$$

jsou konstanty. Vyjádřením kvadratické funkce v proměnné y místo x jsme ji rozdělili na sumu nezávislých kvadratických funkcí jedné proměnné. Minimalizace vzhledem k y_i pak bude nezávislá na minimalizaci v ostatních kojugovaných směrech.

Lze namítnout, že pro konstukci konjugoavných vektorů je zapotřebí znalost matice druhé derivace G a tehdy lze okamžitě použít Newtonovu metodu. Užitečnost užití konjugovaných vektorů však spočívá v tom, že existují metody umožňující jejich výpočet bez úvodní znalosti celé matice G. Jelikož nám jde o výpočet všech konjugovaných vektorů, půjde zřejmně o postupy, ve kterých budeme znalost matice G suplovat ekvivaletními informacemi. Nicméně, takový postup umožňuje zpracovávat informace pro minimalizaci průběžně a to se u zvláště u rozsáhlých výpočtů osvědčuje - takové algoritmy bývají stabilnější.

Konjugované gradienty

Pokud jsou známé první derivace minimalizované funkce, lze užít velmi elegantní metodu konjugoavných gradientů. Předpokladejme, že je funkce vypočtena ve dvou bodech $\underline{x}_0$ a $\underline{x}_1$. Uvažujme nyní rozdíly:

$$\underline{\Delta x} = \underline{x}_1 - \underline{x}_0 \tag{B.19}$$

$$\underline{\Delta g} = \underline{g}_1 - \underline{g}_0. \tag{B.20}$$

Pokud uvažujeme kvadratickou funkci s maticí druhých derivací G, potom platí:

$$\underline{\Delta g} = G \underline{\Delta x}. \tag{B.21}$$

Libovolný vektor $\underline{d}_1$ ortogonální k $\underline{\Delta g}$ je konjugovaný k $\underline{\Delta x}$, nebot:

$$\underline{d}_1^T \underline{\Delta g} = \underline{d}_1^T G \underline{\Delta x} = 0. \tag{B.22}$$

Máme tedy návod na výpočet konjugovaných vektorů bez znalosti G založené na změně gradientu v nějakém zvoleného směru.

Za počáteční směr volíme $d_0 = -g_0$, tedy směr nejstrmějšího spádu v počátečním bodě x_0. Označme minimum nalezené v tomto směru jako x_1 a gradient v tomto směru jako g_1. Další vyhledávací směr bude dán vektorem od kterého požadujeme, aby byl konjugovaný k $\underline{d}_0$. Musí být lineární kombinací vektorů, které máme v danou chvíli k dispozici, konkrétně:

$$\underline{d}_1 = -\underline{g}_1 + b \underline{d}_0. \tag{B.23}$$

Podmínka konjugace dává:

$$\underline{d}_1^T G \underline{d}_0 = \underline{d}_1^T G(\underline{x}_1 - \underline{x}_0) = 0 \tag{B.24}$$

nebo

$$(-\underline{g}_1^T + b\underline{d}_0^T) G \underline{d}_0 = (-\underline{g}_1^T - b\underline{g}_0^T)((\underline{g}_1 - \underline{g}_0) = 0. \tag{B.25}$$

Protože $\underline{x}_1$ je minimum nelezené ve směru $\underline{d}_0 = -\underline{g}_0$, bude směr $\underline{g}_0$ ortogonální ke gradientu funkce v bodě $\underline{x}_1$, tedy $\underline{g}_1^T \underline{g}_0 = 0$. Potom pro koeficient b dostáváme:

$$b = \frac{\underline{g}_1^T \underline{g}_1}{\underline{g}_0^T \underline{g}_0}. \tag{B.26}$$

Tedy nový směr bude dán konjugovaným vektorem:

$$\underline{d}_1 = -\underline{g}_1 + \frac{\underline{g}_1^T \underline{g}_1}{\underline{g}_0^T \underline{g}_0} \underline{d}_0. \tag{B.27}$$

Tímto způsobem můžeme vygenerovat n navzájem konjugovných vektorů, přičemž i-tý bude mít tvar:

$$\underline{d}_{i+1} = -\underline{g}_{i+1} + \frac{\underline{g}_{i+1}^T \underline{g}_{i+1}}{\underline{g}_i^T \underline{g}_i} \underline{d}_i. \tag{B.28}$$

Dodatek C

Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů

V této části se budeme soustředit na vliv volby Q_0^2 v počáteční podmínce na řešení hadronové nesingletní evoluční rovnice. Připomeňme, že hraniční podmínku této rovnice - funkci $q_{NS}(x, Q_0^2)$ - můžeme parametrizovat například takto:

$$q_{NS}(x, Q_0^2) = Ax^{\alpha}(1-x)^{\beta}(1+\gamma x^{\eta}) \tag{C.1}$$

kde A, α, β, γ, η jsou volné parametry.

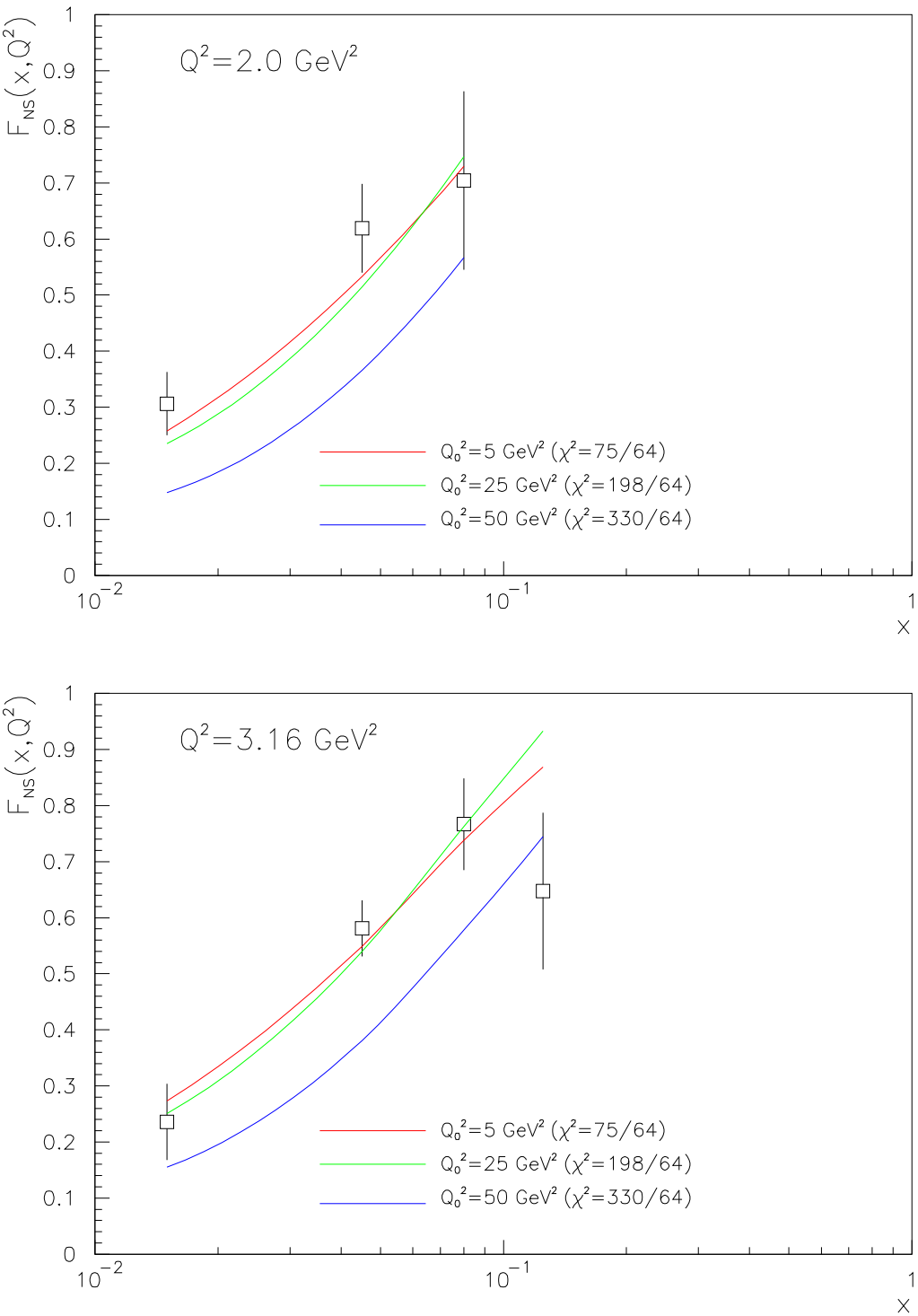
Studován byl výsledek fitů řešení evoluční rovnice s použitím stejných experimentálních dat jako doposud. Tentokrát nás ovšem zajímá jak se na kvalitě fitů odrazí volba Q_0^2.

V prvním kroku bylo k parametrizaci funkce $q_{NS}(x, Q_0^2)$ užito pouze parametrů A, α, β. Postupně se volilo $Q_0^2 = 5,\ 25$ a 50 GeV^2. V dalším se postupně přidávali další parametry.

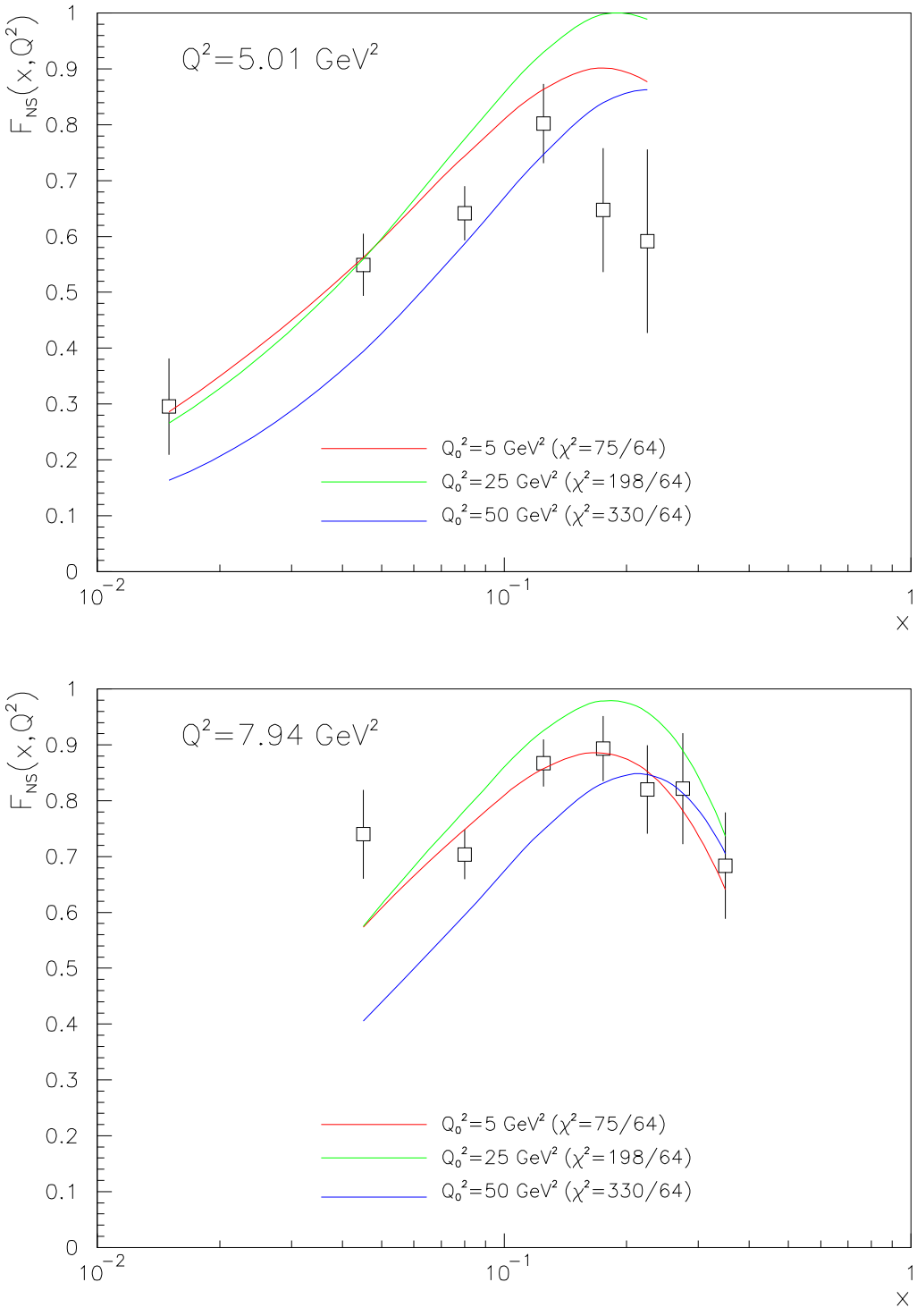
	3 parametry	4 parametry	5 parametrů
$Q_0^2 = 5$ GeV^2	$\chi^2 = 75$	$\chi^2 = 72$	$\chi^2 = 70$
$Q_0^2 = 25$ GeV^2	$\chi^2 = 198$	$\chi^2 = 100$	
$Q_0^2 = 50$ GeV^2	$\chi^2 = 330$	$\chi^2 = 174$	$\chi^2 = 80$

Tabulka C.1: Výsledky fitů pro různá Q_0^2

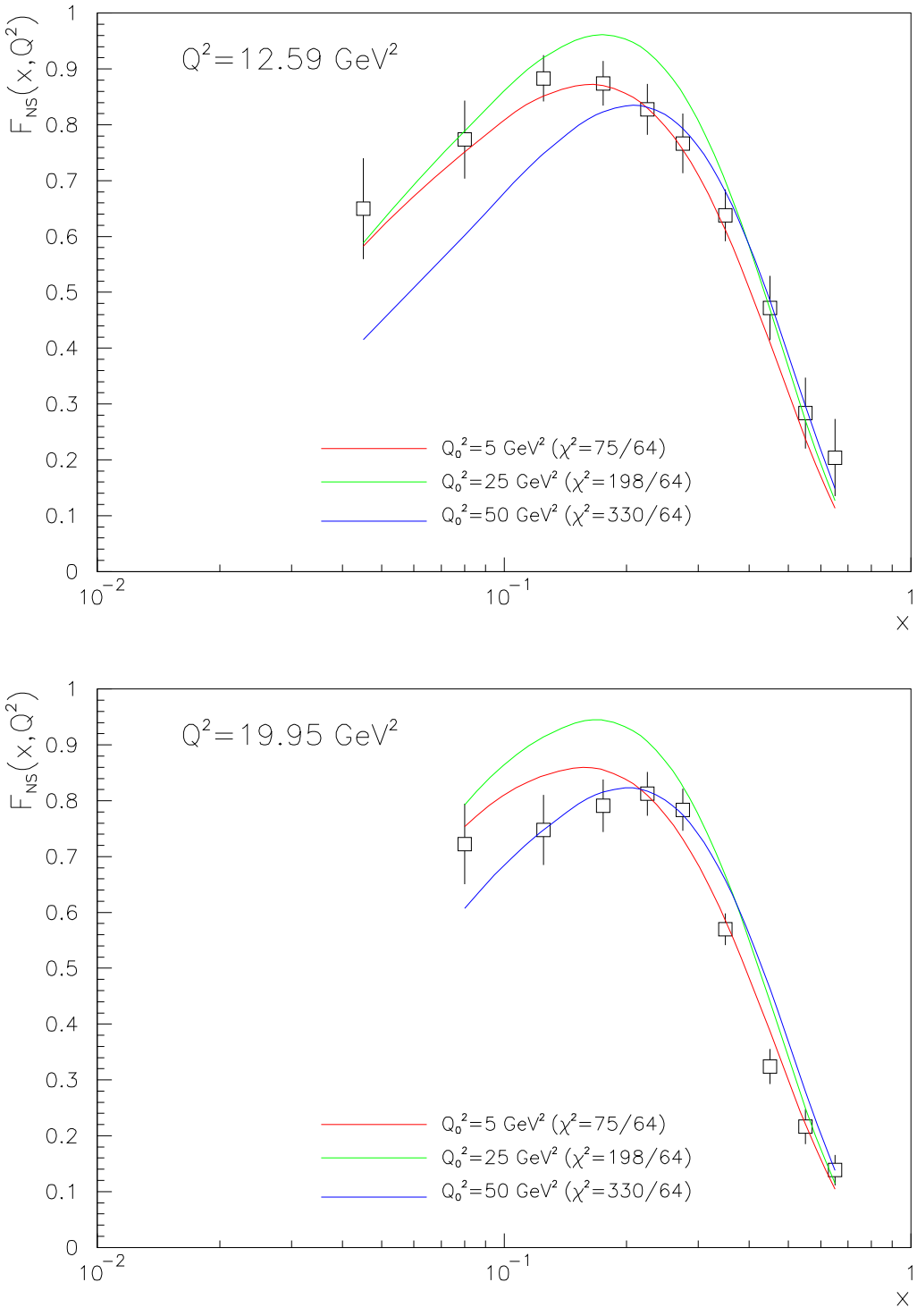
Při nejjednodušší parametrizaci se pro jednotlivá Q_0^2 očekávaly největší rozdíly χ^2. Počáteční podmínka je svými třemi volnými parametry nejvíce svázána a pro různá Q_0^2 tak dává nejméně volnosti pro nalezení optimální počáteční podmínky. S rostoucím počtem parametrů se očekává zkvalitnění fitů při všech třech volbách hodnot Q_0^2. Očekávání se nakonec naplnily. Patrné je to z tab. C. Zároveň bylo zjištěno, že zvolený typ parametrizace je vhodný při volbách malých Q_0^2 v řádech GeV^2. Při všech třech zvolených parametrizacích se se vzrůstajícím Q_0^2 kvalita fitů zhoršuje. Pro grafickou prezentaci získaných výsledků byly zvoleny výsledky fitů při volbě počáteční podmínce se třemi parametry (obr. C.1 - C.5).



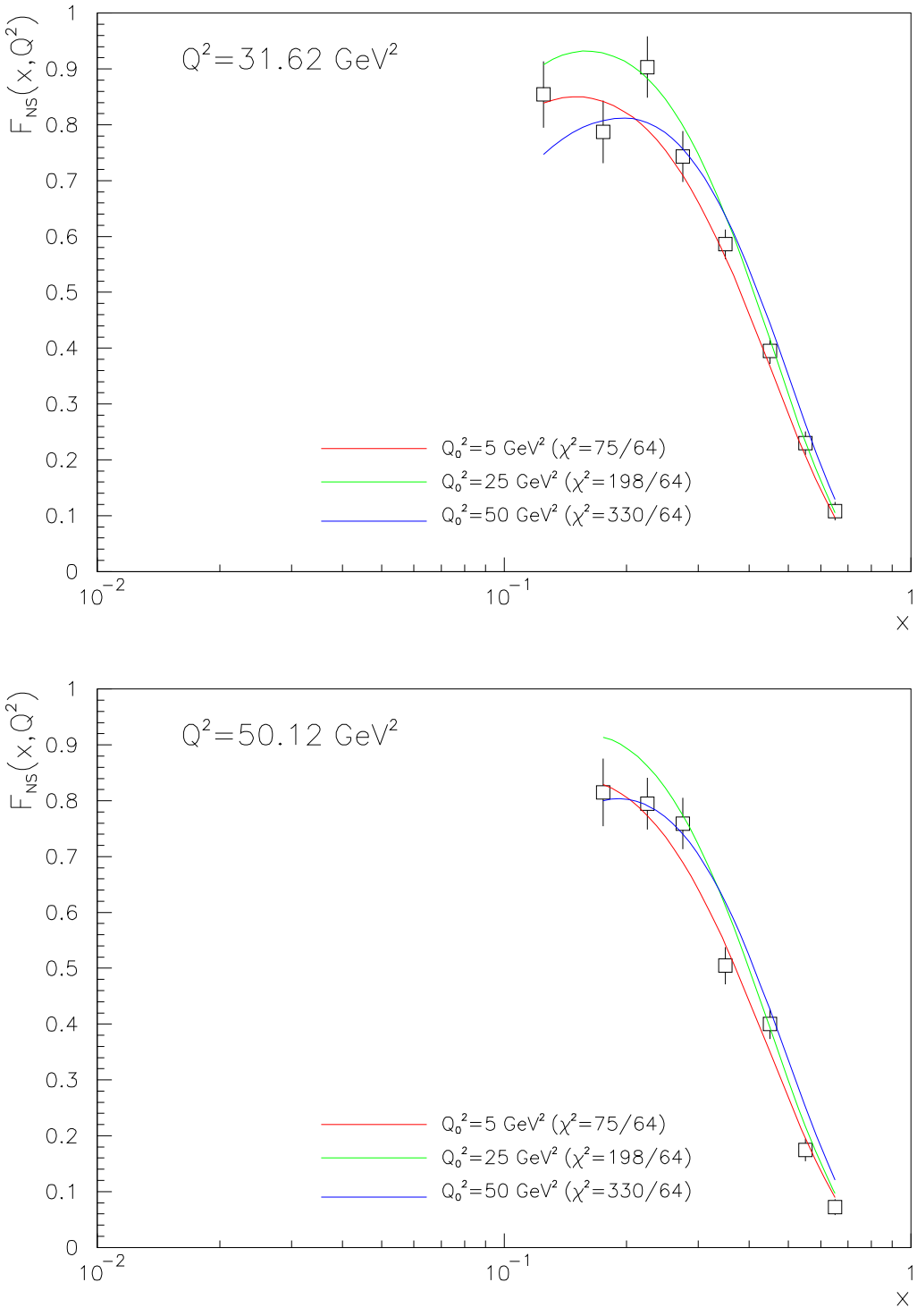
Obrázek C.1: Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů



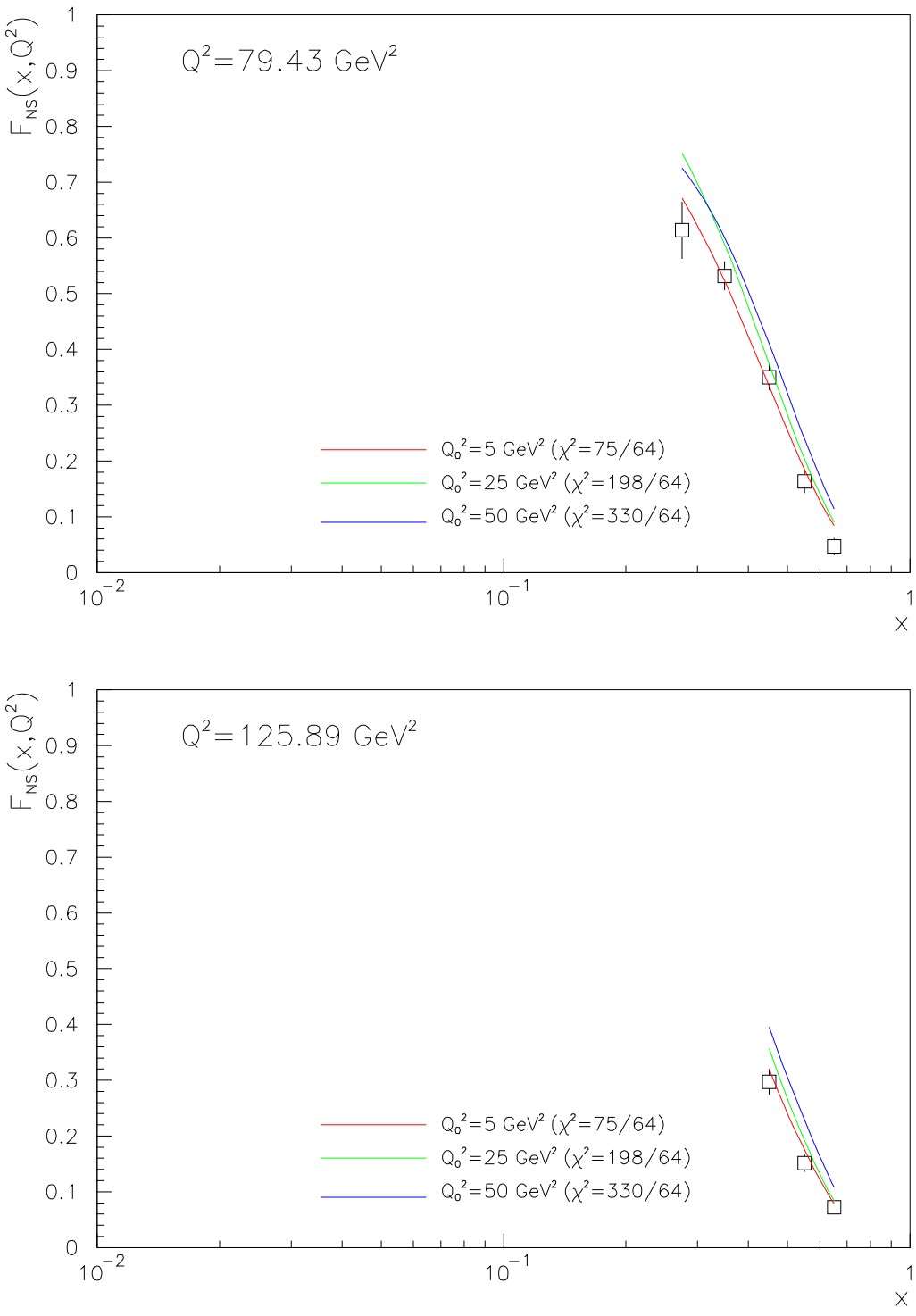
Obrázek C.2: Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů



Obrázek C.3: Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů



Obrázek C.4: Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů



Obrázek C.5: Srovnání výsledků fitů při různých volbách Q_0^2 při analýze dat o struktuře hadronů

Dodatek D

Vývoj parametrů při minimalizaci funkce χ^2 metodou konjugovaných gradientů při analýze dat o struktuře hadronů

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
1	1.994	1.004	2.005	0.502	1.996	1.002	28480.07
2	1.982	1.011	2.015	0.505	1.988	1.007	26124.25
3	1.958	1.025	2.035	0.512	1.971	1.017	21959.70
4	1.910	1.054	2.075	0.526	1.938	1.036	15491.75
5	1.815	1.112	2.156	0.554	1.873	1.075	7935.53
6	1.624	1.228	2.318	0.610	1.743	1.154	3968.34
7	1.619	1.222	2.325	0.613	1.737	1.148	3920.21
8	1.619	1.207	2.335	0.619	1.733	1.146	3831.57
9	1.619	1.177	2.355	0.632	1.723	1.140	3654.14
10	1.619	1.116	2.396	0.657	1.704	1.130	3298.61
11	1.618	0.996	2.477	0.708	1.667	1.109	2585.40
12	1.617	0.754	2.640	0.810	1.592	1.068	1199.40
13	1.615	0.472	2.830	0.930	1.505	1.020	212.25
14	1.573	0.466	2.875	0.955	1.473	1.027	122.19
15	1.576	0.441	2.886	0.962	1.467	1.024	107.70
16	1.562	0.436	2.904	0.971	1.455	1.025	101.11
17	1.563	0.436	2.903	0.970	1.456	1.025	101.05
18	1.564	0.435	2.904	0.971	1.456	1.024	100.32
19	1.564	0.431	2.907	0.972	1.455	1.023	99.38
20	1.564	0.428	2.909	0.973	1.454	1.022	99.13
21	1.563	0.429	2.912	0.974	1.453	1.021	98.63
22	1.562	0.429	2.916	0.975	1.452	1.020	98.46
23	1.565	0.427	2.917	0.975	1.453	1.018	98.16
24	1.564	0.428	2.917	0.975	1.452	1.018	98.13
25	1.565	0.428	2.917	0.975	1.453	1.018	98.02
26	1.565	0.428	2.919	0.976	1.453	1.017	97.81
27	1.567	0.430	2.921	0.976	1.453	1.014	97.38
28	1.569	0.432	2.926	0.977	1.454	1.010	96.57
29	1.574	0.436	2.937	0.979	1.455	1.001	95.06
30	1.584	0.444	2.958	0.982	1.458	0.982	92.48
31	1.604	0.461	2.999	0.989	1.464	0.945	89.09
32	1.623	0.477	3.039	0.996	1.470	0.910	88.04
33	1.620	0.478	3.042	0.999	1.469	0.911	87.74
34	1.619	0.476	3.044	0.998	1.468	0.911	87.39
35	1.616	0.471	3.048	0.996	1.465	0.912	86.93
36	1.614	0.468	3.051	0.995	1.463	0.912	86.82
37	1.613	0.468	3.054	0.990	1.464	0.912	86.48
38	1.611	0.468	3.059	0.979	1.464	0.911	85.86
39	1.608	0.467	3.068	0.959	1.464	0.910	84.83
40	1.600	0.466	3.087	0.918	1.464	0.907	83.74
41	1.598	0.466	3.094	0.905	1.464	0.906	83.67

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
42	1.596	0.465	3.100	0.891	1.464	0.904	82.82
43	1.594	0.462	3.113	0.864	1.466	0.902	81.49
44	1.588	0.456	3.137	0.808	1.468	0.896	80.13
45	1.586	0.454	3.147	0.787	1.469	0.894	80.02
46	1.584	0.461	3.157	0.767	1.471	0.890	79.99
47	1.585	0.457	3.152	0.776	1.470	0.892	79.68
48	1.587	0.459	3.157	0.767	1.472	0.889	79.14
49	1.590	0.463	3.166	0.747	1.477	0.882	78.58
50	1.589	0.464	3.168	0.747	1.476	0.881	78.49
51	1.589	0.464	3.169	0.747	1.476	0.880	78.37
52	1.590	0.464	3.172	0.749	1.476	0.878	78.13
53	1.591	0.465	3.178	0.751	1.476	0.875	77.71
54	1.593	0.466	3.189	0.755	1.477	0.867	77.04
55	1.597	0.468	3.213	0.764	1.477	0.851	76.39
56	1.598	0.468	3.217	0.766	1.477	0.848	76.37
57	1.599	0.471	3.221	0.767	1.478	0.845	76.05
58	1.603	0.476	3.228	0.768	1.481	0.838	75.54
59	1.610	0.486	3.242	0.772	1.485	0.825	75.07
60	1.611	0.488	3.245	0.773	1.486	0.822	75.06
61	1.613	0.489	3.248	0.773	1.488	0.819	74.82
62	1.617	0.492	3.253	0.774	1.490	0.814	74.50
63	1.620	0.494	3.258	0.775	1.492	0.809	74.41
64	1.620	0.495	3.263	0.769	1.494	0.808	74.24
65	1.621	0.498	3.272	0.757	1.496	0.806	74.04
66	1.622	0.497	3.272	0.756	1.497	0.806	73.86
67	1.621	0.496	3.274	0.755	1.496	0.806	73.80
68	1.619	0.493	3.279	0.748	1.496	0.808	73.63
69	1.618	0.491	3.282	0.743	1.496	0.809	73.60
70	1.617	0.490	3.285	0.738	1.496	0.810	73.47
71	1.615	0.488	3.292	0.728	1.496	0.812	73.35
72	1.614	0.488	3.292	0.727	1.496	0.812	73.35
73	1.614	0.488	3.293	0.726	1.496	0.812	73.32
74	1.614	0.488	3.294	0.724	1.497	0.812	73.27
75	1.614	0.488	3.296	0.720	1.497	0.813	73.17
76	1.614	0.488	3.299	0.712	1.498	0.814	72.98
77	1.614	0.488	3.307	0.697	1.501	0.816	72.65
78	1.613	0.488	3.321	0.665	1.506	0.819	72.14
79	1.612	0.488	3.350	0.603	1.516	0.826	71.70
80	1.612	0.487	3.354	0.593	1.518	0.827	71.69
81	1.611	0.488	3.357	0.591	1.518	0.827	71.65
82	1.611	0.488	3.359	0.589	1.518	0.827	71.59

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
83	1.611	0.488	3.364	0.586	1.519	0.826	71.48
84	1.611	0.487	3.375	0.578	1.520	0.824	71.29
85	1.611	0.486	3.395	0.563	1.523	0.821	71.11
86	1.616	0.493	3.395	0.562	1.527	0.817	70.82
87	1.615	0.492	3.395	0.562	1.527	0.818	70.81
88	1.616	0.492	3.395	0.562	1.527	0.817	70.79
89	1.617	0.492	3.395	0.562	1.528	0.817	70.78
90	1.616	0.492	3.395	0.562	1.527	0.817	70.77
91	1.616	0.492	3.395	0.562	1.528	0.817	70.76
92	1.617	0.493	3.396	0.562	1.528	0.816	70.73
93	1.617	0.493	3.398	0.563	1.528	0.815	70.68
94	1.618	0.494	3.401	0.564	1.529	0.813	70.57
95	**1.620**	**0.496**	**3.407**	**0.566**	**1.530**	**0.808**	**70.40**

Dodatek E

Vývoj parametrů při minimalizaci funkce χ^2 simplexní metodou při analýze dat o struktuře fotonů

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
1	0.059	0.511	0.544	0.699	1.940	1.477	468.713
2	0.059	0.511	0.544	0.699	1.940	1.477	468.713
3	0.059	0.511	0.544	0.699	1.940	1.477	468.713
4	0.059	0.511	0.544	0.699	1.940	1.477	468.713
5	0.059	0.511	0.544	0.699	1.940	1.477	468.713
6	0.059	0.511	0.544	0.699	1.940	1.477	468.713
7	0.059	0.511	0.544	0.699	1.940	1.477	468.713
8	0.059	0.511	0.544	0.699	1.940	1.477	468.713
9	0.059	0.511	0.544	0.699	1.940	1.477	468.713
10	0.059	0.511	0.544	0.699	1.940	1.477	468.713
11	0.073	0.427	0.703	0.635	1.599	1.636	359.833
12	0.075	0.504	0.669	0.692	1.565	1.602	355.378
13	0.075	0.504	0.669	0.692	1.565	1.602	355.378
14	0.068	0.388	0.678	0.705	1.598	1.635	205.740
15	0.068	0.388	0.678	0.705	1.598	1.635	205.740
16	0.068	0.388	0.678	0.705	1.598	1.635	205.740
17	0.068	0.388	0.678	0.705	1.598	1.635	205.740
18	0.068	0.388	0.678	0.705	1.598	1.635	205.740
19	0.068	0.388	0.678	0.705	1.598	1.635	205.740
20	0.068	0.388	0.678	0.705	1.598	1.635	205.740
21	0.068	0.388	0.678	0.705	1.598	1.635	205.740
22	0.068	0.388	0.678	0.705	1.598	1.635	205.740
23	0.068	0.388	0.678	0.705	1.598	1.635	205.740
24	0.067	0.464	0.662	0.688	1.704	1.611	199.341
25	0.067	0.458	0.718	0.687	1.622	1.626	187.716
26	0.067	0.458	0.718	0.687	1.622	1.626	187.716
27	0.065	0.407	0.711	0.682	1.693	1.690	171.557
28	0.065	0.407	0.711	0.682	1.693	1.690	171.557
29	0.065	0.407	0.711	0.682	1.693	1.690	171.557
30	0.065	0.407	0.711	0.682	1.693	1.690	171.557
31	0.067	0.399	0.751	0.696	1.634	1.728	164.634
32	0.067	0.399	0.751	0.696	1.634	1.728	164.634
33	0.067	0.399	0.751	0.696	1.634	1.728	164.634
34	0.067	0.399	0.751	0.696	1.634	1.728	164.634
35	0.067	0.399	0.751	0.696	1.634	1.728	164.634
36	0.067	0.399	0.751	0.696	1.634	1.728	164.634
37	0.067	0.399	0.751	0.696	1.634	1.728	164.634
38	0.067	0.399	0.751	0.696	1.634	1.728	164.634
39	0.067	0.399	0.751	0.696	1.634	1.728	164.634
40	0.067	0.399	0.751	0.696	1.634	1.728	164.634
41	0.067	0.399	0.751	0.696	1.634	1.728	164.634

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
42	0.067	0.399	0.751	0.696	1.634	1.728	164.634
43	0.067	0.399	0.751	0.696	1.634	1.728	164.634
44	0.067	0.399	0.751	0.696	1.634	1.728	164.634
45	0.066	0.393	0.717	0.707	1.714	1.741	163.767
46	0.066	0.377	0.748	0.694	1.674	1.748	163.175
47	0.066	0.377	0.748	0.694	1.674	1.748	163.175
48	0.066	0.377	0.748	0.694	1.674	1.748	163.175
49	0.066	0.377	0.748	0.694	1.674	1.748	163.175
50	0.066	0.377	0.748	0.694	1.674	1.748	163.175
51	0.065	0.369	0.704	0.696	1.781	1.742	160.627
52	0.065	0.369	0.704	0.696	1.781	1.742	160.627
53	0.065	0.369	0.704	0.696	1.781	1.742	160.627
54	0.065	0.369	0.704	0.696	1.781	1.742	160.627
55	0.065	0.369	0.704	0.696	1.781	1.742	160.627
56	0.065	0.369	0.704	0.696	1.781	1.742	160.627
57	0.062	0.333	0.642	0.720	1.906	1.774	159.620
58	0.062	0.314	0.673	0.704	1.884	1.810	156.352
59	0.062	0.314	0.673	0.704	1.884	1.810	156.352
60	0.062	0.314	0.673	0.704	1.884	1.810	156.352
61	0.062	0.314	0.673	0.704	1.884	1.810	156.352
62	0.062	0.314	0.673	0.704	1.884	1.810	156.352
63	0.060	0.300	0.603	0.714	2.058	1.788	154.372
64	0.058	0.257	0.559	0.743	2.115	1.868	151.602
65	0.058	0.257	0.559	0.743	2.115	1.868	151.602
66	0.058	0.246	0.577	0.713	2.156	1.882	148.824
67	0.053	0.165	0.462	0.732	2.427	1.930	145.590
68	0.053	0.165	0.462	0.732	2.427	1.930	145.590
69	0.051	0.106	0.449	0.739	2.554	1.996	144.529
70	0.051	0.106	0.449	0.739	2.554	1.996	144.529
71	0.051	0.106	0.449	0.739	2.554	1.996	144.529
72	0.051	0.106	0.449	0.739	2.554	1.996	144.529
73	0.049	0.065	0.394	0.726	2.714	2.034	141.401
74	0.049	0.065	0.394	0.726	2.714	2.034	141.401
75	0.049	0.065	0.394	0.726	2.714	2.034	141.401
76	0.049	0.065	0.394	0.726	2.714	2.034	141.401
77	0.049	0.065	0.394	0.726	2.714	2.034	141.401
78	0.049	0.065	0.394	0.726	2.714	2.034	141.401
79	0.049	0.068	0.393	0.738	2.716	2.019	140.795
80	0.049	0.068	0.393	0.738	2.716	2.019	140.795
81	0.048	0.051	0.367	0.731	2.811	2.032	140.003
82	0.048	0.051	0.367	0.731	2.811	2.032	140.003

	$\sqrt{A}$	α	β	Λ	γ	η	χ^2
83	0.048	0.051	0.367	0.731	2.811	2.032	140.003
84	0.048	0.051	0.367	0.731	2.811	2.032	140.003
85	0.048	0.047	0.371	0.720	2.808	2.032	139.472
86	0.048	0.047	0.371	0.720	2.808	2.032	139.472
87	0.048	0.047	0.371	0.720	2.808	2.032	139.472
88	0.048	0.047	0.371	0.720	2.808	2.032	139.472
89	0.049	0.081	0.375	0.719	2.765	1.994	139.186
90	0.047	0.025	0.345	0.720	2.932	2.046	138.589
91	0.047	0.025	0.345	0.720	2.932	2.046	138.589
92	0.047	0.025	0.345	0.720	2.932	2.046	138.589
93	0.047	0.025	0.345	0.720	2.932	2.046	138.589
94	0.047	0.025	0.345	0.720	2.932	2.046	138.589
95	0.047	0.025	0.328	0.712	2.970	2.041	137.965
96	0.046	0.012	0.328	0.716	2.970	2.056	137.950
97	0.046	0.012	0.328	0.716	2.970	2.056	137.950
98	0.046	0.012	0.328	0.716	2.970	2.056	137.950
99	0.046	0.012	0.328	0.716	2.970	2.056	137.950
100	0.046	0.002	0.311	0.721	3.035	2.062	137.734
101	0.046	0.002	0.311	0.721	3.035	2.062	137.734
102	0.045	0.014	0.296	0.720	3.087	2.075	137.668
103	0.045	0.014	0.296	0.720	3.087	2.075	137.668
104	0.047	0.022	0.329	0.716	2.967	2.043	137.668
105	0.047	0.017	0.323	0.714	2.990	2.048	137.613
106	0.047	0.017	0.323	0.714	2.990	2.048	137.613
107	0.047	0.017	0.323	0.714	2.990	2.048	137.613
108	0.046	0.004	0.312	0.711	3.041	2.057	137.595
109	0.046	0.003	0.309	0.715	3.056	2.065	137.556
110	0.046	0.003	0.309	0.715	3.056	2.065	137.556
111	0.046	0.003	0.309	0.715	3.056	2.065	137.556
112	0.046	0.004	0.307	0.715	3.056	2.066	137.537
113	0.046	0.004	0.307	0.715	3.056	2.066	137.537
114	0.046	0.008	0.316	0.715	3.021	2.056	137.526
115	0.046	0.008	0.316	0.715	3.021	2.056	137.526
116	0.046	0.008	0.316	0.715	3.021	2.056	137.526
117	0.046	0.008	0.316	0.715	3.021	2.056	137.526
118	**0.046**	**0.008**	**0.316**	**0.715**	**3.021**	**2.056**	**137.526**

Dodatek F

Zdrojový kód použitého programu

```
#include <iostream.h>
#include <curses.h>
#include <stdio.h>
#include <math.h>
#include <gsl/gsl_integration.h>
#include <gsl/gsl_complex.h>
#include <gsl/gsl_complex_math.h>
#include <gsl/gsl_diff.h>
#include <gsl/gsl_multimin.h>
#define betanula 4.1

double funkce (double x, void * params)
 {
  double *zk = (double*)params;
  double alpha = zk[0];
  double beta = zk[1];
  double f = x + alpha + beta;
  return f;
}

double RE_psi_integrand (double t, void * aa)
 {
  gsl_complex k1, k2, k22, k3, k33, k4, k5, k6, k7;
  double *jej = (double *)aa;
  double a =  jej[0];
  double b =  jej[1];
  GSL_SET_COMPLEX (&k1, a, b);                //a+ib
  double c =pow(M_E,t*(-1));                  //e^{-t}
  GSL_SET_COMPLEX (&k2, M_E, 0);              //e+i0
  k3 = gsl_complex_mul_real(k1,t*(-1));       //z*(-t)
  k4 = gsl_complex_pow(k2, k3);               //e^{-zt}
  double v1 = c/t;                            //e^{-t}/t
  k5 = gsl_complex_div_real (k4, (1-c));      //e^{-zt}/(1-e^{-t})
  k6 = gsl_complex_mul_real (k5, -1);         // -(e^{-zt}/(1-e^{-t}))
  k7 = gsl_complex_add_real (k6, v1);
  double f = GSL_REAL(k7);
  return f;
 }

double IM_psi_integrand (double t, void * aa)
 {
  gsl_complex k1, k2, k22, k3, k33, k4, k5, k6, k7;
```

```
  double *jej1 = (double *)aa;
  double a =  jej1[0];
  double b =  jej1[1];
  GSL_SET_COMPLEX (&k1, a, b);                  //a+ib
  double c =pow(M_E,t*(-1));                    //e^{-t}
  GSL_SET_COMPLEX (&k2, M_E, 0);                //e+i0
  k3 = gsl_complex_mul_real(k1,t*(-1));         //z*(-t)
  k4 = gsl_complex_pow(k2, k3);                 //e^{-zt}
  double v1 = c/t;                              //e^{-t}/t
  k5 = gsl_complex_div_real (k4, (1-c));        //e^{-zt}/(1-e^{-t})
  k6 = gsl_complex_mul_real (k5, -1);           // -(e^{-zt}/(1-e^{-t}))
  k7 = gsl_complex_add_real (k6, v1);
  double f = GSL_IMAG(k7);
  return f;
 }

double REBetaFunkce (double bzz, void * q)
 {
   double *jej3 = (double *)q;
   double prom =  jej3[0];
   double bb   =  jej3[1];
   double bbeta = jej3[2];
  gsl_complex k1, k2, k3, k4, k5, k6, k7;
  GSL_SET_COMPLEX (&k1, bzz, 0);
  GSL_SET_COMPLEX (&k2, prom, bb);
  GSL_SET_COMPLEX (&k3, (1-bzz), 0);
  GSL_SET_COMPLEX (&k4, bbeta, 0);
  k5 = gsl_complex_pow(k1, k2);
  k6 = gsl_complex_pow(k3, k4);
  k7 = gsl_complex_mul (k5, k6);
  double vysledek = GSL_REAL(k7);
  return vysledek;
 }

double IMBetaFunkce (double bzz, void * q)
 {
   double *jej4 = (double *)q;
   double prom =  jej4[0];
   double bb =    jej4[1];
   double bbeta = jej4[2];
  gsl_complex k1, k2, k3, k4, k5, k6, k7;
  GSL_SET_COMPLEX (&k1, bzz, 0);
  GSL_SET_COMPLEX (&k2, prom, bb);
```

```
  GSL_SET_COMPLEX (&k3, (1-bzz), 0);
  GSL_SET_COMPLEX (&k4, bbeta, 0);
  k5 = gsl_complex_pow(k1, k2);
  k6 = gsl_complex_pow(k3, k4);
  k7 = gsl_complex_mul (k5, k6);
  double vysledek = GSL_IMAG(k7);
  return vysledek;
 }

double f (double b, void * pole)
{
  double *ble = (double *)pole;
  double alpha = ble[0];
  double beta = ble[1];
  double VA = ble[2];
  double QQ0 = ble[3];
  double QQ = ble[4];
  double x = ble[5];
  double a = ble[6];
  double lambda = ble[7];
  double gamma = ble[8];
  double eta = ble[9];

QQ0=4;
gsl_complex PZ, ppodilas, PZnew, vysledek1, vysledek2, vysledek3,
 vysledek4,vysledek5, pointlike, l1, l2, l3, k1, k2, k3, p1, p2,
 p3, losos1, losos2,losos3, n;
double LQ = log((QQ)/(lambda*lambda));
double LQ0 =log((QQ0)/(lambda*lambda));
double asQ = 1/(betanula*LQ);
double asQ0 = 1/(betanula*LQ0);
double podilas = asQ/asQ0;

GSL_SET_COMPLEX (&ppodilas, podilas, 0);

/*Tady zacina vypocet momentu vetvici funkce*/
gsl_integration_workspace * ppsi1 = gsl_integration_workspace_alloc(1000);
double *policKojedna = new double [2];
policKojedna[0] = 1;
policKojedna[1] = 0;
gsl_function psi_1; //bude to funkce se dvema parametry
psi_1.function = &RE_psi_integrand;
psi_1.params = policKojedna;
```

```
double result_1, error_1;
gsl_integration_qags (&psi_1, 0, 30, 0, 1e-7, 1000, ppsi1, &result_1, &error_1);
gsl_integration_workspace_free (ppsi1);

gsl_integration_workspace * ppsi3 = gsl_integration_workspace_alloc(1000);
double *polickodva = new double [2];
polickodva[0] = 3;
polickodva[1] = 0;
gsl_function psi_3; //bude to funkce se dvema parametry
psi_3.function = &RE_psi_integrand;
psi_3.params = polickodva;
double result_3, error_3;
gsl_integration_qags (&psi_3, 0, 30, 0, 1e-7, 1000, ppsi3, &result_3, &error_3);
gsl_integration_workspace_free (ppsi3);

gsl_integration_workspace * ppsinre = gsl_integration_workspace_alloc(1000);
double *polickotri = new double [2];
polickotri[0] = a;
polickotri[1] = b;
gsl_function psi_n_re; //bude to funkce se dvema parametry
psi_n_re.function = &RE_psi_integrand;
psi_n_re.params = polickotri;
double result_n_re, error_n_re;
gsl_integration_qags (&psi_n_re, 0, 30, 0, 1e-7,
 1000, ppsinre, &result_n_re, &error_n_re);
gsl_integration_workspace_free (ppsinre);

gsl_integration_workspace * ppsinim = gsl_integration_workspace_alloc(1000);
double *polickoctyri = new double [2];
polickoctyri[0] = a;
polickoctyri[1] = b;
gsl_function psi_n_im; //bude to funkce se dvema parametry
psi_n_im.function = &IM_psi_integrand;
psi_n_im.params = polickoctyri;
double result_n_im, error_n_im;
gsl_integration_qags (&psi_n_im, 0, 30, 0, 1e-7,
1000, ppsinim, &result_n_im, &error_n_im);
gsl_integration_workspace_free (ppsinim);

gsl_integration_workspace * ppsinplus2re = gsl_integration_workspace_alloc(1000);
double *polickopet = new double [2];
polickopet[0] = (a+2);
polickopet[1] = b;
```

```
gsl_function psi_nplus2_re; //bude to funkce se dvema parametry
psi_nplus2_re.function = &RE_psi_integrand;
psi_nplus2_re.params = policko pet;
double result_nplus2_re, error_nplus2_re;
gsl_integration_qags (&psi_nplus2_re, 0, 30, 0, 1e-7,
1000, ppsinplus2re, &result_nplus2_re, &error_nplus2_re);
gsl_integration_workspace_free (ppsinplus2re);

gsl_integration_workspace * ppsinplus2im = gsl_integration_workspace_alloc(1000);
double *policko sest = new double [2];
policko sest[0] = (a+2);
policko sest[1] = b;
gsl_function psi_nplus2_im;
psi_nplus2_im.function = &IM_psi_integrand;
psi_nplus2_im.params = policko sest;
double result_nplus2_im, error_nplus2_im;
gsl_integration_qags (&psi_nplus2_im, 0, 30, 0, 1e-7,
1000, ppsinplus2im, &result_nplus2_im, &error_nplus2_im);
gsl_integration_workspace_free (ppsinplus2im);

gsl_complex psi_n,psi_nplus2, z1, z2, z3, z4, z5;

GSL_SET_COMPLEX(&psi_n, result_n_re, result_n_im);
GSL_SET_COMPLEX(&psi_nplus2, result_nplus2_re, result_nplus2_im);
/*Tady konci vypocet momentu vetvici funkce*/

double psi1plus3 = result_1 + result_3;

z1 = gsl_complex_mul_real (psi_n, -1);
z2 = gsl_complex_mul_real (psi_nplus2, -1);
z3 = gsl_complex_add (z1, z2);
z4 = gsl_complex_add_real (z3, psi1plus3);
z5 = gsl_complex_mul_real (z4, 4/3);

PZnew = gsl_complex_mul_real (z5, (-1)*(1/betanula));
vysledek1 = gsl_complex_pow (ppodilas, PZnew);

/*tady zacina x na minus n*/

GSL_SET_COMPLEX (&p2, x, 0); /*tohle je input hodnoty x*/
GSL_SET_COMPLEX (&p3, a*(-1), b*(-1));
vysledek2 = gsl_complex_pow (p2, p3);
```

```
//tady zacina ta beta funkce

gsl_integration_workspace * w3 = gsl_integration_workspace_alloc(1000);
double *policko3 = new double [3];
policko3[0] = (a+(alpha)-2);
policko3[1] = b;
policko3[2] = (beta);

gsl_function Funkce3; //bude to funkce s peti parametry
Funkce3.function = &REBetaFunkce;
Funkce3.params = policko3;
double result3, error3;
gsl_integration_qags (&Funkce3, 0, 1, 0, 1e-3, 1000, w3, &result3, &error3);
gsl_integration_workspace_free (w3);

gsl_integration_workspace * w4 = gsl_integration_workspace_alloc(1000);
double *policko4 = new double [3];
policko4[0] = (a+(alpha*alpha)-2);
policko4[1] = b;
policko4[2] = (beta*beta);

gsl_function Funkce4; //bude to funkce s peti parametry
Funkce4.function = &IMBetaFunkce;
Funkce4.params = policko4;
double result4, error4;
gsl_integration_qags (&Funkce4, 0, 1, 0, 1e-3, 1000, w4, &result4, &error4);
gsl_integration_workspace_free (w4);

gsl_integration_workspace * w5 = gsl_integration_workspace_alloc(1000);
double *policko5 = new double [3];
policko5[0] = (a+(alpha*alpha)-2+(eta*eta));
policko5[1] = b;
policko5[2] = (beta*beta);

gsl_function Funkce5; //bude to funkce s peti parametry
Funkce5.function = &REBetaFunkce;
Funkce5.params = policko5;
double result5, error5;
gsl_integration_qags (&Funkce5, 0, 1, 0, 1e-3, 1000, w5, &result5, &error5);
gsl_integration_workspace_free (w5);

gsl_integration_workspace * w6 = gsl_integration_workspace_alloc(1000);
double *policko6 = new double [3];
```

```
policko6[0] = (a+(alpha*alpha)-2+(eta*eta));
policko6[1] = b;
policko6[2] = (beta*beta);

gsl_function Funkce6; //bude to funkce s peti parametry
Funkce6.function = &IMBetaFunkce;
Funkce6.params = policko6;
double result6, error6;
gsl_integration_qags (&Funkce6, 0, 1, 0, 1e-3, 1000, w6, &result6, &error6);
gsl_integration_workspace_free (w6);

GSL_SET_COMPLEX (&losos1, result3, result4);
GSL_SET_COMPLEX (&losos2, result5, result6);

vysledek3 =  gsl_complex_mul_real (losos1, VA*VA);
vysledek4 =  gsl_complex_mul_real (losos2,gamma*gamma*VA*VA);

/*Ted prijde vysledna hodnota te funkce*/
double vvvysledek = GSL_REAL(gsl_complex_mul (vysledek2,gsl_complex_mul
(vysledek1, gsl_complex_add(vysledek3,vysledek4))));

 delete [] polickojedna;
 delete [] polickodva;
 delete [] polickotri;
 delete [] polickoctyri;
 delete [] polickopet;
 delete [] polickosest;
 delete [] policko3;
 delete [] policko4;
 delete [] policko5;
 delete [] policko6;

return vvvysledek;
}

double pointlike (double b, void * pole)
{
  double *ble = (double *)pole;

  double alpha = ble[0];
  double beta = ble[1];
  double VA = ble[2];
  double QQ0 = ble[3];
```

```
  double QQ = ble[4];
  double x = ble[5];
  double a = ble[6];
  double lambda = ble[7];
  double gamma = ble[8];
  double eta = ble[9];

a=0.2;
QQ0=1;

gsl_complex PZ, ppodilas, PZnew, vysledek1, vysledek2, vysledek3,
vysledek4,vysledek5, pointlike, l1, l2, l3, k1, k2, k3, p1, p2, p3, losos1,
losos2,losos3, n;
double LQ = log((QQ)/(lambda*lambda));
double LQ0 =log((QQ0)/(lambda*lambda));
double asQ = 1/(betanula*LQ);
double asQ0 = 1/(betanula*LQ0);
double podilas = asQ/asQ0;

GSL_SET_COMPLEX (&ppodilas, podilas, 0);

/*Tady zacina vypocet momentu vetvici funkce*/
gsl_integration_workspace * ppsi1 = gsl_integration_workspace_alloc(1000);
double *polickojedna = new double [2];
polickojedna[0] = 1;
polickojedna[1] = 0;
gsl_function psi_1; //bude to funkce se dvema parametry
psi_1.function = &RE_psi_integrand;
psi_1.params = polickojedna;
double result_1, error_1;
gsl_integration_qags (&psi_1, 0, 30, 0, 1e-7, 1000, ppsi1, &result_1, &error_1);
gsl_integration_workspace_free (ppsi1);

gsl_integration_workspace * ppsi3 = gsl_integration_workspace_alloc(1000);
double *polickodva = new double [2];
polickodva[0] = 3;
polickodva[1] = 0;
gsl_function psi_3; //bude to funkce se dvema parametry
psi_3.function = &RE_psi_integrand;
psi_3.params = polickodva;
double result_3, error_3;
gsl_integration_qags (&psi_3, 0, 30, 0, 1e-7, 1000, ppsi3, &result_3, &error_3);
gsl_integration_workspace_free (ppsi3);
```

```
gsl_integration_workspace * ppsinre = gsl_integration_workspace_alloc(1000);
double *policKotri = new double [2];
policKotri[0] = a;
policKotri[1] = b;
gsl_function psi_n_re; //bude to funkce se dvema parametry
psi_n_re.function = &RE_psi_integrand;
psi_n_re.params = policKotri;
double result_n_re, error_n_re;
gsl_integration_qags (&psi_n_re, 0, 30, 0, 1e-7,
1000, ppsinre, &result_n_re, &error_n_re);
gsl_integration_workspace_free (ppsinre);

gsl_integration_workspace * ppsinim = gsl_integration_workspace_alloc(1000);
double *policKoctyri = new double [2];
policKoctyri[0] = a;
policKoctyri[1] = b;
gsl_function psi_n_im; //bude to funkce se dvema parametry
psi_n_im.function = &IM_psi_integrand;
psi_n_im.params = policKoctyri;
double result_n_im, error_n_im;
gsl_integration_qags (&psi_n_im, 0, 30, 0, 1e-7,
1000, ppsinim, &result_n_im, &error_n_im);
gsl_integration_workspace_free (ppsinim);

gsl_integration_workspace * ppsinplus2re = gsl_integration_workspace_alloc(1000);
double *policKopet = new double [2];
policKopet[0] = (a+2);
policKopet[1] = b;
gsl_function psi_nplus2_re; //bude to funkce se dvema parametry
psi_nplus2_re.function = &RE_psi_integrand;
psi_nplus2_re.params = policKopet;
double result_nplus2_re, error_nplus2_re;
gsl_integration_qags (&psi_nplus2_re, 0, 30, 0, 1e-7,
1000, ppsinplus2re, &result_nplus2_re, &error_nplus2_re);
gsl_integration_workspace_free (ppsinplus2re);

gsl_integration_workspace * ppsinplus2im = gsl_integration_workspace_alloc(1000);
double *policKosest = new double [2];
policKosest[0] = (a+2);
policKosest[1] = b;
gsl_function psi_nplus2_im; //bude to funkce se dvema parametry
psi_nplus2_im.function = &IM_psi_integrand;
```

```
psi_nplus2_im.params = polickosest;
double result_nplus2_im, error_nplus2_im;
gsl_integration_qags (&psi_nplus2_im, 0, 30, 0, 1e-7,
1000, ppsinplus2im, &result_nplus2_im, &error_nplus2_im);
gsl_integration_workspace_free (ppsinplus2im);

gsl_complex psi_n,psi_nplus2, z1, z2, z3, z4, z5;

GSL_SET_COMPLEX(&psi_n, result_n_re, result_n_im);
GSL_SET_COMPLEX(&psi_nplus2, result_nplus2_re, result_nplus2_im);
/*Tady konci vypocet momentu vetvici funkce*/

double psi1plus3 = result_1 + result_3;

z1 = gsl_complex_mul_real (psi_n, -1);
z2 = gsl_complex_mul_real (psi_nplus2, -1);
z3 = gsl_complex_add (z1, z2);
z4 = gsl_complex_add_real (z3, psi1plus3);
z5 = gsl_complex_mul_real (z4, 4/3);

/*Tady zacina pointlike cast*/

gsl_complex Pnew1 = gsl_complex_mul_real (z5,(-1)*2/betanula );
gsl_complex Pnew = gsl_complex_add_real (Pnew1, 1);

gsl_complex v1 = gsl_complex_pow (ppodilas, Pnew);
gsl_complex v2 = gsl_complex_mul_real (v1, -1);
gsl_complex v3 = gsl_complex_add_real (v2, 1);
gsl_complex v4 = gsl_complex_mul_real (v3,(4*3.142)/(4*3.142*asQ));

double conadra;
conadra = (24*0.02777)/(2*3.142*betanula*137);

GSL_SET_COMPLEX (&n, a, b);
gsl_complex aNS1 = gsl_complex_add (gsl_complex_inverse(n),
gsl_complex_mul_real(gsl_complex_inverse(gsl_complex_add_real(n,2)),2));
gsl_complex aNS2 = gsl_complex_sub
(aNS1,gsl_complex_mul_real(gsl_complex_inverse(gsl_complex_add_real(n,1)),2) );

gsl_complex aNS3 = gsl_complex_div (aNS2, Pnew);
gsl_complex aNS = gsl_complex_mul_real (aNS3, conadra);
pointlike = gsl_complex_mul (aNS, v4);
```

```
/*Tady konci pointlike cast*/

/*tady zacina x na minus n*/

GSL_SET_COMPLEX (&p2, x, 0);
GSL_SET_COMPLEX (&p3, a*(-1), b*(-1));
vysledek2 = gsl_complex_pow (p2, p3);

double vvvysledek = GSL_REAL(gsl_complex_mul (vysledek2,pointlike));

 delete [] polickojedna;
 delete [] polickodva;
 delete [] polickotri;
 delete [] polickoctyri;
 delete [] polickopet;
 delete [] polickosest;

return vvvysledek;
}

/*tady zacina procedura pro vypocet momentu distribucni funkce*/
double moment (double b, double * pole)
{
b=0;
double alpha = * pole;
double beta = *(pole+1);
double VA = *(pole+2);
double QQ0 = *(pole+3);
double QQ = *(pole+4);
double x = *(pole+5);
double a = *(pole+6);
double lambda = *(pole+7);

gsl_complex PZ, ppodilas, PZnew, vysledek1, vysledek2,
 vysledek3, l1, l2, l3, k1, k2, k3, p1, p2, p3,losos;
double LQ = log((QQ)/(lambda*lambda));
double LQ0 =log((QQ0)/(lambda*lambda));
double asQ = 1/(betanula*LQ);
double asQ0 = 1/(betanula*LQ0);
double podilas = asQ/asQ0;

GSL_SET_COMPLEX (&ppodilas, podilas, 0);
```

```
/*Tady zacina vypocet momentu vetvici funkce*/
gsl_integration_workspace * ppsi1 = gsl_integration_workspace_alloc(1000);
double *polickojedna = new double [2];
polickojedna[0] = 1;
polickojedna[1] = 0;
gsl_function psi_1; //bude to funkce se dvema parametry
psi_1.function = &RE_psi_integrand;
psi_1.params = polickojedna;
double result_1, error_1;
gsl_integration_qags (&psi_1, 0, 30, 0, 1e-4,
 1000, ppsi1, &result_1, &error_1);
gsl_integration_workspace_free (ppsi1);

gsl_integration_workspace * ppsi3 = gsl_integration_workspace_alloc(1000);
double *polickodva = new double [2];
polickodva[0] = 3;
polickodva[1] = 0;
gsl_function psi_3; //bude to funkce se dvema parametry
psi_3.function = &RE_psi_integrand;
psi_3.params = polickodva;
double result_3, error_3;
gsl_integration_qags (&psi_3, 0, 30, 0, 1e-4, 1000, ppsi3, &result_3, &error_3);
gsl_integration_workspace_free (ppsi3);

gsl_integration_workspace * ppsinre = gsl_integration_workspace_alloc(1000);
double *polickotri = new double [2];
polickotri[0] = a;
polickotri[1] = b;
gsl_function psi_n_re; //bude to funkce se dvema parametry
psi_n_re.function = &RE_psi_integrand;
psi_n_re.params = polickotri;
double result_n_re, error_n_re;
gsl_integration_qags (&psi_n_re, 0, 30, 0, 1e-4,
1000, ppsinre, &result_n_re, &error_n_re);
gsl_integration_workspace_free (ppsinre);

gsl_integration_workspace * ppsinim = gsl_integration_workspace_alloc(1000);
double *polickoctyri = new double [2];
polickoctyri[0] = a;
polickoctyri[1] = b;
gsl_function psi_n_im; //bude to funkce se dvema parametry
psi_n_im.function = &IM_psi_integrand;
psi_n_im.params = polickoctyri;
```

```
double result_n_im, error_n_im;
gsl_integration_qags (&psi_n_im, 0, 30, 0, 1e-4,
1000, ppsinim, &result_n_im, &error_n_im);
gsl_integration_workspace_free (ppsinim);

gsl_integration_workspace * ppsinplus2re = gsl_integration_workspace_alloc(1000);
double *policko pet = new double [2];
policko pet[0] = (a+2);
policko pet[1] = b;
gsl_function psi_nplus2_re; //bude to funkce se dvema parametry
psi_nplus2_re.function = &RE_psi_integrand;
psi_nplus2_re.params = policko pet;
double result_nplus2_re, error_nplus2_re;
gsl_integration_qags (&psi_nplus2_re, 0, 30, 0, 1e-4,
 1000, ppsinplus2re, &result_nplus2_re, &error_nplus2_re);
gsl_integration_workspace_free (ppsinplus2re);

gsl_integration_workspace * ppsinplus2im = gsl_integration_workspace_alloc(1000);
double *policko sest = new double [2];
policko sest[0] = (a+2);
policko sest[1] = b;
gsl_function psi_nplus2_im; //bude to funkce se dvema parametry
psi_nplus2_im.function = &IM_psi_integrand;
psi_nplus2_im.params = policko sest;
double result_nplus2_im, error_nplus2_im;
gsl_integration_qags (&psi_nplus2_im, 0, 30, 0, 1e-4,
 1000, ppsinplus2im, &result_nplus2_im, &error_nplus2_im);
gsl_integration_workspace_free (ppsinplus2im);

gsl_complex psi_n,psi_nplus2, z1, z2, z3, z4, z5;

GSL_SET_COMPLEX(&psi_n, result_n_re, result_n_im);
GSL_SET_COMPLEX(&psi_nplus2, result_nplus2_re, result_nplus2_im);
/*Tady konci vypocet momentu vetvici funkce*/

double psi1plus3 = result_1 + result_3;

z1 = gsl_complex_mul_real (psi_n, -1);
z2 = gsl_complex_mul_real (psi_nplus2, -1);
z3 = gsl_complex_add (z1, z2);
z4 = gsl_complex_add_real (z3, psi1plus3);
z5 = gsl_complex_mul_real (z4, 4/3);
```

```
PZnew = gsl_complex_mul_real (z5, (-1)*(1/betanula));

vysledek1 = gsl_complex_pow (ppodilas, PZnew);

/*tady zacina ta beta funkce */
gsl_integration_workspace * w3 = gsl_integration_workspace_alloc(1000);
double *policko3 = new double [5];
policko3[0] = VA;
policko3[1] = alpha;
policko3[2] = beta;
policko3[3] = a;
policko3[4] = b;
gsl_function Funkce3; //bude to funkce s peti parametry
Funkce3.function = &REBetaFunkce;
Funkce3.params = policko3;
double result3, error3;
gsl_integration_qags (&Funkce3, 0, 1, 0, 1e-7, 1000, w3, &result3, &error3);
gsl_integration_workspace_free (w3);

gsl_integration_workspace * w4 = gsl_integration_workspace_alloc(1000);
double *policko4 = new double [5];
policko4[0] = VA;
policko4[1] = alpha;
policko4[2] = beta;
policko4[3] = a;
policko4[4] = b;
gsl_function Funkce4; //bude to funkce s peti parametry
Funkce4.function = &IMBetaFunkce;
Funkce4.params = policko4;
double result4, error4;
gsl_integration_qags (&Funkce4, 0, 1, 0, 1e-7, 1000, w4, &result4, &error4);
gsl_integration_workspace_free (w4);

GSL_SET_COMPLEX (&losos, result3, result4); //!!!to je treti cast te funkce
vysledek3 =  gsl_complex_mul_real (losos, VA*VA);

/*Ted prijde vysledna hodnota te funkce*/
double vvvysledek = GSL_REAL(gsl_complex_mul (vysledek1, vysledek3));

 delete [] policko jedna;
 delete [] policko dva;
 delete [] policko tri;
 delete [] policko ctyri;
```

```
 delete [] polickopet;
 delete [] polickosest;
 delete [] policko3;
 delete [] policko4;
return vvvysledek;
}

/*tady konci procedura pro vypocet momentu distribucni funkce*/

/* Tady zacina pripravena funkce v promene alpha*/
double fcealpha (double alpha, void * parametry1)
{
double *kuk1 =(double *)parametry1;
double betapar = kuk1[1];
double Apar = kuk1[2];
double lambdapar = kuk1[3];
double p1 = kuk1[4];
double p2 = kuk1[5];
double gammapar = kuk1[6];
double etapar = kuk1[7];

typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j], &matice[4][j]);
    }
fclose (V);

double *lanalpha1 = new double[10];
double vysledek_r, staterror, resultalpha;
double chiskveralpha = 0;

for(int r=0;r<=48;r++)
{
lanalpha1[0] = alpha;
lanalpha1[1] = betapar;
lanalpha1[2] = Apar;
lanalpha1[3] = p1;
lanalpha1[4] = *(*(matice+1)+r);
```

```
lanalpha1[5] = *(*(matice+0)+r);
lanalpha1[6] = p2;
lanalpha1[7] = lambdapar;
lanalpha1[8] = gammapar;
lanalpha1[9] = etapar;

double resultalpha1, erroralpha1;
double resultalpha2, erroralpha2;

gsl_integration_workspace * alpha1 = gsl_integration_workspace_alloc(1000);

gsl_function funkcealpha1;
funkcealpha1.function = &f;
funkcealpha1.params = lanalpha1;

gsl_integration_qags (&funkcealpha1, 0,
40, 0, 1e-2, 1000,alpha1, &resultalpha1, &erroralpha1);
gsl_integration_workspace_free (alpha1);

gsl_integration_workspace * alpha2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcealpha2;
funkcealpha2.function = &pointlike;
funkcealpha2.params = lanalpha1;

gsl_integration_qags (&funkcealpha2, 0,
80, 0, 1e-2, 1000,alpha2, &resultalpha2, &erroralpha2);
gsl_integration_workspace_free (alpha2);

vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resultalpha = 137*((resultalpha1+resultalpha2)/3.142);
chiskveralpha = chiskveralpha+(((vysledek_r-resultalpha)
*(vysledek_r-resultalpha))/(staterror*staterror));
}
 delete [] lanalpha1;
 delete matice;
return chiskveralpha;
}
/* Tady konci pripravena funkce v promene alpha*/

/* Tady zacina pripravena funkce v promene beta*/
double fcebeta (double beta, void * parametry1)
```

```
{
double *kuk2 =(double *)parametry1;
double alphapar = kuk2[0];
//double betapar = kuk2[1];
double Apar = kuk2[2];
double lambdapar = kuk2[3];
double p1 = kuk2[4];
double p2 = kuk2[5];
double gammapar = kuk2[6];
double etapar = kuk2[7];

typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j], &matice[1][j],
         &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);

double *lanbeta1 = new double[10];
double vysledek_r, staterror, resultbeta;
double chiskverbeta = 0;

for(int r=0;r<=48;r++)
{
lanbeta1[0] = alphapar;
lanbeta1[1] = beta;
lanbeta1[2] = Apar;
lanbeta1[3] = p1;
lanbeta1[4] = *(*(matice+1)+r);
lanbeta1[5] = *(*(matice+0)+r);
lanbeta1[6] = p2;
lanbeta1[7] = lambdapar;
lanbeta1[8] = gammapar;
lanbeta1[9] = etapar;

double resultbeta1, errorbeta1;
double resultbeta2, errorbeta2;

gsl_integration_workspace * beta1 = gsl_integration_workspace_alloc(1000);
```

```
gsl_function funkcebeta1;
funkcebeta1.function = &f;
funkcebeta1.params = lanbeta1;
gsl_integration_qags (&funkcebeta1, 0,
40, 0, 1e-2, 1000,beta1, &resultbeta1, &errorbeta1);
gsl_integration_workspace_free (beta1);

gsl_integration_workspace * beta2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcebeta2;
funkcebeta2.function = &pointlike;
funkcebeta2.params = lanbeta1;
gsl_integration_qags (&funkcebeta2, 0,
80, 0, 1e-2, 1000,beta2, &resultbeta2, &errorbeta2);
gsl_integration_workspace_free (beta2);

vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resultbeta = 137*((resultbeta1+resultbeta2)/3.142);

chiskverbeta = chiskverbeta+(((vysledek_r-resultbeta)
*(vysledek_r-resultbeta))/(staterror*staterror));
}
delete [] lanbeta1;
delete matice;

return chiskverbeta;
}
/* Tady konci pripravena funkce v promene beta*/

/* Tady zacina pripravena funkce v promene A*/
double fceA (double A, void * parametry1)
{
double *kuk =(double *)parametry1;
double alphapar = kuk[0];
double betapar = kuk[1];
//double Apar = kuk[2];
double lambdapar = kuk[3];
double p1 = kuk[4];
double p2 = kuk[5];
double gammapar = kuk[6];
double etapar = kuk[7];
```

```
typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);

double *lanA1 = new double[10];
double vysledek_r,staterror,resultA;
double chiskverA = 0;

for(int r=0;r<=48;r++)
{

lanA1[0] = alphapar;
lanA1[1] = betapar;
lanA1[2] = A;
lanA1[3] = p1;
lanA1[4] = *(*(matice+1)+r);
lanA1[5] = *(*(matice+0)+r);
lanA1[6] = p2;
lanA1[7] = lambdapar;
lanA1[8] = gammapar;
lanA1[9] = etapar;

double resultA1, errorA1;
double resultA2, errorA2;

gsl_integration_workspace * A1 = gsl_integration_workspace_alloc(1000);
gsl_function funkceA1;
funkceA1.function = &f;
funkceA1.params = lanA1;
gsl_integration_qags (&funkceA1, 0,
40, 0, 1e-2, 1000,A1, &resultA1, &errorA1);
gsl_integration_workspace_free (A1);

gsl_integration_workspace * A2 = gsl_integration_workspace_alloc(1000);
gsl_function funkceA2;
funkceA2.function = &pointlike;
```

```
funkceA2.params = lanA1;
gsl_integration_qags (&funkceA2, 0,
80, 0, 1e-2, 1000,A2, &resultA2, &errorA2);
gsl_integration_workspace_free (A2);

vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r))) +
((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resultA = 137*((resultA1+resultA2)/3.142);

chiskverA = chiskverA+(((vysledek_r-resultA)*
(vysledek_r-resultA))/(staterror*staterror));
}
delete [] lanA1;
delete matice;
return chiskverA;
}
/* Tady konci pripravena funkce v promene A*/

/* Tady zacina pripravena funkce v promene lambda*/
double fcelambda (double llambda, void * parametry1)
{
double *kuk4 =(double *)parametry1;
double alphapar = kuk4[0];
double betapar = kuk4[1];
double Apar = kuk4[2];
//double lambdapar = kuk4[3];
double p1 = kuk4[4];
double p2 = kuk4[5];
double gammapar = kuk4[6];
double etapar = kuk4[7];

typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j], &matice[1][j],
         &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);
```

```
double *lanlambda1 = new double[10];
double vysledek_r,staterror, resultlambda;
double chiskverlambda = 0;

for(int r=0;r<=48;r++)
{
lanlambda1[0] = alphapar;
lanlambda1[1] = betapar;
lanlambda1[2] = Apar;
lanlambda1[3] = p1;
lanlambda1[4] = *(*(matice+1)+r);
lanlambda1[5] = *(*(matice+0)+r);
lanlambda1[6] = p2;
lanlambda1[7] = llambda;
lanlambda1[8] = gammapar;
lanlambda1[9] = etapar;

double resultlambda1, errorlambda1;
double resultlambda2, errorlambda2;

gsl_integration_workspace * lambda1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcelambda1;
funkcelambda1.function = &f;
funkcelambda1.params = lanlambda1;
gsl_integration_qags (&funkcelambda1, 0,
40, 0, 1e-2, 1000,lambda1, &resultlambda1, &errorlambda1);
gsl_integration_workspace_free (lambda1);

gsl_integration_workspace * lambda2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcelambda2;
funkcelambda2.function = &pointlike;
funkcelambda2.params = lanlambda1;
gsl_integration_qags (&funkcelambda2, 0,
 80, 0, 1e-2, 1000,lambda2, &resultlambda2, &errorlambda2);
gsl_integration_workspace_free (lambda2);

vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resultlambda = 137*((resultlambda1+resultlambda2)/3.142);
chiskverlambda = chiskverlambda+(((vysledek_r-resultlambda)
*(vysledek_r-resultlambda))/(staterror*staterror));
}
```

```
 delete [] lanlambda1;
 delete matice;

return chiskverlambda;
}
/* Tady konci pripravena funkce v promene lambda*/

/* Tady zacina  pripravena funkce v promene gamma*/
double fcegamma (double gamma, void * parametry1)
{
double *kuk5 =(double *)parametry1;
double alphapar = kuk5[0];
double betapar = kuk5[1];
double Apar = kuk5[2];
double lambdapar = kuk5[3];
double p1 = kuk5[4];
double p2 = kuk5[5];
//double gammapar = kuk5[6];
double etapar = kuk5[7];
typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);

double *langamma1 = new double[10];
double vysledek_r,staterror, resultgamma;
double chiskvergamma = 0;

for(int r=0;r<=48;r++)
{

langamma1[0] = alphapar;
langamma1[1] = betapar;
langamma1[2] = Apar;
langamma1[3] = p1;
langamma1[4] = *(*(matice+1)+r);
langamma1[5] = *(*(matice+0)+r);
```

```
langamma1[6] = p2;
langamma1[7] = lambdapar;
langamma1[8] = gamma;
langamma1[9] = etapar;

double resultgamma1, errorgamma1;
double resultgamma2, errorgamma2;

gsl_integration_workspace * gamma1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcegamma1;
funkcegamma1.function = &f;
funkcegamma1.params = langamma1;
gsl_integration_qags (&funkcegamma1, 0,
40, 0, 1e-2, 1000,gamma1, &resultgamma1, &errorgamma1);
gsl_integration_workspace_free (gamma1);

gsl_integration_workspace * gamma2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcegamma2;
funkcegamma2.function = &pointlike;
funkcegamma2.params = langamma1;
gsl_integration_qags (&funkcegamma2, 0,
80, 0, 1e-2, 1000,gamma2, &resultgamma2, &errorgamma2);
gsl_integration_workspace_free (gamma2);


vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resultgamma = 137*((resultgamma1+resultgamma2)/3.142);

chiskvergamma = chiskvergamma+(((vysledek_r-resultgamma)
*(vysledek_r-resultgamma))/(staterror*staterror));
}

delete [] langamma1;
delete matice;
return chiskvergamma;
}
/* Tady konci pripravena funkce v promene gamma*/

/* Tady zacina  pripravena funkce v promene eta*/

double fceeta (double eta, void * parametry1)
```

```
{
double *kuk5 =(double *)parametry1;
double alphapar = kuk5[0];
double betapar = kuk5[1];
double Apar = kuk5[2];
double lambdapar = kuk5[3];
double p1 = kuk5[4];
double p2 = kuk5[5];
double gammapar = kuk5[6];
//double eta = kuk5[7];

typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);

double *laneta1 = new double[10];
double vysledek_r,staterror, resulteta;
double chiskvereta = 0;

for(int r=0;r<=48;r++)
{

laneta1[0] = alphapar;
laneta1[1] = betapar;
laneta1[2] = Apar;
laneta1[3] = p1;
laneta1[4] = *(*(matice+1)+r);
laneta1[5] = *(*(matice+0)+r);
laneta1[6] = p2;
laneta1[7] = lambdapar;
laneta1[8] = gammapar;
laneta1[9] = eta;

double resulteta1, erroreta1;
double resulteta2, erroreta2;
```

```
gsl_integration_workspace * eta1 = gsl_integration_workspace_alloc(1000);
gsl_function funkceeta1;
funkceeta1.function = &f;
funkceeta1.params = laneta1;
gsl_integration_qags (&funkceeta1, 0,
40, 0, 1e-2, 1000,eta1, &resulteta1, &erroreta1);
gsl_integration_workspace_free (eta1);

gsl_integration_workspace * eta2 = gsl_integration_workspace_alloc(1000);
gsl_function funkceeta2;
funkceeta2.function = &pointlike;
funkceeta2.params = laneta1;
gsl_integration_qags (&funkceeta2, 0,
80, 0, 1e-2, 1000,eta2, &resulteta2, &erroreta2);
gsl_integration_workspace_free (eta2);

vysledek_r = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
resulteta = 137*((resulteta1+resulteta2)/3.142);
chiskvereta = chiskvereta+(((vysledek_r-resulteta)*
(vysledek_r-resulteta))/(staterror*staterror));
}
delete [] laneta1;
delete matice;

return chiskvereta;
}
/* Tady konci pripravena funkce v promene eta*/

double my_f (const gsl_vector *v, void *params)
{
double *dp = (double *)params;
double chiskver=0;
double vysledek_i, staterror;
double alpha, beta, A, Q0, lambda,gamma,eta;

A      = gsl_vector_get(v, 0);
alpha  = gsl_vector_get(v, 1);
beta   = gsl_vector_get(v, 2);
lambda = gsl_vector_get(v, 3);
gamma  = gsl_vector_get(v, 4);
eta    = gsl_vector_get(v, 5);
```

```
typedef double radek[49];
radek* matice = new radek[5];

FILE *V = fopen("datafotonIII.txt", "r");
for(int j=0; j<=48;j++)
    {
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j],&matice[4][j]);
    }
fclose (V);

double *policko = new double [10];
double jelen;

for(int r=0;r<=48;r++)
{
policko[0] = alpha; //c1;
policko[1] = beta; //c2;
policko[2] = A; //c3;
policko[3] = dp[0]; //p1;
policko[4] = *(*(matice+1)+r);
policko[5] = *(*(matice+0)+r);
policko[6] = dp[1];  //p2; //c7;
policko[7] = lambda;
policko[8] = gamma;
policko[9] = eta;

double rresultt, eerrorr, rr2,ee2;

gsl_integration_workspace * ww = gsl_integration_workspace_alloc(1000);
gsl_function tafunkcex;
tafunkcex.function = &f;
tafunkcex.params = policko;
gsl_integration_qags (&tafunkcex, 0,
40, 0, 1e-2, 1000,ww, &rresultt, &eerrorr);
gsl_integration_workspace_free (ww);

gsl_integration_workspace * www = gsl_integration_workspace_alloc(1000);
gsl_function tafunkcex2;
tafunkcex2.function = &pointlike;
tafunkcex2.params = policko;
gsl_integration_qags (&tafunkcex2, 0,
```

```
80, 0, 1e-2, 1000,www, &rr2, &ee2);
gsl_integration_workspace_free (www);

vysledek_i = *(*(matice+2)+r);
staterror = sqrt( ((*(*(matice+3)+r)) * (*(*(matice+3)+r)))
+ ((*(*(matice+4)+r)) * (*(*(matice+4)+r)))  );
jelen = 137*((rresultt+rr2)/3.142);

chiskver = chiskver+(((vysledek_i-jelen)*
(vysledek_i-jelen))/(staterror*staterror));
}
delete [] policko;
delete matice;

return chiskver;
}

/* The gradient of f, df = (df/dx, df/dy,...). */
void my_df (const gsl_vector *v, void *params, gsl_vector *df)
{
double *dp = (double *)params;
double alpha, beta, A, Q0, llambda,gamma, eta;

A       = gsl_vector_get(v, 0);
alpha   = gsl_vector_get(v, 1);
beta    = gsl_vector_get(v, 2);
llambda = gsl_vector_get(v, 3);
gamma   = gsl_vector_get(v, 4);
eta     = gsl_vector_get(v, 5);

double *parametry1 = new double[8];
parametry1[0]= alpha;
parametry1[1]= beta;
parametry1[2]= A;
parametry1[3]= llambda;
parametry1[4]= dp[0];
parametry1[5]= dp[1];
parametry1[6]= gamma;
parametry1[7]= eta;

// Tady zacina derivace te funkce podle alpha
gsl_function Falpha;
double resultalpha, abserralpha;
```

```
Falpha.function = &fcealpha;
Falpha.params = parametry1;
gsl_diff_central (&Falpha, alpha, &resultalpha, &abserralpha);
// Tady konci derivace te funkce podle alpha

// Tady zacina derivace te funkce podle beta
gsl_function Fbeta;
double resultbeta, abserrbeta;
Fbeta.function = &fcebeta;
Fbeta.params = parametry1;
gsl_diff_central (&Fbeta, beta, &resultbeta, &abserrbeta);
// Tady konci derivace te funkce podle beta

// Tady zacina derivace te funkce podle A
gsl_function FA;
double resultA, abserrA;
FA.function = &fceA;
FA.params = parametry1;
gsl_diff_central (&FA, A, &resultA, &abserrA);
// Tady konci derivace te funkce podle A


// Tady zacina derivace te funkce podle lambda
gsl_function Flambda;
double resultlambda, abserrlambda;
Flambda.function = &fcelambda;
Flambda.params = parametry1;
gsl_diff_central (&Flambda, llambda, &resultlambda, &abserrlambda);
// Tady konci derivace te funkce podle lambda

// Tady zacina derivace te funkce podle gamma
gsl_function Fgamma;
double resultgamma, abserrgamma;
Fgamma.function = &fcegamma;
Fgamma.params = parametry1;
gsl_diff_central (&Fgamma, gamma, &resultgamma, &abserrgamma);
// Tady konci derivace te funkce podle gamma

// Tady zacina derivace te funkce podle eta
gsl_function Feta;
double resulteta, abserreta;
Feta.function = &fceeta;
Feta.params = parametry1;
```

```
gsl_diff_central (&Feta, eta, &resulteta, &abserreta);
// Tady konci derivace te funkce podle eta

delete [] parametry1;

gsl_vector_set(df, 0,resultA);
gsl_vector_set(df, 1,resultalpha);
gsl_vector_set(df, 2,resultbeta);
gsl_vector_set(df, 3,resultlambda);
gsl_vector_set(df, 4,resultgamma);
gsl_vector_set(df, 5,resulteta);
}

/* Compute both f and df together. */
void
my_fdf (const gsl_vector *x,void *params,
double *f, gsl_vector *df)
{
*f = my_f(x, params);
my_df(x, params, df);
}

double overeni (double x)
{
//double q = 0.0003*pow(x,-0.5)*pow(1-x, 3)*(1+pow(x,1));
double q = (x*x)+((1-x)*(1-x));
return q;
}

/*****************************************************************/

double klavesa, novy, komplex, real;
int d,k,volba;

main()
{
while(1==1)
{
cout << endl << endl << endl << endl;
cout << "Vitejte v mem programu version 1.0" << endl ;
cout << "---------------------------------"<< endl << endl;
cout << "1. Vypocet integralu" << endl;
cout << "2. Testovani" << endl;
```

```
cout << "3. Minimalizace funkce" << endl;
cout << "4. Konec" << endl << endl;
cout << "Vase volba:";

//d=3;
cin >> d;

switch(d)
{
case 3:
{
double Kve;
cout << "FITOVANI FUNKCE" << endl;
cout << "---------------------------" << endl <<endl;
cout << "tak jedem" << endl;

/*

size_t iter = 0;
int status;
const gsl_multimin_fdfminimizer_type *T;
gsl_multimin_fdfminimizer *s;
 double * par = new double[2];
 par[0] = 1.5;
 par[1] = 2.5;
gsl_vector *x;
gsl_multimin_function_fdf my_func;
my_func.f = &my_f;
my_func.df = &my_df;
my_func.fdf = &my_fdf;
my_func.n = 6;
my_func.params = par;

// Starting point, x = (5,7)

 x = gsl_vector_alloc (6);
gsl_vector_set (x, 0, 0.090);
gsl_vector_set (x, 1, 1.668);
gsl_vector_set (x, 2, 1.104);
gsl_vector_set (x, 3, 0.026);
gsl_vector_set (x, 4, 1.136);
gsl_vector_set (x, 5, 1.645);
```

```
T = gsl_multimin_fdfminimizer_conjugate_fr;
s = gsl_multimin_fdfminimizer_alloc (T, 6);

gsl_multimin_fdfminimizer_set (s, &my_func, x, 0.01, 1e-2);

do
{
iter++;
status = gsl_multimin_fdfminimizer_iterate (s);
if (status)
break;
status = gsl_multimin_test_gradient (s->gradient, 1e-3);
if (status == GSL_SUCCESS)
  {
FILE *sou = fopen("t3.dta", "a");
if(!sou) return 1;
fprintf(sou, "Mimimum found at:\n");
fclose(sou);
printf ("Minimum found at:\n");
  }
 {
FILE *sou1 = fopen("t3.dta", "a");
if(!sou1) return 1;
fprintf(sou1, "%5d %.5f %.5f %.5f %.5f %.5f %.5f %10.5f\n",iter,
gsl_vector_get (s->x, 0),
gsl_vector_get (s->x, 1),
gsl_vector_get (s->x, 2),
gsl_vector_get (s->x, 3),
gsl_vector_get (s->x, 4),
gsl_vector_get (s->x, 5),
s->f);
fclose(sou1);

printf ("%5d %.5f %.5f %.5f %.5f %.5f %.5f %10.5f\n", iter,
gsl_vector_get (s->x, 0),
gsl_vector_get (s->x, 1),
gsl_vector_get (s->x, 2),
gsl_vector_get (s->x, 3),
gsl_vector_get (s->x, 4),
gsl_vector_get (s->x, 5),
s->f);
}
}
```

```
while (status == GSL_CONTINUE && iter < 100);
gsl_multimin_fdfminimizer_free (s);
gsl_vector_free (x);
*/

size_t np = 6;
double par[2] = {5, 2.5};
  const gsl_multimin_fminimizer_type *T =
  gsl_multimin_fminimizer_nmsimplex;
  gsl_multimin_fminimizer *s = NULL;
  gsl_vector *ss, *x;
  gsl_multimin_function minex_func;

  size_t iter = 0, i;
  int status;
  double size;

//Initial vertex size vector
    ss = gsl_vector_alloc (np);
//Set all step sizes to 1
    gsl_vector_set_all (ss, 1);
//Starting point
  x = gsl_vector_alloc (np);
  gsl_vector_set (x, 0, 0.097);
  gsl_vector_set (x, 1, 1.642);
  gsl_vector_set (x, 2, 1.1215);
  gsl_vector_set (x, 3, -0.084);
  gsl_vector_set (x, 4, 1.167);
  gsl_vector_set (x, 5, 0.757);
//Initialize method and iterate
  minex_func.f = &my_f;
  minex_func.n = np;
  minex_func.params = par;
  s = gsl_multimin_fminimizer_alloc (T, np);
  gsl_multimin_fminimizer_set (s, &minex_func, x, ss);
  do
    {
      iter++;
      status = gsl_multimin_fminimizer_iterate(s);
      if (status)
      break;
      size = gsl_multimin_fminimizer_size (s);
      status = gsl_multimin_test_size (size, 1e-2);
```

```
      if (status == GSL_SUCCESS)
      {
       FILE *sou1 = fopen("simplex3.dta", "a");
       if(!sou1) return 1;
       fprintf(sou1, "Konverguje k minimu v \n");
       fclose(sou1);
       printf ("converged to minimum at\n");
      }

FILE *sou2 = fopen("simplex3.dta", "a");
if(!sou2) return 1;
fprintf(sou2, "%5d ",iter);
fclose(sou2);
  printf ("%5d ", iter);
  for (i = 0; i < np; i++)
        {
          FILE *sou3 = fopen("simplex3.dta", "a");
          if(!sou3) return 1;
          fprintf(sou3,"%.3f ", gsl_vector_get (s->x, i));
          fclose(sou3);
          printf ("%.3f ", gsl_vector_get (s->x, i));
        }
FILE *sou4 = fopen("simplex3.dta", "a");
if(!sou4) return 1;
fprintf(sou4,  "& %7.3f size = %.3f\n", s->fval, size);
fclose(sou4);
    printf ("f() = %7.3f size = %.3f\n", s->fval, size);
    }
  while (status == GSL_CONTINUE && iter < 200);

  gsl_vector_free(x);
  gsl_vector_free(ss);
  gsl_multimin_fminimizer_free (s);

  return status;
break;
 }

case 2:
{
    cout << "ted prijde zkouska" << endl;
    gsl_integration_workspace * w
    = gsl_integration_workspace_alloc (1000);
```

```
  double result, error;
  double expected = -4.0;
  double * bezec = new double[2];
  bezec[0] = 2;
  bezec[1] = 3;

  //double alpha = 1.0;

  gsl_function ZK;
  ZK.function = &funkce;
  ZK.params = bezec;

  gsl_integration_qags (&ZK, 0, 1, 0, 1e-7, 1000,
                        w, &result, &error);

  printf ("result          = % .18f\n", result);
  printf ("exact result    = % .18f\n", expected);
  printf ("estimated error = % .18f\n", error);
  printf ("actual error    = % .18f\n", result - expected);
  printf ("intervals =  %d\n", w->size);

 /*
cout << "nulty moment vetvici funkce (mela by to byt nula):" <<endl << endl;
cout << "vysledek pro n=0:" << pomoc(0) << endl;
cout << "vysledek pro n=1:" << pomoc(1) <<endl << endl << endl;
cout << "vypocet nekolika momentu hledane distribucni funkce:" << endl;

double * kun = new double[8];
double parez;
for(int n =1;n<=5;n++)
{
for(int g=1;g <= 1000;g=g+20)
{
kun[0] =  0.85;
kun[1] =  3.94;
kun[2] =  8.6;
kun[3] =  5;
kun[4] =  g;
// kun[5] = 0.3;
kun[6] = n;
kun[7] = 0.337;
```

```
parez = moment(0,kun);

FILE *D = fopen("mprg2real.dta", "a");

if(!D) return 1;
fprintf(D, "%.5f %.5f \n",kun[4], parez);
fclose(D);
}
}
cout << "prvnich pet momentu distribucni funkce bylo
zapsano do souboru mprg2real.dta" << endl <<endl << endl;

gsl_integration_workspace * koza = gsl_integration_workspace_alloc(1000);
double *datel = new double [5];
double sojka1, sojka2;
datel[0] = 1;
datel[1] =1;
datel[2] = 0;
datel[3] =0;
datel[4] = 0;
datel[5] =0;

cout << endl << endl <<"overeni spravnosti inverzniho transformovani" << endl;
double * orel = new double[10];
orel[0] = 0.5;
orel[1] = 3;
orel[2] = 0.017320508;
orel[3] = 5;
orel[4] = 5.5;
//orel[5] = 0.3;
orel[6] = 0.1;
orel[7] = 0.3;
orel[8] = 1;
orel[9] = 1;

for(double h=0;h <= 1;h=h+0.02)
{
orel[5]=0.01+h;
double zz,zzz,uu,uuu;

gsl_integration_workspace * wwwwww = gsl_integration_workspace_alloc(1000);
gsl_function samuraj;
samuraj.function = &pointlike;
```

```
samuraj.params = orel;
gsl_integration_qags (&samuraj, 0, 80, 0, 1e-2, 1000,wwwwww, &uu, &uuu);
gsl_integration_workspace_free (wwwwww);

double uu1 = (uu*137*2*3.1415926)/(0.6665*log(5.5/5));

cout << orel[5] << " " <<((uu1)/3.1415926535) << " " << overeni(orel[5]) << endl;
}

cout << "vypocet prubehu integrandu pro ruzne polohy int. krivky:" << endl << endl;
double * sul = new double[10];
double kul,o;
cout << "Zadej vzdalenost integracni krivky:";
cin >> o;

for(double s=1;s <= 200;s=s+2)
{
sul[0] =  1;
sul[1] =  2;
sul[2] =  0.1;
sul[3] =  1.5;
sul[4] =  17;
sul[5] =  0.05;
sul[6] =  o;
sul[7] =  1;
sul[8] =  1;
sul[9] =  1;

kul = pointlike(s,sul);

cout << s << " " << kul << endl;

}
*/

cout <<"**************************************" << endl;
cout << "srovnani exp. dat a num. vyp.:" << endl;

typedef double radek[53];
radek* matice = new radek[5];
FILE *V = fopen("datafotonII.txt", "r");
for(int j=0; j<=52;j++)
    {
```

```
     fscanf(V, "%lf %lf %lf %lf %lf", &matice[0][j],
         &matice[1][j], &matice[2][j],&matice[3][j], &matice[4][j]);
    }
fclose (V);

double * l1 = new double[10];
for(int rr=0;rr<=52;rr++)
{
l1[0] = 1.717;
l1[1] = 1.5;
l1[2] = 0.035;
l1[3] = 1.9;
l1[4] = *(*(matice+1)+rr);
l1[5] = *(*(matice+0)+rr);
l1[6] = 2.5;
l1[7] = 0.239;
l1[8] = 0.642;
l1[9] = 1.938;

gsl_integration_workspace * A1 = gsl_integration_workspace_alloc(1000);
double resultA1, errorA1,resultA2, errorA2;
gsl_function funkceA1;
funkceA1.function = &f;
funkceA1.params = l1;
gsl_integration_qags (&funkceA1, 0, 30, 0, 1e-2, 1000,A1, &resultA1, &errorA1);
gsl_integration_workspace_free (A1);

gsl_integration_workspace * A2 = gsl_integration_workspace_alloc(1000);
gsl_function funkceA2;
funkceA2.function = &pointlike;
funkceA2.params = l1;
gsl_integration_qags (&funkceA2, 0, 80, 0, 1e-2, 1000,A2, &resultA2, &errorA2);
gsl_integration_workspace_free (A2);

 cout << *(*(matice+0)+rr) << " " << *(*(matice+1)+rr) << " "
<< *(*(matice+2)+rr) << " " << 137*((resultA1+resultA2)/3.1415926535) << endl;
}
//tady se bude pokracovat s vypoctem celyho integrandu...
break;
}

case 1:
{
```

```
while (1==1)
{
cout << endl << endl;
cout << "Vypocet integralu" << endl ;
cout << "-----------------"<< endl << endl;
cout << "1. Zobrazit aktualni nastaveni parametru" << endl;
cout << "2. Zmena parametru" << endl;
cout << "3. Vypocet integralu" << endl << endl;
cout << "Vase volba:";

cin >> k;

switch(k)
{
case 1:
{
double a1, a2, a3, a4, a5, a6, a7, a8;
FILE *V = fopen("para.dta", "r");
fscanf(V, "%lf %lf %lf %lf %lf %lf %lf %lf", &a1, &a2, &a3, &a4, &a5, &a6, &a7, &a8)

cout << endl;
cout << "Parametr A: " << a1 << endl;
cout << "Parametr alpha: " << a2 << endl;
cout << "Parametr beta: " << a3 << endl;
cout << "Hodnota Q0: " << a4 << endl;
cout << "Vzdalenost integracni krivky: " << a5 << endl;
cout << "Hodnota parametru lambda: " << a6 << endl;
cout << "Hodnota gamma: " << a7 << endl;
cout << "Hodnota eta:" << a8 << endl;

fclose (V);
break;
}
case 2:
{
double *ppolicko = new double [8];

cout << "Vlozte parametr A:" << endl;
cin >> ppolicko[0];

cout << "Vlozte parametr alpha:" << endl;
cin >> ppolicko[1];
```

```
cout << "Vlozte parametr beta:" << endl;
cin >> ppolicko[2];

cout << "Vlozte hodnotu Q0:" << endl;
cin >> ppolicko[3];

cout << "Zadejte vzdalenost integracni krivky:" << endl;
cin >> ppolicko[4];

cout << "Vlozte hodnotu parametru lambda:" << endl;
cin >> ppolicko[5];

cout << "Vlozte hodnotu parametru gamma:" << endl;
cin >> ppolicko[6];

cout << "Vlozte hodnotu parametru eta:" << endl;
cin >> ppolicko[7];

//double i;
FILE *R = fopen("para.dta", "w");

if(!R) return 1;

fprintf(R, "%f\n", *(ppolicko));
fprintf(R, "%f\n", *(ppolicko+1));
fprintf(R, "%f\n", *(ppolicko+2));
fprintf(R, "%f\n", *(ppolicko+3));
fprintf(R, "%f\n", *(ppolicko+4));
fprintf(R, "%f\n", *(ppolicko+5));
fprintf(R, "%f\n", *(ppolicko+6));
fprintf(R, "%f\n", *(ppolicko+7));

fclose(R);
cout << endl << "Parametry byly zapsany do souboru para.dta" << endl << endl;
break;
}

case 3:
{
  cout << endl;
  cout << "1. Vypocet nekolika grafu" << endl;
  cout << "2. Vypocet pri pevnem Q" << endl;
  cout << "3. Vypocet pro fitovani" << endl << endl;
```

```
  cout << "Vase volba:";

cin >> volba;
  switch(volba)
    {

    case 1:
    }
double *polex = new double [10];

cout << endl << "Vlozte hodnotu Q:" << endl;
cin >> polex[4];

double ii,jj;
double cc1, cc2, cc3, cc4, cc5, cc6, cc7, cc8;

polex[0] = 1.952;
polex[1] = 1.548;
polex[2] = 0.091;
polex[3] = 676767;
//policko[4] = 7;
//policko[5] = 0.3;
polex[6] = 2.5;
polex[7] = 0.0001;
polex[8] = 1.826;
polex[9] = 2.282;

for(jj=0.1;jj <= 1;jj=jj+0.02)
{
  polex[5]= jj;
double ju1, eu1;
double ju2, eu2;

gsl_integration_workspace * gamma1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcegamma1;
funkcegamma1.function = &f;
funkcegamma1.params = polex;
gsl_integration_qags (&funkcegamma1, 0, 40, 0, 1e-2, 1000,gamma1, &ju1, &eu1);
gsl_integration_workspace_free (gamma1);

gsl_integration_workspace * gamma2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcegamma2;
funkcegamma2.function = &pointlike;
```

```
funkcegamma2.params = polex;
gsl_integration_qags (&funkcegamma2, 0, 80, 0, 1e-2, 1000,gamma2, &ju2, &eu2);
gsl_integration_workspace_free (gamma2);

FILE *HH = fopen("fitfotvyp.dta", "a");
if(!HH) return 1;
fprintf(HH, "%.5f %.5f \n",polex[5], 137*(ju1+ju2)/3.1415926535);
fclose(HH);
}

cout << "Hotovo! Vysledek byl zapsan do souboru pokus.dta" << endl << endl;
delete [] polex;
return 0;
}

case 2:
      {
double *policko = new double [10];
double i;
double c1, c2, c3, c4, c5, c6, c7, c8;

FILE *A = fopen("para.dta", "r");
fscanf(A, "%lf %lf %lf %lf %lf %lf %lf %lf",
&c1, &c2, &c3, &c4, &c5, &c6, &c7, &c8);
fclose(A);
policko[0] = c2;
policko[1] = c3;
policko[2] = c1;
policko[3] = c4;
//policko[4] = 7;
//policko[5] = 0.3;
policko[6] = c5;
policko[7] = c6;
policko[8] = c7;
policko[9] = c8;

 double kuk=0;

for(double ii=50;ii <= 1000;ii=ii+190)
{
for(double jj=0.01;jj <= 1;jj=jj+0.02)
{
policko[4]=ii;
```

```
policko[5]=jj;

gsl_integration_workspace * www = gsl_integration_workspace_alloc(1000);
double result, error;
gsl_function tafunkce;
tafunkce.function = &f;
tafunkce.params = policko;
gsl_integration_qags (&tafunkce, 0, 25, 0, 1e-2, 1000,www, &result, &error);
gsl_integration_workspace_free (www);

FILE *H = fopen("datareal2.dta", "a");
if(!H) return 1;
fprintf(H, "%.5f %.5f \n",policko[5], policko[5]*result/3.1415926535);
fclose(H);

 kuk = kuk + policko[5]*policko[5]*result/3.1415926535;
}
}
cout << "Hotovo! Vysledek byl zapsan do souboru datareal2.dta" << endl << endl;
FILE *X = fopen("overeni.dta", "a");
if(!X) return 1;
fprintf(X, "%.5f \n",kuk);
fclose(X);
 delete [] policko;
return 0;
      }//konec case2 (vypocet integralu pri pevnem Q)

case 3:
{
  //tady to je pro Q=1.9************************************************
double * fit = new double[10];
fit[0] = 1.682;
fit[1] = 1.131;
fit[2] = 0.095;
fit[3] = 50;
fit[6] = 2.5;
fit[7] = 0.001;
fit[8] = 1.184;
fit[9] = 1.644;

double resultfit, errorfit;
double resultfit1,errorfit1;
```

```
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 1.9;
fit[5] = hh;

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit1;
funkcefit1.function = &f;
funkcefit1.params = fit;
gsl_integration_qags (&funkcefit1, 0, 200, 0, 1e-2,
 1000,fitt1, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt1);

gsl_integration_workspace * fitt2 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit2;
funkcefit2.function = &pointlike;
funkcefit2.params = fit;
gsl_integration_qags (&funkcefit2, 0, 250, 0, 1e-2,
1000,fitt2, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt2);

FILE *HHH = fopen("fitfotvyp1.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5],137*(resultfit+resultfit1)/3.142);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit1.dta";
  cout << endl;

//tady to je pro Q=3.7***********************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 3.7;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit3;
funkcefit3.function = &f;
funkcefit3.params = fit;
gsl_integration_qags (&funkcefit3, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);
```

```
gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit4;
funkcefit4.function = &pointlike;
funkcefit4.params = fit;
gsl_integration_qags (&funkcefit4, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp2.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.142);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit2.dta";
  cout << endl;

//tady to je pro Q=3.76*********************************************

for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 3.76;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit5;
funkcefit5.function = &f;
funkcefit5.params = fit;
gsl_integration_qags (&funkcefit5, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit6;
funkcefit6.function = &pointlike;
funkcefit6.params = fit;
gsl_integration_qags (&funkcefit6, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp3.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.142);
fclose(HHH);
```

```
}
 cout << "Vysledek byl zapsan do souboru fit3.dta";
 cout << endl;

  //tady to je pro Q=7.5*********************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 7.5;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit7;
funkcefit7.function = &f;
funkcefit7.params = fit;
gsl_integration_qags (&funkcefit7, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit8;
funkcefit8.function = &pointlike;
funkcefit8.params = fit;
gsl_integration_qags (&funkcefit8, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp4.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.142);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit4.dta";
 cout << endl;

  //tady to je pro Q=8.9*********************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 8.9;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit9;
funkcefit9.function = &f;
```

```
funkcefit9.params = fit;
gsl_integration_qags (&funkcefit9, 0, 200, 0, 1e-2,
 1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit10;
funkcefit10.function = &pointlike;
funkcefit10.params = fit;
gsl_integration_qags (&funkcefit10, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp5.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.142);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit5.dta";
 cout << endl;

  //tady to je pro Q=9************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 9;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit11;
funkcefit11.function = &f;
funkcefit11.params = fit;
gsl_integration_qags (&funkcefit11, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit12;
funkcefit12.function = &pointlike;
funkcefit12.params = fit;
gsl_integration_qags (&funkcefit12, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);
```

```
FILE *HHH = fopen("fitfotvyp6.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit6.dta";
 cout << endl;

  //tady to je pro Q=9.9*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 9.9;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit13;
funkcefit13.function = &f;
funkcefit13.params = fit;
gsl_integration_qags (&funkcefit13, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit14;
funkcefit14.function = &pointlike;
funkcefit14.params = fit;
gsl_integration_qags (&funkcefit14, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp7.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit7.dta";
 cout << endl;

  //tady to je pro Q=10.7*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 10.7;
fit[5] = hh;
```

```
gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit15;
funkcefit15.function = &f;
funkcefit15.params = fit;
gsl_integration_qags (&funkcefit15, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit16;
funkcefit16.function = &pointlike;
funkcefit16.params = fit;
gsl_integration_qags (&funkcefit16, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp8.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit8.dta";
 cout << endl;

  //tady to je pro Q=10.8*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 10.8;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit17;
funkcefit17.function = &f;
funkcefit17.params = fit;
gsl_integration_qags (&funkcefit17, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit18;
funkcefit18.function = &pointlike;
funkcefit18.params = fit;
```

```
gsl_integration_qags (&funkcefit18, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp9.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit9.dta";
 cout << endl;

  //tady to je pro Q=13.7************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 13.7;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit19;
funkcefit19.function = &f;
funkcefit19.params = fit;
gsl_integration_qags (&funkcefit19, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit20;
funkcefit20.function = &pointlike;
funkcefit20.params = fit;
gsl_integration_qags (&funkcefit20, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp10.dta", "a");

if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit10.dta";
 cout << endl;
```

```
  //tady to je pro Q=14.5**********************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 14.5;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit21;
funkcefit21.function = &f;
funkcefit21.params = fit;
gsl_integration_qags (&funkcefit21, 0, 200, 0, 1e-2,
 1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit22;
funkcefit22.function = &pointlike;
funkcefit22.params = fit;
gsl_integration_qags (&funkcefit22, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp11.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit11.dta";
 cout << endl;

  //tady to je pro Q=14.7**********************************************************

for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 14.7;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit23;
funkcefit23.function = &f;
funkcefit23.params = fit;
gsl_integration_qags (&funkcefit23, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
```

```
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit24;
funkcefit24.function = &pointlike;
funkcefit24.params = fit;
gsl_integration_qags (&funkcefit24, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp12.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit12.dta";
 cout << endl;

  //tady to je pro Q=15.3************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 15.3;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit25;
funkcefit25.function = &f;
funkcefit25.params = fit;
gsl_integration_qags (&funkcefit25, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit26;
funkcefit26.function = &pointlike;
funkcefit26.params = fit;
gsl_integration_qags (&funkcefit26, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp13.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
```

```
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit13.dta";
 cout << endl;

  //tady to je pro Q=17.5*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 17.5;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit27;
funkcefit27.function = &f;
funkcefit27.params = fit;
gsl_integration_qags (&funkcefit27, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit28;
funkcefit28.function = &pointlike;
funkcefit28.params = fit;
gsl_integration_qags (&funkcefit28, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp14.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit14.dta";
 cout << endl;

  //tady to je pro Q=17.8*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 17.8;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit29;
```

```
funkcefit29.function = &f;
funkcefit29.params = fit;
gsl_integration_qags (&funkcefit29, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit30;
funkcefit30.function = &pointlike;
funkcefit30.params = fit;
gsl_integration_qags (&funkcefit30, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp15.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
cout << "Vysledek byl zapsan do souboru fit15.dta";
cout << endl;

  //tady to je pro Q=20.7*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 20.7;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit31;
funkcefit31.function = &f;
funkcefit31.params = fit;
gsl_integration_qags (&funkcefit31, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit32;
funkcefit32.function = &pointlike;
funkcefit32.params = fit;
gsl_integration_qags (&funkcefit32, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);
```

```
FILE *HHH = fopen("fitfotvyp16.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit16.dta";
 cout << endl;

  //tady to je pro Q=23.1************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 23.1;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit33;
funkcefit33.function = &f;
funkcefit33.params = fit;
gsl_integration_qags (&funkcefit33, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit34;
funkcefit34.function = &pointlike;
funkcefit34.params = fit;
gsl_integration_qags (&funkcefit34, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp17.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit17.dta";
 cout << endl;

  //tady to je pro Q=120************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 120;
```

```
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit35;
funkcefit35.function = &f;
funkcefit35.params = fit;
gsl_integration_qags (&funkcefit35, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit36;
funkcefit36.function = &pointlike;
funkcefit36.params = fit;
gsl_integration_qags (&funkcefit36, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp18.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
cout << "Vysledek byl zapsan do souboru fit18.dta";
cout << endl;

  //tady to je pro Q=135*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 135;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit37;
funkcefit37.function = &f;
funkcefit37.params = fit;
gsl_integration_qags (&funkcefit37, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit38;
funkcefit38.function = &pointlike;
```

```
funkcefit38.params = fit;
gsl_integration_qags (&funkcefit38, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp19.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit19.dta";
 cout << endl;

  //tady to je pro Q=284*************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 284;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit39;
funkcefit39.function = &f;
funkcefit39.params = fit;
gsl_integration_qags (&funkcefit39, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit40;
funkcefit40.function = &pointlike;
funkcefit40.params = fit;
gsl_integration_qags (&funkcefit40, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp20.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit20.dta";
 cout << endl;
```

```
  //tady to je pro Q=30************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 30;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit41;
funkcefit41.function = &f;
funkcefit41.params = fit;
gsl_integration_qags (&funkcefit41, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
gsl_integration_workspace_free (fitt);


gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit42;
funkcefit42.function = &pointlike;
funkcefit42.params = fit;
gsl_integration_qags (&funkcefit42, 0, 250, 0, 1e-2,
1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp21.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit21.dta";
 cout << endl;

  //tady to je pro Q=59************************************************************
for(double hh=0.1;hh<=1;hh=hh+0.018)
{
fit[4] = 59;
fit[5] = hh;

gsl_integration_workspace * fitt = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit43;
funkcefit43.function = &f;
funkcefit43.params = fit;
gsl_integration_qags (&funkcefit43, 0, 200, 0, 1e-2,
1000,fitt, &resultfit, &errorfit);
```

```
gsl_integration_workspace_free (fitt);

gsl_integration_workspace * fitt1 = gsl_integration_workspace_alloc(1000);
gsl_function funkcefit44;
funkcefit44.function = &pointlike;
funkcefit44.params = fit;
gsl_integration_qags (&funkcefit44, 0, 250, 0, 1e-2,
 1000,fitt1, &resultfit1, &errorfit1);
gsl_integration_workspace_free (fitt1);

FILE *HHH = fopen("fitfotvyp22.dta", "a");
if(!HHH) return 1;
fprintf(HHH, "%.5f %.5f \n",fit[5], 137*(resultfit+resultfit1)/3.1415926535);
fclose(HHH);
}
 cout << "Vysledek byl zapsan do souboru fit22.dta";
 cout << endl;

 delete [] fit;
 return 0;
}
    }//konec switch pro zpusob vypoctu integralu (pri pevnem x or Q)
    } //konec case3 pro vypocet integralu
default:
break;
} //konec switch
} //konec while
break;
} //konec case

case 4:
// clrscr();
cout << endl << "Bye bye" <<endl <<endl;
return 0;
}
}
}
```

Literatura

[1] Chýla J.: *Quarks, partons a Quantum Chromodynamics*, Fyzikální ústav AV ČR, 2001

[2] Aitchinson I.G., Hoy A.J.: *Gauge Theories in Particle Physics*, Bristol, 1982

[3] Edin, A.: *Interplay between Soft and Hard Processes in Quantum Chromodynamics*, Acta Universitatis Upsaliensis, Uppsala, 1998

[4] Budnev V. et al.: *The two foton particle production mechanism*, Physics reports **15C** (1975), 181

[5] Friberg C.: *Aspects of QCD and the photon structure*, PhD Thesis, Lund University, 2000

[6] Krawcyk M.: *Survey of present data on photon strukture and resolved photon processes*, Physics Reports **345** (2001), 265

[7] Chýla J.: *When semantic turns to substance: reformulating QCD analysis of* $F_2^\gamma(x, Q^2)$, JHEP, 2000.

[8] Chýla J.. Taševský M.: *Interpreting virtual photon interactions in terms of parton distribution function*, DESY, 1998-1999.

[9] Chýla J., Rameš J.: *On Methods of Analyzing Scaling Violation in Deep Inelastic Scattering*, Fyzikální ústav AV ČR, 1985

[10] Blumlein J.: *Analytic Continuation of Mellin Transforms up to Two-Loop Order*, DESY 98-149, 2000

[11] Graudenz D. et al. : *The Mellin Transform Technique for the extraction of the Gluon Density*, DESY 95-107, 1995

[12] Kolář K.: *Evoluční rovnice kvantové chromodynamiky a jejich řešení*, diplomová práce, 2003

[13] Weinzierl S.: *Fast evolution of parton distributions*, Univerzita Parma, 2002

[14] Botje M.A.J.: *QCDNUM16: A fast QCD evolution program*, Amsterdam, 1998

[15] Botje M.A.J.: *A QCD analysis of HERA and fixed target structure function data*, Eur. Phys. J. C 14, 2000

[16] Bock R.K. et al.: *Formulae and Methods in Experimental Data Evaluation*, Vol 1., European Physical Society, 1984

[17] Bock R.K. et al.: *Formulae and Methods in Experimental Data Evaluation*, Vol 2., European Physical Society, 1984

[18] Bock R.K. et al.: *Formulae and Methods in Experimental Data Evaluation*, Vol 3., European Physical Society, 1984

[19] James F. : *Function Minimization*, CERN, 1972

[20] http://durpdg.dur.ac.uk/cgi-hepdata/struct3/CCFR/ZP53C51/xf3nucl