POGLAVJE I

2. Naprednejši koraki z *Mathematico*

1. Uporaba indeksiranih spremenljivk s podpisovanjem (subscriptom)
2. Shranjevanje slik za kasnejšo uporabo v Mathematici
3. Prenos enačb v urejevalnike besedil
4. Dostop do rezultatov
5. Manipuliranje z nizi

1. Uporaba indeksiranih spremenljivk s podpisovanjem (subscriptom)

Programski paket *Mathematica* omogoča tudi uporabo indeksiranih spremenljiv s subscriptom, vendar je potrebna posebna previdnost pri izbiri indeksa. Indeksiran spremenljivka se lahko zapiše bodisi z uporabo ustreznega menija  še pred zapisom imena spremenljivke (ime spremenljivke in indeks se nato zapišeta v ustrezni okenci), ali pa se najprej zapiše spremenljivka in se nato s kombinacijo tipk *Ctrl in* - aktivira meni za zapis indeksa. Ena izmed pogosto uporabljanih spremenljivk v dinamiki konstrukcij je *frekvenca dušenega nihanja*, ω_d, ki se tako npr. definira kot:

```
In[1]:= ωd = 9

Out[1]= 9
```

Uporaba spremenljivke je enostavna in logična:

```
In[2]:= ωd

Out[2]= 9
```

in s spremenljivko je mogoče tudi dalje manipulirati:

```
In[3]:= √ωd

Out[3]= 3
```

Nepričakovani zapleti pa nastopijo, ko se definira spremenljivka, ki v indeksiranih spremenljivkah nastopa kot indeks. V dinamiki konstrukcij se z *d* običajno označuje *podajnost*, ki je pri sistemih z eno prostostno stopnjo obratnosorazmerna togosti, npr.:

```
In[4]:= d = 1 / 12345

Out[4]=   1
        -----
        12345
```

Navidezno nedolžna definicija spremenljivke *d* pa povzroči zaplete pri manipulaciji s frekvenco dušenega nihanja, saj ob uporabi indeksirane spremenljivke sledi :

```
In[5]:= ωd

Out[5]= ω 1/12345
```

kar praktično onemogoči dostop do vrednosti definirane spremenljivke, saj se vse operacije odslej izvajajo v simbolni obliki:

```
In[6]:= √ωd

Out[6]= √(ω 1/12345)
```

Tudi zapis spremenljivke z vrednostjo indeksa ne omogoči dostopa do prej definirane vrednosti:

```
In[7]:= ω 1/12345

Out[7]= ω 1/12345
```

Če pa se sedaj ponovno dodeli vrednost spremenljivki ω_d, pa že definirana vrednost spremenljivke *d* nima vpliva:

```
In[8]:= ωd = 9

Out[8]= 9
```

saj je izpis spremenljivke v skladu s pričakovanji:

```
In[9]:= ωd

Out[9]= 9
```

mogoča pa je tudi normalna manipulacija s spremenljivko:

```
In[10]:= √ωd

Out[10]= 3
```

pri čemer pa je spremenljivka d vseskozi ohranila svojo vrednost:

```
In[11]:= d

Out[11]= 1/12345
```

Še večje težave nastopijo, če je bila vrednost spremenljivke, ki je indeksirana (npr. ω v prikazanem delu analize), že definirana pred definiranjem vrednosti spremenljivke, ki se pojavi v indeksu.

2. Shranjevanje slik za kasnejšo uporabo v Mathematici

Pri uporabi različnih pristopov za analizo istega problema se primerjava dobljenih rezultatov izkaže kot dobra kontrola uspešnosti posameznega pristopa, še posebej uporabna pa je grafična predstavitev rezultatov, npr. spreminjanje odziva skozi čas. Ker je smiselno za vsak pristop uporabiti ločeno datoteko (notebook) v *Mathematici*, je tako za končno (ali sprotno) primerjavo iz vsake datoteke potrebno prenesti pripadajočo sliko. Če so vse datoteke aktivne (izvedene ustrezne celice), je to mogoče storiti tako, da se slike za primerjavo direktno izrišejo na osnovi rešitev, shranjenih v spominu (Kernelu), ali pa se že izrisane slike zgolj ponovno prikažejo. Vse to pa zahteva tudi ohranjevanje postopkov oz. korakov analize v spominu, pa čeprav zgolj za primerjavo ti niso več zanimivi. Zato je smiselno poiskati možnosti, ki bodo iz datotek prenesli zgolj sliko, vendar v obliki, ki bo omogočala kombiniranje z drugimi slikami in tudi spreminjanje stila izrisa (zaradi lažje primerjave). Slike je namreč mogoče prenašati med notebooki kar enostavno z odložiščem, vendar take slike ni mogoče naknadno obdelovati ali kombinirati z drugimi.
Mathematica omogoča izvažanje (eksportiranje) objektov v različnih formatih, ki so odvisni od verzije programa. Celoten spisek vseh formatov je mogoče videti z ukazom *$ExportFormats*:

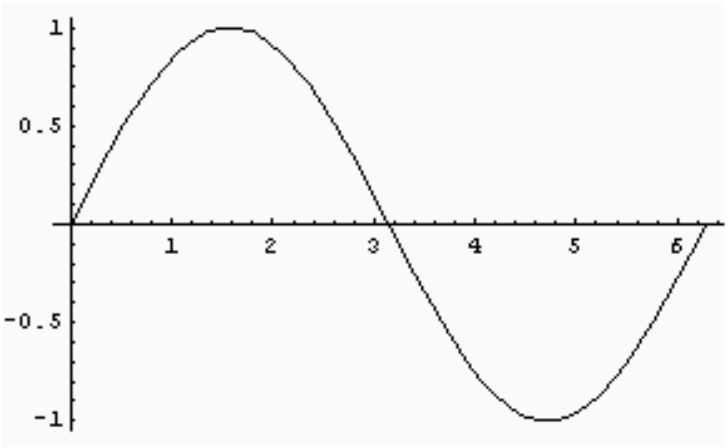
```
In[12]:= $ExportFormats

Out[12]= {AI, AIFF, AU, BMP, Dump, DXF, EPS, EPSI, EPSTIFF,
          Expression, GIF, HDF, HTML, JPEG, Lines, List,
          MAT, MGF, MPS, PBM, PCL, PDF, PGM, PICT, PNM,
          PPM, PSImage, RawBitmap, SND, Table, TeX, Text,
          TIFF, UnicodeText, WAV, WMF, Words, XBitmap}
```

Ker pa bo sliko, shranjeno v enem izmed gornjih formatov, potrebno tudi včitati nazaj v program, je potrebno izbrati takšen format, ki ga je *Mathematica* sposobna tudi uvoziti (importirati). Seznam podprtih formatov pa se izpiše z ukazom *$ImportFormats*, ki pa je očitno drugačen od seznama izvoznih formatov:

```
In[13]:= $ImportFormats

Out[13]= {AIFF, AU, BMP, Dump, EPS, EPSI, EPSTIFF, Expression,
          GIF, HDF, JPEG, Lines, List, MAT, MGF, MPS, PBM,
          PGM, PNM, PPM, PSImage, RawBitmap, SND, Table,
          Text, TIFF, UnicodeText, WAV, Words, XBitmap}
```

Za demonstracijo je potrebno sedaj pripraviti neko sliko, ki se shrani npr. kar pod imenom *slika*:

```
In[14]:= slika = Plot[Sin[x], {x, 0, 2 Pi}];
```



Pri shranjevanju je potrebno sedaj izbrati ustrezni grafični format. Obstajajo formati, ki so neodvisni od izbrane grafične resolucije, saj operirajo z vektorji (AI, EPS, MPS, PCL, PDF, PICZ, WMF); ter formati, ki so odvisni od izbrane grafične resolucije, saj operirajo z bitno sliko (BMP, EPSI, EPSTIFF, GIF, JPEG, MGF, PBM, PGM, PNG, PPM, PSImage, TIFF, XBitmap).
Čeprav je na grafično resolucijo in s tem na kvaliteto slike mogoče vlivati z opcijo *ImageResolution*, s temi slikami ni več mogoče manipulirati, in zato se izbere format, ki je neodvisen od grafične resolucije, npr. EPS (Encapsulated Post Script). Za shranjevanje se uporabi ukaz *Export[“imedatoteke“, imeobjekta]*. Ime datoteke lahko zajame tudi celotno pot (path) na mediju shranjevanja do datoteke, drugače pa se datoteka shrani v mapo, kjer je inštalirana *Mathematica* (npr. c:\Program Files\Wolfram Research\Mathematica\4.0\). Vidni rezultat uporabe ukaza pa je izpisano ime datoteke s sliko.

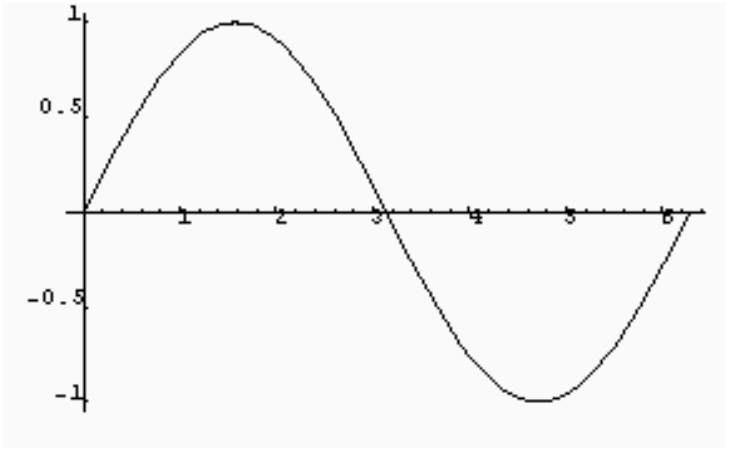
```
In[15]:= Export["slikaM.eps", slika]

Out[15]= slikaM.eps
```

Shranjeno sliko je sedaj mogoče včitati z ukazom *Import[“imedatoteke.ekstenzija“]*. Če je mogoče že iz ekstenzije imena razbrati, za kakšen tip oz. format gre, formata ni potrebno posebej zapisati, drugače pa se uporabi ukaz v obliki *Import[“imedatoteke“,“format“]*. Ker se včita objekt, se mu dodeli ime (zaradi kasnejšega referiranja in manipulacije):

```
In[16]:= vcitana = Import["slikaM.eps"];
```

Včitano grafiko je seveda mogoče tudi prikazati z ukazom *Show[objekt]*:

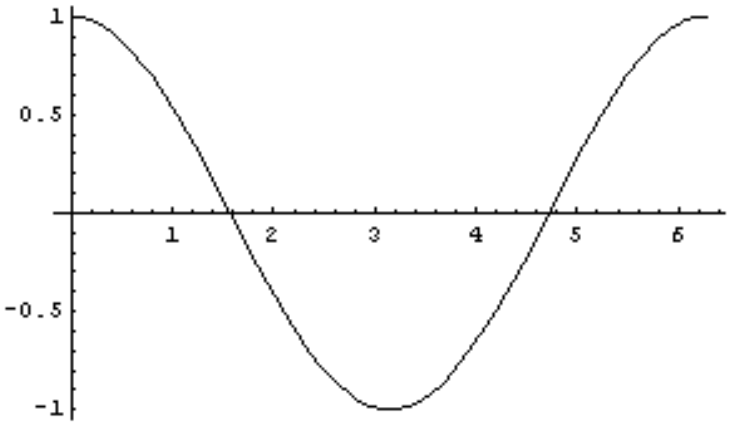
```
In[17]:= Show[vcitana];
```



in vizualnih razlik med originalno in včitano sliko ni.

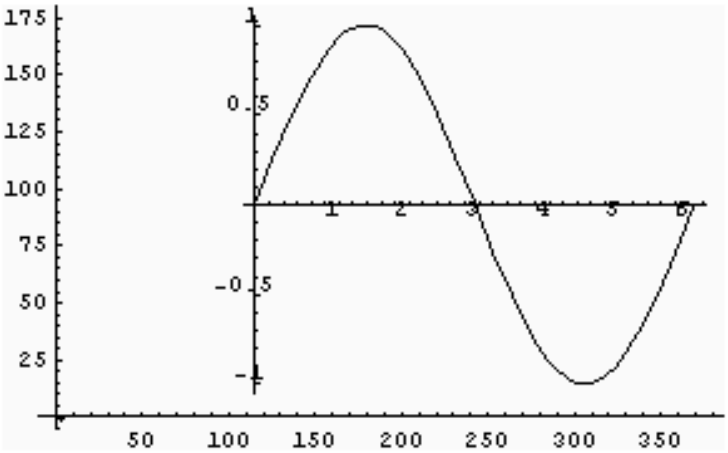
Ker pa je zaradi primerjave zanimivo kombiniranje včitanih slik z drugimi, se za primerjavo pripravi še slika druge funkcije:

```
In[18]:= nova = Plot[Cos[x], {x, 0, 2 Pi}];
```



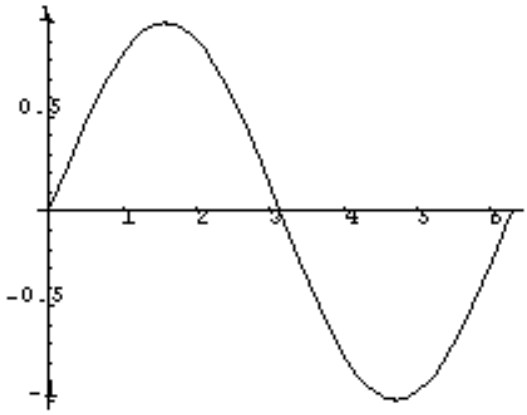
Primerjava obeh slik pa vodi do nepričakovanih rezultatov:

```
In[19]:= Show[nova, vcitana];
```



ki pa se celo spremenijo (!), če se zamenja vrstni red prikazovanja obeh slik (drugi objekt v ukazu Show se namreč prikaže za prvim):

```
In[20]:= Show[vcitana, nova];
```



Odgovor, zakaj sploh pride do takšnega nepričakovanega odziva, se lahko uvidi, če slika včitane grafike (Out[17]) pretvori v vhodno obliko, kar se izvede tako, da se klikne na sliko, nato pa se s kombinacijo tipk *Ctrl+Shift+I* pretvori iz izhodne v vhodno obliko (prikazan je zgolj del obsežnega izpisa):

```
In[17]:= Show[vcitana];

Show[Graphics[{
AbsoluteThickness[1],
AbsoluteThickness[0.25],
Line[{{118.5, 94.5}, {118.5, 96.125}}],
Text[FontForm["1", {"Courier", 10}],
{117.25000, 89.37500}, {-1.000, -0.700}],
Line[{{156.938, 94.5}, {156.938, 96.125}}],
Text[FontForm["2", {"Courier", 10}],
{156.18750, 89.37500}, {-1.000, -0.700}],
Line[{{195.438, 94.5}, {195.438, 96.125}}],
Text[FontForm["3", {"Courier", 10}],
{194.68750, 89.37500}, {-1.000, -0.700}],
Line[{{233.938, 94.5}, {233.938, 96.125}}],
```

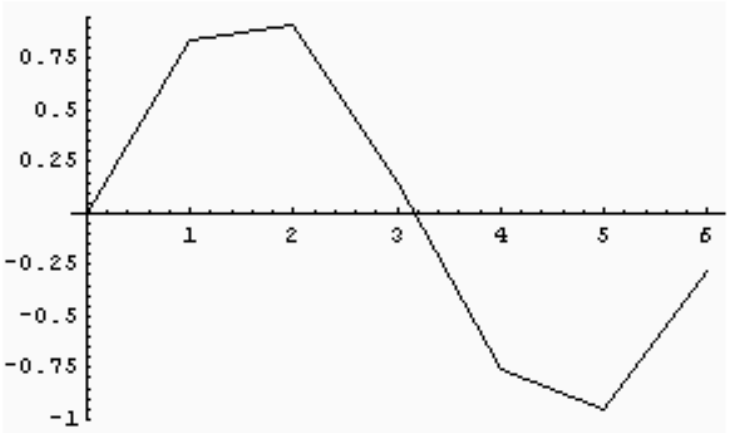
Iz izpisa je jasno razvidno, da *Mathematica* pri shranjevanju zapisa ne shrani v s strani uporabnika uporabljenem koordinatnem sistemu, temveč v nekem lastnem, kar sicer omogoči korekten izris, onemogoči pa primerjavo z drugimi funkcijami.

Zato je mnogo primerneje shraniti na disk vektor diskretnih vrednosti funkcije v obliki parov (ki zajemejo vrednost abscise in pripadajočo vrednost ordinate). Tak vektor se generira z ukazom *Table[{vrednost na abscisi, vrednost na odinati},{neodvisna spremenljivka, začetna vrednost, končna vrednost}]*, kjer se členi vektorja (odvisni vrednosti) generirajo oz. izračunajo s pomočjo neodvisne spremenljivke, ki se spreminja od njene začetne do končne vrednosti).

```
In[21]:= tabelaslike = Table[{x, Sin[x]}, {x, 0, 2 Pi}]

Out[21]= {{0, 0}, {1, Sin[1]}, {2, Sin[2]}, {3, Sin[3]},
          {4, Sin[4]}, {5, Sin[5]}, {6, Sin[6]}}
```

Ker *Mathematica* sama izbere število računskih točk med začetno in končno mejo, je smiselno generirani vektor izrisati in tako preveriti njegovo podobo. Ker gre sedaj za množico diskretnih točk in ne več za analitično funkcijo, se mora uporabiti ukaz *ListPlot[imeliste]*, če pa se želi diskretne točke medsebojno povezati, kar omogoči občutek zveznosti funkcije, pa se uporabi opcija *ListPlot[imeliste;PlotJoined->True]*:

```
In[22]:= ListPlot[tabelaslike, PlotJoined → True];
```

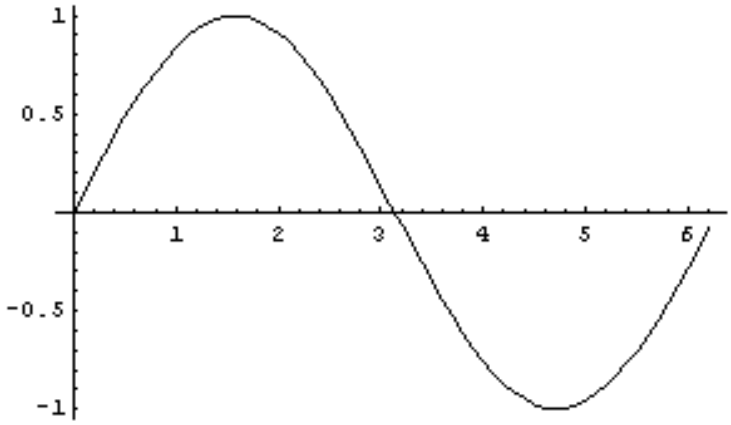


Iz slike je razvidno, da je bilo očitno izbranih premalo računskih točk in zato se ponovi izračun vektorja, vendar tokrat v naprednejši obliki, ki bo definirala interval med posameznimi točkami abscise: *Table[{vrednost na abscisi, vrednost na odinati},{neodvisna spremenljivka, začetna vrednost, končna vrednost, korak}]*. Ker je sedaj mogoče pričakovati večje število računskih točk, se njihov izpis, ki ni zanimiv, lahko izpusti (kar se doseže s podpičjem na koncu ukaza):

```
In[23]:= tabelaslike = Table[{x, Sin[x]}, {x, 0, 2 Pi, 0.1}];
```

Sedaj izrisana slika pokaže, da je izbrani interval dovolj dober:

```
In[24]:= ListPlot[tabelaslike, PlotJoined → True];
```



Zato se sedaj dobljeni podatki shranijo na disk, pri čemer se ponovno uporabi ukaz *Export*, vendar tokrat z opcijo, ki definira zapis objekta v obliki tabele *Export[“imedatoteke“, imeobjekta, “format“]*:

```
In[25]:= Export["tabela", tabelaslike, "Table"]
Out[25]= tabela
```

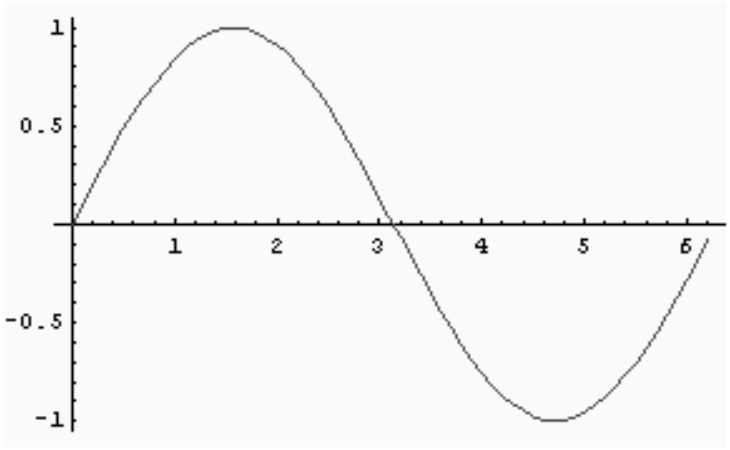
Namesto direktnega shranjemvanja slike funkcije se je tako izvedlo shranjevanje informacij (o krivulji), ki bodo omogočile »rekonstruiranje« slike, seveda brez možnosti kompletnega analitičnega matematičnega manipuliranja s funkcijo.

Tako zapisani podatki se včitajo z že znanim ukazom *Import[“imedatoteke“,“format“]*:

```
In[26]:= vcitanatabela = Import["tabela", "Table"];
```

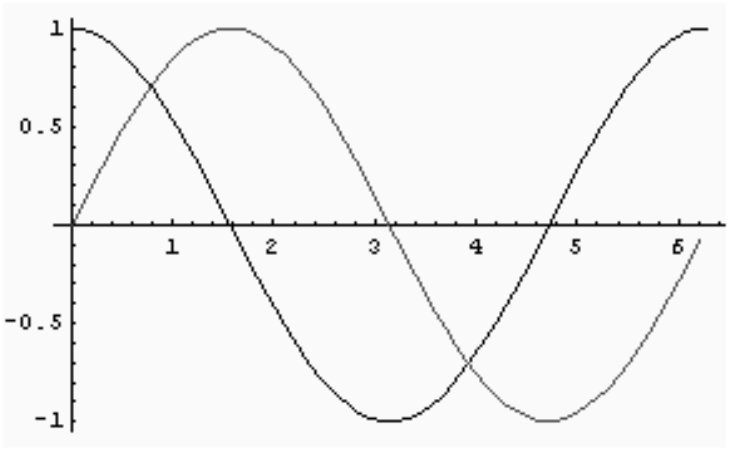
Na njihovi osnovi se izriše slika, ponovno z ukazom *ListPlot*, vendar tokrat z opcijo, ki izvede izris npr. v rdeči barvi, *PlotStyle->RGBColor[1,0,0]*:

```
In[27]:= izris = ListPlot[vcitanatabela, PlotJoined → True,
            PlotStyle → RGBColor[1, 0, 0]];
```



Tako dobljeni izris pa je sedaj mogoče brez problemov tudi kombinirati z drugimi izrisi:

```
In[28]:= Show[nova, izris];
```



3. Prenos enačb v urejevalnike besedil

Dobljene rezultate analize (torej izpeljane izraze in ne samo grafične prikaze) je včasih potrebno tudi prenesti v urejevalnike besedil. *Mathematica 4* tako omogoča izvoz izrazov v obliki *HTML* in *TeX*, *Mathematica 5* pa še v formatih *NB* in *XHTML+MathML*. Najpogosteje uporabljana sta prva dva formata, katerima sta na voljo tudi alternativni obliki *HTMLSave* in *TeXSave*.

Če se želi nek izraz oz. rezultat shraniti na disk, je najprimerneje, da se shrani v neko spremenljivko:

```
In[29]:= rezultat = Integrate[x^3 Cos[x], x]

Out[29]= 3 (-2 + x^2) Cos[x] + x (-6 + x^2) Sin[x]
```

Zapis v *Hyper Text Markup Language* ali *HTML* obliki se doseže z ukazom *Export[“imedatoteke“, imeobjekta, “HTML“]*:

```
In[30]:= Export["enacba", rezultat, "HTML"]

Out[30]= enacba
```

ki v mapi, kjer je inštalirana *Mathematica*, kreira novo mapo (folder) z imenom *enacba*, v kateri se nahaja datoteka *index.html*, ter mapi (folderju) z imenom *Images* in *Links*. Tako ustvarjena HTML datoteka se lahko pregleda s standardnimi brskalniki za internet, kot npr. *IE, Netscape*, itd.:

$$3\,(-2+x^2)\,\text{Cos}[x]+x\,(-6+x^2)\,\text{Sin}[x]$$

Converted by *Mathematica* November 17, 2004

lahko pa se tudi včita v urejevalnik besedil (*Vstavljanje/Slika/Iz datoteke*):

$3\,(-2+x^2)\,\text{Cos}[x]+x\,(-6+x^2)\,\text{Sin}[x]$

Converted by *Mathematica* November 17, 2004

V kolikor pa je napis pod izrazom odveč ali moteč, pa je mogoče včitati zgolj sliko izraza, ki se nahaja v mapi *Images* v obliki *index_gr_1.gif*:

$3\,(-2+x^2)\,\text{Cos}[x]+x\,(-6+x^2)\,\text{Sin}[x]$

Če pa se želi enačba izvoziti v formatu, ki ga uporablja urejevalnik besedil LaTeX, pa se uporabi ukaz:

```
In[31]:= Export["enacba.tex", rezultat, "TeX"]

Out[31]= enacba.tex
```

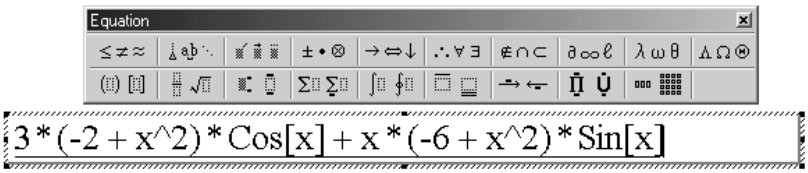
V primeru, ko pa zadošča kar oblika izpisa, ki jo prikaže *Mathematica*, se lahko z miško označi desni konec izhodne celice, celica pa se preko odložišča prilepi v urejevalnik besedil (brez vmesnega shranjevanja na disk):

$3\,(-2+x^2)\,\text{Cos}[x]+x\,(-6+x^2)\,\text{Sin}[x]$

kjer ga je načeloma mogoče urejati kot sliko z *Urejanje/Uredi sliko* (*Edit/Picture Object*), vendar se rezultati urejanja ne ujemajo s pričakovanimi, saj se vsak člen izraza obravnava kot samostojna slika. Zato je smiselno izhodno vrstico najprej pretvoriti v izhodno (s *Ctrl+Shift+I*),

```
In[29]:= rezultat = Integrate[x^3 Cos[x], x]

Out[29]= 3*(-2 + x^2)*Cos[x] + x*(-6 + x^2)*Sin[x]
```

enačbo nato prenesti v odložišče, v WinWordu odpreti urejevalnik besedil in vanjo odložiti enačbo:



Enačbo je mogoče še dodatno urejati (npr. oglate oklepaje zamenjati z okroglimi, urediti potence in predznake, vstaviti znak za množenje, ipd.) v želeno obliko, npr. v:

$$3\cdot(-2+x^2)\cdot\cos(x)+x\cdot(-6+x^2)\cdot\sin(x)$$

4. Dostop do rezultatov

Mathematica posamezne rezultate analize (posameznih enačb ali tudi sistemov enačb, rešitve diferencialnih enačb, ničle funkcij), ne dodeli avtomatično spremenljivkam, ampak vse rezultate oz. bolje potencialne rešitve zapiše v posamezne člene niza oz. vektorja. S tem sicer omogoči, da se izmed večjega števila matematičnih rešitev izberejo zgolj inženirsko uporabne rešitve, hkrati pa oteži njihovo manipuliranje, kar je za uporabnike, ki se šele srečujejo s programom, nenavadno in moteče.
Zaradi lažje manipulacije z dobljenimi rezultati je zato v primeru, ko je pričakovanih več matematičnih rešitev, smiselno direktno shraniti rezultate analize v spremenljivko z ustreznim imenom.
Rešitvi npr. kvadratne enačbe, zapisane v splošni obliki, se tako poiščeta z ukazom *Solve[enačba, neznanka]*. Pomembno je, da je spremenljivka, ki nastopa kot neznanka, sedaj npr. x, še nedefinirana (saj drugače ne more biti neznanka).

```
In[32]:= resitvi = Solve[a x^2 + b x + c == 0, x]
```

$$\text{Out[32]= } \left\{\left\{x \to \frac{-b-\sqrt{b^2-4ac}}{2a}\right\}, \left\{x \to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$$

Dobljeni (matematično korektni) rešitvi sta sedaj shranjeni v spremenljivko *resitvi*, ki ima obliko matrike z dvema vrsticama (ki sta pravzaprav vektorja, vsak z enim členom), v kar se je mogoče prepričati z uporabo ukaza *TableForm[spremenljivka]*:

```
In[33]:= TableForm[resitvi]
```

Out[33]//TableForm=

$$x \to \frac{-b-\sqrt{b^2-4ac}}{2a}$$

$$x \to \frac{-b+\sqrt{b^2-4ac}}{2a}$$

Posamezna rešitev se torej skriva v posamezni vrstici oz. posameznem členu vektorja *resitvi*, ki se dosega tako, da se za imenom spremenljivke, v dvojnem oglatem oklepaju zapiše številka člena, npr. za drugi člen oz. drugo rešitev:

```
In[34]:= resitvi[[2]]
```

$$\text{Out[34]= } \left\{x \to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}$$

Rezultat je sicer korekten, vendar se še vedno nahaja v zavitih oklepajih, saj se je v bistvu izpisal kot vektor (resda zgolj z enim členom). Do vsebine med oklepajema je mogoče priti, je potrebno eksplicitno navesti, da gre za prvi člen drugega vektorja:

```
In[35]:= resitvi[[2, 1]]

Out[35]= x → (-b + √(b^2 - 4 a c)) / (2 a)
```

Rezultat sedaj nakazuje (in ne definira), da je rešitev za spremenljivko *x* lahko izraz oz. rezultat za puščico. Če se torej želi prikazati zgolj rezultat, je potrebno to jasno definirati, in sicer tako, da se doda še dodatni indeks za drugi del rešitve:

```
In[36]:= resitvi[[2, 1, 2]]

Out[36]= (-b + √(b^2 - 4 a c)) / (2 a)
```

Če je to rešitev, ki jo je uporabnik iskal, se ta vrednost rešitve lahko dodeli neki spremenljivki, kar se lahko izvede na dva načina. V prvem načinu se vrednost spremenljivke definira tako, da se vrednost iskane spremenljivke nadomesti z rešitvijo z ukazom /.. Če se ime tako definirane spremenljivke razlikuje od spremenljivke, katere vrednost je bila iskana, torej neznanke, ostane ta spremenljivka nedefinirana (ker bo prva rešitev shranjena npr. v spremenljivko x_1, bo vrednost originalne spremenljivke x še vedno ostala nedefinirana):

```
In[37]:= x1 = x /. resitvi[[1, 1]]

Out[37]= (-b - √(b^2 - 4 a c)) / (2 a)
```

Vrednost neznanke *x* iz enačbe je torej še vedno nedefinirana:

```
In[38]:= x

Out[38]= x
```

Seveda pa je mogoče vrednost spremenljivke definirati tudi direktno, z neposredno dodelitvijo rešitve spremenljivki:

```
In[39]:= x2 = resitvi[[2, 1, 2]]

Out[39]= (-b + √(b^2 - 4 a c)) / (2 a)
```

5. Manipuliranje z nizi

Niz je urejena množica (vektor, matrika) vrednosti, ki se nahajajo v spremenljivki s skupnim imenom, posamezne vrednosti pa se dosegajo oz. odčitajo z uporabo številke-indeksa, ki označuje lokacijo iskane vrednosti v nizu. Posameznih členov niza pa seveda ni mogoče zgolj odčitati, ampak je z njimi mogoče tudi direktno manipulirati.

Za demonstracijske potrebe bo tako definirana periodično spreminjajoča se obtežba, ki bo najprej v poglavju II.14 uporabljena pri iskanju odziva konstrukcije z razvojem v Fourierovo vrsto, v poglavju

II.15 pa pri analizi odziva z Duhamelovim integralom, in sicer s podatki, pripravljenimi v tem poglavju. Obravnavana obtežba ima konstantno pozitivno vrednost P_o v prvi polovici trajanja periode, ter konstantno negativno vrednost v drugi polovici trajanja periode. Zato se najprej definirata perioda T in amplituda obtežbe P_o:

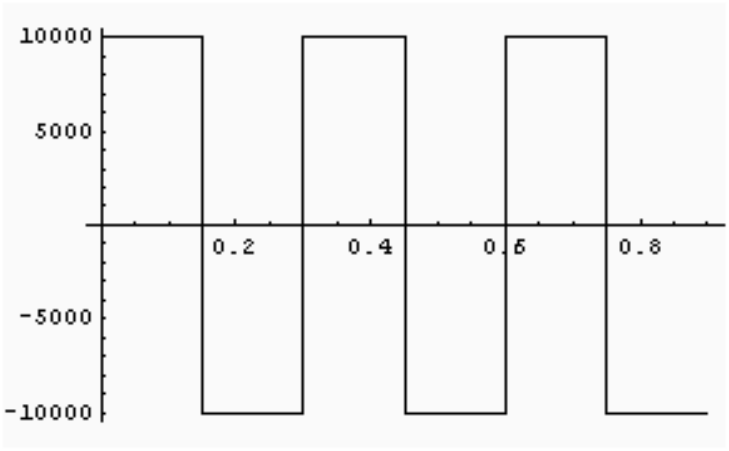
```
In[40]:= T = 0.3;
         P0 = 10000;
```

Najpreprostejša, a hkrati tudi najmanj avtomatična definicija funkcijskih parov se izvede z neposrednim zapisom posameznega člena – para diskretnih točk v zavitem oklepaju in zajame zgolj točke, kjer pride do spremembe obtežbe, kar omogoči najkompaktnejši zapis (vsi pari se zapišejo v dodatni par zavitih oklepajev):

```
In[42]:= obtezba0 = {{0, 0}, {0, P0}, {T / 2, P0}, {T / 2, -P0},
           {T, -P0}, {T, P0}, {3 / 2 T, P0}, {3 / 2 T, -P0},
           {2 T, -P0}, {2 T , P0}, {5 / 2 T, P0}, {5 / 2 T, -P0},
           {3 T, -P0}};
```

Korektnost definiranega vektorja je mogoče enostavno preveriti z njegovim izrisom:

```
In[43]:= pristop0 = ListPlot[obtezba0, PlotJoined → True];
```



in podatke je tako mogoče shraniti na disk:

```
In[44]:= Export["obtezba0", obtezba0, "Table"]

Out[44]= obtezba0
```

Takšno kreiranje podatkov sicer privede do najmanjše možne oblike vektorja obtežbe, vendar posledično vodi do najmanj uporabnega odziva, saj bo »pomanjkanje« točk vodilo do zelo popačene slike odziva. Za pridobitev realne slike odziva je torej potrebno pripraviti večje število podatkov. Ker je jasno, da je direktni zapis takih podatkov (zaradi obsega dela) nesmiseln (a mogoč), se bodo taki podatki iz demonstracijskih razlogov pripravili na različne načine.
Skupni podatek pa bo število časovnih podintervalov, s katerimi se bo delila celotna perioda obtežbe. Na osnovi tega podatka pa se lahko tudi izračuna trajanje takšnega podintervala:

```
In[45]:= Nint = 100;
         ΔT = T / Nint;
```

Ker je osnovna oblika obtežbe očitno podana v dveh različnih oblikah oz. delih, se tako definira vrednost obtežbe za prvi del kot dvostolpčni vektor. V prvi koloni so vrednosti časa, v drugi pa pripadajoče vrednosti obtežbe. Tak vektor se definira z ukazom *Table[{vrednost v prvem stolpcu, vrednost v drugem stolpcu},{neodvisna spremenljivka, začetna meja, končna meja, korak}]*:

```
In[47]:= del1 = Table[{t, P0}, {t, 0, T / 2, ΔT}];
```

ter drugi del:

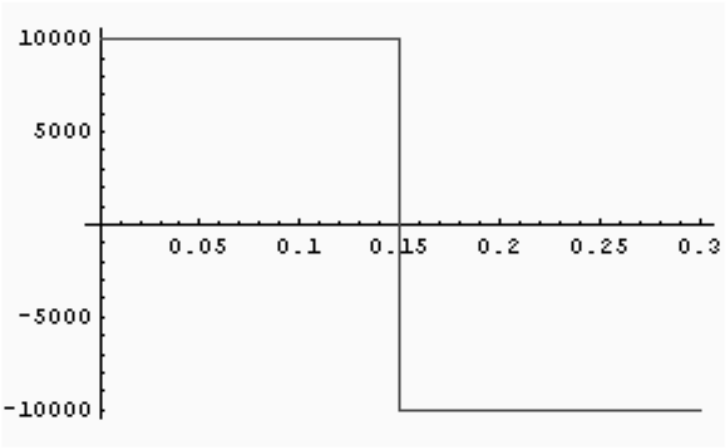
```
In[48]:= del2 = Table[{t, -P0}, {t, T / 2, T, ΔT}];
```

Celotna obtežba v prvi periodi pa se dobi z združitvijo obeh (delnih) obtežb, kar se izvede z ukazom *Join[spisek_vektorjev]:*

```
In[49]:= FazaI = Join[del1, del2];
```

ki se lahko tudi izriše (zaradi lažje primerjave npr. z osnovno rdečo barvo):

```
In[50]:= ListPlot[FazaI, PlotJoined → True,
           PlotStyle → RGBColor[1, 0, 0]];
```



Druga faza se od prve faze razlikuje zgolj po temu, da je čas povečan za periodo T, in tako se lahko členi druge faze pridobijo tako, da se abscisam, ki so zapisane kot členi prve kolone oz. stolpca, prišteje vrednost trajanja periode T. Da pa se lahko prištejejo vrednosti zgolj abscisam, je potrebno vektor najprej transponirati, z ukazom *Transpose[vektor]*, nato pa členom (sedaj) prve vrstice prišteti vrednost T. Vrednosti ordinat (zapisane v drugi vrstici) se sicer ne spremenijo, vendar ker bi preprosto prištetje vrednosti T v obliki skalarja spremenilo vse člene (tako abscise kot tudi ordinate), je potrebno prišteti vektor, katerega drugi člen je enak nič. Zaradi podobnosti imen *FazaI* in *FazaII Mathematica* sicer izpiše opozorilo o morebitni napaki, operacijo pa vseeno izvede korektno:

```
In[51]:= FazaII = Transpose[FazaI] + {T, 0};

General::spelll :
 Possible spelling error: new symbol name "FazaII" is
   similar to existing symbol "FazaI".
```

Zaradi nadaljnjih operacij je potrebno sedaj ležeče postavljen vektor spet povrniti v pokončno lego, kar se izvede z ukazom *Transpose[vektor]*:

```
In[52]:= FazaII = Transpose[FazaII];
```

Podobni koraki se izvedejo še za tretjo fazo, le da se lahko vsi kar združijo v eno samo vhodno vrstico (zaradi podobnosti uporabljenih imen se ponovno izpiše opozorilo):

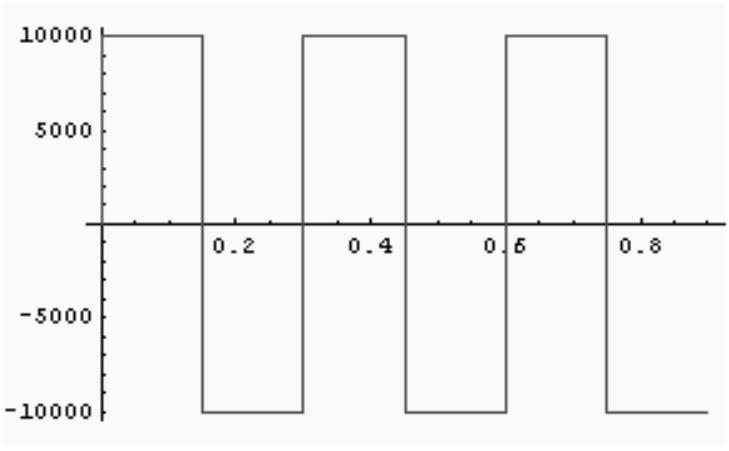
```
In[53]:= FazaIII = Transpose[Transpose[FazaII] + {T, 0}];

General::spelll :
 Possible spelling error: new symbol name "FazaIII" is
   similar to existing symbol "FazaII".
```

Celotno obtežbo pa tvorijo vsi trije nizi oz. faze skupaj, doda pa se še začetna, ničelna vrednost obtežbe, definirana kot vektor. Uporabi se ukaz *Join[vektor1,vektor2,..]*:

```
In[54]:= obtezba1 = Join[{{0, 0}}, FazaI, FazaII, FazaIII];
```

V korektnost opisa obtežbe se je mogoče najenostavneje prepričati z njenim izrisom v izbrani rdeči barvi, slika pa se shrani npr. v spremenljivko *pristop1*:

```
In[55]:= pristop1 = ListPlot[obtezba1, PlotJoined → True,
           PlotStyle → RGBColor[1, 0, 0]];
```



Generirani podatki pa se še shranijo na disk za kasnejšo uporabo:

```
In[56]:= Export["obtezba1", obtezba1, "Table"]

Out[56]= obtezba1
```

Na tak način je bilo sedaj generirano bistveno večje število podatkov kot v direktnem pristopu, kljub temu pa je bil način njihovega kreiranja že elegantnejši in krajši.
Obstajajo pa še drugačni načini, ki omogočajo še hitrejše kreiranje takega niza podatkov, npr. z avtomatskim združevanjem podatkov nizov za posamezne faze. Zato se najprej definira spremenljivka, ki definira število vseh obravnavanih faz oz. period:

```
In[57]:= Nfaz = 3;
```

nato pa se še definira vektor, kjer bodo shranjeni vsi diskretni pari obtežbe (torej vseh faz obtežbe skupaj). V prvi stolpec se zapišejo vrednosti časa, v drugo kolono pa npr. vrednosti 0, ker se bodo vanjo šele naknadno zapisale vrednosti obtežbe:

```
In[58]:= obtezba2 = Table[{t, 0}, {t, 0, Nfaz T - ΔT, ΔT}];
```

Sedaj se s pomočjo dvojne zanke Do v obliki *Do[operacije,{iterator1,začetna vrednost, končna vrednost},{oterator2,začetna vrednost, končna vrednost}]* izvede prenos podatkov o obtežbi iz že prej definiranih vektorjev *del1* in *del2* (posamezne operacije se ločijo s podpičjem). Prvi iterator, spremenljivka *i*, zajame vse faze; drugi iterator, spremenljivka *j*, pa zajame vse člene znotraj obravnavane faze.

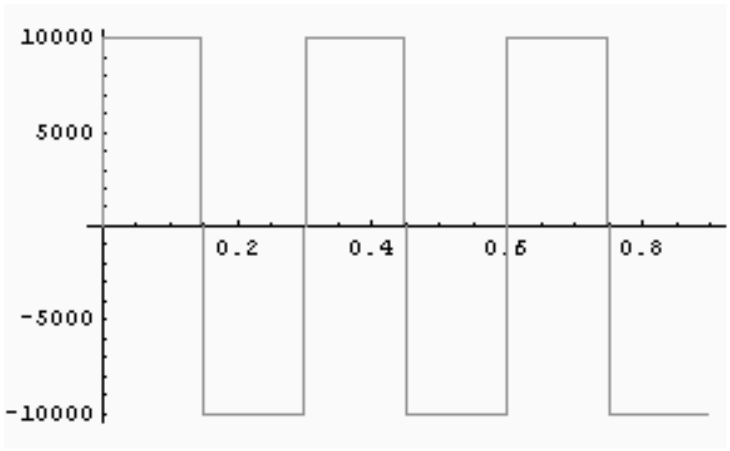
```
In[59]:= Do[obtezba2[[(i - 1) Nint + j, 2]] = del1[[j, 2]];
          obtezba2[[(i - 1) Nint + Nint / 2 + j, 2]] = del2[[j, 2]],
          {i, 1, Nfaz}, {j, 1, Nint / 2}]
```

Tako generiranemu nizu se doda še začetna vrednost ob času 0:

```
In[60]:= obtezba2 = Join[{{0, 0}}, obtezba2];
```

in celoten niz se lahko tudi izriše, npr. v osnovni zeleni barvi:

```
In[61]:= pristop2 = ListPlot[obtezba2, PlotJoined → True,
            PlotStyle → RGBColor[0, 1, 0]];
```



ter shrani na disk:

```
In[62]:= Export["obtezba2", obtezba2, "Table"]

Out[62]= obtezba2
```

Tak pristop že elegantneje omogoča zapis obtežbe preko poljubnega števila faz (in ne ravno za 3), deluje pa na principu dodelitve vrednosti obtežbe k že definiranim diskretnim vrednostim časa.

Pristop je mogoče modificirati tudi tako, da se v vektor obtežbe dodajajo celotni bloki parov podatkov, torej hkrati podatki o času in pripadajoči obtežbi, definirani v vektorjih *del1* in *del2*. Zato se najprej definira začetna vrednost vektorja obtežbe (brez definirane začetne vrednosti vektorja ni mogoče združiti z drugim):

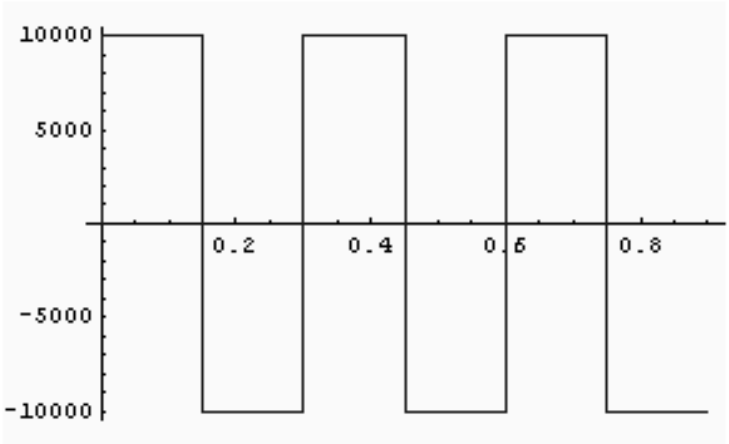
```
In[63]:= obtezba3 = {{0, 0}};
```

nato pa se dodajata posamezna dela prve periode, ustrezno premaknjena v času (zaradi podobnosti imen originalnih in transponiranih vekorjev *Mathematica* izpiše obvestilo o morebitni napaki):

```
In[64]:= Do[del1T = Transpose[Transpose[del1] + {(i - 1) T, 0}];
          del2T = Transpose[Transpose[del2] + {(i - 1) T, 0}];
          obtezba3 = Join[obtezba3, del1T, del2T], {i, 1, Nfaz}]

General::spell1 :
 Possible spelling error: new symbol name "del1T" is
   similar to existing symbol "del1".

General::spell1 :
 Possible spelling error: new symbol name "del2T" is
   similar to existing symbol "del2".
```

in vektor obtežbe je že pripravljen na izris (npr. v osnovni modri barvi):

```
In[65]:= pristop3 = ListPlot[obtezba3, PlotJoined → True,
           PlotStyle → RGBColor[0, 0, 1]];
```



in shranjevanje:

```
In[66]:= Export["obtezba3", obtezba3, "Table"]

Out[66]= obtezba3
```

To rutino pa je mogoče še direktno pospešiti, in sicer tako, da se ne dodajata posamezna dela prve periode (vektorja *del1* in *del2*), ampak kar celotna prva perioda (vektor *FazaI*). Zato se (zaradi primerjave rezultatov) ponovno definira začetni člen podobnega imena (kar posledično ponovno povzroči izpis obvestila o morebitni napaki):

```
In[67]:= obtezba3b = {{0, 0}};

      General::spelll : Possible spelling
         error: new symbol name "obtezba3b" is
         similar to existing symbol "obtezba3".
```

nato pa se temu definiranemu členu dodaja še celotna obtežba prve faze (podobnost imen povzroči pričakovano reakcijo *Mathematice*, torej obvestilo o morebitni napaki):

```
In[68]:= Do[FazaIT = Transpose[Transpose[FazaI] + {(i - 1) T, 0}];
          obtezba3b = Join[obtezba3b, FazaIT], {i, 1, Nfaz}]

      General::spell :
       Possible spelling error: new symbol name "FazaIT" is
         similar to existing symbols {FazaI, FazaII}.
```

Seveda je mogoče tudi to obtežbo izrisati in tako izvesti (ne popolno) grafično kontrolo, vendar je še natančnejša direktna oz. neposredna primerjava obeh vektorjev obtežbe. Taka kontrola se lahko naredi sicer tako, da se oba vektorja odštejeta in v primeru enakosti obeh vektorjev mora slediti vektor, kjer so vsi členi enaki nič. Z *Mathematico* pa je to mogoče storiti še elegantneje z uporabo logičnega operaterja za primerjavo *leva stran==desna stran*. Primerjava obeh strani se zagotovi z dvojnim enačajem = =), uporaba enojnega enačaja bi namreč zgolj levi strani priredila vrednost desne strani.

```
In[69]:= obtezba3 == obtezba3b

Out[69]= True
```

Obravnavana obtežba pa se lahko definira še na drugačen način, če se definira funkcija, ki zavzame vrednost 1, kadar se opazovani čas *t*, ki je neodvisni parameter, nahaja v prvi polovici periode *T*, in vrednost –1, kadar se čas nahaja v drugi polovici periode. Funkcija se imenuje npr. *vrednost*, njena parametra pa sta čas *t* in trajanje periode *T*. Parametroma sledi _, kar pomeni, da ju definiramo kot lokalni spremenljivki (funkcija se tako lahko uporabi tudi za druge parametre in ne zgolj za t in T). Nato sledi ukaz *If[pogoj, operacija ob izpolnitvi pogoja, operacija ob neizpolnitvi pogoja]*.
Če bi se funkcija želela definirati zgolj za čase znotraj prve periode *T*, bi se pogoj v *If* ukazu npr. zapisal kar kot *t<T/2*, ker pa se želi funkcija zapisati splošneje, torej tudi za čase t>T, je potrebno pogoj zapisati splošneje. Razmislek pokaže, da je mogoče zaznati velikost obtežbe za poljubni čas tako, da se obravnavani čas *t* deli s trajanjem periode *T*. Če je decimalni del rezultata manjši od 0.5, pripada opazovanemu času pozitivna vrednost obtežbe, v nasprotnem primeru pa negativna.
Zaznavanje, ali se čas nahaja v prvi ali drugi polovici posamezne periode (torej v prvi ali drugi fazi), se tako izvede s primerjavo rezultatov dveh ukazov. Prvi ukaz je *IntegerPart[argument]*, ki vrne zgolj celoštevilsko vrednost argumenta (torej brez zaokroževanja). Drugi ukaz je *Round[argument]*, ki

argument zaokroži na celoštevilsko vrednost (vrednosti, manjše od 0.5, zaokroži navzdol, ostale vrednosti pa navzgor). Če je opazovani čas v prvi polovici periode, med celoštevilsko vrednostjo in zaokroženo vrednostjo seveda ni razlike:

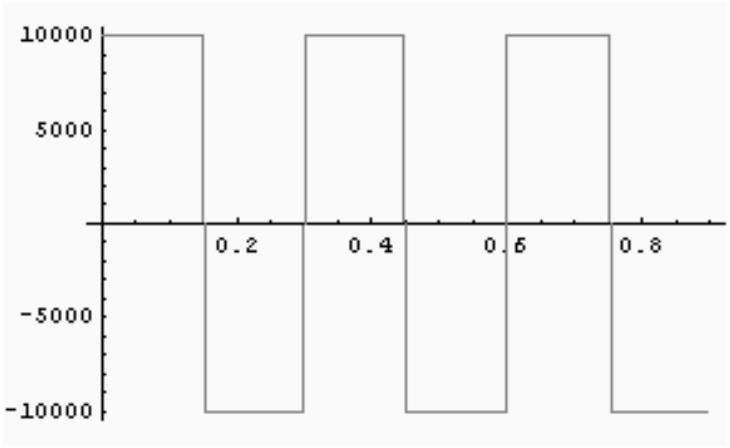
```
In[70]:= vrednost[t_, T_] :=
           If[IntegerPart[t / T] == Round[t / T], 1, -1]
```

Obtežba se sedaj direktno generira v vektor s pomočjo zanke po času, ki zajame vse faze naenkrat:

```
In[71]:= obtezba4 = Table[{t, P0 vrednost[t, T]},
             {t, 0, Nfaz T - ΔT, ΔT}];
```

Tudi tako definirana funkcija se lahko izriše:

```
In[72]:= pristop4 = ListPlot[obtezba4, PlotJoined → True,
             PlotStyle → RGBColor[1, 0.2, 1]];
```



ter seveda shrani na disk:

```
In[73]:= Export["obtezba4", obtezba4, "Table"]

Out[73]= obtezba4
```

Ker so se vsi izrisi sproti shranjevali v ločene spremenljivke, jih je mogoče sedaj zgolj za primerjavo vse prikazati hkrati:

```
In[74]:= Show[pristop0, pristop1, pristop2, pristop3, pristop4];
```