CESKÉ VYSOKÉ UCENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Petr Štefko

Numerické algoritmy pro polynomiální matice pro TI-89

Katedra rídicí techniky
Vedoucí diplomové práce: **Ing. Zdenek Hurák**

PRAHA 2004

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatne a použil jsem pouze podklady (literaturu, projekty, software atd.) uvedené v priloženém seznamu.

Nemám závažný duvod proti užití tohoto školního díla ve smyslu § 60 Zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících správem autorským a o zmene nekterých zákonu (autorský zákon).

V Praze dne **14. cervna 2004** …….

podpis

Abstrakt

V této diplomové práci jsem vytvoril balícek funkcí pro polynomiální matice pro grafický kalkulátor firmy Texas Instruments Inc. TI-89. Balícek obsahuje sadu funkcí pro výpocty s polynomiálními maticemi a knihovnu pomocných funkcí. Práce obsahuje popis kalkulátoru TI-89 a jeho hardwarových verzí, vývoj jeho operacního systému AMS (Advanced Mathematical Software) a jeho omezení, dále pak prehled dostupných rozšírení AMS (Kernel) a jejich parametry. Vysvetluje strukturu reprezentace císel, promenných a výrazu vkalkulátoru a jejich uložení v Zásobníku výrazu (Expression Stack). V záveru práce uvádím jednoduchý príklad využití vytvorených polynomiálních funkcí a diskutuji vhodnost využití polynomiální teorie na tomto kalkulátoru.

Abstract

In this diploma thesis I designed a package of functions for polynomial matrices for a calculator TI-89 produced by Texas Instruments Inc. This package includes a set of functions for computing with polynomial matrices and a library of auxiliary functions. This thesis also brings a description of calculator TI-89 and its hardware versions, history of operation system AMS (Advanced Mathematical Software) and its limitations, kernel programs which remove AMS limitations. Explain tagged representation of numbers, variables and expressions on the Expression Stack. At the end of this diploma thesis is simple example demonstrating use of the polynomial functions for a simple controller design via pole placement. This thesis is concluded by a discussion of suitability of the calculator for education in control theory.

Obsah

1 Úvod

1.1 Proc polynomiální matice?

Polynomy a polynomiální matice jsou prirozeným a mnohostranným nástrojem pro popis dynamických systému (popsaných diferenciálními rovnicemi) jako elektrické obvody, servomotory, ramena robotu, letadla a dokonce prenosové kanály mobilních sítí. Príklad polynomiální matice a její možný zápis viz. (Obrázek 1.1). Pro popis spojitých systému se používají polynomy promenných s, p a pro systémy diskrétní z, q, d .

Studium algebraických vlastností polynomiálních matic odkrývá mnoho vlastností dynamických systému odpovídajících fyzickému systému a to bez rešení techto diferenciálních rovnic. Systém G lze popsat pomocí maticového zlomku $G = N \cdot D^{-1}$. Napríklad zkoumáním pólu (korenu D) zjištujeme, jestli je systém stabilní.

Teorie polynomiálních matic muže být využita pro návrh regulátoru (LQG, H_2, H_{inf}, I1, atd.) a filtru pro zpracování signálu (Wieneruv filtr , Kalmanuv filtr, atd.). Klícovým teoretickým i numerickým nástrojem je pak lineární diofantická rovnice $AX \cdot BY = C$ a spektrální faktorizace $X \cdot X' = A$.

$$P(s) = \begin{bmatrix} 1+s & s^2 \\ 2s^3 & 1+2s+s^2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{P_0} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}}_{P_1} s + \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}}_{P_2} s^2 + \underbrace{\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}}_{P_3} s^3$$

Obrázek 1.1 Príklad polynomiální matice

1.2 Proc grafický kalkulátor TI-89?

Kalkulátor TI-89 je velice levný, mobilní a výkonný matematický pomocník, který umožnuje mimo jiné maticové operace, symbolické výpocty a lze pro nej tvorit vlastní programy ve strojovém kódu, vprogramovacím jazyce C nebo vobdobe programovacího jazyka BASIC. Toto jsou základní predpoklady pro možnou implementaci numerických algoritmu pro polynomiální matice.

1.3 Budoucí uživatelé

Tento balícek ocení studenti CVUT, kteří se venují teorii rízení. Budou moci v predmetech zabývajících se teorií rízení, jako jsou Robustní rízení, Moderní teorie rízení a Optimální rozhodování a rízení, rešit jednodušší problémy s použitím kalkulátoru TI-89, který je mezi nimi hojne využívaný . Pro složitejší problémy rízení budou muset použít jeden zkomplexnejších polynomiálních balícku, který jim poskytne jak více dostupných polynomiálních funkcí tak kratší dobu výpoctu. Tento balícek umožnuje vyzkoušení polynomiální teorie i tem, kteří nemají pristup k matematickým programum. K otestování tohoto balícku nebude uživatel vázán ani vlastnictvím kalkulátoru. Postací mu jedna z virtuálních verzí, které jsou bežne dostupné na internetu [2].

1.4 Podobné existující balícky

V soucasné dobe existuje nekolik nástroju pro práci spolynomiálními maticemi. Nejznámejší jsou :

- *Polynomial Toolbox for Matlab* [10] – komercní balicek vyvinutý a distribuovaný firmou Polyx Ltd. [11]
- *Scilab* [12] – balícek pro obdobu matematického produktu Matlab (volne šíritelný) vyvíjený lidmi z INRIA.
- *Maple* [13] – komercní matematický produkt obsahující CAS (Computer Algebra System), který podporuje polynomiální matice pouze okrajove.

Práce na vývoji balícku pro polynomiální matice stále pokracují. Na katedre Rídicí techniky Elektrotechnické fakulty CVUT jsou vyvíjeny nástroje pro další platformy.

- Java – *Polynomial Matrices in Java* [14] – Michal Padera
- C++ - *PolPack++* [15] – Leoš Halmo
- Mathematica – *Polynomiální balícek* [18] – Petr Kujan
- MuPAD – *Polmat* [16]– Petr Augusta

2 Popis TI-89

Grafický kalkulátor TI-89 firmy Texas Instruments patrí k nejvýkonnejším kalkulátorum na trhu. Vychází z výkonove stejného kalkulátoru TI-92, který navíc obsahuje klasickou qwerty klávesnici a vetší displej. TI-92 je vsoucasné dobe nahrazen novým produktem Voyage 200. Z konkurencních znacek kalkulátoru lze TI-89 srovnat s kalkulátorem HP48g vyrábeným firmou Hewlett Packard. Svými schopnostmi a relativne nízkou cenou 4500Kc (v roce 2004) je dobrou volbou stredoškolského a vysokoškolského studenta. Dá se ríci, že díky možnosti aktualizace operacního systému a stále vzrustající programátorské základny z rad studentu a zapálených programátoru, má i do budoucna co nabídnout.

2.1 Hardware

Tento kalkulátor v sobe kombinuje dostatecný výkon, velkou zobrazovací plochu i pro 3D zobrazení, relativne výkonný matematický software za prijatelnou cenu.

2.1.1 CPU

Grafický kalkulátor TI-89 vyrábí firma Texas Instruments. Srdcem tohoto kalkulátoru je 16 bitový CISC procesor Motorola MC68000. Vyrábí se od roku 1982. Osvedcil se ve starších pocítacích ATARI a AMIGA. V dnešní dobe stále nalézá uplatnení v zarízeních jako tiskárny, PDA nebo jako radic ve složitejších systémech. Celá poslední rada grafických kalkulátoru Texas Instruments je postavena na tomto procesoru. Je to výborný mikropocítac pro programy psané v programovacím jazyce C.

Specifikace CPU:

- 16 32bitových datových a adresních registru
- 16 MByte adresní prostor
- 56 instrukcí
- 2 MIPS pri 20 MHz

2.1.2 Kalkulátor

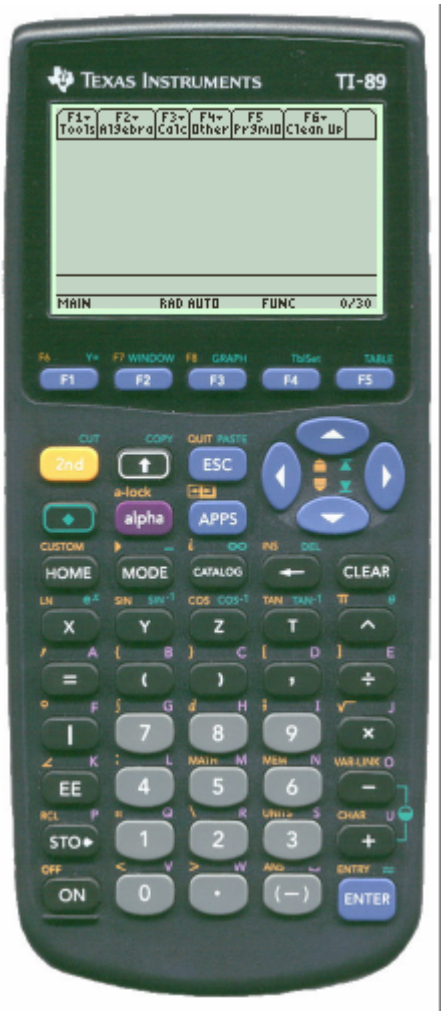
Kalkulátor se vyrábí od roku 1996 a od doby svého vzniku prošel nekolika zmenami. Asi rok po uvedení na trh se objevila Hardwarové verze 2, která zvýšila výpocetní výkon, ale pridala pár nesmyslných omezení. V roce 2003 kalkulátor prodelal zmenu designu, ale pouze pro evropský trh.

Specifikace TI-89:

- 256 kB RAM (188 kB prístupno pro uživatele)
- 2 MB pamet FLASH (pamet pro AMS - Advanced Mathematics Software a pro uživatelské programy - archiv)
- 384 kB archivu (702 kB pri AMS 2.03 a vyšším)
- display 100 x 160 bodu monochromatický, lze programem emulovat odstíny šedi
- výkonný matematický software
- symbolický výpocet
- pokrocilá lineární algebra (charakteristické hodnoty, vektory, rozklad matic)

- 3D grafika, obrysové nákresy a rotace ve trech smerech
- soustavy rovnic
- výpocet s fyzikálními jednotkami
- interaktivní numerický rešitel
- statistické funkce
- pripojení k dalšímu kalkulátoru nebo PC
- 4 x AAA baterie + záložní baterie lithium

Podrobná specifikace uvedena v [1].

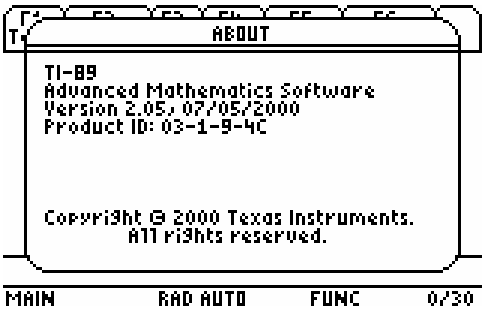


Obrázek 2.1 Kalkulátor TI-89

Zjištení hardwarové verze lze provést v úvodní obrazovce kalkulátoru stiskem klávesy F1 (Tools) a následne A (About).

2.1.3 Hardwarová verze 1

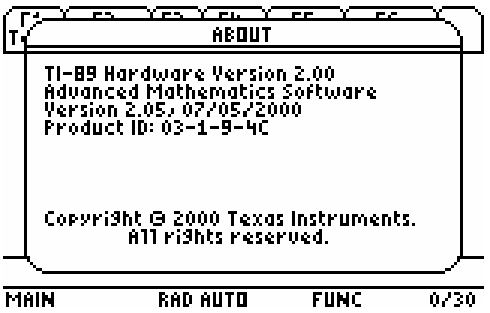
Puvodní hardwarová verze (HW1), která nemela omezení uživatelského programu. Zobrazování nebylo optimalizováno a provádelo se vždy pri obnove LCD, což zatežovalo procesor. Takt procesoru byl 9,4 MHz



Obrázek 2.2 Hardwarová verze 1

2.1.4 Hardwarová verze 2

Pozdejší verze (HW2) TI-89 je shodná s TI-92 Plus až na velikost pameti. Vykreslování video pameti probíhá jen v prípade zmeny. Tímto se snížila zátež procesoru, ale nastaly problémy s vykreslováním stupnu šedi (grayscale). Takt procesoru byl zvýšen na 13,1 MHz. Dále probehla zmena správy pameti, kdy lze pamet archivu využít pro promenné i programy. Bylo zavedeno striktní omezení velikosti ASM programu, které je možno odstranit použitím jednoho z rozšírení AMS. Byl pridán obvod reálného casu, jež lze využít s AMS 2.07 a vyšším.



Obrázek 2.3 Hardwarová verze 2

2.2 Operacní systém

Operacní systém nese název AMS (Advanced Mathematical Software). Zjištení verze AMS lze provést vúvodní obrazovce kalkulátoru stiskem klávesy F1 (Tools) a následne A (About). Behem svého vývoje prošel mnoha zmenami, souhrn zmen je v [4] . Strucný prehled zmen následuje.

AMS 1.00/1.01 - tyto verze jsou prvními a vyznacují se s vysokou kompatibilitou, neomezenou velikostí programu. Použití pouze pro hardwarovou verzi 1.

AMS 1.05 - první verze, která je kompatibilní s HW2, taktéž bez omezení velikosti programu, nestabilní.

AMS 2.01/2.02 - verze, které nebyly nikdy oficiálne uvolneny, velice podobné následující verzi 2.03.

AMS 2.03 – pridán limit 8 kB pro programy, který zpusobil mnoho problému a inicioval vznik rozšírení AMS zvaných Kernel. Pridána vlastnost obnovy dat z archivu po násilném restartu kalkulátoru.

AMS 2.04 – zvýšen limit programu na 24 kB, ale obsahoval chybu která zpusobovala havárii systému pri spouštení programu z pameti archivu.

AMS 2.05 – limit programu zustává na 24 kB, obnova pameti archivu zachována, plne kompatibilní s rozšírením AMS jménem Universal OS, ostatní rozšírení jsou v nekterých prípadech nestabilní. Pro kalkulátory HW2 nutno instalovat opravný program HW2Patch [3].

AMS 2.07 – první verze s grafickým menu, datem, hodinami, nekompatibilita s HW2Patch [3], uvolnena pouze pro Voyage 200.

AMS 2.08 – obdoba verze 2.07 ale pro TI-89 a TI-92 Plus.

AMS 2.09 – jen male zmeny a vylepšení oproti predcházející verzi.

2.3 Rozšírení AMS

Rozšírení AMS (Kernel) odstranuje urcitá omezení dané ruznými verzemi AMS. Jeho instalace je nutná pro korektní beh uživatelských programu. Po instalaci tento program beží na pozadí a dokud nedojde ke spuštení programu, který jej požaduje, tak se navenek neprojevuje. V dnešní dobe pomalu ztrácejí význam, protože jejich funkce jsou implementovány do vývojových prostredí. Je zcela na tvurci programu jestli se rozhodne vytvorit program, který vyžaduje rozšírení AMS (kernel-based , stub) nebo jeho funkce zahrne do svého programu a vytvorí nezávislý (no stub) program. Starší programy byly psány jmenovite pro jedno z rozšírení AMS, nejcasteji DoorsOs. Protože všechny rozšírení AMS provádejí priblizne stejné úkony, je velká pravdepodobnost, že program psaný pro starší rozšírení AMS bude schopen bezproblémove fungovat i na jiném.

2.3.1 DoorsOS

První z rozšírení AMS a na dlouho jediné. Byl k dispozici spolecne se souborovým manažerem a jako první rešil problémy s velikostí programu a s generováním stupnu šedi na monochromatickém displeji. Jeho vývoj ustal, poslední verze je kompatibilní s AMS 2.05.

2.3.2 Universal OS

Plne kompatibilní s HW1 a všemi verzemi AMS (1.00 – 2.09). Pro HW2 a AMS vyšší než 2.01 musí být instalován HW2Patch, který je ale kompatibilní pouze do verze AMS 2.05, proto je jeho využití omezené.

2.3.3 PreOS

Novejší rozšírení AMS, kompatibilní se všemi AMS (1.00 – 2.09) a s obemi hardwarovými verzemi. Specifikace a samotný program je dostupný v[5]. Aktuální verze je v0.67. Obsahuje v sobe funkce, které jsou nutné pro HW2, proto není nutné instalovat HW2Patch nebo rezidentní verzi H220xTSR.

2.3.4 KerNo

Nedá se presne nazvat rozšírením AMS. Je to velmi malý program, který odstranuje nekterá nepríjemná omezeni AMS. Umožnuje programum nezávislým na rozšírení AMS (no stub), aby plne využívaly možnosti kalkulátoru. Reší kritické chyby, které v jiných prípadech vedou k havárii kalkulátoru (chyba adresy, neplatná instrukce, delení nulou atd.). Umožnuje uživatelským funkcím být použity ve výrazech a být do sebe vnoreny atd. Je dostupný v [6].

2.3.5 PedroM

Toto rozšírení plne nahrazuje AMS, dá se ríci, že je to plnohodnotný operacní systém. Zajištuje funkce, které byly obsaženy v AMS 1.00. Tyto funkce byly kompletne prepsány a optimalizovány. Výsledkem je nový operacní systém o velikosti 192 kB. Tento OS posouvá hranice kalkulátoru mnohem dále. Uživatel má dostupnou pamet 228 kB RAM a 1900 KB archivu. Tato volná pamet vznikla odstranením systému algebraického výpoctu (Computer Algebra Systém - CAS). PedroM do budoucna nebude podporovat Flash aplikace. Obsahuje cásti kódu PreOs, a proto není nutno instalovat další rozšírení AMS. Aktuální verze je v0.8 je dostupná v [5] stejne jako PreOs.

2.4 Vnitrní reprezentace dat v TI-89

Urcitou zvláštností TI-89 je vnitrní interpretace matematických výrazu. Jak numerické tak symbolické výrazy jsou reprezentovány ve vnitrní znackové reprezentaci (tagged internal representation) nazývané také znacková forma (tokenized form). Tento zpusob zápisu explicitne vyjadruje hierarchickou závislost mezi operacemi a operandy. Operacní systém obsahuje funkce pro prevod textového zápisu do znackové formy (tokenizer) a zpet (detokenizer), pro zjednodušení (simplifier) a vycíslení znackové formy. Každá promenná, císlo, text, operace nebo funkce má svoje oznacení (tag), které ji specifikuje. Zretezením techto výrazu vznikají retezce, které se ukládají na zásobník výrazu (expression stack). Tokenizer vytvárí formu zvanou spojitá znacková polská reprezentace (contiguous tokenized polish representation). Má dve základní výhody, a to efektivitu co do obsazeného místa a informace o hierarchické strukture bez dalších informací, není nutné vnitrní závorkování.

2.4.1 Znacková polská reprezentace

Znacková polská reprezentace umistuje operandy hloubeji než operátory. Pak následný zápis souctu dvou promenných vypadá takto.

Klasický zápis výrazu	Znaková polská reprezentace
A + B	+ (vyšší adresa)
	B
	A (nižší adresa)

Je duležité poznamenat, že výraz vždy zacíná na vyšší adrese a pri vyhodnocování narazíme nejdríve na operátory a pak až na operandy. Tato metoda je odlišná od známé zpetné polské formy (reverse polish form), která narazí nejprve na operandy následované operátory. Zde je pár dalších príkladu.

Výraz	Polská reprezentace
A * B + C	A B * C +
A * (B + C)	A B C + *
A * B + C / D	A B * C D / +
A * (B + C) / D	A B C + * D /
X * Y^N –Z	Z X N Y ^ * -

Je duležité si uvedomit, že tokenizer vytvárí znackovou polskou formu ctením textového výrazu zleva doprava, ale následne vyhodnocuje zprava doleva.

2.4.2 Reprezentace čísel

Operacní systém podporuje dva oddelené systémy čísel. Racionální číselný systém, pracující s oznaceným celocíselným typem (integer) nebo oznaceným zlomkem dvou celocíselných typu. Rozsah možných čísel je $(-10^{614},+10^{614})$, což je vetší než rozsah reprezentovaný celocíselnými typy v implementaci programovacího jazyka C v TIGCC [9]. Číselný systém s pohyblivou radovou cárkou (floating-point), který používá binárne kódované dekadické čísla s pohyblivou radovou cárkou (BCD). Zde na rozdíl od C, které podporuje BCD16 (16 platných číslic a exponent), podporuje kalkulátor pouze BCD14. Proto dochází k zaokrouhlení. Rozdíl mezi temito systémy reprezentace čísel je, že racionální systém je už zdefinice presný, kdežto čísla s pohyblivou radovou cárkou jsou vždy pouhé aproximace daného čísla. Avšak výpocet v celocíselném tvaru je casove nárocnejší, protože velikost jejich reprezentace není pevná, s rostoucí velikostí reprezentace roste i vyhodnocovaný výraz a není možné stanovit indexy, kde jednotlivé argumenty zacínají jinak než procházením seznamu a zjištováním délky každého celocíselného argumentu. Reprezentace jednotlivých čísel na zásobníku výrazu pak vypadá následovne.

Hodnota	Znacková forma
X (integer)	nižší B..vyšší B pocet B xxxINT_TAG
0	0 POSINT_TAG
-5	5 1 NEGINT_TAG
256	0 1 POSINT_TAG
65538	2 0 1 3 POSINT_TAG
1 / 2	2 1 1 1 POSFRAC_TAG
-5/256	0 1 2 5 1 NEGFRAC_TAG
X (real)	znaménko + exponent (2B) 14 platných číslic (7B) FLOAT_TAG
?	0x40 0x00 0x31 0x41 0x59 0x26 0x53 0x58 0x98 FLOAT_TAG

2.4.3 Externí a interní znacková forma

Systém má dve znackové polské formy. Externí, kterou vytvárí tokenizer. Touto formou lze zapsat jeden a tentýž výraz více zpusoby. Jako napríklad výraz a/b a výraz $a \cdot b^{-1}$. Pak tento výraz zpracuje funkce pro zjednodušení, která vytvorí z každého výrazu nejkratší možný retezec a tím rozpozná shodnost výrazu. Takto zpracované výrazy se nacházejí v Interní polské forme. Pri této operaci dochází k eliminaci nekterých znacek (tag) a jejich náhrade za znacky vyskytující se v Interní polské forme.

2.4.4 Zásobník výrazu

Pro práci se retezenými výrazy používá Operacní systém Zásobník výrazu (Expression Stack - ES). Jeho maximální velikost je omezená na 15 kB a umístnení urceno speciálními registry (top_estack, bottom_estack). Organizace ES viz. Obrázek 3.3 a Obrázek 3.4. Operacní systém obsahuje mnoho definovaných funkcí pro práci s ES. Tyto funkce se dají rozdelit do nekolika skupin podle operací, které provádejí.

- push_(..) - vkládá argument, nebo výsledek operace s argumentem na vrchol ES.
- replace_top_(..) – zamení argumenty na vrcholu ES výsledkem dané operace.
- is_(..) – užívá se k získání informace o argumentu, neprovádí zmeny.
- index_(..) – provádí pohyb po ES, vrací indexy jednotlivých výrazu na ES.

3 Implementace

Je nekolik možností vytvorení vlastního programu pro kalkulátor TI-89. První z možností je vytvorit program v asembleru, tím je dosaženo nejlepšího výkonu za cenu vetších nároku na programátora.

Zcela opacným prístupem je použít obdobu jazyka TI-BASIC, jehož omezený prekladac se nachází i v samotném kalkulátoru. Tento prístup nevyžaduje k tvorbe programu nic víc než samotný kalkulátor a dovoluje editaci programu kdykoli. Protože preklad takto vytvoreného programu probíhá pri jeho spuštení, jsou tyto programy velmi pomalé. Programátorovy možnosti jsou dále omezené pouze na funkce implementované v kalkulátoru. Tyto dve nevýhody nedovolují využít TI-BASIC k rešení složitejších matematických problému. Ukázka kódu jednoduché funkce viz. Obrázek 3.1.

```
prand(row,col,varr,order)
Func
©deg,row,col,varr
Local i,j,k,mat
newMat(row,col)—>mat
For i,1,row
 For j,1,col
  For k,0,order
   randNorm(0,1)*varr^k+m[i,j]—>mat[i,j]
  EndFor
 EndFor
EndFor
mat
EndFunc
```

Obrázek 3.1 Ukázka kódu v TI-BASIC

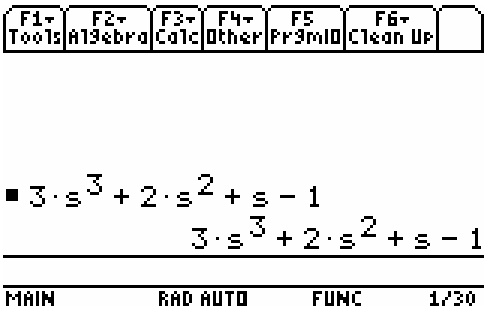
Tretí možností je program vytvorit v programovacím jazyce C, který poskytuje programátorský komfort, srovnatelný výkon kódu s kódem programovaném v asembleru a v neposlední rade dostatecnou volnost ve výužívání jeho funkcí. Tato možnost se jeví jako nejschudnejší.

3.2 Programátorské prostredí

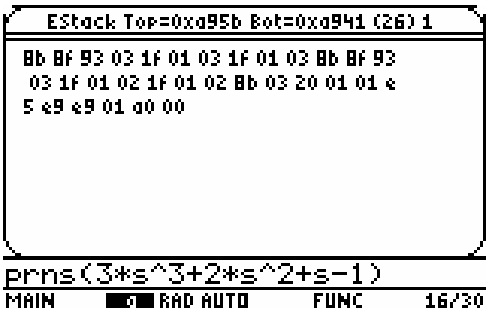
Dostupné programátorské prostredí jsou pouze TI Flash Studio [6], vytvoreno prímo výrobcem kalkulátoru firmou Texas Instruments v soucasné dobe verze v1.1 a TIGCC IDE [9] programovaným skupinou TIGCC Team dostupné ve verzi v0.95 Beta 8. Oba tyto programátorská prostredí jsou velice mladá a prochází vývojem. Po krátkém testování obou prostredí jsem se rozhodl pro TIGCC IDE. Mé rozhodnutí bylo ovlivneno informacemi získanými v internetovém fóru TIGCC Programming [17] , kde podobnou otázku rešilo více zacínajících programátoru TI-89.

Kámen úrazu u tohoto prostredí je ješte nedokoncená dokumentace, vetšina funkcí dostupných behem programování nebyla ješte zdokumentována, a proto bylo nutno kombinovat dokumentace obou vývojových prostredí. Velkou pomocí byla dokumentace vydaná pro TI Flash Studio kterou najdete na [8] a poslední možností, kde nalézt odpoved bylo internetové fórum TIGCC Programming [17].

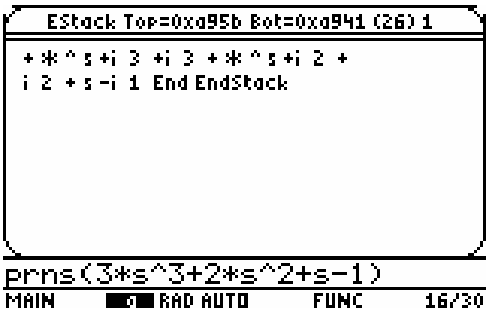
Další programátorskou nesnází bylo, že ješte nebyl vytvoren debugger. Proto hledání chyb v kódu bylo zdlouhavé. Byl jsem nucen vytvorit vlastní funkce pro výpis pameti a transformaci techto informací do srozumitelnejší podoby. Ladení pak probíhalo vkládáním techto funkcí do kritických míst programu a kontrola údaju ve sledovaných promenných. Príklad dvou z techto funkcí pro výpis obsahu zásobníku výrazu viz. Obrázek 3.3 a Obrázek 3.4.



Obrázek 3.2 Reprezentace polynomu – klasický výpis



Obrázek 3.3 Príklad funkce print_estack



Obrázek 3.4 Príklad funkce print_estack2

3.3 Knihovna plib.dll

Z pocátku byly polynomiální funkce programovány nezávisle na jakékoli knihovne. Problém byl s jejich velikostí, kdy každá z nich zabírala více než 10 kB pameti kalkulátoru. Proto byl prehodnocen prístup a hledaly se jiná rešení. Protože prvotní verze vývojového prostredí neobsahovaly podporu dynamických knihoven, byl vytvoren archiv funkcí, ze kterého se pripojovaly ke kódu polynomiální funkce pouze ty, které byly nutné. Tímto prístupem se zachovala nezávislost jednotlivých funkcí, ale úspora pameti nebyla stále dostatecná, funkce stále zabíraly 6 – 8 kB . S príchodem novejší verze vývojového prostredí TIGCC, která podporovala dynamické knihovny byly všechny funkce spolecné pro funkce polynomiální presunuty do dynamické knihovny plib.dll. Bylo docíleno snížení velikosti techto funkcí na 2 – 4 kB výmenou za nezávislost jednotlivých polynomiálních funkcí.

3.3.1 Struktura polynomu a polynomiální matice

Bylo nutno zvolit strukturu polynomiální matice na úrovni kalkulátoru. Díky tomu, že TI-89 dovoluje symbolické výpocty, lze vytvorit matici s prvky jako polynomy. Nebyl tedy duvod vytváret vlastní datový typ na úrovni kalkulátoru. Výhodou tohoto rešení je, že na takovéto objekty lze používat stejné operace, jako na matice s císelnými prvky. Tudíž lze s nimi dále pracovat a využívat funkce implementované v kalkulátoru.

Pro efektivnejší práci v programovaných funkcích byly vytvoreny vlastní struktury pro vnitrní uchování polynomu a polynomiální matici viz. Obrázek 3.5. Vždy pri predávání parametru polynomiálním funkcím se volají funkce pro prevod polynomu a polynomiálních matic do techto vnitrních struktur. Výpocty pak probíhají až na výjimky ve vnitrních strukturách a výsledek se opet prevádí na standardní tvar matice s polynomiálními prvky.

```
typedef struct{
 unsigned char cVar;   // promenná v polynomu
 unsigned int nSize;   // pocet koeficientu
 int nTdeg;   // dolní stupen polynomu
 int nLdeg;   // horní stupen polynomu
 double * pfVector;    // pole koeficientu polynomu
} TPolynom;
typedef TPolynom * PTPolynom;

typedef struct{
    unsigned char cVar;
    unsigned int nRow;      // pocet rádku matice
    unsigned int nCol;      // pocet sloupcu matice
    unsigned int nPolSize;  // pocet koeficientu polynomu
    int nTdeg;   // dolní stupen pol. matice
    int nLdeg;   // horní stupen pol. matice
    double * pfMatrix;    // pole koeficientu pol. matice
    } TPolynomial_matrix;
typedef TPolynomial_matrix * PTPolynomial_matrix;
```

Obrázek 3.5 Vnitrní struktury:

3.3.2 Funkce knihovny pro práci s polynomy

Knihovna plib.dll obsahuje více než 20 funkcí, které lze volat z vytvorených polynomiálních funkcí.

```
PTPolynom create_polynom(
unsigned char cVar,
int nTdeg,
int nLdeg
)
```

- alokuje pamet pro strukturu TPolynom, naplní vnitrní promenné charakterizující vlastnosti polynomu, vynuluje koeficienty polynomu a vrací ukazatel na tuto strukturu.

```
void delete_polynom(
PTPolynom pPol
)
```

- zajištuje dealokaci pameti struktury TPolynom.

```
unsigned char variable_of_polynom(
ESI pcEStack
)
```

- kontroluje zda výraz zadaný ve znackové forme je polynom jediné promenné a tuto promennou vrací.

```
void deg_of_polynom(
ESI pcEStack,
unsigned char cVar,
int * nT,
int * nL
)
```

- zjištuje dolní a horní stupen polynomu.

```
void parse_polynom(
ESI pcEStack,
PTPolynom pPol
)
```

- získává koeficienty polynomu ze znackové formy a ukládá je do predem alokované struktury TPolynom.

```
PTPolynom read_polynom(
ESI pcEStack
)
```

- voláním ruzných funkcí zajištuje alokaci, kontrolu a naplnení struktury TPolynom hodnotami získanými ze zadaného polynomu ve znackové forme.

```
PTPolynom zeroing_polynom(
PTPolynom pPol,
float fTol
)
```

- nuluje koeficienty polynomu, jejichž absolutní hodnota je menší než zadaná tolerance.

```
void push_polynom(
PTPolynom pPol
)
```

- z obsahu struktury TPolynom vytvorí výraz ve znackové forme a vloží ho na vrchol zásobníku výrazu.

```
PTPolynomial_matrix create_polynomial_matrix(
unsigned int nRow,
unsigned int nCol,
unsigned char cVar,
int nTdeg,
int nLdeg
)
```

- alokuje pamet pro strukturu TPolynomial_matrix, naplní vnitrní promenné charakterizující vlastnosti polynomiální matice, vynuluje koeficienty polynomiální matice a vrací ukazatel na tuto strukturu.

```
void delete_polynomial_matrix(
PTPolynomial_matrix pMat
)
```

- zajištuje dealokaci pameti struktury TPolynomial_matrix.

```
void get_param_polynomial_matrix(
ESI pcEStack,
unsigned int * pnRow,
unsigned int * pnCol,
unsigned char * pcVar,
int * pnTdeg,
int * pnLdeg
)
```

- provádí kontrolu a zjištuje parametry polynomiální matice zadané znackovou formou.

```
PTPolynomial_matrix read_polynomial_matrix(
ESI pcEStack
)
```

- voláním ruzných funkcí zajištuje alokaci, kontrolu a naplnení struktury TPolynomial_matrix hodnotami získanými ze zadané polynomiální matice ve znackové forme.

```
PTPolynomial_matrix zeroing_polynomial_matrix(
PTPolynomial_matrix pM,
float fTol
)
```

- nuluje koeficienty polynomiální matice, jejichž absolutní hodnota je menší než zadaná tolerance.

```
unsigned int is_matrix_zero(
PTPolynomial_matrix pM
)
```

- zjištuje jestli všechny koeficienty polynomiální matice jsou nulové.

```
PTPolynomial_matrix get_pm_from_TIvar(
unsigned char const * pcVarName
)
```

- obdoba funkce read_polynomial_matrix, ale zdrojem znackové formy polynomiální matice je globální promenná typu MAT. Používá se pro získání výsledku pri volání funkcí kalkulátoru, které vracejí více než jeden parametr.

```
PTPolynomial_matrix get_copy_pm(
PTPolynomial_matrix pM
)
```

- vytvorí kopii polynomiální matice.

```
PTPolynomial_matrix get_pm_from_cm(
PTPolynomial_matrix pM,
unsigned int nDeg,
unsigned char cVar
)
```

- transformace matici, která obsahuje pouze císla, která odpovídají koeficientum polynomiální matice, na polynomiální.

```
PTPolynomial_matrix get_augment_pm(
PTPolynomial_matrix pM1,
PTPolynomial_matrix pM2,
unsigned char const cStyle
)
```

- spojení dvou matic do jedné. Umožnuje jak sloupcové([A,B]) tak rádkové([A;B]) spojení.

```
void push_polynomial_matrix(
PTPolynomial_matrix pMat
)
```

- z obsahu struktury TPolynomial_matrix vytvorí výraz ve znackové forme a vloží ho na vrchol zásobníku výrazu.

```
void push_polynomial_matrix_ext(
PTPolynomial_matrix pMat,
unsigned int r1,
unsigned int r2,
unsigned int c1,
unsigned int c2,
unsigned int transpose
)
```

- rozšírení predchozí funkce, znackovou formu tvorí jen z cásti polynomiální matice a dovoluje vložení její transponované podoby.

```
void expand_arg(
unsigned int n
)
```

- provádí roznásobení polynomu v zásobníku výrazu, které jsou ve znackové forme reprezentovány jako soucin souctu.

```
float get_tolerance(
void
)
```

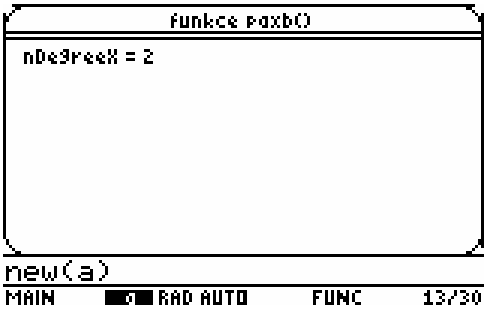
- vrací hodnotu z globální promenné kalkulátoru používanou jako globální tolerance.

3.3.3 Funkce knihovny pro ladení polynomiálních funkcí

Tyto funkce jsou dostupné pouze tehdy, pokud je knihovna zkompilována v režimu ladení.

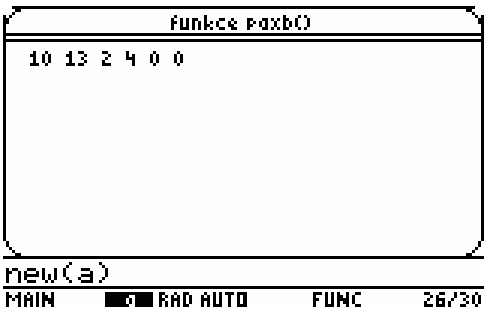
```
void print_debug_int(
unsigned char const * cMsg,
unsigned char const * cWhat,
long a
)
```

- zobrazí na displeji okno s textem popisující promennou typu integer a její hodnotu.



Obrázek 3.6 print_debug_int

```
void print_debug_6int(
unsigned char const * cMsg,
long a,
long b,
long c,
long d,
long e,
long f
)
```

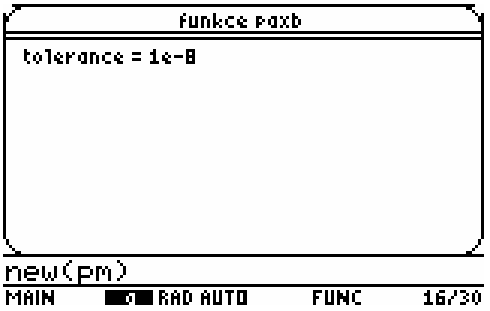


Obrázek 3.7 print_debug_6int

- zobrazí na displeji okno s textem a s hodnotami 6ti promenných typu integer. Obdoba predchozí funkce využívaná pri ladení cyklu.

```
void print_debug_double(
unsigned char const * cMsg,
unsigned char const * cWhat,
double a
)
```

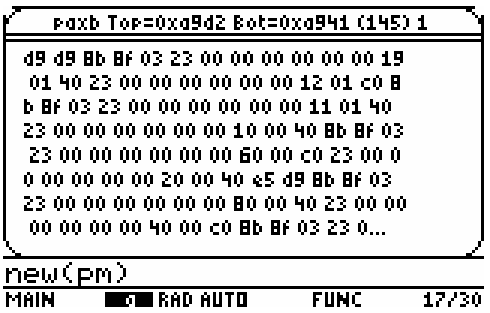
- zobrazí na displeji okno s textem popisující promennou typu double a její hodnotu.



Obrázek 3.8 print_debug_double

```
void print_estack(
unsigned char const * cMsg,
ESI pcEStack
)
```

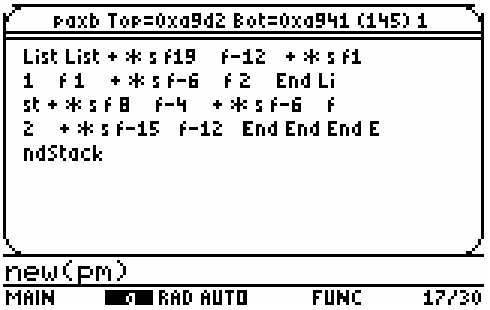
- zobrazí okno s parametry zásobníku výrazu a vypíše jeho obsah v číselné forme.



Obrázek 3.9 print_estack

```
void print_estack2(
unsigned char const * cMsg,
ESI pcEStack
)
```

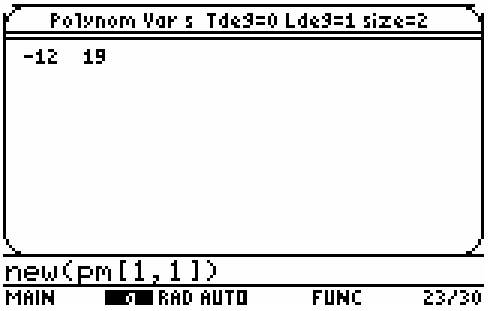
- zobrazí okno s parametry zásobníku výrazu a vypíše jeho obsah v transformované podobe pro lepší citelnost .



Obrázek 3.10 print_estack2

```
void print_debug_pol(
unsigned char const * cMsg,
PTPolynom pPol
)
```

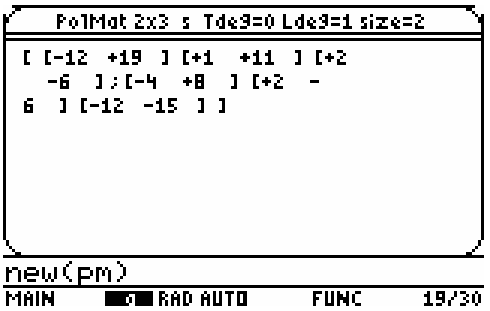
- zobrazí okno s parametry polynomu a vypíše jeho koeficienty.



Obrázek 3.11 print_debug_pol

```
void print_debug_mat(
unsigned char const * cMsg,
PTPolynomial_matrix pMat
)
```

- zobrazí okno sparametry polynomiální matice a vypíše její koeficienty.



Obrázek 3.12 print_debug_mat

```
void store_to_TIvar(
PTPolynomial_matrix pM,
unsigned char const * pcVarName
)
```

- vytvorí globální promennou kalkulátoru a uloží do ní znackovou formu polynomiální matice.

3.4 Polynomiální funkce

Tyto funkce nám dovolují provádet výpocty s polynomy a polynomiálními maticemi na kalkulátoru TI-89. Všechny funkce jsou závislé na dynamické knihovne plib.dll a nekterá funkce ke své cinnosti vyžadují prítomnost i dalších polynomiálních funkcí. Funkce pinit() provede inicializaci kalkulátoru a zkontroluje prítomnost polynomiálních funkcí a na chybející vás upozorní. Ukázka jednoduché polynomiální funkce se nachází v príloze I. Následuje seznam vytvorených polynomiálních funkcí a jejich detailní popis.

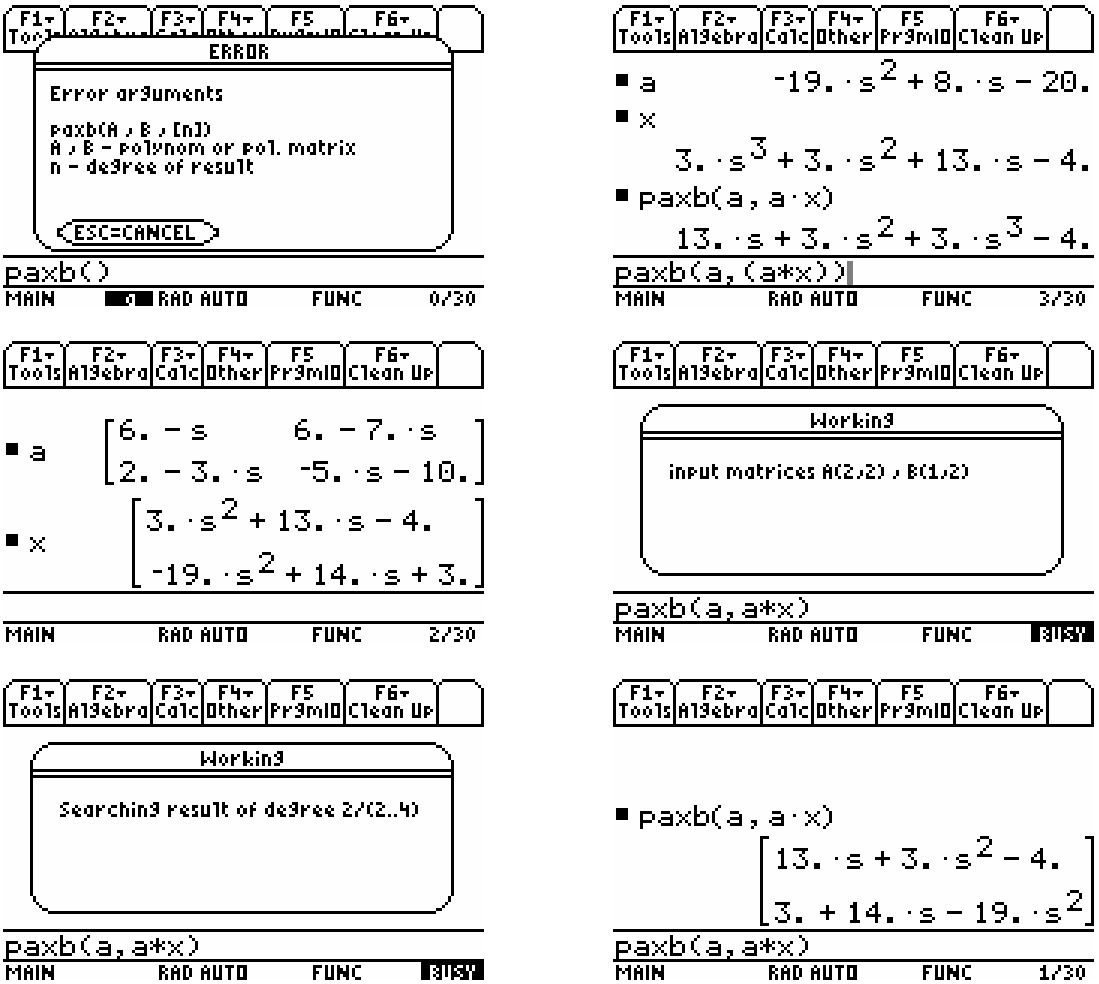
Seznam:

- paxb - reší rovnici $A \cdot X = B$
- paxbyc - reší rovnici $A \cdot X + B \cdot Y = C$
- pcoef - vrací koeficienty u zadané mocniny promenné .
- pctrans - vrací konjugovanou transpozici polynomiální matice.
- pdeg - vrací stupne polynomu nebo pol. matice.
- pinit - inicializuje polynomiální toolbox.
- piscred - zjištuje, zda-li je pol. mat sloupcove redukovatelná.
- pisrred - zjištuje, zda-li je pol. mat rádkove redukovatelná.
- pisstab - zjištuje, zda-li je polynom nebo pol. mat stabilní.
- plcoef - vrací koeficienty u horní mocniny polynomu nebo pol. matice.
- prand - generuje náhodný polynom nebo pol. matici.
- prank - vrací hodnost matice.
- proots - vrací koreny polynomu nebo pol. matice.
- psetzer - nastavuje hodnotu globální tolerance.
- psylv - vrací Sylvestrovu matici odpovídající polynomu nebo pol. matici.
- ptcoef - vrací koeficienty u dolní mocniny polynomu nebo pol. matice.
- ptrans - vrací transpozici matice, nebere ohled na typ prvku matice.
- pxab - reší rovnici $X \cdot A = B$
- pxaybc - reší rovnici $X \cdot A + Y \cdot B = C$
- pzer - provádí nulování koeficientu, jejichž abs. hodnota je menší než tolerance.

3.4.1 paxb

Funkce reší rovnici $A \cdot X = B$ pro zadané hodnoty A , B a vrací hodnotu X .
Syntaxe:

```
paxb(A,B,[n])
A,B  …  polynom nebo polynomiální matice
n  …  požadovaný stupen X, není-li zadán, vrací nejnižší
      možný stupen X

príklad:
        paxb(A,B,2)
        paxb(A,B)
```

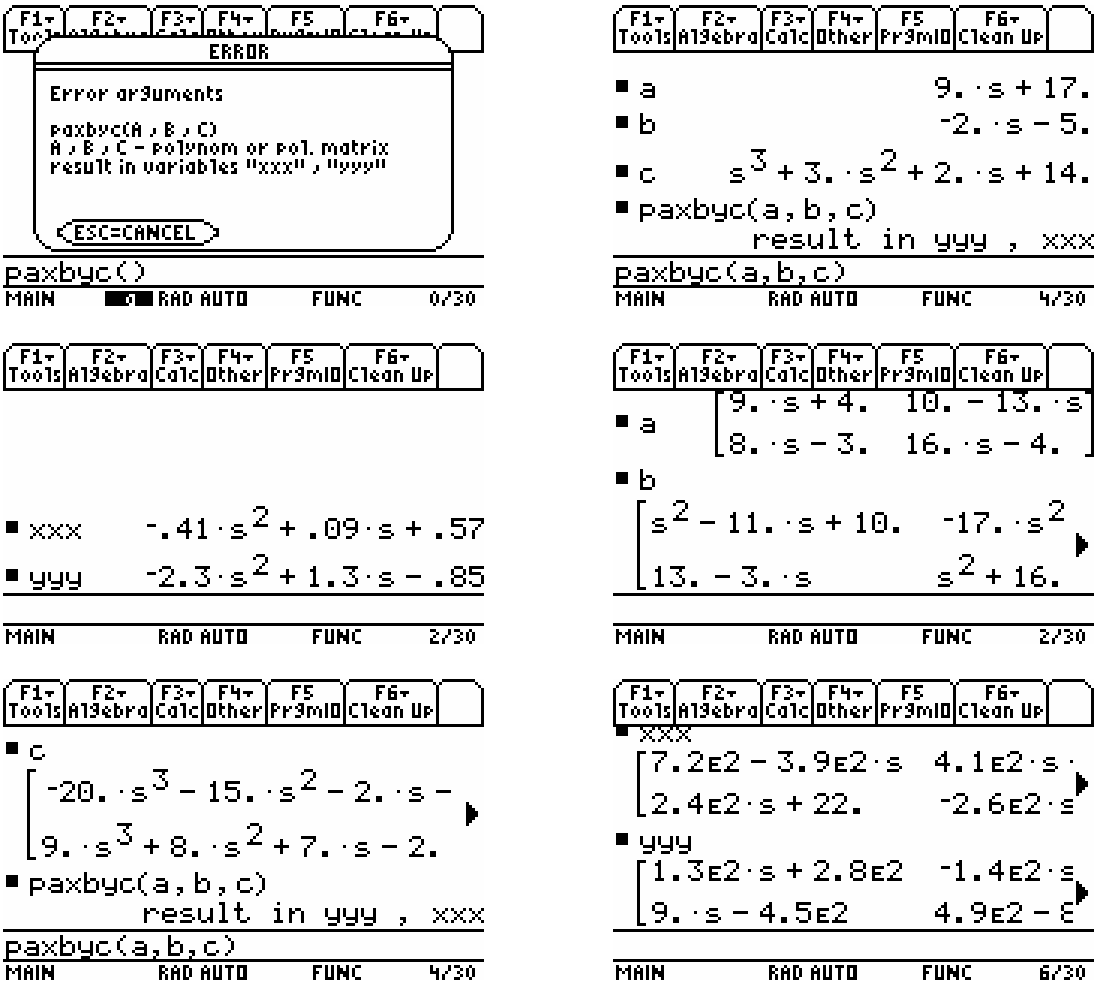


Obrázek 3.13 Príklad funkce paxb

3.4.2 paxbyc

Funkce reší rovnici $A \cdot X + B \cdot Y = C$ pro zadané hodnoty A , B , C a vrací hodnoty X a Y .
Syntaxe:

```
pxaybc(A,B,C)
A,B,C  …  polynom nebo polynomiální matice

príklad:
        pxaybc(A,B,C)
```

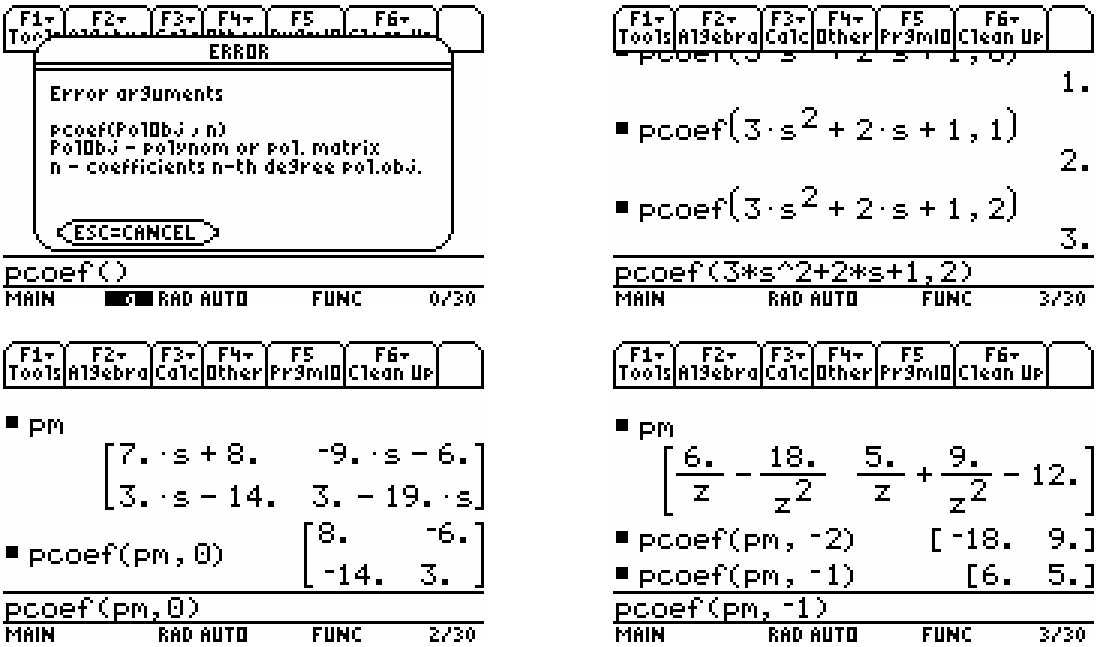


Obrázek 3.14 Príklad funkce paxbyc

3.4.3 pcoef

Funkce vrací koeficienty u zadané mocniny promenné polynomu nebo polynomiální matice.
Syntaxe:

```
pcoef(polObj,n)
polObj  …  polynom nebo polynomiální matice
n  …  stupen promenné požadovaného koeficientu

príklad:
        pcoef(2s+1,1)
```



Obrázek 3.15 Príklad funkce pcoef

3.4.4 pctrans

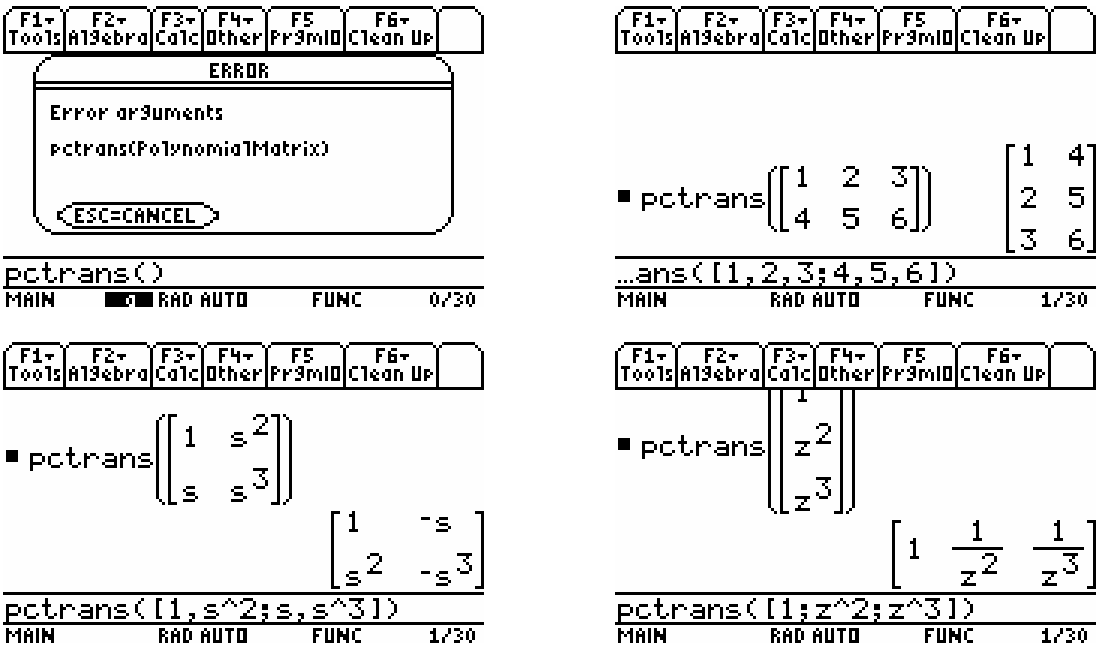
Provede komplexne sdruženou transpozici zadané matice.

Pro polynomiální matici $A(s)$ provede $A_{CT} = (A(-s))^T$.

Pro polynomiální matici $A(z)$ provede $A_{CT} = \left(A\left(\frac{1}{z}\right)\right)^T$.

Syntaxe:

```
pctrans(polMat)
polMat  …  polynomiální matice

príklad:
        pctrans([s,2s^2])
```



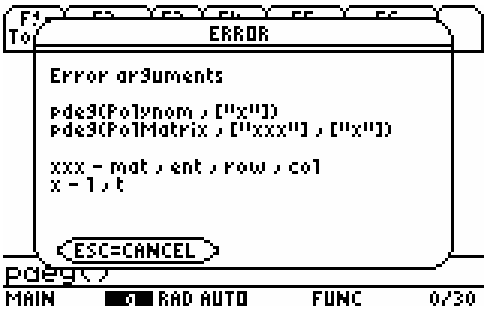
Obrázek 3.16 Príklad funkce pctrans

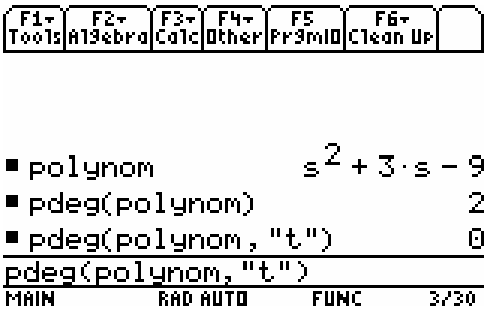
3.4.5 pdeg

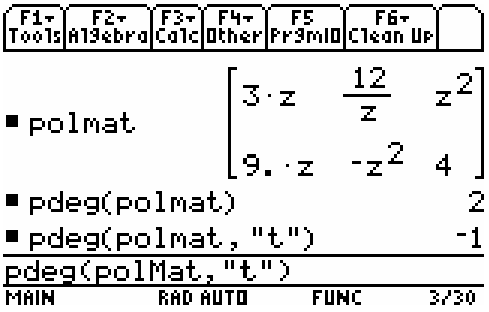
Funkce vrací stupen polynomu nebo stupne polynomiální matice podle zadaného parametru

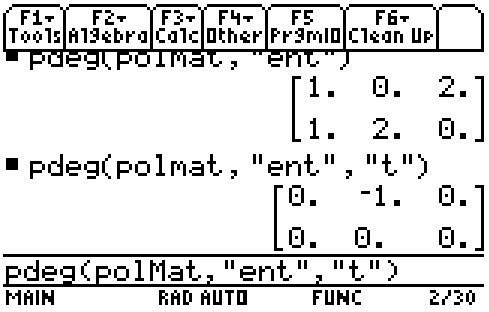
Syntaxe:

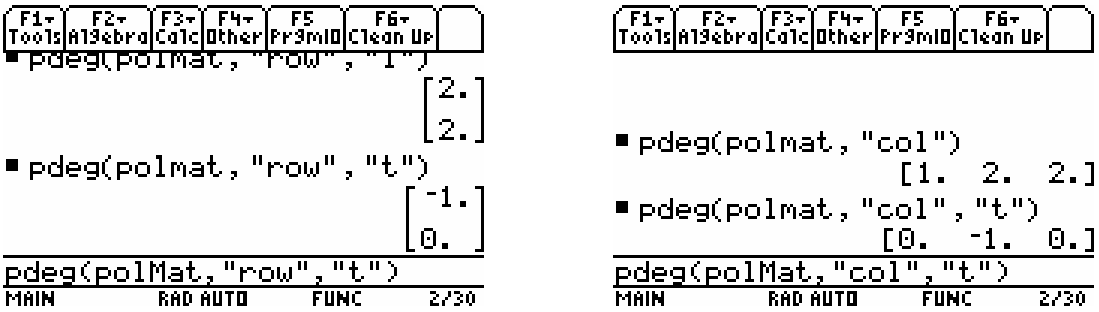
```
pdeg(polynom,[x])
pdeg(polMat,[xxx],[x])
polMat  …  polynomiální matice
xxx – mat  …  vrátí nejvyšší stupen z celé matice
    - ent  …  vrátí matici stupnu jednotlivých polynomu
    - row  …  vrátí sloupec rádkových maxim stupnu
              jednotlivých polynomu
    - col  …  vrátí rádek sloupcových maxim stupnu
              jednotlivých polynomu
x – l  …  horní stupen polynomu
  - t  …  dolní stupen polynomu

implicitne: xxx = mat
            x = l

príklad: pdeg(polynom) = pdeg(polynom,“l“)
         pdeg(polMat) = pdeg(polMat,“mat“,“l“)
         pdeg(polMat,“t“) = pdeg(polMat,“mat“,“t“)
```









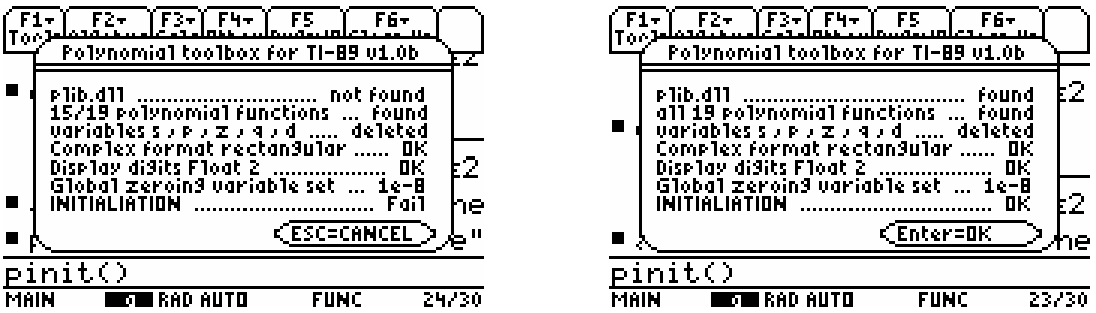


Obrázek 3.17 Príklad funkce pdeg

3.4.6 pinit

Inicializace polynomiálního toolboxu pro TI-89. Provádí se

- kontrola prítomnosti a verze dynamické knihovny PLIB.DLL
- kontrola prítomnosti polynomiálních funkcí
- odstranení predefinovaných promenných s, p, z, q, d
- nastavení výpoctu v oboru komplexních císel v obdélníkovém tvaru
- nastavení zobrazení císel na dve platné císlice



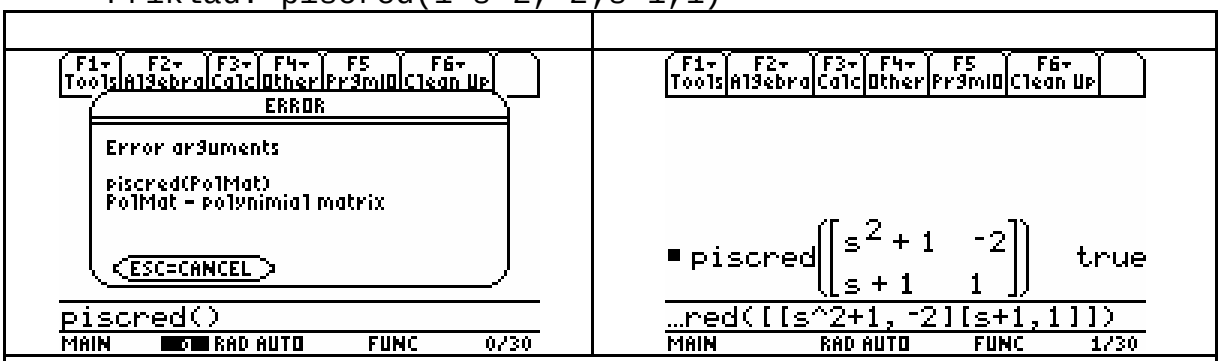
Obrázek 3.18 Príklad funkce pinit

3.4.7 piscred

Funkce zjištuje, je-li polynomiální matice sloupcove redukovatelná.

Syntaxe:

```
piscred(PolMat)
PolMat  …  polynomiální matice

Príklad: piscred(1+s^2,-2;s+1,1)
```



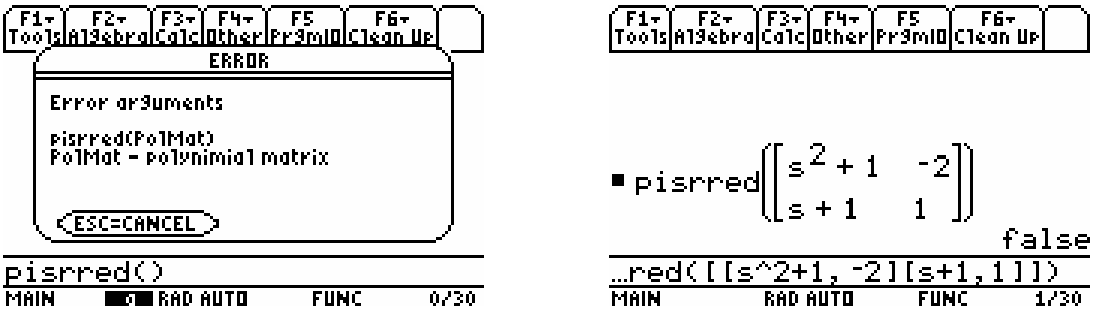
Obrázek 3.19 Príklad funkce piscred.

3.4.8 pisrred

Funkce zjištuje, je-li polynomiální matice rádkove redukovatelná.

Syntaxe:

```
pisrred(PolMat)
PolMat  …  polynomiální matice

Príklad: pisrred(1+s^2,-2;s+1,1)
```



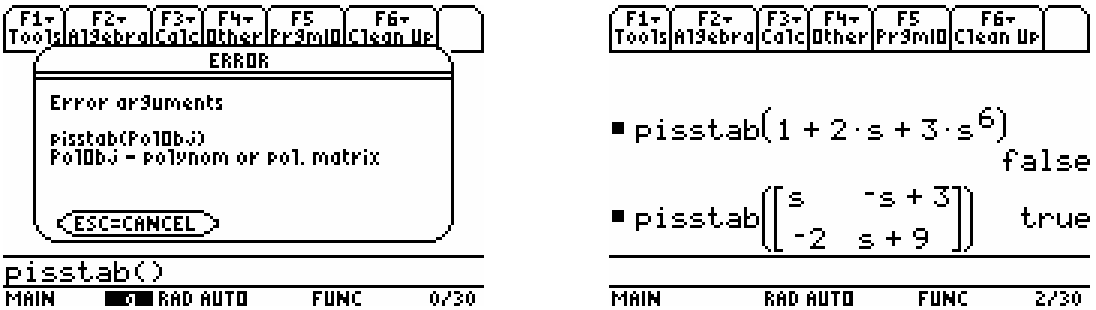
Obrázek 3.20 Príklad funkce pisrred.

3.4.9 pisstab

Funkce zjištuje stabilitu polynomu nebo polynomiální matice.

Syntaxe:

```
pisstab(PolObj)
PolObj  …  polynom nebo polynomiální matice

príklad: pisstab(1+2s+3s^6)
         pisstab([s,3-s;-2,s+9])
```



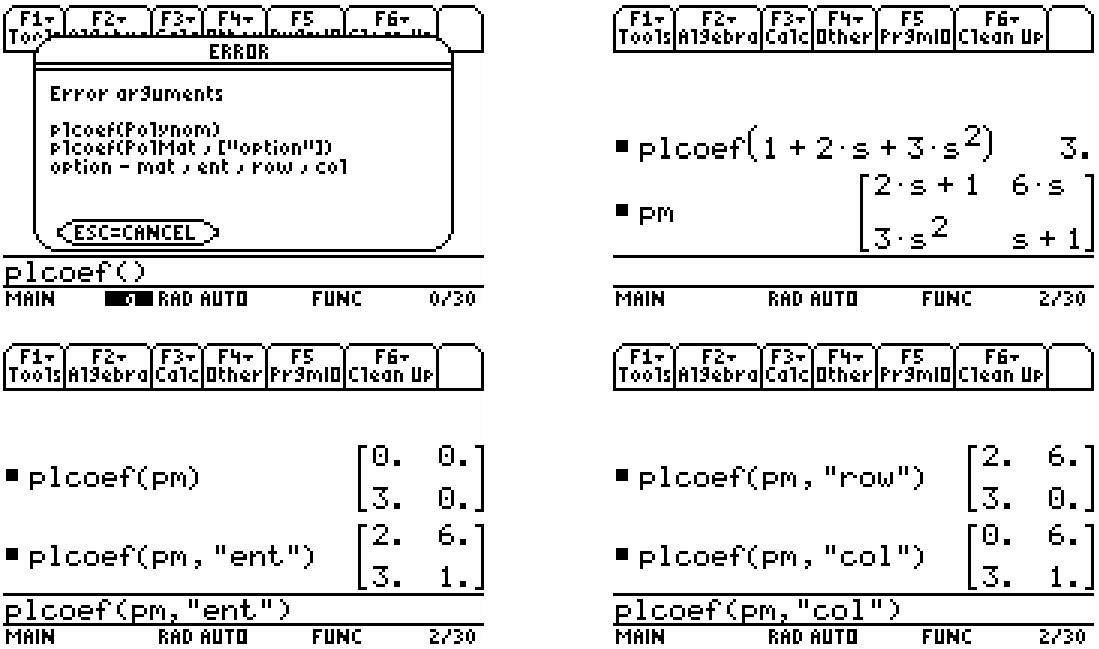
Obrázek 3.21 Príklad funkce pisstab

3.4.10 plcoef

Vrací koeficienty u horních mocnin polynomu.

Syntaxe:

```
plcoef(polynom)
plcoef(polMat,[option])
polMat  …  polynomiální matice
option – mat  …  koeficienty u horní mocniny
                 z celé matice
         ent  …  koeficienty u horní mocniny
                 jednotlivých prvku polynomiální matice
         row  …  koeficienty u horní mocniny
                 z rádku polynomiální matice
         col  …  koeficienty u horní mocniny
                 ze sloupcu polynomiální matice

implicitne: option = mat

príklad: ptcoef(polMat)
         ptcoef(polMat,“ent“)
```



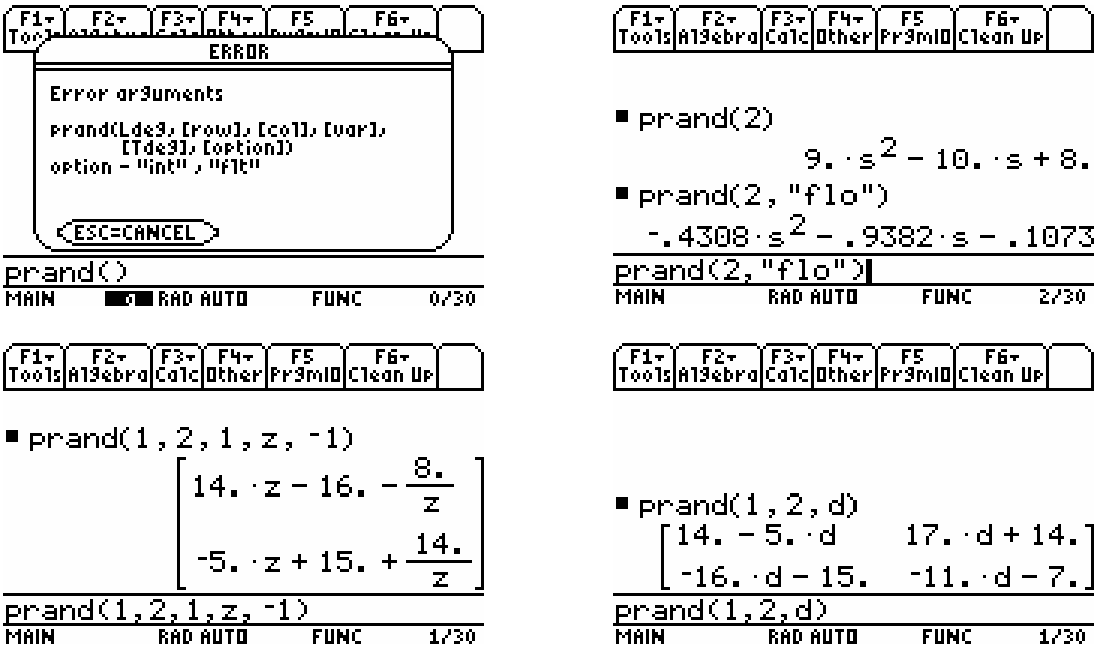
Obrázek 3.22 Príklad funkce plcoef

3.4.11 prand

Generuje náhodný polynom, respektive náhodnou polynomiální matici se zadanými parametry.

Syntaxe:

```
prand (Ldeg,[row],[col],[var],[Tdeg],[options])
Ldeg  ...  horní stupen polynomiální matice
row   ...  pocet rádku polynomiální matice
col   ...  pocet sloupcu polynomiální matice
var   ...  promenná polynomiální matice
Tdeg  ...  dolní stupen polynomiální matice
options – int ... celocíselné koeficienty
        - flo ... reálné koeficienty

implicitne: var = s
            options = int

príklad: prand(2) = prand(2,2,2,s,0,int)
         prand(1,2,p,"flo") = prand(1,2,2,p,0,"flo")
         prand(1,3,z,-1) = prand(1,3,3,z,-1,int)
```



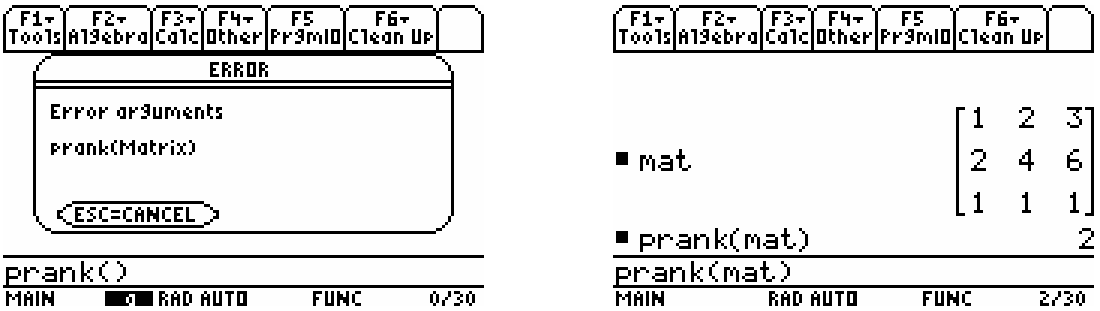
Obrázek 3.23 Príklad funkce prand

3.4.12 prank

Funkce vrací hodnost matice.

Syntaxe:

```
prank(matice)
matice … číselná nebo polynomiální matice

príklad: prank([1,2;3,4])
```



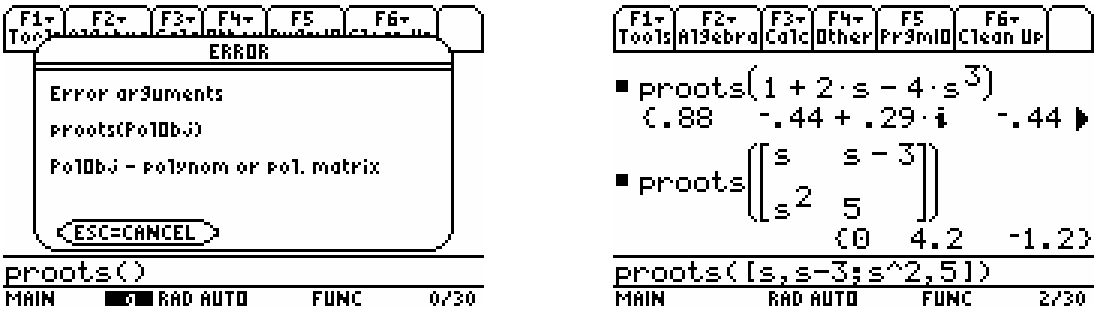
Obrázek 3.24 Príklad funkce prank

3.4.13 proots

Funkce vrací koreny polynomu nebo polynomiální matice. U polynomiální matice reší rovnici $\det(PolMat) = 0$.

Syntaxe:

```
proots(PolObj)
PolObj  …  polynom nebo polynomiální matice

Príklad: proots(1+2s-4s^3)
         proots([s,s-3;s^2,5])
```



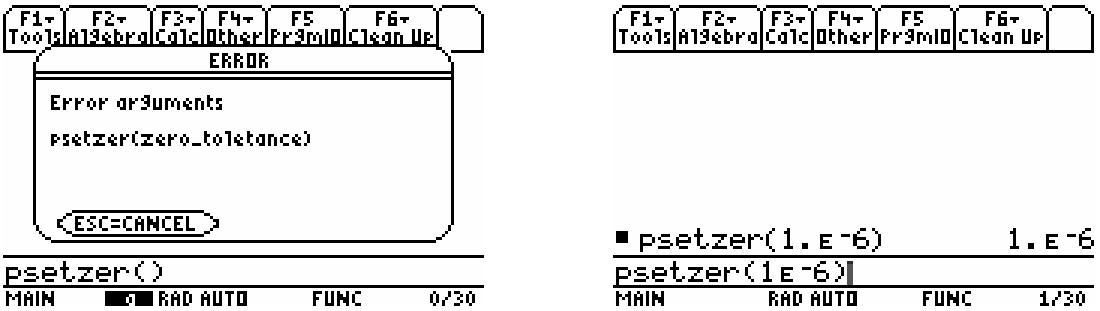
Obrázek 3.25 Príklad funkce proots

3.4.14 psetzer

Nastavuje hodnotu globální nulové tolerance.

Syntaxe:

```
psetzer(tolerance)
tolerance  …  nová hodnota globální nulové tolerance

implicitne: funkce pinit() nastaví toleranci 1e-8

príklad: psetzer(1e-6)
```



Obrázek 3.26 Príklad funkce psetzer

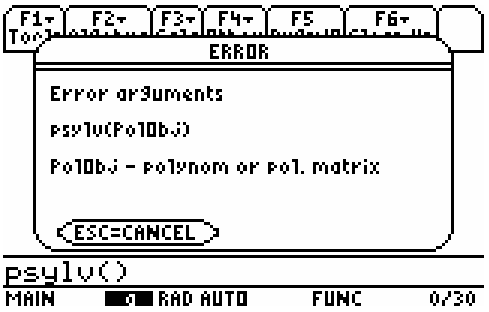
3.4.15 psylv

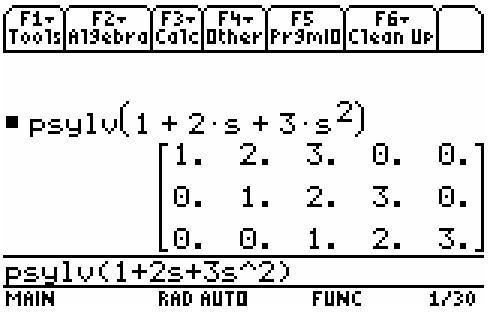
Funkce vytvorí Sylvestrovu matici z polynomu nebo z polynomiální matice.
Pro polynomiální matici $A = A0 + A1 \cdot s + A2 \cdot s^2 + ... + Ad \cdot s^d$

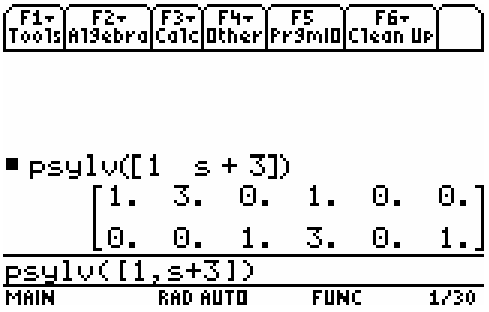
bude Sylvestrova matice $S = \begin{bmatrix} A0 & A1 & A2 & \cdots & Ad & 0 & 0 & \cdots & 0 \\ 0 & A0 & A1 & A2 & \cdots & Ad & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & A0 & \cdots & Ad \end{bmatrix}$

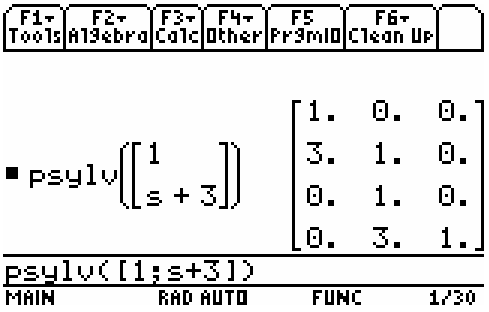
Syntaxe:

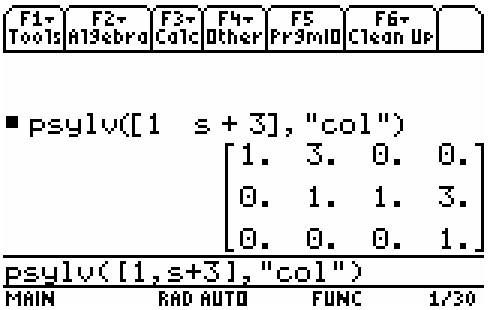
```
psylv(polObj,[K],[type])
polObj  … polynom nebo polynomiální matice
K  …  pocet bloku nul v rádku
type – row  …  rádková Sylvestrova matice
     - col  …  sloupcová Sylvestrova matice

implicitne: K = d
            type = row

príklad: psylv(1+2s)
         psylv([1,s+4;2s+2,3])
```

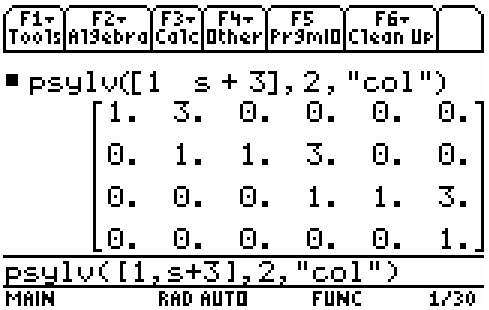












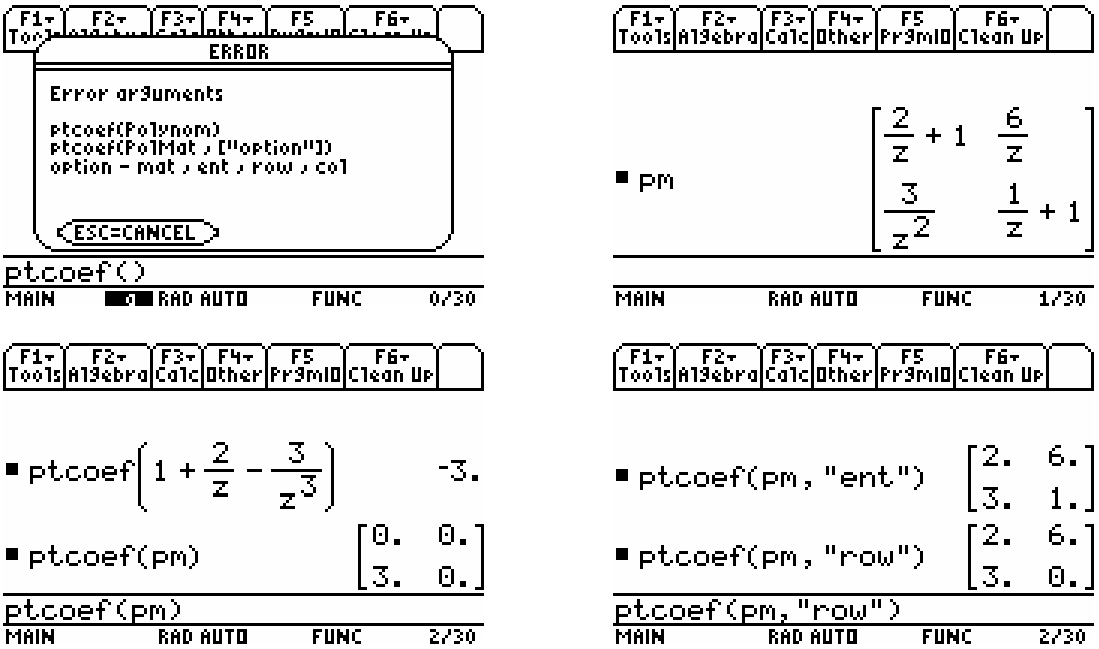
Obrázek 3.27 Príklad funkce psylv

3.4.16 ptcoef

Vrací koeficienty u dolních mocnin polynomu.

Syntaxe:

```
ptcoef(polMat,[option])
polMat  …  polynomiální matice
option – mat  …  koeficienty u dolní mocniny
                 z celé matice
         ent  …  koeficienty u dolní mocniny
                 jednotlivých prvku polynomiální matice
         row  …  koeficienty u dolní mocniny
                 z rádku polynomiální matice
         col  …  koeficienty u dolní mocniny
                 ze sloupcu polynomiální matice

implicitne: option = mat

príklad: ptcoef(polMat)
         ptcoef(polMat,“ent“)
```



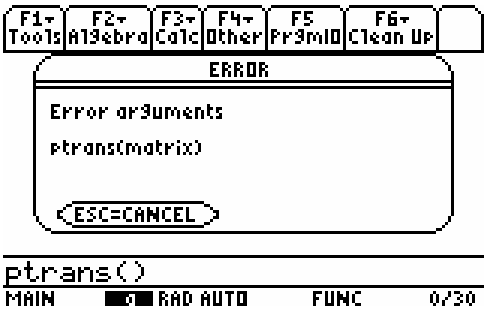
Obrázek 3.28 Príklad funkce ptcoef

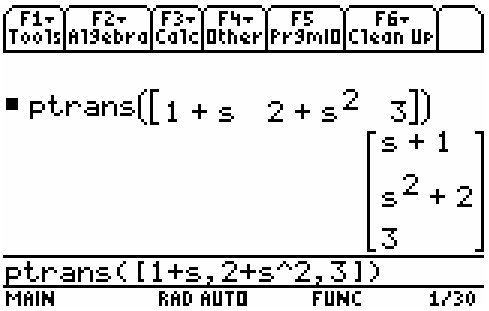
3.4.17 ptrans

Provádí transpozici matice jak s číselnými prvky, tak polynomy.

Syntaxe:

```
ptrans(matice)
matice – matice s číselnými prvky nebo polynomy

príklad: ptrans([1,2;3,4]]
```



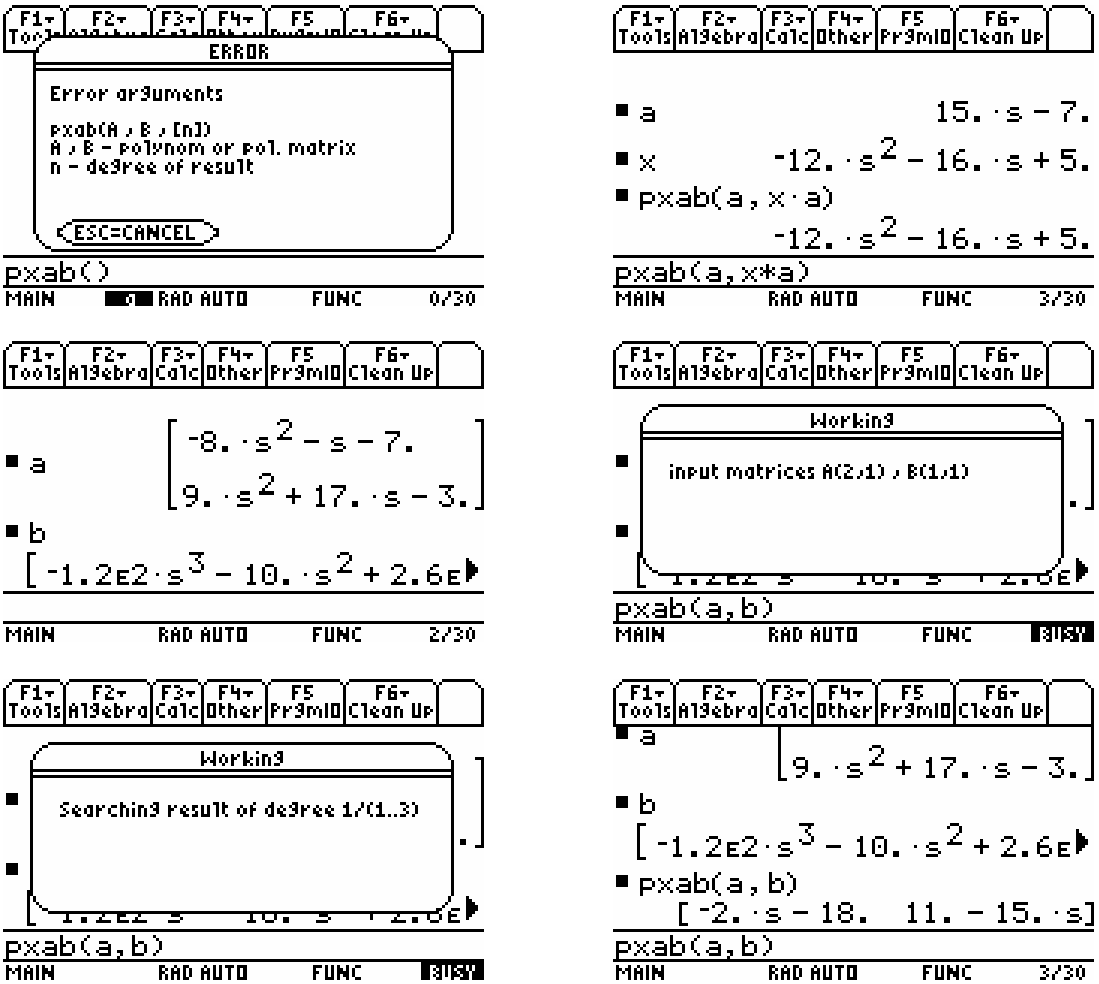


Obrázek 3.29 Príklad funkce ptrans

3.4.18 pxab

Funkce reší rovnici $X \cdot A = B$ pro zadané hodnoty A , B a vrací hodnotu X .
Syntaxe:

```
pxab(A,B,[n])
A,B  …  polynom nebo polynomiální matice
n  …  požadovaný stupen X, není-li zadán, vrací nejnižší
      možný stupen X

príklad:
        pxab(A,B,2)
        pxab(A,B)
```

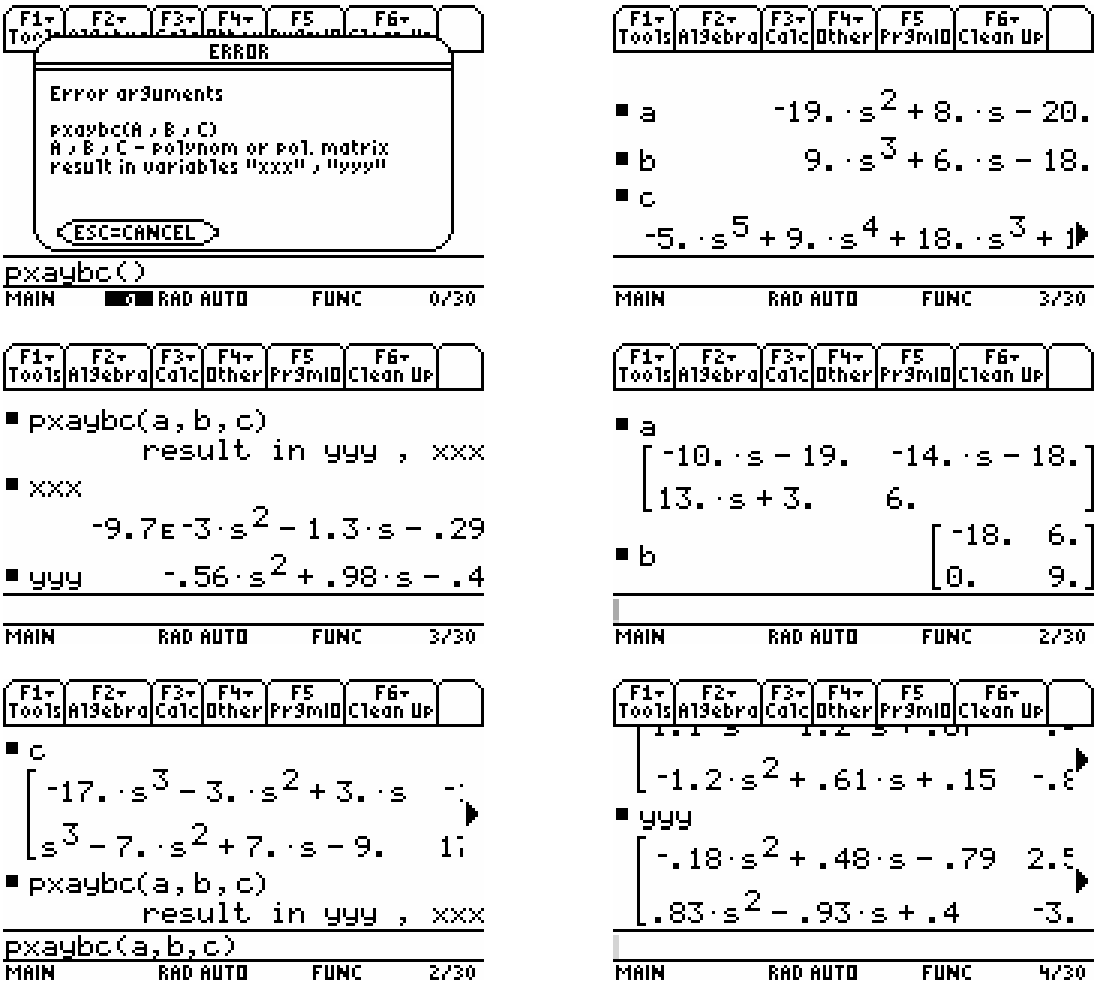


Obrázek 3.30 Príklad funkce pxab

3.4.19 pxaybc

Funkce reší rovnici $X \cdot A + Y \cdot B = C$ pro zadané hodnoty A , B , C a vrací hodnoty X a Y .
Syntaxe:

```
pxaybc(A,B,C)
A,B,C  …  polynom nebo polynomiální matice

príklad:
        pxaybc(A,B,C)
```



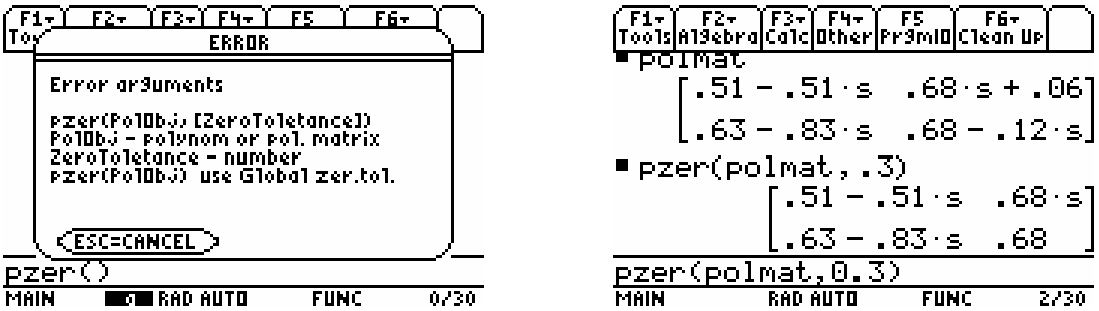
Obrázek 3.31 Príklad funkce pxaybc

3.4.20 pzer

Nuluje koeficienty u polynomu nebo polynomiální matice, pokud jejich absolutní hodnota je menší než zadaná tolerance.

Syntaxe:

```
pzer(polObj,[tolerance])
polObj  …  polynom nebo polynomiální matice
tolerance  …  reálné číslo

implicitne: tolerance = Globální nulová tolerance

príklad: pzer(polMat,1e-3)
         pzer(polMat)
```

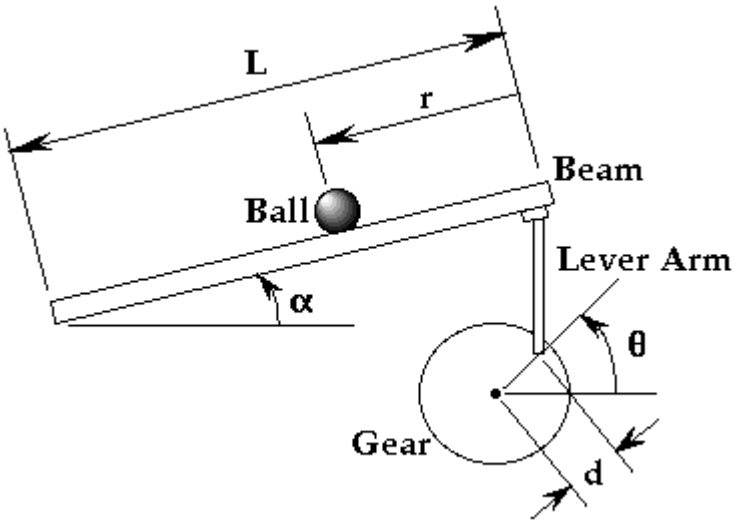


Obrázek 3.32 Príklad funkce pzer

4 Príklad

Využití polynomiálního balícku lze demonstrovat na jednoduchém príkladu návrhu regulátoru pomocí umistování pólu. Mám systém viz. Obrázek 4.1, kde

hmotnost kulicky	M = 0,11 kg
polomer kulicky	R = 0,015 m
délka páky motoru	d = 0,03 m
gravitacní zrychlení	g = 9,8 m / s^2
délka tyce	L = 1 m
moment setrvacnosti	J = 9,99e-6 kg m^2
poloha kulicky	r
úhel naklonení tyce	a
úhel servo motoru	?



Obrázek 4.1 Kulicka na tyci

Tento systém je popsán pohybovou rovnicí

$$0 = \left(\frac{J}{R^2} + m\right)\ddot{r} + mg\sin\alpha - mr(\dot{\alpha})^2$$

linearizovaný tvar rovnice v okolí bodu (a = 0)

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = -mg\alpha$$

dosazením vztahu

$$\alpha = \frac{d}{L}\theta$$

získám rovnici

$$\left(\frac{J}{R^2}+m\right)\ddot{r} = -mg\frac{d}{L}\theta$$

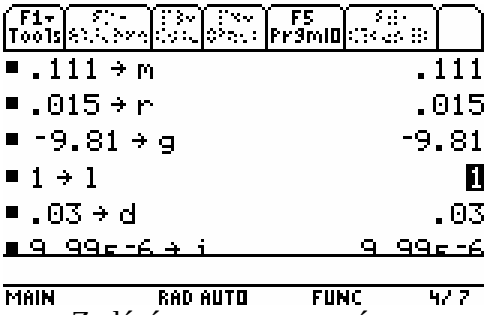
Laplaceovou transformaci získám prenosovou funkci

$$\left(\frac{J}{R^2}+m\right)R(s)s^2 = -\frac{mgd}{L}\Theta(s)$$

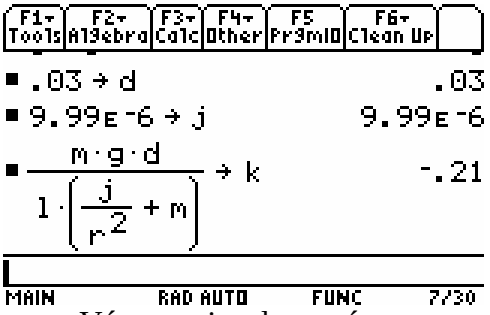
Preusporádáním získám vztah mezi úhlem motoru a polohou kulicky

$$\frac{R(s)}{\Theta(s)} = -\frac{mgd}{L\left(\frac{J}{R^2}+m\right)}\frac{1}{s^2}$$

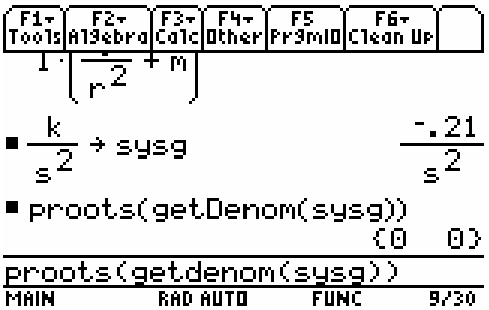
Návrh regulátoru pomocí kalkulátoru TI-89.



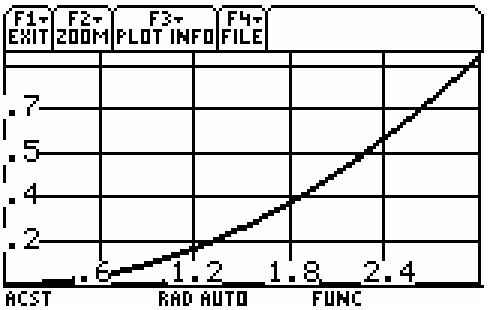
Zadání parametru systému.



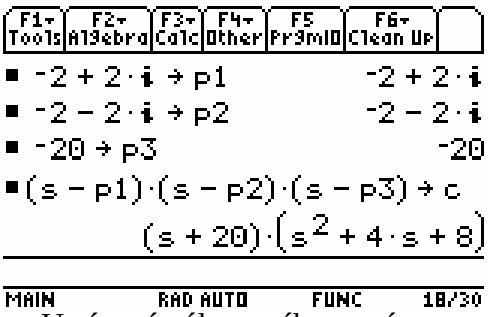
Výpocet citatele systému.



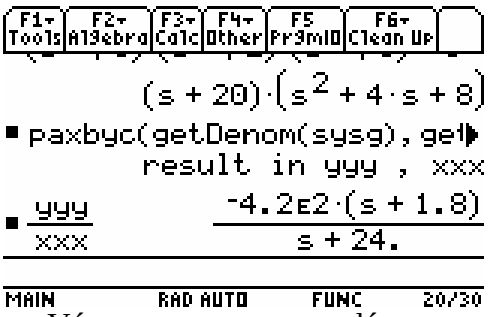
Definice systému a kontrola polohy pólu.



Prechodová charakteristika systému. Systém je na mezi stability, jakákoliv výchylka vede k pohybu do nekonecna.
Použita funkce step() ze systému ACST. viz. [22]

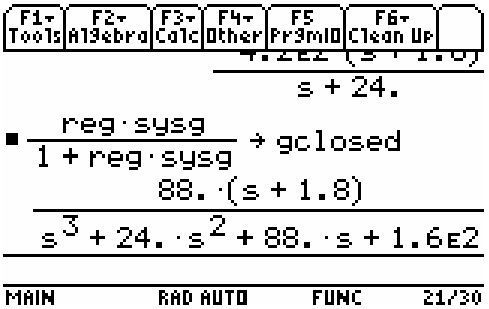


Umístení pólu nového systému.

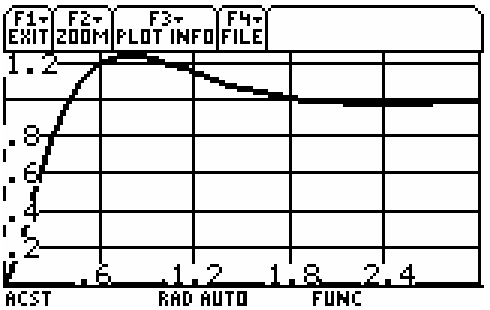


Výpocet parametru regulátoru.

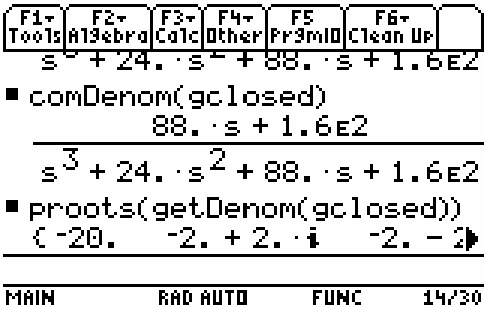
Póly volím podle požadavku na parametry zpetnovazebního systému (doba nábehu, doba regulace, preregulování). Pro aproximaci systému 2 rádu platí ($t_s = 4{,}6/s$ a preregulování 5% odpovídá $V = 0{,}7 \Rightarrow 45°$) Pro $t_s = 3 \Rightarrow s = 1.5 = -\mathrm{Re}(p_i)$ volím približne $p_{12} = -2 \pm 2$. Tretí pól predstavuje dynamik filtru (pozorovatele). Volím $p_3 = -20$, aby dynamika systému druhého rádu zustala dominantní.



Regulacní smycka.



Prechodová charakteristika regulacní smycky. Doba regulace pod 3 sekundy je splnena. Zadané preregulování je prekroceno asi o 17%, což lze vysvetlit výskytem nuly v -1,8 , která snižuje váhu dominantních pólu a tím zvolený systém vzdaluje od aproximace druhého rádu. Použita funkce step() ze systému ACST. viz. [22]



Kontrola pólu systému. Póly jsou presne, kde jsem je umístil.
Reálné cásti jsou záporné => systém je stabilní, což je videt i v prechodové charakteristice.

5 Záver

V diplomové práci jsem se zabýval možností implementace numerických algoritmu pro polynomiální matice pro kalkulátor TI-89. Vytvoril jsem sadu funkcí pro polynomiální matice a knihovnu pomocných funkcí. Je nutné podotknout, že informace ohledne problematiky programování tohoto kalkulátoru byly v pocátcích práce málo dostupné a neúplné. Nejlepším zdrojem bylo internetové diskusní fórum TIGCC Programming [17], kterého jsem se aktivne úcastnil a na základe toho jsem si zvolil i typ vývojového prostredí (TIGCC IDE). Toto prostredí je dosud dostupné pouze jako beta verze, ale na jeho vývoji se aktivne pracuje. Je vytvárena celistvá dokumentace o problematice programování tohoto kalkulátoru a v blízké dobe se ocekává i implementace debuggeru.

Zvažoval jsem který programovací jazyk použiji. Nekteré jednoduché funkce jsem vytvoril v programovacím jazyce TI-Basic, ale se zvyšující se složitostí polynomiálních funkcí jsem byl nucen prejít na programovací jazyk C. Výpocetní cas se snížil, ale velikost jednotlivých funkcí byla neprijatelná (10 kB a více). Uskutecnil jsem nekolik kroku k optimalizaci velikosti, kdy posledním bylo vytvorení dynamické knihovny obsahující funkce spolecné pro programované polynomiální funkce.

Polynomiální funkce byly behem svého vývoje nekolikrát optimalizovány jak na velikost, tak na rychlost. Kladl jsem duraz na bezpecnost techto funkcí, kdy chybné uvolnení pameti muže vést k zatuhnutí kalkulátoru a ztráte dat. Na požadavek vzešlý z internetového fóra jsem do casove nárocných funkcí implementoval možnost okamžitého ukoncení výpoctu.

Vytvorený balícek polynomiálních funkcí je pouhým zlomkem funkcí, které by bylo nutno vytvorit pro plnohodnotné využití. Ale na základe mých zkušeností bych vývoj techto funkcí, zvlášte pak složitejších, pozastavil. A to do doby než bude do programovacího prostredí TIGCC IDE implementován debugger. Využití kalkulátoru pro kompletní návrh regulátoru je možný, ale díky nesrovnatelne nižšímu výkonu v porovnání s dnešními osobními pocítaci je nutno na výsledek složitejších operací cekat rádove desítky sekund.

Literatura

[1] Texas Instruments Inc. *TI-89 Bid Specifications* <http://education.ti.com/us/product/tech/89/features/89bid.html>. 2004.

[2] Wagner, R. *Virtual TI v2.5 beta 5* <http://www.ticalc.org/archives/files/fileinfo/84/8442.html>. 2000.

[3] Muchembled, J. *HW2Patch v2.30* <http://www.ticalc.org/archives/files/fileinfo/113/11331.html>. *2001.*

[4] Davidson, P. *TheUltimate TI Calculator FAQ - Index* <http://www.ocf.berkeley.edu/~pad/faq/>. 2004.

[5] TimeToTeam *PreOs* <http://arche.dyndns.org/~technic/index.php?id=11>. 2002.

[6] Dietsche, G. *KerNo* <http://www.ticalc.org/archives/files/fileinfo/260/26093.html>. *2003.*

[7] Texas Instruments Inc.*TI Flash Studio* <http://education.ti.com/us/resources/developer/8992/hilight/hilight.html>. 2004

[8] Texas Instruments Inc. *TI Flash Studio – Guide* <http://education.ti.com/downloads/pdf/us/sdk8992pguide.pdf>. 2001.

[9] Juric, Z., Kofler, K., Reichel, S. *TIGCC –IDE* <http://tigcc.ticalc.org/>. 2004.

[10] Polyx Ltd. *The Polynomial Toolbox for Matlab – Manual* <http://www.polyx.com/download/manual.pdf.gz>. *1999.*

[11] Polyx Ltd. *Polynomial Toolbox for Matlab* <http://www.polyx.com>. *2004.*

[12] INRIA. *Scilab* <http://scilabsoft.inria.fr>. *2004.*

[13] Maplesoft. *Maple* <http://www.maplesoft.com/products/maple>. *2004.*

[14] Padera, M. *Polynomial Matrices in Java* <http://klokan.sh.cvut.cz/~padera/polynomial/>. *2004.*

[15] Halmo, L. *PolPack++* <http://sourceforge.net/projects/polpackplusplus>. *2004.*

[16] Augusta, P. *Polmat* <http://polmat.wz.cz>. *2004.*

[17] TI-Chess Team HQ *TIGCC Programming* <http://pub26.ezboard.com/ftichessteamhqfrm5>. *2004.*

[18] Kujan, P. *Polynomiální balícek* Diplomová práce FEL CVUT v Praze. 2004.

[19] Kailath, T., *Linear Systems*, Prentice Hall. 1980.

[20] Henrion, D. *Reliable Algorithms for Polynomial Matrices* <http://www.laas.fr/~henrion/Papers/thesis.ps.gz> Ph. D. Thesis, Institute of Information Theory and Automation, Czech Academy of Sciences, Prague, Czech Republic, December 1998. LAAS-CNRS Research Report No. 99003.

[21] Herout, P. *Ucebnice jazyka C*. Ceské Budejovice, 3. upravené vydání, 2000.

[22] Troiani, G. *Automatic Control Systems Toolbox v3.0* <http://www.ticalc.org/archives/files/fileinfo/155/15562.html>. 2001.

Príloha I. – Kód jednoduché polynomiální funkce

```
// C Source File
// tento priklad je pouze pro demonstraci ukonu, ktere je
// nutno provest pro pro spravnou funkci programovane funkce

#define EXECUTE_IN_GHOST_SPACE
// direktiva zajistujici moznost volani jedne funkce v druhe
// a moznost ulozeni vysledku do globalni promenne kalkulatoru

#include "pstruct.h"
// obsahuje makra, struktury a funkce potrebne v polynomialnich
// funkcich pred pripojenim dynamicke knihovny

#include "plib.h"
// seznam funkci obsazenych v dynamicke knihovne

void _main(void)
{

#ifdef TEST
test("new",1);
// funkce zjistujici a uchovavajici hodnotu volne pameti
// slouzi k testu spravne dealokace
#endif

  volatile PTPolynomial_matrix pM, pM2, pM3;
  volatile int nErrorCode;
  // promenne jejichz hodnotu potrebuji v bloku TRY nesmi byt
  // typu register

  nErrorCode = 1; pM = NULL; pM2 = NULL; pM3 = NULL;
  // protoze staticke promenne se nachazeji v tele programu
  // a inicializace probiha pouze v dobe prekladu
  // nelze pouzit (int A = 0;) protoze pri naslednem spusteni
  // programu by byla hodnota A rovna posledni hodnote A
  // pred ukoncenim funkce.

TRY

  if ((nErrorCode = LoadDLL(DLL_name, DLL_id, DLL_major,
DLL_minor)) != DLL_OK) ER_throwVar(nErrorCode);
  // kontrola dynamicke knihovny a jeji pripojeni

  int nArgs = ArgCount();
  // zjisteni poctu argumentu v zasobniku vyrazu
```

```
  if ((nArgs < 1)||(nArgs > 2))
ER_throwVar(ER_f_augment_bad_arguments);
// pokud je pocet argumentu jiny nez se ocekava
// vyvola se definovana chyba

  ESI pBegin, pArgs, pEmpty;
  pBegin = top_estack;  pArgs = top_estack;
  // inicializace ukazatelu na zasobnik vyrazu

  pEmpty = end_params(top_estack);
  // nalezeni ukazatele na konec zasobniku

  pM2 = read_polynomial_matrix(pArgs);
  // kontrola a nacteni polynomialni matice
  // do struktury TPolynomial_matrix

  if (OSCheckBreak()) ER_throwVar(ER_BREAK);
  // test stisku klavesy Break, tento test se vklada
  // pouze do casove narocnych bloku funkce
  // umoznuje prime ukonceni funkce

  pM3 = read_polynomial_matrix(next_expression_index(pArgs));
  if (OSCheckBreak()) ER_throwVar(ER_BREAK);

  delete_between(pEmpty,pBegin);
  // vymazani obsahu zasobniku vyrazu

  pM = get_augment_pm(pM2,pM3,';');
  // radkove spojeni dvou matic

  push_polynomial_matrix(pM);
  // vlozeni polynomialni matice na zasobnik vyrazu
  // ve znackove forme

  delete_polynomial_matrix(pM); pM = NULL;
  delete_polynomial_matrix(pM2); pM2 = NULL;
  delete_polynomial_matrix(pM3); pM3 = NULL;
  // dealokace pameti

  UnloadDLL();
  // odpojeni dynamicke knihovny

ONERR
  clear_estack();   push_zstr("Error");
  // smazani zasobniku vyrazu a vlozeni textu "Error"

  if (nErrorCode != DLL_OK){
    error("DLL_ERROR\n\nrun pinit() for more information");
    // pripojeni DLL se nepodarilo, nelze volat jeji funkce
```

```
  }else{
     delete_polynomial_matrix(pM); pM = NULL;
     delete_polynomial_matrix(pM2); pM2 = NULL;
     delete_polynomial_matrix(pM3); pM3 = NULL;
     // dealokace pameti

     error(my_find_error_message(errCode));
     // zobrazeni okna s textem, ktery odpovida cislu chyby

     UnloadDLL();
     // odojeni dynamicke knihovny
  }
ENDTRY

   if (OSCheckBreak()) OSClearBreak();
   // vymazani priznaku stisknuti Break

#ifdef TEST
test("new",3);
// kontrola zda byla uvolnena vsechna pamet
// pokud se lisi od hodnoty funkce volane na zacatku programu
// vypise okno s hodnotami pameti a jejich rozdilem.
#endif
}
```

Príloha II. – Obsah priloženého kompaktního disku

V následující tabulce je uveden abecední seznam všech korenových adresáru priloženého kompaktního disku s popisem obsahu adresáru.

Adresár	Obsah
DIPLOMOVA_PRACE	*Tato diplomová práce.*
DOKUMENTACE	*Dokumentace ke kalkulátoru TI-89 a k jeho programování.*
ROZSIRENI_AMS	*Aktuální verze rozšírení AMS (Kernel).*
PTOOL	*Funkce polynomiálního balícku pripravené pro použití v TI-89.*
PTOOL_SOURCE	*Zdrojové kódy polynomiálních funkcí a dynamické knihovny plib.dll.*
TI_FLASH_STUDIO	*Vývojové prostredí v1.1.*
TIGCC_IDE	*Vývojové prostredí ve verzi v0.95b8.*
WWW	*Internetové stránky tohoto projektu v anglictine.*

Príloha III. – Internetové stránky projektu.

Internetové stránky projektu v anglictine jsou dostupné na adrese <http://ptoolti89.wz.cz>.